Some Comments on the Security of RSA

Debdeep Mukhopadhyay

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

Objectives

- **Computing Φ(n) is no easier than factoring n.**
- **Computing the Decryption exponent is no easier than factoring n.**
- **Bit Security of RSA.**

Computing Φ(n)

- **For if n and Φ(n) are known, and n is the product of two primes p, q:**
  - **then n can be factored by solving:**

$$n=pq$$

$$\Phi(n)=(p-1)(q-1)$$

Combining we obtain:

$$p^2-(n-\Phi(n)+1)p+n=0$$

The roots are p and q.

Decryption Exponent

- **If the decryption exponent is known, then n can be factored:**
  - **we will see a randomized algorithm**
  - **there is a deterministic algorithm published in Crypto 2004 by Alexander May if:**
    - **a and b are of the same bit size**
    - **$ab < n^2$**
    - **Run Time: $O(\log^2 n)$**

Importance of this result

- **This result is important from practical point of view:**
  - **if b (the RSA secret key) is leaked, then changing it does not suffice**
  - **one needs to change the modulus n.**
- **We shall study a Las Vegas Algorithm which can factor n with a probability of at least ½**

Non-trivial Square Roots

- **How many roots does $x^2 \equiv 1 \pmod{n}$ has, where n=pq?**
- **There are 4 roots:**

  Trivial Roots: $x = \pm 1 \pmod{n}$

  Other two are Non-trivial roots

Example

n=403=13 x 31

y^2=1 (mod 403) is equivalent to the system of equations:

y^2=1 (mod 13): there are two roots x=+1, -1 mod 13

y^2=-1 (mod 31) : there are two roots x=+1, -1 mod 13

Example

- **The root x=92 is a solution to:**
  - **x=+1 mod 13**
  - **x=-1 mod 31**
- **The root x=311 is a solution to:**
  - **x=-1 mod 13**
  - **x=+1 mod 31**

Example

- **If x is a non-trivial square root of 1 modulo n,**

  thus $x^2=1^2 \bmod n$, but $x \neq 1 \bmod n$

Then we compute the gcd of (x+1,n) and gcd(x-1,n) to get the factors of n.

Here, gcd(93,403)=31 and gcd(91,403)=13.

The Algorithm

1. choose w at random such that $1 \leq w \leq n-1$
2. compute $x = \gcd(w, n)$
3. **if** $1 < x < n$ **then quit** (success: $x = p$ or $x = q$)
4. compute $a = \mathrm{A}(b)$
5. write $ab - 1 = 2^s r$, r odd
6. compute $v = w^r \bmod n$
7. **if** $v \equiv 1 \pmod{n}$ **then quit** (failure)
8. **while** $v \not\equiv 1 \pmod{n}$ **do**
9. $\quad v_0 = v$
10. $\quad v = v^2 \bmod n$
11. **if** $v_0 \equiv -1 \pmod{n}$ **then**
    quit (failure)
    else
    compute $x = \gcd(v_0 + 1, n)$ (success: $x = p$ or $x = q$)

Analysis: Termination

- **We do the analysis for w not being a multiple of p or q.**
- **w is relatively prime to n:**
  - **we compute successive squares:**

    w^r, w^{2r}, w^{4r}, …, $w^{(2^s)r}$

    Now, since ab-1=(2^s)r=0 (mod Φ(n)),

    $w^{(2^s)r}$=1 mod n. Hence the while loop terminates after at most s iterations.

Analysis: Correctness

- **At the end of the while loop we have found out a v_0 st.:**

  $(v_0)^2$=1 mod n, v_0≠1 mod n

If v_0=-1 mod n, the algorithm fails.

Otherwise v_0 is indeed a non-trivial square root of 1. Hence, gcd(v_0+1,n) is a factor of n.

Partial Information of Plaintexts

- **Sometimes the attacker has modest goals:**
  - **to achieve partial information of the plaintext: partial break**
- **We shall consider two types of partial information of the plaintext bits:**
  - **Jacobi of the plaintext**
  - **Parity of the lowest bit of the plaintext**
  - **Whether the plaintext is less than or more than n/2, where n is the RSA modulus.**

Jacobi of the Plaintext

- **$y=x^b \bmod n$, where x is the plaintext.**
- **Note that b is co-prime to $\Phi(n)=(p-1)(q-1)$, which is even:**
  - **thus b is odd.**

From definition:

$$\left(\frac{y}{n}\right)=\left(\frac{x}{n}\right)^b$$

Note, $\left(\frac{x}{n}\right)=\pm 1,$ thus $\left(\frac{x}{n}\right)^b=\left(\frac{x}{n}\right),$

as b is odd.

Thus RSA leaks some information of the plaintext

Parity(y) and Half(y)

- **Parity(y) denotes the low order bit of x, that is parity(y)=0, if x is even and parity(y)=1 if x is odd.**
- **Half(y)=0, if $0 \leq x < n/2$ and half(y)=1 if $n/2 < x \leq n-1$.**

Reductions

- **Existence of a polynomial time algorithm that computes half(y)=>Existence of a polynomial time algorithm for RSA decryption.**
- **RSA Hard => Computing half(y) is hard.**

The Proof Idea

- **Let there be an oracle HALF, which computes half(y).**
  - **if half(y)=0, then x ε [0, n/2)**
  - **Now, $y=x^b \bmod n$. Compute,**
    $y=2^b y \bmod n$
    $=(2x)^b \bmod n$
  - **if half(y)=0, then 2x ε [0, n/2)**
    =>x ε [0, n/4) U [n/2, 3n/4)

Continuing in this fashion we obtain distinct boundaries of x. Then the actual x value, can be found out using binary search technique.

Algorithm

Algorithm: Oracle RSA Decryption(n,b,y)
external HALF
$k \leftarrow floor(\log_2 n)$
for i=0 to k,
{
$h_i = HALF(n, b, y)$
$y = y \times 2^b \pmod n$
}

Binary Search Routine

```
for i=0 to k
{
    mid=(hi+lo)/2
    if(h_i == 1)
        lo=mid
    else
        hi=mid
}
return floor(hi)
```

Parity ?

- **Computing parity(y) is polynomially equivalent to computing half(y):**
  - **half(y)=parity((y x $e_K(2)$) mod n)**
  - **parity(y)=half((y x $e_k(2^{-1})$) mod n)**

Proof Sketch

- **Parity((y x $e_k(2)$) mod n)**
  = Parity($e_k(2x)$ mod n)
  = 0, if 2x is even
  1, if 2x is odd

Now, n=2t+1, where t is an integer
and 0≤x<n/2 => 0 ≤ x ≤ t => 0 ≤ 2x ≤ 2t=n-1
All these number are even
If, n/2 ≤x ≤n-1 => t+1≤x ≤2t => 2t+2 ≤2x ≤4t
or, n+1 ≤2x ≤2n-2
Taking, modulo n we have: 1 ≤ 2x ≤ n-2
All these numbers are odd. This proves first eqn.

Points to Ponder

- **Prove that the algorithm to factor n, from the decryption exponent succeeds with at-least ½ probability.**
- **Prove the second relation to show that parity(y) and half(y) are polynomially equivalent.**

References

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

Next Days Topic

- **Discrete Logarithm Problem (DLP)**