Crack the RSA Encryption

Ben Chen
chenb2022@mail.sustech.edu.com
Student, Dept of CSE, SUSTech
12212231

1 Introduction

The security of modern cryptography relies on the difficulties on solving the NP problems within a acceptable cost. Generally, modern cryptography is categorised into three majar parts, Symmtric(ECC), Asymmetric(RSA) and Homomorphic(Lattice). However, the cryptography system is never absolutely secure and brilliant mathematicians and hackers have proposed several methods to crash into the wall built by cryptography. In this report, we mainly focus on the RSA algorithm which is one of the most well-known and widely applied cryptography algorithm nowadays. The following chapters will give a brief introdution to the RSA and then we will move to the most exciting part, the techniques to hack the cryptography where we also provides the python codes as proof of concept. Enjoy hacking!

2 Background

The RSA algorithm is a typical asymmetric cryptography algorithm, which is known for the public key and private key. The reliability has its base on the hardness of factoring large numbers which are usually in length of 1024 bits or more. Therefore, the brute force algorithm is unlikely to achieve an accpetable solution, unless you have a quantum computer. Nevertheless, crack the encryption rudely is not cool for a hacker or computer science lover. In this chapter, we just simply introduce how RSA works and attack methods will not be included.

2.1 Generation

Basically, the generation of the keys follow the three steps

1. Randomly choose two distinct prime number p and q, calculate $N = p \times q$
2. From the Euler function, $\varphi(N) = (p-1)(q-1)$
3. Choose a number e coprime to $\varphi(N)$, calculate the inverse $ed \equiv 1 \bmod \varphi(N)$

Then the public key will be (N, e) and the private key is (N, d). Note that in the next chapter the first step might be different, largely to improve the reliability.

2.2 Encryption

At first, the sender Alice should ensure that the length of message is less than N. Otherwise, he/she may need to split it into several block that smaller than N. You may wonder how message can be turned into a number. The common method is to simply treat the bytes of printable strings into numbers in little endian. Then we can do the encryption by calculating

$$c \equiv m^e \bmod N$$

where c is the encrypted message and m is original message.

2.3 Decryption

When the receiver Bob gets the encrypted message, he needs to retore the message which is called decrytion. In this process, he will do the computing

$$m \equiv c^d \bmod N$$

to obtain the original message.

2.4 Correctness

The proof has been well taught and examined, so we barely go through it. If the message m is coprime to N then it holds obviously that

$$m^{ed} = m^{\varphi(N)} \equiv m \bmod N$$

and if they are not coprime, then m must be a multiple of p or q. Suppose it is p

$$m = xp$$

then q must be greater than x and

$$m^{\varphi(q)} \bmod q$$

so we have

$$m^{k\varphi(N)} = m^{k(p-1)(q-1)} = \left(m^{\varphi(q)}\right)^{k(p-1)} \equiv 1 \bmod q$$

which gives

$$m^{k\varphi(N)+1} = m + uqm$$

and

$$m^{k\varphi(N)+1} = m + uqxp = m + uxN$$

so the algorithm is correct for $m < N$.

3 Attack on RSA

The objective of attacker is to get the private key from the public key or the opposite. In this chapter, we just focus on the former one since the process is similar by simply exchanging d and e. To crack the RSA encryption, we generally aim at three vulnerabilities, which are the moduli N, the power of e and the private key d. For each vulnerabilities, we will prove the theoritical feasibility and present the proof-of-concept code written in Python.

3.1 $p-1$ Smooth

The smooth number is the number that can be factorized into small primes and the number is called B-smooth if the maximum power does not exceed B, which is

$$p - 1 = a^n \times b^m \times c^k$$
$$\text{where } B > n, m, k$$

So we can apply Pollard's $p-1$ algorithm[1] to factor N. After getting p and q, we can reproduce the generation of RSA keys to get the private key, and then the original message is recovered.

