EE575 Term Project

Explanation and Implementation of Wiener's Attack

EKREM FATİH YILMAZER

Department of Electrical and Electronics Engineering

Boğaziçi University

Boğaziçi University

Bebek, Istanbul 34342

Lecturer: Emin ANARIM

May 27, 2020

1 RSA Systems

1.1 Overview

RSA is one of the early public-key cryptosystem and is still used widely.Unlike the symetric cryptosystems, encryption and decryption keys are not same. Decryption key is kept private whereas the enryption key is public. The mechanism of RSA is based on the difficulty of factoring the product of two large numbers.

Sender publishes a public encryption key based on two large prime numbers which are kept secret. Anyone can encrypt the message, decryption can only be done with the knowledge of prime numbers can decrypt the message. Breaking RSA system is called RSA problem which is actually a factoring problem. There isn't any method for RSA encryption if large enough key is used.

1.2 Key Distribution

Note that, decryption key must be known by the receiver. How is it done? Suppose Bob wants to send a message to Alice via RSA encryption system.He first sends message to Alice to inform her that he is going to send message. Then Alice sends the public key (n, e) of her via secure channel. This way, Alice's private key is never distributed.

1.3 Encryption and Decryption

Suppose Bob wants to send message M to Alice. Firstly it converts the plaintext to an integer m by applying a padding scheme. Here M is called unpadded plaintext whereas m is padded plaintext. $(0 <= m < n)$ Afterwards ciphertext is computed by public keys.

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

Modular exponentiation is used for faster computation. In order to decrypt the message Alice applies the following formula.

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

Note that Given the padding scheme, Alice can find out the message M with ease.

1.4 Key Generation

The keys $(n, e), d$ are generated by the following procedure.

- Generate two distinct prime numbers (p, q) which are kept secret.
- Compute $N = pq$ where n is part of the public key.
- Compute $\lambda(N)$ where $\lambda(n)$ is Carmichael's totient function. One can use Euler's totient function $\phi(N)$ since it is divisible by $\lambda(n)$.
- Choose e such that $1 < n < \lambda(n)$ and $gcd(e, \lambda(n)) = 1$ (n and $\lambda(n)$ are coprime.)
- Compute $d = e^{-1}(mod \, \lambda(n))$

1.5 Proof of Correctness

The proof below uses Fermat's Little Theorem. The equivalency that we want to show is that,

$$(m^e)^d \equiv m(mod \, pq) \tag{3}$$

for every integer m and p and q distinct prime numbers satisfying $ed \equiv 1(mod \lambda(pq))$. Since $\lambda(pq) = lcm(p-1, q-1)$, it is both divisible by $p-1$ and $q-1$.

$$ed - 1 = h(p-1) = k(q-1) \tag{4}$$

We first will show, $m^{ed} = m(mod \, p)$.

- $m = 0(mod\, p) \rightarrow$ m is multiple of p. Then,$(m^e)^d \equiv m(mod\, p)$
- $m \neq 0(mod\, p) \rightarrow m^{ed} \equiv m^{ed-1}m \equiv (m^{p-1})^h m = 1^h m \equiv m(mod\, p)$

Likewise, $m^{ed} = m(mod\, q)$ so, $(m^e)^d \equiv m(mod\, pq)$

2 Wiener's Attack

In order to faster decryption performance, one can use small d. However, for small value of d, Wiener's Attack shows that the system is insecure.

2.1 Methodology

From now on, Euler's Toitent function $\phi(N)$ will be used instead of Charmichael's toitent function. Note that,

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

We also know that,

$$ed \equiv 1(mod\, \phi(pq)) \tag{6}$$

So there is an integer k such that, $ed = k\lambda(N) + 1 \rightarrow ed = k(p-1)(q-1) + 1$ If we divide both sides with dpq;

$$\frac{e}{pq} = \frac{k}{d}(1-\delta) \tag{7}$$

where $\delta = \frac{p+q-1-1/k}{pq}$. Note that, δ is a small number. So, $\frac{k}{d}$ is an approximate value of $\frac{e}{N}$.

2.2 Wiener's Theorem

Let $N = pq$ with $q < p < 2q$. Let $d < \frac{1}{3}N^{1/4}$. Given (e, N), $e < \phi(N)$ and $ed \equiv 1(mod\, \phi(N))$, d can be efficiently recovered by searching the right $\frac{k}{d}$ among the convergents of $\frac{e}{N}$.

So the theorem guarantees that $\frac{k}{d}$ is among the convergents of $\frac{e}{N}$ if d is small.!!!

2.3 Continued Fractions and Rational Approximations

Convergents of a number can be calculated using continued fractions. A simple continued fraction is in the following form,

$$x = a_0 + \cfrac{1}{a_1 + \frac{1}{a_2 + \frac{1}{a_3 + \ldots}}} \tag{8}$$

where a_is are called quotients. $x = [a_0, a_1, a_2, ..., a_n]$ Then convergents of x is as follows.

- $c_0 = a_0$
- $c_1 = a_0 + \frac{1}{a_1}$
- $c_2 = a_0 + \frac{1}{a_1 + \frac{1}{a_2}}$

  $\vdots$

These approximations are used in Wiener's Attack.

