The RSA Cryptosystem: Factoring the public modulus

Debdeep Mukhopadhyay

Assistant Professor
Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur
INDIA -721302

Objectives

- **The Pollard p-1 Algorithm**
- **The Pollard RHO Algorithm**
- **Dixon's Random Squares Algorithm**

Factoring Algorithms

- Most obvious way to attack RSA would be to try to factor the public modulus, n
- Modern Algorithms: Quadratic Sieve, Elliptic Curve Factoring Sieve, Number field Sieve.
- Other well-known algorithms: p-1 algorithm, Pollard's rho algorithm etc.
- Of course we have trial division.

Complexity of Trial Division

- If n is composite, then n has a prime factor less than $\sqrt{n}$.
- Good if n is less than 2^{40}.
- We need to do better than trial division for larger composite numbers
- We shall study two algorithms.
- Note we are just searching for a non-trivial factor.
- If we desire for complete prime factorizations, then we need to test for primality of the obtained factors, and if composite further factorize them

The Pollard p-1 algorithm

POLLARD $p-1$ FACTORING ALGORITHM(n, B)

```
a ← 2
for j ← 2 to B
  do a ← a^j mod n
d ← gcd(a − 1, n)
if 1 < d < n
  then return (d)
  else return ("failure")
```

- **Two inputs:**
  n: odd integer
  B: Prescribed bound

Explanation of the Algorithm

- **Suppose p is a prime divisor of n.**
- **Consider the prime factors of (p-1)**
- **Suppose for every prime power q|(p-1), q≤B**

Prime Facorization of (p-1):

$$(p-1) = q_1^{e_1} q_2^{e_2} \ldots q_k^{e_k}$$

wlog let $q_1^{e_1} < q_2^{e_2} < \ldots < q_k^{e_k} \leq B$

then, $(p-1) \mid B!$

This is because, all the prime powers exist in the terms of B! at least once.

At the end of the for loop, the algorithm computes:

$a \equiv 2^{B!} (\text{mod } n)$.

Hence, $a = kn + 2^{B!}$, where k is an integer.

Now, n=pq. Thus, $a = kpq + 2^{B!}$.

Thus, $a \equiv 2^{B!} (\text{mod } p)$.

Since, we have $2^{p-1} \equiv 1 (\text{mod } p)$ and (p-1)|B!

$\Rightarrow a \equiv 2^{B!} \equiv 1 (\text{mod } p)$

Thus, p|(a-1) and p|n, thus p|gcd(a-1,n).

Thus we have a non-trivial factor of n, unless a=1.

Example

- **n=15770708441**
- **Set, B=180**
- **a=11620221425**
- **d=gcd(a-1,n)=135979**
- **1577078441=135979x115979**

Finer Points

- **There are B-1 modular exponentiations each requiring at most $2\log_2 B$ modular multiplications, using square and multiply.**
- **The gcd can be computed in $O(\log_2 n)^3$ using the Extended Euclidean algorithm.**
- **Overall complexity=$O(B\log B(\log n)^2+(\log n)^3)$. If $B=O(\log n)^l$, then we have a polynomial time algorithm.**
- **However, if B increases the success probability increases, but the algorithm becomes as slow as the trial division.**
- **Hence, the modulus n should be such that p-1 does not have all prime powers small.**

Pollard's Rho Method

- **Say, n=7171**
  - **What is p|n? (We know that $p \le \sqrt{n}$)**
  - A possible method: **Start picking up a and b at random ($0 \le a,b<n$). Since, p is small there is a good chance that $a \equiv b \pmod p$. Thus p|(a-b) and we know p|n.**
  - **Thus, gcd(a-b,n) gives a non-trivial factor of n.**
  - **From Birthday paradox, if the number of elements picked are $O(\sqrt{p})$, then we have a large chance of a collision.**