3.1.1 Proof

From the Fermet's Little Theorem

$$p \nmid a \Rightarrow a^{p-1} \equiv 1 \bmod p$$

then we have

$$a^{t(p-1)} \equiv 1^t \equiv 1 \bmod p$$

which is

$$a^{t(p-1)} - 1 = k * p$$

according to Pollard's $p-1$ algorithm, then there exists

$$M = \prod_{q \leq B} q^{[\log_q B]}$$

which makes

$$(p-1) \mid M$$

hold and then

$$\gcd(a^M - 1, N)$$

if the result is neither 1 nor N, it successfully factored the N since N only contains two prime factors. We must not care about the M since letting $M = k!$ for all positive number will cover all possible M. In actual calculation, we can reduce the power by

$$a^{n!} \bmod N = \begin{cases} (a \bmod N)^2 \bmod N & n = 2 \\ \left(a^{(n-1)!} \bmod N\right)^n \bmod N & n \geq 3 \end{cases}$$

3.1.2 Code

```
from gmpy2 import *

def pollard(N: int):
    a = 2
    n = 2
    while True:
        a = powmod(a, n, N)
        res = gcd(a - 1, N)
        if res != 1 and res != N:
            q = N // res
            d = invert(e, (res - 1)*(q - 1))
            m = powmod(c, d, N)
            return m
        n += 1
```

3.2 $p+1$ Smooth

When $p+1$ is a smooth number, we can apply the William's $p+1$ algorithm[2] to factor N. The definition of Lucas sequence is similar to Fibonaci's

$$L_n = \begin{cases} 2 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ L_{n-1} + L_{n-2} & \text{if } n > 1 \end{cases}$$

3.2.1 Proof

Given that p is a prime factor of N and $p+1$ is smooth

$$p = \left(\prod_{i=1}^{k} q_i^{\alpha_i} \right) + 1$$

where q_i is the i-th prime factor and $q_i^{\alpha_i} < B_1$. Find the β_i such that

$$q_i^{\beta_i} \leq B_1 \text{ and } q_i^{\beta_i - 1} > B_1$$

then we let

$$R = \prod_{i=1}^{k} q_i^{\beta_i}$$

it is obvious that $p-1 \mid R$ and when $\gcd(N,a)=1$, we have $a^{p-1} \equiv 1 \bmod p$ so

$$a^R \equiv 1 \bmod p \Rightarrow p \mid \gcd(N, a^R - 1)$$

Let P, Q be a integer, α, β be the root for the equtaion $x^2 - Px + Q = 0$, and define

$$U_n(P,Q) = \frac{\alpha^n - \beta^n}{\alpha - \beta}$$
$$V_n(P,Q) = \alpha^n + \beta^n$$

the discriminant is $\Delta = (\alpha - \beta)^2 = P^2 - 4Q$, then we have the following equations

$$\begin{cases} U_{n+1} = PU_n - QU_{n-1} \\ V_{n+1} = PV_n - QV_{n-1} \end{cases}$$

and

$$\begin{cases} U_{2n} = V_n U_n \\ V_{2n} = V_n^2 - 2Q_n \end{cases}$$

and

$$\begin{cases} U_{2n-1} = U_n^2 - QU_{n-1}^2 \\ V_{2n-1} = V_n V_{n-1} - PQ_{n-1} \end{cases}$$

and

$$\begin{cases} \Delta U_n = PV_n - 2QV_{n-1} \\ V_n = PU_n - 2QU_{n-1} \end{cases}$$

and

$$\begin{cases} U_{m+n} = U_m U_{n+1} - QU_{m-1}U_n \\ \Delta V_{m+n} = V_m V_{n-1} - QV_{m-1}V_n \end{cases}$$

and

$$\begin{cases} U_n(V_k(P,Q), Q^k) = U_{nk}(P,Q)/U_k(P,Q) \\ V_n(V_k(P,Q), Q^k) = V_n(P,Q) \end{cases}$$