2.4 Whole Attack Flow

Following steps indicate the attack process of Wiener.

- Generate vulnarable keypair with short private key exponent.$(d < \frac{1}{3}N^{\frac{1}{4}})$
- Find the convergents of $\frac{e}{N}$ with continued fraction expansion.
- Iterate over all possible convergents for determining d_i/k_i
- Test your guess. It could be done with 2 diffirent ways. First one is based on factoring N with $\phi(N)$. Other one just encrypts a random message then

decrypts it with d_i and checks that whether the decrypted message is the original message. In the implementation the second method is used.

Note that since Wiener's theorem guarantees that k_i/d_i is among the convergents of e/N, the attack always find a right solution. If the key is not vulnarable, attack may fail. Suppose the generated key is (N,e)=(90581,17993).

$$\frac{e}{N} = \frac{17993}{90581} = 0 + \frac{1}{5 + \frac{1}{29 + \frac{1}{4 + \dots}}} = [0, 5, 4, 29, 4, 1, 3, 2, 4, 3] \tag{9}$$

By the convergents of $\frac{e}{N}$, possible $\frac{k}{d}$ values are,

$$\frac{k_i}{d_i} = 0, \frac{1}{5}, \frac{29}{146}, \frac{117}{589}, \dots \tag{10}$$

It is obvious that the first convergent does not yield to private key. For the second convergent $\frac{1}{5}$,

$$\phi(N) = \frac{ed - 1}{k} = \frac{17993 \times 5 - 1}{1} = 89994 \tag{11}$$

By solving the equation,

$$p^2 + p(\phi(N) - N - 1) + N = 0 \tag{12}$$

the roots are found as $p_1, p_2 = 379, 279$. $p_1 p_2 = 90581 = N$ so $d = 5$ is the right key. Note that $\frac{1}{3}N^{\frac{1}{4}} = 5.7828$ which is smaller than d.

2.5 Proof of Wiener's Theorem

Theorem 1 *Assume that $gcd(a, b) = 1$. If r, s are any natural numbers such that $gcd(r, s) = 1$, and $|\frac{a}{b} - \frac{r}{s}| < 1/(2s^2)$ then r/s is one of the convergents of a/b.*

We will use the theorem above in order to prove the Wiener's Theorem.We know that $ed \equiv 1(mod(\phi(N))$. So there exist a k such that,

$$ed - k\phi(N) = 1 \rightarrow \left| \frac{e}{\phi(N)} - \frac{k}{d} \right| \tag{13}$$

Attacker does not know $\phi(N)$, so he might approximate it with N.

$$\left|\frac{e}{N}-\frac{k}{d}\right| = \left|\frac{ed-kN}{Nd}\right| = \left|\frac{ed-k\phi(N)+k\phi(N)-kN}{Nd}\right| = \left|\frac{1-k(N-\phi(N))}{Nd}\right| \quad (14)$$

$|N-\phi(N)| = |p+q-1|$ and also $|p+q-1| < |3q-1| < 3\sqrt{pq}$ so, $|N-\phi(N)| < 3\sqrt{N}$. If we use this inequality in above equation,

$$\left|\frac{e}{N}-\frac{k}{d}\right| < \frac{3k}{d\sqrt{N}} \quad (15)$$

In RSA systems, we have $e < \phi(N)$, thus $k\phi(N) < ed < d\phi(N) \rightarrow k < d$. Note that vulnerability condition is $d < \frac{1}{3}N^{1/4}$, so

$$\left|\frac{e}{N}-\frac{k}{d}\right| < \frac{1}{dN^{1/4}} \quad (16)$$

With using the inequality $d < \frac{1}{3}N^{1/4} < \frac{1}{2}N^{1/4}$ we conclude that $\frac{1}{2d} > \frac{1}{N^{1/4}}$. So,

$$\left|\frac{e}{N}-\frac{k}{d}\right| < \frac{1}{2d^2} \quad (17)$$

This is the inequality that we wanted to show. From Theorem 1, it is seen that k/d must be one of the convergents of e/N. QED

2.6 Defense Mechanisms Against Wiener's Attack

2.6.1 Not Using Low Private Key Exponent

This defend mechanism is a bit obvious but it is crucial to applying it. When low private exponent is used, the cryptosystem is not just vulnerable to Wiener's Attack but some others, too. As long as $d > \frac{1}{3}N^{1/4}$, Wiener's attack is not guaranteed to succeed.

2.6.2 Increasing the Public Key

Instead of using e, one can use $e' = e + k\phi(N)$. Note that this change of variable does not break the multiplicative inverse condition and e' and $\phi(N)$ are still coprime.

Note that when we first generated e, we assumed that $1 < e < \phi(n)$. So Wiener's theorem is not violated. By changing $e \rightarrow e'$, we also change the ratio $\frac{e}{N}$. It is shown that if k is large enough ($e' > N^{3/2}$), Wiener's Attack is useless, regardless of how small d is. But encryption performance suffers because of this change.

