Wieners's Attack on RSA

Jacob Shin

December 2020

1 Background on RSA

RSA is a widely used cryptosystem that allows secure encryption and decryption of data. Even though RSA was first released in 1977 by MIT researchers (whose names make up the RSA acronym), and also independently discovered in 1973 and kept secret by British Intelligence, RSA still plays a crucial role in securing today's Internet traffic and digital transactions [**?**, **?**]. It is *asymmetric* meaning that there are two separate keys – one for encrypting and one for decrypting. The key used for encryption is called the *public key* while the decryption key is called the *private key*, since everyone can have access to the public key while only the receiver should have access to the private key.

Let us first define some terminology. The message we want to encrypt is called the *plaintext* and the encrypted message is the *ciphertext*. They are denoted by m and c, respectively. We use also have two exponents when using RSA – one used solely for encryption called the public exponent, e, and a private exponent, d, used only for decryption. The private exponent is kept private/secret as the name suggests, while the public exponent is not. The RSA algorithm also utilizes a modulus, n, in both encryption and decryption operations. The following formulas allow us to encrypt and decrypt data.

Encryption:

$$c \equiv m^e \ (mod\, n) \tag{1}$$

Decrpytion:

$$m \equiv c^d \ (mod\, n) \tag{2}$$

Notice that for encryption both e and n are present in addition to the message. Together these form the *public key* and, which can be denoted as the pair $\langle n, e\rangle$. Similiarly, decryption uses both d and n, which together form the *private key*. Both the public key and private key are also often encoded with base64 to save space and for easier packaging of the keys.

The following result can be derived from the above [**?**].

$$m \equiv c^d \equiv (m^e)^d \ \equiv m^{ed} \ (mod\, n) \tag{3}$$

Next, we will try to show that expression 3 this is actually true and that decryption actuall works.. In order for the above expression to be true, there are some more definitions for some of the variables defined above.

$$n = p \cdot q \tag{4}$$

The modulus, n, is the product of two distinct primes, p and q. The modulus, n, is known to everyone since it's public, while the factors, p and q, are kept secret since they're used to derive the private key as we'll see later. One of the reasons why RSA is secure when done properly is due to how difficult it is to factor large numbers. Thus an attacker would have a difficult time factoring n to get p and q.

Next we have a function called **Euler's totient** represented by ϕ.

$$\phi(n) = (p-1)(q-1) \tag{5}$$

This function is used to get the value for e, the public exponent. The public exponent only has to meet the following two conditions [**?**].

$$1 < e < \phi(n)$$

$$gcd(\phi(n), e) = 1$$

The second condition means $\phi(n)$ and e have to be relative primes or coprime. Finally, in order to obtain a proper value for d, the private exponent, we need the following to be true [**?**].

$$d \cdot e \equiv 1 \ (mod\ \phi(n)) \tag{6}$$

This also means that there exists an integer k so that the following is true.

$$d \cdot e = 1 \ + k \cdot \phi(n) = 1 \ + k \cdot (p-1)(q-1) \tag{7}$$

Now we can plug in the value of $d \cdot e$ we got above to get the following.

$$c^d \equiv (m^e)^d \ \equiv m^{ed} \ (mod\ n) \equiv m^{1 \ +k\cdot(p-1)(q-1)} \ (mod\ n) \tag{8}$$

Fermat's little theorem states that the following is true for all integers, a, if p is a prime number.

$$a^p \equiv a \ (mod\ p) \tag{9}$$

And if a is not divisible by p the following is also true [**?**].

$$a^{p-1} \equiv 1 \ (mod\ p) \tag{10}$$

We can use Fermat's little theorem to get some more expressions. First we can rewrite expression 8 as follows:

$$c^d \equiv m^{1 \ +k\cdot(p-1)(q-1)} \equiv m^1 \cdot m^{k\cdot(p-1)(q-1)} \ (mod\ n) \tag{11}$$

$$\equiv m^1 \cdot (m^{k\cdot(p-1)})^{(q-1)} \ (mod\ n) \tag{12}$$

$$\equiv m^1 \cdot (m^{k\cdot(q-1)})^{(p-1)} \ (mod\ n) \tag{13}$$

When we apply Fermat's little theorem to expression 12 we get the following:

$$c^d \equiv m^1 \cdot (m^{k\cdot(p-1)})^{(q-1)} \equiv m \ (mod\ q) \tag{14}$$

And we can also apply the theorem to expression 13:

$$c^d \equiv m^1 \cdot (m^{k\cdot(q-1)})^{(p-1)} \equiv m \ (mod\ p) \tag{15}$$

Finally, we can apply the Chinese Remainder Theorem to verify that decryption does indeed produce the original message [**?**]:

$$c^d \equiv m^1 \ (mod\ p \cdot q) \equiv m \ (mod\ n) \tag{16}$$

Even though many attacks on RSA exist, most of the attacks are only effective when there is an improper use or generation of the public key and/or private key. Thus as long as RSA is not used improperly, it will be secure against any classical, non-quantum computer based attacks. In this paper, "breaking" RSA does not mean breaking the whole RSA cryptosystem, but rather just the implementation of RSA and the improper use of public/private keys to recover a secret. The rest of this paper will explore a classical, common attack on RSA.