If we have $\gcd(N,Q) = 1$ and $P'Q \equiv P^2 - 2Q \bmod N$, then

$$P' \equiv \frac{\alpha}{\beta} + \frac{\beta}{\alpha} \text{ and } Q' \equiv \frac{\alpha}{\beta} + \frac{\beta}{\alpha} = 1 \bmod N$$

which is

$$U_{2m}(P,Q) \equiv PQ^{m-1}U_m(P', 1) \bmod N$$

From the extended Lucas Theorem, if p is prime, $p \nmid Q$ and the Lerendre symbol $(\Delta/p) = \varepsilon$, then

$$U_{(p-\varepsilon)m}(P,Q) \equiv 0 \bmod p$$
$$V_{(p-\varepsilon)m}(P,Q) \equiv 2Q^{m(1-\varepsilon)/2} \bmod p$$

Considering the first condition, given that p is a factor of N and $p+1$ is smooth

$$p = \left(\prod_{i=1}^{k} q_i^{\alpha_i}\right) - 1$$

we have $p+1 \mid R$ and when $\gcd(Q,N)=1$ and $(\Delta/p)=-1$ we have

$$p \mid U_R(P,Q) \Rightarrow p \mid \gcd(U_R(P,Q),N)$$

to find $U_R(P,Q)$ we can use the formula

$$U_{2n-1} = U_n^2 - QU_{n-1}^2 - 1$$
$$U_{2n} = U_n(PU_n - 2QU_{n-1})$$
$$U_{2n+1} = PU_{2n} - QU_{2n-1}$$

but it is hard to calculate, so we can simply it by the following approach. If $p \mid U_R(P,1)$, from the formular above, we have

$$p \mid U_{2R}(P,Q)$$
$$p \mid U_R(P',1)$$

Let $Q=1$, then

$$V_{(p-\varepsilon)m}(P,1) \equiv 2 \bmod p$$

which is

$$p \mid V_R(P,1) - 2$$

so the first condition is equivlant to:

Let $R = r_1 r_2 r_3 ... r_m$, we should find P_0 such that $\gcd(P_0^2-4,N)=1$, define that

$$V_n(P) = V_n(P,1)$$
$$U_n(P) = U_n(P,1)$$
$$P_j \equiv V_{r_j}(P_{j-1}) \bmod N$$

from the formular above

$$P_m \equiv V_R(P_0) \bmod N$$

and to calculate $V_r = V_r(P)$ we have

$$\begin{cases} V_{2f-1} \equiv V_f V_{f-1} - P \\ V_{2f} \quad\ \equiv V_f^2 - 2 \\ V_{2f+1} \equiv PV_f^2 - V_f V_{f-1} - P \bmod N \end{cases}$$

Let

$$r = \sum_{i=0}^{t} b_t 2^{t-i} \quad (b_i = 0, 1)$$

$f_0 = 1, f_{k+1} = 2f_k + b_{k+1}$ then $f_t = r$, as $V_0(P) = 2, V_1(P) = P$, then the formular is

$$\left(V_{f_{k+1}}, V_{f_{k+1}-1}\right) = \begin{cases} \left(V_{2f_k}, V_{2f_k-1}\right) \text{ if } b_{k+1} = 0 \\ \left(V_{2f_k+1}, V_{2f_k}\right) \text{ if } b_{k+1} = 1 \end{cases}$$

Considering the second condition, given $p + 1$ is a smooth number

$$p = s\left(\prod_{i=1}^{k} q_i^{\alpha_i}\right) - 1$$

where s is prime and $B_1 < s \leq B_2$, we have

$$p \mid \gcd(a_m^s - 1, N)$$

Define s_j and $2d_j$ as

$$2d_j = s_{j+1} - s_j$$

If $(\Delta/p) = -1$ and $p \nmid P_m - 2$, then we have

$$p \mid (U_s(P_m), N)$$

Let

$$U[n] \equiv U_n(P_m) \bmod N$$
$$V[n] \equiv V_n(P_m) \bmod N$$

calculate $U\left[2d_j - 1\right], U\left[2d_j\right]$ and $U\left[2d_j + 1\right]$ by

$$U[0] = 0, U[1] = 1, U[n+1] = P_m U[n] - U[n-1]$$

and to calculate

$$T[s_i] \equiv \Delta U_{s_i}(P_m) = \Delta U_{s_i R}(P_0)/U_R(P_0) \bmod N$$

from the formular above we have

$$\begin{cases} T[s_1] \equiv P_m V[s_1] - 2V[s_1 - 1] \bmod N \\ T[s_1 - 1] \equiv 2V[s_1] - P_m V[s_1 - 1] \bmod N \end{cases}$$