Number of gcd computations too large

- **Pick a and b: compute gcd(a,b)**
- **Pick up c: compute gcd(a,c), gcd(b,c)**
- **Pick up d: compute gcd(d,a),gcd(d,b),gcd(d,c)**
- **Thus if |X|=O(√p) is the number of elements chosen, number of gcds is:**

$$C_2^{|X|} = O(p) = O(\sqrt{N})$$
$$Memory = O(\sqrt{N})$$
$$Time = O(\sqrt{N})$$

Improvement

- **We wish to compute less gcd's.**
- **We choose a polynomial f(x)=x²+a, to randomly choose the numbers mod n.**
  - **note a is not 0 or -2 mod n. Why?**

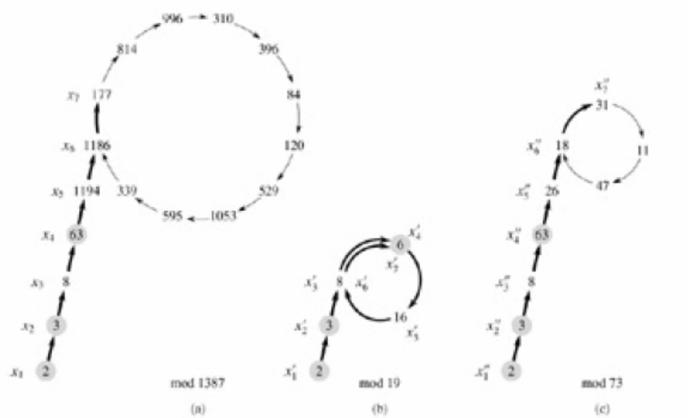
Suppose, $x_i \equiv x_j (\text{mod p}) \Rightarrow f(x_i) \equiv f(x_j) \bmod p$

$\because x_{i+1} \equiv f(x_i) \bmod n$, we have $x_{i+1} \bmod p \equiv [f(x_i) \bmod n] \bmod p \equiv f(x_i) \bmod p$

Similarly, $x_{j+1} \bmod p \equiv [f(x_j) \bmod n] \bmod p \equiv f(x_j) \bmod p \equiv x_{i+1} \bmod p$

Repeating, if $x_i \equiv x_j \bmod p$, we have $x_{i+\delta} \equiv x_{j+\delta} \bmod p, \forall \delta \ge 0$

Looks like the letter ρ (rho)



mod 1387	mod 19	mod 73

Reducing number of gcds

- **Our goal is to find two terms $x_i \equiv x_j \pmod p$, $i<j$.**

$x_{i+\delta} \equiv x_{j+\delta} \bmod p, \forall \delta \geq 0$

$l = j - i,$ and l is the length of the cycle.

Now in l consecutive terms,

$x_i, x_{i+1}, \ldots, x_{j-1}$

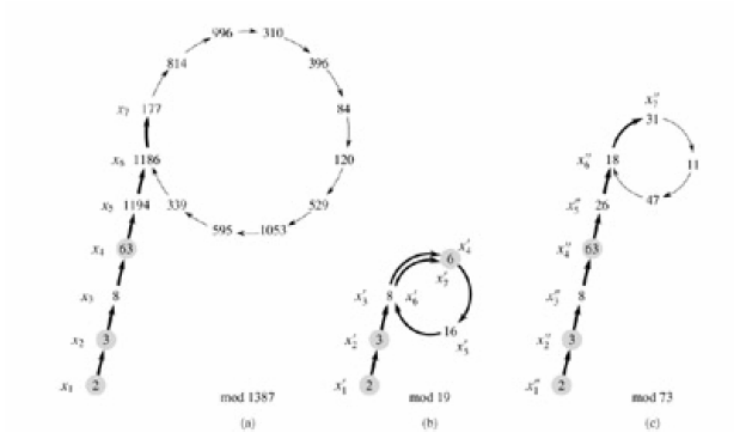
there is one index say i' which is divisible by l.

If $l \mid i' \Rightarrow l \mid (2i' - i')$

Thus as $i' > i$ and $(2i' - i')$ is a multiple of l,

$x_{2i'} \equiv x_{i'} \pmod p$

Thus we compute gcd only when the current index is even and $d = \gcd(x_{2i} - x_i, n)$ gives a non-trivial factor of n.