2.6.3 Using Chinese Remainder Theorem

Suppose that we want to have fast decryption performance as we have when we use small private key. But we do not want to have large public key since it decreases encryption performance. With using Chinese Remainder Theorem, we can achieve it by following procedure.

- Choose large d such that $d_p \equiv d(mod\, p-1)$ and $d_q \equiv d(mod\, q-1)$ is small.
- Then decrypt message with these new keys. $M_p = C^{d_p}(modp), M_q = C^{d_q}(modq)$
- Using the Chinese Remainder Theorem, compute unique M such that, $M \equiv M_p(mod\, p)$ and $M \equiv M_q(mod\, q)$
- We found our decrypted message since $M = C^d(mod\, pq)$

We used small d_p and d_q, so d is not necessarily small.

3-Wiener's Attack in MATLAB

```
clc;
clear;
import java.security.*;
import java.math.*;

wienersAttack(2^16);

function p= prime_random(size)
%generates prime random number less
%than size value
while 1
p=randi(size);
if isprime(p)
    break;
end
end
end


function p=randomNumber(max)
%generates random number less
%than max value
while 1
    a=de2bi(max);
    p=randi([0 1],1,length(a));
    p=bi2de(p);
    if p<max && p>1
        break;
    end
end
end

function [a,exist]= multInverse(b,m)
[div,c1,c2] = gcd(b,m);
% returns the greatest common divisor
%"div" and the two integer constants that solve
% c1*b + c2*m = div

if div==1
    exist=1;
    a = mod(c1,m);
else
    exist=0;
```

```
        a=-1;
end
end


function [N,e,d,p,q,phi_N]=vulnarableKey(size)
%generates keypairs which are vulnarable
%to Wiener's ATTACK!!
while 1
        p=prime_random(size/2);
        q=prime_random(size/2);
        if q<p && q<2*p
            break;
        end
end


N=(p*q);
phi_N=(p-1)*(q-1);
%max d according to Wiener's Theorem
max_d=floor((1/3)*double(N)^(1/4));
while 1
        d=randomNumber(max_d);
        [e,coprime]= multInverse(d,phi_N);
        if coprime && mod((e*d),phi_N)==1
            break;
        end
end
end

function hash = num2hash(n)
%hashes the integer according to 'sha-1'
string=int2str(n);
persistent md
if isempty(md)
        md = java.security.MessageDigest.getInstance('SHA-1')
            ;
end
hash = sprintf('%2.2x', typecast(md.digest(uint8(string))
    , 'uint8')');
end


function e=continuedFractions(m,n)
% calculates the continued Fractions of m/n
e=[];
```

```
a=int16(floor(m/n));
b=mod(m,n);
e=[e a];
while b~=0
    m=n;
    n=b;
    a=int16(floor(m/n));
    b=mod(m,n);
    e=[e a];

end
end
function [n,d]=convergents(e)
%does rational approximation using
%continued fractions.
n=[];
d=[];
for i=1:length(e)
    if i==1
        n=[n e(1)];
        d=[d 1];
    elseif i==2
        d=[d e(2)];
        n=[n e(1)*e(2)+1];
    else
        n=[n e(i)*n(i-1)+n(i-2)];
        d=[d e(i)*d(i-1)+d(i-2)];
    end

end
end

function wienersAttack(size)
%whole Attack Flow of Wiener!!
[N,e,d,p,q,phi]=vulnarableKey(2^16);
isSuccess=0;
fprintf('Vulnerable RSA keys are generated. \n');

hash_N=num2hash(N);
hash_e=num2hash(e);
hash_d=num2hash(d);

fprintf('Hashed N value --> %s \n',hash_N);
fprintf('Hashed e value --> %s \n',hash_e);
fprintf('Hashed d value --> %s \n',hash_d);

```

```
contFrac=continuedFractions(e,N);
[numer,denom]=convergents(contFrac);

%iterating over possible k,d values
for i=1:length(denom)
    psbl_k=numer(i);
    psbl_d=denom(i);
    if psbl_k==0
        continue;
    end
    % checking the guess values
    message=randi(1024);
    crypt_msg=int64(powermod(message,e,N));
    if message==powermod(crypt_msg,psbl_d,N)
        %the true value is found.
        hash_d_found=num2hash(psbl_d);
        fprintf('Wiener Attack Succeed!!! \n');
        fprintf('Found Hashed d value --> %s \n',
            hash_d_found);
        isSuccess=1;
        break;
    end
end
%print failure
if isSuccess==0
    fprintf('Wiener Attack Failed!!');
end
end
```

References

[1] 'RSA (cryptosystem),' Wikipedia, 26-May-2020. [Online]. Available: https://en.wikipedia.org/wiki/RSA(cryptosystem). [Accessed: 27-May-2020].

[2] 'Wiener's attack,' Wikipedia, 24-Feb-2020. [Online]. Available: https://en.wikipedia.org/wiki/Wiener'sattack. [Accessed: 27-May-2020]

[3] https://math.boisestate.edu/ liljanab/ISAS/coursematerials/AttacksRSA.pdf