Dual RSA and it's Security Analysis

Submitted

in partial fulfillment of

the requirements of the degree of

Bachelor of Engineering (Electronics and Communication)

By

Group No 33

Pratyush Bisht 2017AAPS0301H

Vaishnavi Ramapalli 2017AAPS0383H

Kanishk Katara 2017AAPS0416H

2020-21

Dr. Runa Kumari

(Supervisor)

Department of Electrical and Electronics Engineering

2020-21

1. Introduction

A cryptosystem is an efficient implementation of cryptographic algorithms to achieve security in an information system. There are three primary functions of any cryptography algorithm: privacy, authentication & integrity. A cryptography algorithm is made up of six components:

1. Plaintext: The message that can be understood by the reader

2. Ciphertext: Encrypted message that flows through the network

3. Encryption algorithm: An algorithm to convert the plaintext into ciphertext

4. Decryption algorithm: An algorithm that converts ciphertext back into plaintext

5. Encryption key: A key that a sender uses to encrypt plaintext

6. Decryption key: A key a receiver uses to decrypt the ciphertext

The key used in the encryption and decryption process may be the same or different. When different keys are involved in encryption and decryption, Such algorithms are classified under public-key cryptography. RSA (Rivest–Shamir–Adleman) is one of the first and most popular public-key cryptography algorithms. RSA can be used in data encryption/decryption, key exchange, and digital signatures. This project is to understand the working and implementation of RSA and its variants. The key generation algorithm of dual RSA, a variant of RSA with reduced key storage requirements will be looked into and the algorithm will be implemented in a high-level programming language (e.g. Python 3.6). The conditions required to implement dual RSA will also be probed along with its applications in the real world. Finally, the security analysis

of the RSA variant will be done, and its performance will be compared to the original version of the algorithm.

2. Objective

The objective of this paper is to implement the Dual RSA algorithm and also to perform the security analysis of the proposed Dual RSA algorithm. Thus we would also compare the security boundaries of different variants of Dual RSA namely small-e, small-d and rebalanced RSA to the standard RSA and its variants.

3. Literature Survey

Fast Variants of RSA – Dan Boneh & Hovav Shacham

This paper takes into account 3 different variants of RSA, which help us in speeding up the process of encrypting and decrypting. The paper starts by giving a brief review of the original RSA algorithm, then delves into these said variants emphasizing backward compatibility. This means that any system that works these three variants of RSA should be able to interoperate with the standard algorithm of RSA. The three variants discussed in this paper are, Batch RSA, Multifactor RSA and Rebalanced RSA.

Batch RSA does a number of decryptions for the cost of 1 decryption. Multifactor RSA uses a modulus of the form $N=p*q*r$ or $N =(p^2)*q$

Rebalanced RSA makes the decryption process faster by shifting the workload to the encryption side. This paper but does to deal with the security of these variants, and there is very little correlation between the security of the original

RSA algorithm and these variants. The 3 variants reviewed in this paper are very useful in speeding up the RSA implementation and become highly useful when one has to do thousands of decryptions at once.

Fast Decipherment Algorithm for RSA Public Key Cryptosystems – J.J. Quisquater and C.Couvreur

This paper discusses an algorithm that is used for faster deciphering of cryptosystems. It deciphers the given text about 4-8 times faster than the standard algorithm. As RSA's main goal is to transmit the message secretly, the deciphering of that message needs to be as difficult as possible and should be immune to attacks by enumerative techniques. Both the encrypting and decrypting algorithm require several hundred multiplications of integers of length 500 to 600 bits long, which makes it time a consuming process. So to reduce the time of operation the enciphering key is taken to be small value and the deciphering key is taken to a large value in order to prevent the above mentioned enumerative attacks. The algorithm presented in this paper tries to minimize the deciphering time by using the Chinese Remainder Theorem and improved modular multiplication algorithms. Even though this reduces the time of deciphering by 4-8 times it's still very high and further improvements are required.

