Project Report on RSA Attack

Boshi An, Bingxuan Li

June 2022

1 Introduction

The RSA cryptography system [2] had been the most commonly used system providing security and privacy for a variety of scenes. It is used by web services to secure all the email we send, all the password we type in, all the private sites we browse and all the payment we offer. Without such a cryptography system, privacy and security would be no where to discuss.

There is never a truly secure system. With the power of quantum computation, attacking RSA is a problem yet proved to be in the class QBP , which can be solved in polynomial time. More precisely, a n bit key can be cracked in $O(n^2 \log n \log\log n)$ time. However, in the recent future, practical quantum computation is not capable of having any impact on our data security and privacy.

However, there are still some classical ways to crack a mere portion of vulnerable RSA keys.

In this project, we focus on the evil side, which we will try our best to crack 21 RSA keys generated in vulnerable but different ways.

1.1 Fundamental Concepts

To encrypt a message represented by an integer m, the RSA system requires two primes p, q whose product is larger then m (usually as large as 1024 or more bits), and an integer e (can be as small as 3) as the *encryption exponent*. Let $N = pq$, then the cipher text will be $m^e \bmod N$.

To decipher an encrypted message m^e , one need an integer d as the *decryption exponent* . d should satisfy $m^{ed} \bmod N = m$.

One simplified transmission protocol can be described as below:

- Alice want to send Bob m.
- Bob secretly generates p, q and announces an arbitrary e along with the product of $N = pq$ publicly.
- Alice encrypts her message m with the public *encryption exponent* e . The data she sends to Bob is $c = m^e \bmod N$.

- Bob receives the encrypted message c , and decipher it using the decryption exponent d he calculated secretly. The decoded message is $m' = c^d \bmod N = m^{ed} \bmod N$.

Since Bob knows p, q, e , he can efficiently compute d to satisfy $ed \equiv 1 \pmod{\phi(N)}$. By Euler's theorem, this d will recover the m , since $m^{ed} = 1$.

1.2 Practice Routines

In practice, there are several routines a yet still secure RSA cryptography system should follow:

1. The product N need to be large. A typical size is 1024 .
2. The private primes p, q need to be large but not too close. A typical separation is $|p - q| = \Theta(N^{1/2})$
3. The private primes must satisfy: $p-1$ is not B-smooth for all B too small. (*i.e.* have at least one prime factor larger than B.)
4. The message m should be concatenated with some random prefix. The resulting sequence should match the length of N .
5. In multiple transactions, never use the same padding prefix.
6. In multiple transactions, never send the same message with different N and same e , or same N and different e.
7. In multiple transactions, a prime can be only used once.
8. Make the message difficult to guess when given part of it. One solution is to insert random characters.

The reason why each of the rules exists is due to the corresponding attack methods.

2 Project Task

The task consists of 21 transactions, each transaction is an encrypted message sending by Alice and intercepted by Eve (us) . Our task is to decipher all the transactions, evilly.

2.1 Factorizing Attack

This type of attack aims on factorizing N thus knowing everything about the transmission.

2.1.1 Pollard Rho

For task 1,3 and 12. 12 takes longer time (about 5 seconds).

Pollard Rho algorithm is a famous factorizing algorithm. We first use $f(x) = (x^2 + c) \bmod N$, as our pseudo random number generator, and we set two variables $t, r (t \neq r$ initially). In each loop, if $\gcd(|t - r|, N) > 1$, then N is factorized. Otherwise we update t and r: $t = f(t), r = f(f(t))$. If $t == r$, then we choose another c and return to the beginning. It runs at expectation time of $O(\min\{p, q\}^{1/2})$

However the given N in all tasks except 3 of them all have no small primes, thus unable to factorize easily.

This method suggests us follow routine 2.

2.1.2 Pollard's p-1

For task 1 and 3. More effective than Pollard Rho.