which is to calculate $T[s_i]$ by

$$\begin{cases} T[s_{i+1}] \equiv T[s_i]U[2d_i + 1] - T[s_i - 1]U[2d_i] \bmod N \\ T[s_{i+1} - 1] \equiv T[s_i]U[2d_i] - T[s_i - 1]U[2d_i - 1] \bmod N \end{cases}$$

and then calculate

$$H_t = \left(\prod_{i=0}^{c} T[s_{i+t}], N\right)$$

where $t = 1, c + 1, 2c + 1, ..., c[B_2/c] + 1$, when $(\Delta/p) = -1$ we have

$$p \mid H_i$$

so end of proof.

3.2.2 Code

```
from gmpy2 import *

def mlucas(v, a, n):
    v1, v2 = v, (v**2 - 2) % n
    for bit in bin(a)[3:]:
        v1, v2 = ((v1 ** 2 - 2) % n, (v1 * v2 - v) % n) if bit == "0" else ((v1 * v2 -
v) % n, (v2 ** 2 - 2) % n)
    return v1

def william(N: int):
  for v in count(1):
      for p in primegen():
          e = ilog(isqrt(n), p)
          if e == 0:
              break
          for _ in xrange(e):
              v = mlucas(v, p, n)
          g = gcd(v-2, n)
          if 1 < g < n:
              return g
          if g == n:
              break
```

3.3 Common Modulus

If the sender Alice generates two RSA keys with the same moduli N and encrypted the same message, then we can exploit it with common modulus attack.

3.3.1 Proof

Suppose Alice has two public key e_1 and e_2, which are coprime, and the message m, to generate the encrypted message c_1 and c_2 which are

$$\begin{aligned} c_1 &\equiv m^{e_1} \bmod N \\ c_2 &\equiv m^{e_2} \bmod N \end{aligned}$$

by the extended Eulicean algorithm, we can obtain r and s such that

$$re_1 + se_2 = q \bmod N$$

then we can get the message without knowing the private key by

$$\begin{aligned} c_1^r c_2^s &\equiv m^{re_1} m^{se_2} \bmod N \\ &\equiv m^{(re_1 + se_2)} \bmod N \\ &\equiv m \bmod N \end{aligned}$$

so the original message m is recovered from two encrypted message.

3.3.2 Code

```
from gmpy2 import *

def common_modulus(n, e1, e2, c1, c2):
    gcd, s, t = gcdext(e1, e2)
```

```
    if s < 0:
        s = -s
        message1 = invert(message1, n)
    if t < 0:
        t = -t
        message2 = invert(message2, n)
    plain = powmod(message1, s, n) * powmod(message2, t, n) % n
    return plain
```

3.4 Moduli $N = p^r q$

If Alice generates p and q from the same small prime g, and obtain them by

$$p = k_1 g + 1$$
$$q = k_2 g + 1$$

where k_1, k_2 are large primes. And she gets the moduli by

$$N = p^r q$$

it can be attacked by the Lift algorithm, which is when the public key e statisfies $ex - \varphi(N)y = z$, if

$$|xz| < N^{\frac{r(r-1)}{(r+1)^2}}$$

with small parameters x and $|z|$, then we can factor N. [3]

3.4.1 Proof

Suppose that e satisfies $ex - \varphi(N)y = z$ with $|x| < N^\delta$ and $|z| < N^\gamma$. Then, since

$$\varphi(N) = p^{r-1}(p-1)(q-1)$$

we get $ex - z \equiv 0 \bmod p^{r-1}$. And from the Theorem 1 of the paper[3], we can solve the equation in polynomial time if

$$\delta + \gamma < uv\beta^2 = \frac{r(r-1)}{(r+1)^2}$$

that is $|xz| < N^{\frac{r(r-1)}{(r+1)^2}}$. Since $\frac{e}{\varphi(N)} < 1$, then using x and z in the equation $ex - \varphi(N)y = z$, we get for sufficiently large N comparatively to r

$$y = \frac{ex - z}{\varphi(N)} < \frac{e|x|}{\varphi(N)} + \frac{|z|}{\varphi(N)} < |x| + |z| \leq 1 + |xz| < 1 + N^{\frac{r(r-1)}{(r+1)^2}} < N$$