An Approach Towards Rebalanced RSA-CRT with Short Public Exponent - Hung-Min Sun & Mu-En Wu

This paper aims to find a version of rebalanced RSA-CRT whose encryption time is much lesser than the normal rebalanced RSA-CRT and is still secure enough to use.

The standard RSA algorithm has the value of e and d of the magnitude of Euler totient of N and is usually very large. The RSA-CRT algorithm tries to reduce the decryption time by splitting the operation and then combine it with the use of the Chinese Remainder Theorem. Rebalanced RSA-CRT makes the decryption further faster by choosing the value d such that it results in a small value of CRT exponents and hence shifts the workload to the encryption side. This makes the encryption process very time consuming as the value of encryption key e is of the other of totient of N. The encryption time can be reduced if small values of e are considered, but that makes the algorithm susceptible to attacks and decreases its security, so this paper tries to find a sufficiently small e so that the encryption process becomes much faster as well as keeps the algorithm invulnerable to outside attacks. It finds the value of such e to be equal to, this makes the algorithm safe as well as reduces the encryption time by 3 times that of the original rebalanced RSA-CRT.

4. RSA Algorithm

The algorithm of the original RSA can be broken down into three parts - key generation, encryption and decryption. Key generation algorithm is used to generate two keys for the users (sender and receiver), public key and private key. These keys are used in encryption and decryption algorithms to encipher/decipher the plane text and cipher text. The detailed algorithms are given below.

Key generation algorithm:

- Choose two distinct prime numbers
- Compute n = pq
- Compute the totient function of the n = phi(n) = (p-1)(q-1)

- Choose any number 1 < e < phi(n) that is coprime to phi(n)
- Compute d, the modular multiplicative inverse of e (mod phi(n))
- The public key is (n, e)
- The private key is (n, d)

Example:

- We randomly choose p = 61 and q = 53.
- We get the value of n = 61*53 = 3233
- Totient function phi(n) = 3120
- We choose e = 17
- Using multiplicative inverse, we get d = 2753
- The public key is (3233, 17)
- The private key is (3233, 2753)

Encryption and decryption algorithms:

- Let the plaintext message be m and the generated ciphertext be c
- The encryption function is given by
    - $c(m) = m^e \bmod(n)$
- The decryption function is given by
    - $m(c) = c^d \bmod(n)$

Example: To make this simple we take e = 13, d = 37 and n = 143

- Let the message be m = 13
- The ciphertext is given by
- $c = 13^{13} \bmod 143 = 53$
- The plaintext can be obtained from ciphertext
- $p = 53^{37} \bmod 143 = 13$

5. Work done

The team has implemented the RSA algorithm and its variants. Following is the list of the algorithms of RSA variants that have been implemented:-

1. Small exponent

1.1. Small e RSA - When e is chosen as a random integer which is much smaller than the Euler totient of N(N=p*q), which makes the encryption process much faster as it reduces the mathematical complexity of the encryption process the algorithm is known as RSA small-e. In this process however, the decryption process remains the same as in the original RSA algorithm and hence its computational time is unaffected. The result after implementing the algorithm is as follows:-

```
kanishk@dell:~$ /usr/bin/python3.7 /home/kanishk/Desktop/rsa small e.py
RSA Encryption/Decryption small-e implementation
p =  107 , q =  29
Generating your public/private keypairs now . . .
Your public key is  (957, 3103)  and your private key is  (2509, 3103)
Enter a message to encrypt: secret message
Your encrypted message is:  10722418224512552418116024171397241810721072294422112418
Decrypting message . . .
Your message after decryption is:  secret message
```

The code of this algorithm can be found in the following link:
https://pastebin.com/1vdLLAD5

1.2. Small d RSA - This is the opposite of small-e as here d is chosen to be a random integer much smaller than Euler totient of N, and hence the decryption process now is faster and the encryption process remains unaffected.