- **Consider, x'_3, x'_4, x'_5 in the cycle for mod 19, there is one index namely 3 which is divisible by 3, the cycle length. So, gcd(x_6-x_3,1387)=gcd(1186-8,1387)=19.**

The Pollard Rho Algorithm

POLLARD RHO FACTORING ALGORITHM(n, x_1)

external f
$x \leftarrow x_1$
$x' \leftarrow f(x) \bmod n$
$p \leftarrow \gcd(x - x', n)$
while $p = 1$
do
- **comment:** in the ith iteration, $x = x_i$ and $x' = x_{2i}$
- $x \leftarrow f(x) \bmod n$
- $x' \leftarrow f(x') \bmod n$
- $x' \leftarrow f(x') \bmod n$
- $p \leftarrow \gcd(x - x', n)$

if $p = n$
  then return ("failure")
  else return (p)

Example

Suppose n=7171=71×101, $f(x) = x^2 + 1, x_1 = 1$

The sequence of x_i's begins as follows:

1	2	5	26	677	6557	4105	
6347	4903	2218	219	4936	4210	4560	
4872	375	4377	4389	2016	5471	88	574

The above values when reduced modulo 71 are:

1	2	5	26	38	25	58
28	4	17	6	37	21	16
44	20	46	58	28	4	17

The first collision in the above list is:

$x_7 \bmod 71 = x_{18} \bmod 71 = 58$

Since, (18-7)=11, therefore the algorithm computes at some stage $\gcd(x_{11} - x_{22}, 71) = \gcd(574 - 219, 7171)$

$= 71$

Complexity

- **You have to compute gcd j number of times.**
- **From Birthday Paradox, maximum value of j is $O(\sqrt{p})=O(n^{1/4})$**

Dixon’s Random Squares Algorithm

- **Simple Idea**

Suppose we can find, $x \neq y (\mathrm{mod}\, n), st.\ x^2 = y^2 (\mathrm{mod}\, n)$.
Then, $n \mid (x-y)(x+y)$.
But neither (x-y), nor (x+y) is divisible by n.
Hence, gcd(x+y,n) is a non-trivial factor of n.
So, is gcd(x-y,n).
Consider, n=77. Choose 10 and 32, as
$10^2 \equiv 32^2 (\mathrm{mod}\, 77)$, but $10 \neq 32 (\mathrm{mod}\ 77)$.
Computing gcd(10+32,77)=7 gives us one factor
of n=77.

Dixon’s Random Squares Algorithm

Suppose, n=1829.
Consider a factor base, B={-1,2,3,5,7,11,13}
Compute, $\sqrt{kn} = \{42.77, 60.48, 74.07, 85.53\}$.
We take, z={42,43,61,74,85,86}.
Consider the following congruences modulo n,

$z_1^2 \equiv 42^2 \equiv -65 = (-1)(5)(13)$

$z_2^2 \equiv 43^2 \equiv 20 = (2)^2(5)$

$z_3^2 \equiv 61^2 \equiv 63 = (3)^2(7)$

$z_4^2 \equiv 74^2 \equiv -11 = (-1)(11)$

$z_5^2 \equiv 85^2 \equiv -91 = (-1)(7)(13)$

$z_6^2 \equiv 86^2 \equiv 80 = (2)^4(5)$

Considering the congruence,

$(42 \times 43 \times 61 \times 85)^2 \equiv (2 \times 3 \times 5 \times 7 \times 13)^2 (\mathrm{mod} 1829) \Rightarrow$

$\Rightarrow 1459^2 \equiv 901^2 \Rightarrow \mathrm{gcd}(1459 + 901, 1829) = 59$

References

- **D. Stinson, Cryptography: Theory and Practice, Chapman & Hall/CRC**

Next Days Topic

- **Some Comments on the Security of RSA**