Hence

$$\gcd(ex - z, N) = \gcd(p^{r-1}(p-1)(q-1)y, p^r 1) = g$$

with $g = p^{r-1}$, $g = p^r$ or $g = p^{r-1}q$ and

$$g = p^{r-1} \Rightarrow p = q^{\frac{1}{r-1}}$$
$$g = p^r \Rightarrow p = q^{\frac{1}{r}}$$
$$g = p^{r-1}q \Rightarrow p = \frac{N}{g}$$

This leads to the factorization of N.

3.4.2 Code

```
from Crypto.Util.number import *
import itertools, gmpy2

hint = 251 # gcd(e * g, phi)
known_msg = b"test"

def lift(n, e, c):
    R.<x> = PolynomialRing(Zmod(n))
    f = e*x - 251
    f = f.monic()
    root = f.small_roots(X = 2^256, beta=0.16)
    tmp = int(f(root[0]))
    t = gmpy2.gcd(tmp,n)
    p = gmpy2.iroot(t,4)[0]
    q = n // p ^ 5
    p_list = [p,q]
    mi = [5,1]
    n_list = [ZZ(p_list[i]) ** mi[i] for i in range(len(mi))]
    res = []

    for pi in n_list:
        d = inverse(int(e//251),euler_phi(pi))
        m = pow(c,d,pi)
        res.append(Zmod(pi)(m).nth_root(251, all=True))

    for vc in itertools.product(*res):
        _c = [int(x) for x in vc]
        m = crt(_c, n_list)
        msg = long_to_bytes(int(m))
        if known_msg in msg:
            return msg
```

3.5 Small Public Key

If Alice accidentally generates a small public key e, then we can attack it even with large N. For $e = 2$, we have the Rabin algorithm.

3.5.1 Proof

Suppose we have the encrypted message c

$$c \equiv m^2 \bmod N$$

calculate m_p and m_q

$$m_p = \sqrt{c} \bmod p$$
$$m_q = \sqrt{c} \bmod q$$

then solve the congruence equation by extended Eucliean algorithm

$$y_p \cdot p + y_q \cdot q = 1$$

and we can solve the original message

$$
\begin{aligned}
a &= \left(y_p \cdot p \cdot m_q + y_q \cdot q \cdot m_p\right) \bmod n \\
b &= n - a \\
c &= \left(y_p \cdot p \cdot m_q - y_q \cdot q \cdot m_p\right) \bmod n \\
d &= n - c
\end{aligned}
$$

If $p \equiv q \equiv 3 \bmod 4$ then

$$
\begin{aligned}
m_p &= c^{\frac{1}{4}(p+1)} \bmod p \\
m_q &= c^{\frac{1}{4}(q+1)} \bmod q
\end{aligned}
$$

otherwise, we can only use brute force algorithm.

3.5.2 Code

```
import gmpy2, string
from Crypto.PublicKey import RSA

def rabin(p, q):
    inv_p = gmpy2.invert(p, q)
    inv_q = gmpy2.invert(q, p)

    mp = pow(cipher, (p + 1) / 4, p)
    mq = pow(cipher, (q + 1) / 4, q)

    a = (inv_p * p * mq + inv_q * q * mp) % N
    b = N - int(a)
    c = (inv_p * p * mq - inv_q * q * mp) % N
    d = N - int(c)

    result = ''
    for i in (a, b, c, d):
        s = '%x' % i
        if len(s) % 2 != 0:
            s = '0' + s
        result += s
    return result
```

3.6 Low Private Exponent

When the private key d is small, it is possible to factor N by the Wiener's Approach[4]. In this section, the proof will not be detailed since the original proof is too long to put here. Wiener shows that when

$$
d < \frac{1}{3} N^{\frac{1}{4}}
$$

N can be factorized by his approach definitely.

