Network Security
Project 1
Hacking the Cipher

Louie Lu

September 30, 2017

1 Objective

To study and understand RSA common factor attacks.

1.1 Introduction

RSA is a public-key cryptosystem create at 1978[1]. Using RSA public key (n, e) we can encrypt the message **M** to ciphertext **C**

$$C = M^e \pmod{n}$$

Experimental Data Where n is the product of two large primes, and e is the public exponent, an odd integer $e >= 3$ that is relatively prime to $\phi(n)$.

We can decrypt the ciphertext by correspond RSA private key (n, d), since $de = 1 \pmod{\phi(n)}$ implies that

$$M = C^d \pmod{n}$$

So here comes the RSA Problem:

The RSA Problem : Given an RSA public key (n, e) and a ciphertext $C = M^e \pmod{\phi(n)}$, to compute M.[2]

2 RSA Common Factor Attack

Given an RSA public key (n, e), we will know about the product of two primes n, the main problem is, how to find out the factor p and q.

We can find the prime in two key via common factor attack[3], which is based on the fact of the coprime of the RSA public key.

If we got two different public key (*n1*, *e1*) and (*n2*, *e2*), if the choose of the factor is purely random, then *p1*, *q1*, *p2*, *q2* will be four different big prime numbers, e.g.

$$n1 = 1809632459 \times 2402636221 = 4347888492690697439$$

$$n2 = 1488286753 \times 1800980219 = 2680375002352738907$$

We can manipulate *n1* and textitn2 with greatest common divisor (gcd)

$$gcd(4347888492690697439, 2680375002352738907) = 1$$

But if the prime numbers didn't choose carefully, it may be choose the same prime number in different keys, e.g.

$$n1 = 1809632459 \times 1488286753 = 2693252016528515627$$

$$n2 = 1488286753 \times 1800980219 = 2680375002352738907$$

If we do the gcd on these number, it will find out the greatest common divisor

$$gcd(2693252016528515627, 2680375002352738907) = 1488286753$$

And that is the key of the RSA, we got the factor of *n1* and *n2*, we can then calculate the q and the private key of the two public keys.

3 Manipulate exist RSA key in Linux

Most of the manipulation of RSA key can be done by **openssl** command line tool[4].

3.1 Generate RSA Private Key

Using **openssl** to generate a 2048-bits RSA private key

```
$ openssl genpkey -algorithm RSA -out rsa.pem \
          -pkeyopt rsa_keygen_bits:2048
```

3.2 Extract RSA Public Key from Private Key

To extract public key from private key, use the following command

```
$ openssl rsa -pubout -in rsa.pem -out rsa.pub
```

3.3 Read the RSA Private Key

To read out the RSA private key attributes, use the following command

```
$ openssl rsa -text -noout -in rsa.pem
```

This will output all the private key attributes: modulus (n), public exponent (e), private exponent (d), prime1 (p), prime2 (q), exponent1 ($d \pmod{(p-1)}$), exponent2 ($d \pmod{(q-1)}$), coefficient (inverse of q) mod p).

3.4 Read the RSA Public Key

To read out the RSA public key attributes, use the following command

```
$ openssl rsa -text -pubin -noout -in rsa.pub
```

This will output the public key attributes: modulus (n) and exponent (e).

4 Manipulate RSA with Python

We can manipulate RSA with python, using **rsa** package[5].

5 Solving the RSA Common Factor Problem

Our mission is to find out the weak RSA key in 12 public key, and regenerate their correspond private key.

Function `FindCommonKey(`*path*`):`
- $pubkeys \leftarrow GetPubkeys(path)$
- **foreach** $p1, p2 \in Premutation(pubkeys)$ **do**
  - $g = gcd(p1.n, p2.n)$
  - **if** $g \neq 1$ **then**
    - **return** *p1, p2, g*
  - **end**
- **end**

Algorithm 1: Find the common key in the path

Function **FindCommonKey** help us to find out the public keypair that have the same prime number in the path.
Function **Main** provide the skeleton of the program to find out the key pair, and automate generate the private keys for the correspond public keys.

You can find the source code and result at my personal GitLab[6].

Function `Main()`**:**

foreach $pub1, pub2, g \in$ `FindCommonKey`$(path)$ **do**

`/* generate private key */`

$priv1 = GeneratePrivateKey(pub1, g)$

$priv2 = GeneratePrivateKey(pub2, g)$

`/* save private key */`

$SavePrivateKey(priv1)$

$SavePrivateKey(priv2)$

`/* verify keypair */`

$VerifyKey(pub1, priv1)$

$VerifyKey(pub2, priv2)$

end

Algorithm 2: Main program to generate, save and verify

References

[1] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[2] R. L. Rivest and B. Kaliski, "Rsa problem," in *Encyclopedia of cryptography and security*. Springer, 2005, pp. 532–536.

[3] S. Shoen, "Understanding common factor attacks:an rsa-cracking puzzle." [Online]. Available: http://www.loyalty.org/ schoen/rsa/

[4] "The openssl project: open source toolkit for ttl/stl." [Online]. Available: https://www.openssl.org

[5] "python-rsa: Python-rsa is a pure-python rsa implementation." [Online]. Available: https://stuvel.eu/rsa

[6] L. Louie, "Network security project 1." [Online]. Available: http://gitlab.nems.cs.nctu.edu.tw/louielu/ns-p1