This method assumes that $p-1$ is B-smooth (*i.e.*, have prime factors all less then a small number B) . This allows us to compute $2^{B!} \equiv 2^{p-1} \equiv 1 (\text{mod p})$, so $p|2^{B!} - 1$. Thus we can get p by simply calculating $\gcd(2^{B!} - 1, N)$.

This method suggests us follow routine 3.

2.1.3 GCD Attack

For task 2 and 19.

This method discovers primes that are used in multiple transactions. If $\gcd(N_1, N_2) > 1$ for N_1 and N_2, we can simply factorize both of them using Euclid's method in a short time.

This method suggests us follow routine 7.

2.1.4 Squaring Attacks

For task 8 and 10.

Squaring attacks include Fermat attack and Londahl attack. These methods factorizes N if p, q are too close. By enumerating p, q from the middle, we can efficiently find them.

This method suggests us follow routine 2.

2.2 Multi-transaction Attacks

This type of attack reveals loopholes within multiple transactions.

2.2.1 Common Modulus Attack

For task 11 and 14.

If a single message is encrypted with same N and $\gcd(e_1, e_2) = 1$, then we can efficiently recover m by calculating s_1, s_2 such $s_1 e_1 + s_2 e_2 = 1$ and then $m = c_1^{s_1} c_2^{s_2} \bmod N = m^{e_1 s_1 + e_2 s_2} \bmod N = m$.

This method suggests us follow routine 6.

2.2.2 Hastad Broadcast Attack

For task 4, 5, 7, 13 and 18.

If a single message is encrypted with different N s and same e , then we can efficiently recover m using CRT (Chinese Remainder Theorem) . Suppose $m^e \equiv c_i \pmod{N}_i$, by CRT, we can have $m^e \equiv x \pmod{\prod N_i}$. If $\prod N_i > m^e$, then simply compute the e-th root of x would produce m .

This method suggests us follow routine 6.

2.2.3 Coppersmith's Linear Padding Attack

For task 0, 6, 17.

Assume the padding procedure of encoding is simply a linear transformation of m , which is $c = (m + s)^e$, here s is the padding string. Our target is to find the root x of the polynomial $f(x) = (x + s)^e - c$.

We can create a lattice with basis $f(x)^u x^v N^{m-u}$ where $0 \leq u \leq m, 0 \leq v \leq e-1$ and m is a predefined constant relative to the precision of the algorithm. It's obvious that in the lattice spanned by this basis, each corresponding polynomial has a root m . By applying LLL algorithm [1] to the lattice, a polynomial with coefficients all smaller than $N^{1/m+\epsilon}$ can be found, thus the root of the polynomial can be computed by considering it in $\mathbb{Z}$ rather than $\mathbb{Z}_N$.

This method suggests us follow routine 4 and 5.

3 Conclusion

3.1 Project Result

We finally solved 17 of 21 total tasks. The remaining tasks have the same padding as others (which is a loophole from ignoring routine 5 and 8), thus can be guessed and verified. The final result of the attack forms a sentence:

My secret is a famous saying of Albert Einstein. That is "Logic will get you from A to B. Imagination will take you everywhere."

3.2 Further Thinking

This project focuses on a small area of cryptography, which is the hacking of a specific protocol. This aim encourages students to dig deeper in to the area, however, may allow us exploit open-source software without knowing how it works. For example, I have found a Github repository implemented every single method we need in this project here.

3.3 Workload

- Boshi An:
  - Implemented the methods in section 2.2
    (The code is uploaded here)

  - Prepared and gave a half of the presentation.
  - Wrote the most part of the final report.

- Bingxuan Li:
  - Implemented the methods in section 2.1
    (The code is uploaded here)
  - Prepared and gave a half of the presentation.
  - Appended a part of the final report.

References

[1] Arjen K. Lenstra. Factoring multivariate polynomials over algebraic number fields. *SIAM Journal on Computing*, 16(3):591–598, 1987.

[2] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, feb 1978.