2 Wiener's Attack

Wiener's attack utilizes an improperly chosen private exponent, d, in order to recover the secrets from just the public key. Wiener proved that if $d < \frac{1}{3}n^{\frac{1}{4}}$, it is guaranteed that an attacker can recover d [?]. Thus to prevent an attack, a user simply has to choose a value for d that is big enough to be secure. The rest of this section will cover how to use Wiener's findings to recover d.

$\phi(n)$ can be approximated as n [?]:

$$\phi(n) = (p-1)(q-1) = pq - (p+q) + 1 = n - (p+q) + 1$$

Since n is the product of two very large prime numbers, n is also very large. Compared to the sum of p and q, n is large enough that $p+q$ is neglibile and so we can approximate $\phi(n)$ as follows:

$$\phi(n) \approx n$$

After rearranging expression 7 we have the following:

$$d \cdot e = 1 \ + k \cdot \phi(n)$$

$$e = \frac{1}{d} + \frac{k \cdot \phi(n)}{d}$$

$$e - \frac{k \cdot \phi(n)}{d} = \frac{1}{d}$$

$$\frac{e}{\phi(n)} - \frac{k}{d} = \frac{1}{d \cdot \phi(n)}$$

Since both $\phi(n)$ is a large number and $\phi(n) \cdot d$ is even larger, $\frac{1}{\phi(n) \cdot d}$ is approximately zero.

$$\frac{e}{\phi(n)} - \frac{k}{d} \approx 0$$

$$\frac{e}{\phi(n)} \approx \frac{k}{d}$$

Previously we showed that $\phi(n)$ was approximately n, so we get the following result:

$$\frac{e}{n} \approx \frac{k}{d}$$

We can use this result to get d with only the public key, e and n. In order to get the value of $\frac{k}{d}$, we can generate a set of possible values for $\frac{k}{d}$ that approximate $\frac{e}{n}$ using **continued fractions**. Any real number can be represented as a continued fraction. The continued fraction converges to the value of the actual number. For example, π can be represented as an infinite continued fraction.

$$\pi = 3 + \cfrac{1}{7 + \cfrac{1}{15 + \cfrac{1}{1 + \cfrac{1}{292 + \cfrac{1}{\ddots}}}}}$$

After each fraction, the value of π comes closer and closer to the actual value of π.

$$\pi \approx 3 + \frac{1}{7} = 3.142857143$$

$$\pi \approx 3 + \cfrac{1}{7 + \cfrac{1}{15}} = 3.141509434$$

$$\pi \approx 3 + \cfrac{1}{7 + \cfrac{1}{15 + \cfrac{1}{1}}} = 3.14159292$$

and so on.

Each fraction at each step is called a *convergent.* We can also use continued fractions to get approximations for rational numbers as well like $\frac{e}{n}$. Since we know that $\frac{k}{d}$ is only *approximately* $\frac{e}{n}$, we need to use continued fractions to generate a list of approximations (i.e. convergents of $\frac{e}{n}$) and get the actual value of $\frac{k}{d}$. Wiener proved that one of the convergents of the continued fraction for $\frac{e}{n}$ *must* be equal to $\frac{k}{d}$ [**?**]. Thus in order to get d, all we need to do is generate convergents for $\frac{e}{n}$ and then check each convergent to determine if it is equal to $\frac{k}{d}$. The first part is simple; we can use Euclid's algorithm to generate convergents. In order to check to see if the convergent is valid we can use three checks. If the convergent passes all the checks, then the convergent is equal to $\frac{k}{d}$; if the convergent fails any one of the tests, then it is not the right one and thus we can move on to the next convergent [**?**]. Here are the checks:

1. d must be odd. We know that $\phi(n) = (p-1)(q-1)$ is even. Since both p and q are primes, they are odd. One less than an odd number is always even and the product of two even numbers is also always even. Thus $(p-1)(q-1)$ is always even. Since $d \cdot e = 1 + k \cdot \phi(n)$, we know that one plus an even number is always odd. So both sides of the expression must be odd, since $\phi(n) \cdot k + 1$ is odd. This also means $e \cdot d$ is odd, and since e is odd, then d must also be odd in order for the product of e and d to be odd.

2. We know that $\phi(n) = \frac{d \cdot e - 1}{k}$ after rearranging expression 7. Since $\phi(n)$ is a positive integer, our second check is to make sure that $\frac{d \cdot e - 1}{k}$ is a whole number.

3. The final check is to make sure that the p and q we obtain from our solution are positive integers. We can solve for p and q by using a simple quadratic formula. The formula and derivation are shown below:

$$\phi(n) = n - (p+q) + 1$$

$$p + q = n - phi(n) + 1$$

$$(x-p)(x-q) = 0$$

$$x^2 - (p+q)x + pq = 0$$

$$x^2 - (n - \phi(n) + 1)x + pq = 0$$

An example of using Wiener's attack to find d is provided in Appendix A.