3.6.1 Proof

Given

$$
e * d = k * \lambda(N) = 1
$$

where

$$
\lambda(N) = lcm(p-1, q-1) = \frac{\varphi(N)}{g}
$$

let $s = 1 - p - q$, then

$$edg - kN = g + ks$$

divide both sides with dgN

$$\frac{e}{N} - \frac{k}{dg} = g + k\frac{s}{dgN} = \left(\frac{k}{dg}\right)\left(\frac{s}{N}\right) + \frac{1}{dN}$$

it is known that $e \approx N$ and $s \approx N^{\frac{1}{2}}$, so we have

$$\frac{k}{dg} \approx 1$$

and the right side approximates to $N^{-\frac{1}{2}}$ so when

$$\left|x - \frac{a}{b}\right| < \frac{1}{2b^2}$$

$\frac{a}{b}$ is a continued fraction approximation of x. So when

$$d < \frac{\sqrt{2}}{2g}N^{\frac{1}{4}}$$

we have $\frac{k}{dg}$ is a continued fraction approximation of $\frac{e}{N}$, and we can cover all the possible private keys d by expanding the coninued fractions.

3.6.2 Code

The code requires to run from the SageMath with `gmpy2` installed.

```
import gmpy2
from sage.all import *

def transform(x, y):
    res = []
    while y:
        res.append(x // y)
        x, y = y, x % y
    return res

def continued_fraction(sub_res):
    numerator, denominator = 1, 0
    for i in sub_res[::-1]:
        denominator, numerator = numerator, i * numerator + denominator
    return denominator,numerator

def sub_fraction(x, y):
    res = transform(x, y)
    res = list(map(continued_fraction, (res[0:i] for i in range(1, len(res)))))
    return res

def get_pq(a, b, c):
    par = gmpy2.isqrt(b * b - 4 * a * c)
    x1, x2 = (-b + par) // (2 * a), (-b - par) // (2 * a)
    return x1, x2

def wiener_attack(e,n):
    for (d, k) in sub_fraction(e, n):
```

```
    if k == 0:
        continue
    if (e * d - 1) % k != 0:
        continue

    phi = (e * d - 1) // k
    px, qy = get_pq(1, n - phi + 1, n)
    if px * qy == n:
        p, q = abs(int(px)), abs(int(qy))
        d = gmpy2.invert(e, (p - 1) * (q - 1))
        return d
```

4 Summary

In this report, we introduce six approaches to attack RSA encryption under the conditions. The vulnerabilities are caused by the improper generation of moduli, private key and public key. If one of them goes wrong, the RSA key has a relatively high risk of being exploited. The first four approaches attack on the moduli which is the most common vulnerability, and the fifth approach tries to crack encryption with small public key, and the last one obtains the private key if it has low exponent.

For now, the explore is over. Since we are new to the world of cryptography, that's the farthest place that we can reach presently. The approaches mentioned are for those improper construction of RSA keys. However, is there any RSA construction that is hard to crack? We don't know. We do know that the war between hackers and just mathematicians are promoting the progress of cryptography, a world that is exquisite. Anyway, thank you for reading this. May we go further in the way of learning discrete math.

Reference

[1] J. M. Pollard, "Theorems on factorization and primality testing", *Mathematical Proceedings of the Cambridge Philosophical Society*, vol. 76, no. 3, pp. 521–528, 1974, doi: 10.1017/s0305004100049252.

[2] H. C. Williams, "A $p+1$ method of factoring", *Mathematics of Computation*, vol. 39, no. 159, pp. 225–234, 1982, doi: 10.1090/s0025-5718-1982-0658227-7.

[3] A. Nitaj and T. Rachidi, "New attacks on RSA with Moduli N=p^rq .", *IACR Cryptology ePrint Archive*, vol. 2015, p. 399--, 2015, [Online]. Available: http://dblp.uni-trier.de/db/journals/iacr/iacr2015.html#NitajR15a

[4] M. J. Wiener, "Cryptanalysis of short RSA secret exponents", *IEEE Transactions on Information Theory*, vol. 36, no. 3, pp. 553–558, 1990, doi: 10.1109/18.54902.