Attacks on Textbook RSA

By Diya Bansal

October 2020

1 Introduction

[1] Invented by Ron Rivest, Adi Shamir, and Len Adleman [2], the RSA cryptosystem was first revealed in the August 1977 issue of Scientific American. The RSA is most commonly used for providing privacy and ensuring authenticity of digital data. RSA is used by many commercial systems. It is used to secure web traffic, to ensure privacy and authenticity of Email, to secure remote login sessions, and it is at the heart of electronic credit-card payment systems.

Since its initial release, the RSA has been analyzed for vulnerabilities. Twenty years of research have led to a number of intriguing attacks, none of them is devastating. They mostly show the danger of wrong use of RSA. Our objective is to explorer some of these attacks.

RSA encryption in its simple form is explained as follow. Let N = pq be the product of two large primes of the same size (n/2 bits each). As [2] explains, a typical size for N is n=1024 bits, i.e. 309 decimal digits. Let e, d be two integers satisfying ed = 1 mod φ(N) where φ(N) = (p-1) (q-1). N is called the RSA modulus, e is called the encryption exponent, and d is called the decryption exponent. The pair (N, e) is the public key. The pair (N, d) is called the secret key and only the recipient of an encrypted message knows it.

A message M is encrypted by computing $C = M^e$ mod N. To decrypt the ciphertext C, the authentic receiver computes C^d mod N. The following equality is based on Euler's theorem.

$$C^d = M^{ed} = M \pmod{N}$$

My Java coded implementation of the attacks in this report are at:
https://github.com/dabsgts1/CryptoAttacks

2 Attack With Small e

2.1 Description

As stated in the introduction, the pair (N, e) is the public key where e is known as the encryption exponent. A message M is encrypted by computing $C = M^e$ mod N.

$$\text{If } M < N^{1/e}, \text{ then } M^e < N^1.$$

In such a case, M does not undergo any modular reduction. Hence, M can be directly found by calculating the eth root of C. Therefore, M can be calculated without knowing the value of d, the decryption exponent. Encrypted messages are vulnerable to such attacks for small values of e which satisfy the condition $M^e < N^1$.

2.2 Implementation

Coding language: Java.

My Java coded implementation of this attack is at:
https://github.com/dabsgts1/CryptoAttacks/blob/master/Attack%20With%20Small%20e

```
import java.util.*;
class AttackWithSmallE
{
    public static void main()
    {
        Scanner s=new Scanner(System.in);
        System.out.println("Condition: M < \(N^{1/e}\)");
        System.out.println("Enter e");
        long e=s.nextLong();
        System.out.println("Enter C");
        long C=s.nextLong();
        double x1 = C;
        double x2 = C / e;
        while (Math.abs(x1 - x2) > 0.0001)
        {
            x1 = x2;
            x2 = ((e - 1.0) * x2 + C / Math.pow(x2, e - 1.0)) / e;
        }
        System.out.println("M = "+(long)x2);
    }
 }
```

2.3 Results

Due to limitation in the size of the Java variables, the following values were selected to test the implementation of the code:

Given:
$e = 3$
$C = 4188852928$

The output achieved on running the Java code on BlueJ IDE was:
$M = 1612$

3 Wiener's Attack for Small Exponent d

3.1 Description

As stated in the introduction, the pair (N, e) is the public key where e is known as the encryption exponent and N=pq, the product of two large primes of the same size. A message M is encrypted by computing $C = M^e$ mod N. If $d < N^{1/4}/3$, then Wiener's Attack can be implemented to find p, q, and d. As [3] and [4] explain in detail, Wiener's Attack makes use of the mathematical concepts of continued fractions and the Chinese Remainder Theorem.

3.2 Implementation

Coding language: Java.

3.2.1 Version 1: Decimal Version (less accurate)

My Java coded implementation of this attack:
https://github.com/dabsgts1/CryptoAttacks/blob/master/Wiener's%20Attack%20(Java)

3.2.2 Version 2: Fractional Version (more accurate)

My Java coded implementation of this attack:
https://github.com/dabsgts1/CryptoAttacks/blob/master/Wiener's%20Attack%20 2%20(Java))

3.3 Results

The following examples have been taken from [3] and [4]. Due to limitation in the size of the Java variables, the following values were selected to test the implementation of the code:

3.3.1 Example 1

Given:
$N = 90581$
$e = 17993$

The output achieved on running the Java code on BlueJ IDE was:
$p = 379$
$q = 239$

$d = 5$
$k = 1$
$\varphi(N) = 89964$
$d < 5.782798011487551 = N^{1/4}/3$

3.3.2 Example 2

Given:
$N = 160523347$
$e = 60728973$

The output achieved on running the Java code on BlueJ IDE was:
$p = 13001$
$q = 12347$
$d = 37$
$k = 14$
$\varphi(N) = 160498000$
$d < 37.520040361274155 = N^{1/4}/3$