Appendix A Wiener's Attack Example

Let $e = 42667$ and $n = 64741$. Thus we will need to find approximations of $\frac{e}{n} = \frac{42667}{64741}$ using continued fractions. We can find the convergents of the continued fraction of $\frac{e}{n}$ using Euclid's algorithm:

$$\begin{gathered}
42667 = 0 \cdot 64741 + 42667 \\
64741 = 1 \cdot 42667 + 22704 \\
42667 = 1 \cdot 22704 + 20593 \\
22704 = 1 \cdot 20593 + 1481 \\
20593 = 13 \cdot 1481 + 1340 \\
1481 = 1 \cdot 1340 + 141 \\
\cdots
\end{gathered}$$

Now we can express $\frac{e}{n}$ as a continued fraction:

$$\frac{e}{n} = 0 + \cfrac{1}{1 + \cfrac{1}{1 + \cfrac{1}{1 + \cfrac{1}{13 + \cfrac{1}{1 + \cfrac{1}{\ddots}}}}}}$$

Thus the convergents are as follows:

$$0 + \frac{1}{1}$$

$$0 + \cfrac{1}{1 + \cfrac{1}{1}} = \frac{1}{2}$$

$$0 + \cfrac{1}{1 + \cfrac{1}{1 + \cfrac{1}{1}}} = \frac{2}{3}$$

- The first convergent fails because d and e can't be 1.
- The second convergent fails the first test. In this convergent, $\frac{1}{2} = \frac{k}{d}$, but this is wrong since d has to be odd.
- The third convergent passes all the checks. d is 3 so it is odd, which passes the first test. $\phi(n) = \frac{d \cdot e - 1}{k} = \frac{3 \cdot 42667 - 1}{2} = 64000$, which a whole number, so this passes the second check. Finally, the solution to the following equation produces whole number values for x: $x^2 - (n - \phi(n) + 1)x + n = x^2 - (64741 - 64000 + 1)x + 64741 = 0$. The solutions for x are 641 and 101, which are p and q. We have now found all the secret information since all the checks pass. Thus d is 3 for this particular example.

Note that in this examples, the convergents are precomputed ahead of time for clarity; however, it is more efficient in an actual algorithm to test the convergents before computing the next one, since the algorithm can be halted once a d is found.

Appendix B Wiener's Attack Code

```
#/usr/bin/env python3
from fractions import Fraction # Helps with contined fractions
import math

e = 42667
n = 64741

# Returns whether true or false for whether the solutions to quadratic are integers
def check_quadratic(phi, n):
    a = 1
    b = -(n - phi + 1)
    c = n

    determinant = b**2 - 4*a*c

    # No imaginary solutions allowed
    if determinant < 0:
        return False

    root = math.isqrt(determinant)

    # If determinant is not perfect square then that means the roots are not integers
    if root ** 2 != determinant:
        return False

    # Check to make sure roots are integers
    if (-b + root) % (2*a) != 0 or (-b - root) % (2*a) != 0:
        return False

    q = (-b + root) // (2*a)
    p = (-b - root) // (2*a)

    print("q: " + str(q))
    print("p: " + str(p))

    # If nothing else return False then the solutions are integers
    return True

# Returns whether totient is whole number or not
def totient_whole_number(d, k, e):
    if k == 0:
        return False
    return (d * e - 1) % k == 0

# Performs the three checks to make sure we have the right value for d and k
def perform_check(d, k):
    if d == 1:
        return False

    # First check: Check for odd parity
    if d % 2 == 0:
        return False
    # Second check: Check to make sure totient is whole number
    if not totient_whole_number(d, k, e):
        return False

    # If we pass 2nd test then calculate the possible phi value
    phi = (d * e - 1) // k

    # Third check: Check for integer solution to quadratic
    if not check_quadratic(phi, n):
        return False

    return True

# Converts the sequence to a convergent which might be a possibility for k/d
def sequence_to_d_and_k(sequence):
    # If only one element in sequence
    if len(sequence) == 1:
        return 1, sequence[0]

    convergent = 0

    # Traverse sequence in reverse order
    for elem in sequence[::-1]:
        if (convergent == 0):
            convergent = elem
        else:
            convergent = elem + Fraction(1, convergent)

    k = convergent.numerator
    d = convergent.denominator

    return d, k

def continued_fraction(e, n):
    # Stores the sequence from euclid's algorithm
    sequence = []
    a = e
    b = n

    # Use euclid's algorithm to get the convergents/continued fraction
    while (a != 0 and b != 0):
        quotient = a // b
        r = a % b
        a = b
        b = r

        sequence.append(quotient)

        # Converts the sequence to the convergent
        d, k = sequence_to_d_and_k(sequence)

        # print(d, k)

        # If all the checks are passed, there's no need to continue since we've found d
        if (perform_check(d, k)):
           return d

    # print(sequence)

d = continued_fraction(e, n)

print("Your value for d is " + str(d))
```

Appendix C A Visual Representation of Wiener's Attack