4 Chinese Remainder Theorem Attack

4.1 Description

As stated in the introduction, the pair (N, e) is the public key where e is known as the encryption exponent. [5] Let e = 3 and let's say that M was sent to three different parties holding 3 different public keys:

$$pk_1 = (N_1, 3)$$
$$pk_2 = (N_2, 3)$$
$$pk_3 = (N_3, 3)$$

An eavesdropper can obtain the following corresponding ciphertexts:

$$C_1 = [M^3 \bmod N_1]$$
$$C_2 = [M^3 \bmod N_2]$$
$$C_3 = [M^3 \bmod N_3]$$

Assuming that gcd(N_i,N_j) $\neq$ 1 for all (i, j), let N*=$N_1N_2N_3$. An extended version of the Chinese Remainder Theorem states that there exists a unique $\hat{c} < N*$ such that:

$$\hat{c} = C_1 \bmod N_1$$
$$\hat{c} = C_2 \bmod N_2$$
$$\hat{c} = C_3 \bmod N_3$$

With $\hat{c}$, the message M can be decrypted and RSA is, thus, cracked. A detailed explanation of the Chinese Remainder Theorem can be found at [6].

4.2 Implementation

Coding language: Java.

My Java coded implementation of this attack is at:
https://github.com/dabsgts1/CryptoAttacks/blob/master/Chinese%20Remainder %20Theorem%20Attack

4.3 Results

The following example has been taken from [7]. Due to limitation in the size of the Java variables, the following values were selected to test the implementation of the code:

Given:
$C_1 = 330 \pmod{377}$
$C_2 = 34 \pmod{391}$
$C_3 = 419 \pmod{589}$

The output achieved on running the Java code on BlueJ IDE was:
M=102

5 Common Modulus Attack

5.1 Description

As stated in the introduction, the pair (N, e) is the public key where e is known as the encryption exponent. Let there be a message M which is encrypted with two different public keys where gcd(e_1,e_2) = 1 [5].

$$pk_1 = (N, e_1)$$
$$pk_2 = (N, e_2)$$

An eavesdropper can obtain the following corresponding ciphertexts:

$$C_1 = M^{e_1} \bmod N$$
$$C_2 = M^{e_2} \bmod N$$

[8] Next, there exists some a and b such that

$$a * e_1 + b * e_2 = 1$$

In such a case, the message M can be recovered as follows:

$$C_1^a * C_2^b$$
$$= \mathrm{M}^{e_1 * a} * M^{e_2 * b}$$
$$= \mathrm{M}^{e_1 * a + e_2 * b}$$
$$= \mathrm{M}^1 = M$$

A detailed explanation of the Common Modulus Attack can be found at [9].

5.2 Implementation

Coding language: Java.

My Java coded implementation of this attack is at:
https://github.com/dabsgts1/CryptoAttacks/blob/master/Common%20Modulus %20Attack

5.3 Results

Due to the limitations with Java's decimal data type, examples for this attack cannot be successfully executed with the accurate output on the BlueJ IDE at the moment. I am currently working on implementing the same attack on Python with SageMath.

6 Acknowledgements

I would like to thank Professor Somitra Sanadhya for his invaluable guidance over the past few months. Under his direction, I was able to self-study the cryptography course [10] by Professor Dan Boneh and was able to gain a profound understanding of the complex concepts with Prof. Sanadhya. I would also like to thank him for his constructive suggestions and aid during the planning and development of this research work. His willingness to give his time so generously is much appreciated.

7 References

[1] https://www.utc.edu/center-academic-excellence-cyber-defense/pdfs/course-pa per-5600-rsa.pdf

[2] D. Boneh, Twenty Years of Attacks on the RSA Cryptosystm

[3] https://www.infsec.cs.uni-saarland.de/teaching/16WS/Cybersecurity/lecture_ notes/cysec16-ln05.pdf

[4] https://en.wikipedia.org/wiki/Wiener%27s_attack

[5] http://cs.wellesley.edu/~cs310/lectures/26_rsa_slides_handouts.pdf

[6] https://www.youtube.com/watch?v=zIFehsBHB8o

[7] https://crypto.stackexchange.com/questions/6713/low-public-exponent-attack-f or-rsa

[8] https://crypto.stackexchange.com/questions/16283/how-to-use-common-modulus-attack

[9] https://medium.com/bugbountywriteup/rsa-attacks-common-modulus-7bdb34f3 31a5

[10] https://crypto.stanford.edu/~dabo/courses/OnlineCrypto/