The result after implementing the algorithm is as follows:-

```
kanishk@dell:~$ /usr/bin/python3.7 /home/kanishk/Desktop/rsa small d.py
RSA Encryption/Decryption small-d implementation
p =  11 , q =  13
Generating your public/private keypairs now . . .
Your public key is  (67, 143)  and your private key is  (43, 143)
Enter a message to encrypt: secret message
Your encrypted message is:  151404475140511342114015151246414O
Decrypting message . . .
Your message after decryption is:  secret message
```

The code of this algorithm can be found in the following link:

https://pastebin.com/ftMMHfpU

2. CRT Decryption

In this algorithm, Chinese remainder theorem is used for the process of decryption. Here the key generation step is modified to output (d, p, q) instead of (d, N) (N = p*q). Also, the decryption process is altered to calculate plaintext using dp (dp = d*mod (p-1)) and dq (dq= d*mod (q-1)) instead of using simply d, and then applying the Garner's algorithm to compute the result of decryption, dp and dq are called CRT exponents. This reduces the computational cost of the decryption process to about 1/4th of the original algorithm, leading to faster implementation.

The result after implementing the algorithm is as follows:-

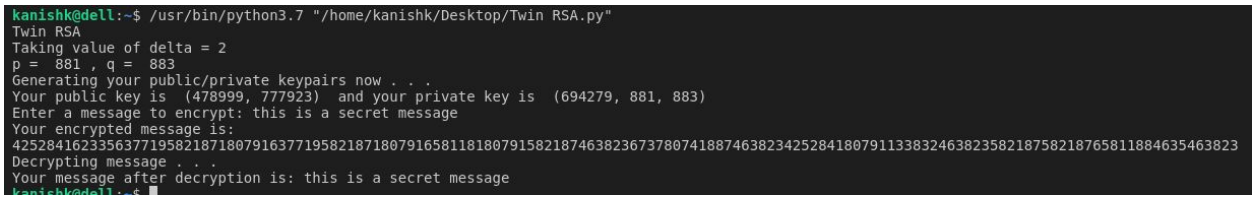
```
kanishk@dell:~$ /usr/bin/python3.7 "/home/kanishk/Desktop/CRT  decryption.py"
RSA Encryption/Decryption - CRT Decryption
p =  599 , q =  887
Generating your public/private keypairs now . . .
Your public key is  (353139, 531313)  and your private key is  (257155, 599, 887)
Enter a message to encrypt: secret message
Your encrypted message is:
317879508494245327100506508494420173741824703085084943178793178794046184143895084 94
Decrypting message . . .
Your message after decryption is: secret message
```

The code of this algorithm can be found in the following link:
https://pastebin.com/y4MZEdq8

3. Twin RSA

Twin RSA is a variant of RSA which consists of the pair of modulus N and N + delta. The value of delta is a small integer, usually taken as 2. As the storage requirement of delta is negligible, only one moduli N needs to be stored instead of 2. Hence, the storage requirements get reduced by the size of one RSA modulus.

The result after implementing the algorithm is as follows:-



The code of this algorithm can be found in the following link:
https://pastebin.com/W2pk9Gpq

4. Dual RSA

Dual RSA has 2 instances of RSA. It can be applied in ecash, untraceable email, electronic election systems, time stamping, anonymous access control and wherever authentication with secrecy of msg is required. It requires 2 pairs of primes (p1,q1) and (p2,q2) to form N1 and N2. The public key will be (e, N1, N2). Private key will be (d,p1,q1,p2,q2).

The team has implemented the small e variant of the RSA algorithm in python to get public and private keys.

In small e implementation, ne and n are taken as inputs with ne < n/2. Let no1=n/2-ne. Random numbers (x1,y1) of length ne and (x2,y2) of length no1 such that p1 = x1x2 + 1, p2 =x1y2 + 1 and q1 = y1y2 + 1 are primes are considered.Find a random ne bit integer e such that gcd(e,x1y1x2y2)=1. Now, find d such that ed = 1+k1(p1 - 1)(q1 - 1). If q2 = k1x2 + 1 is not prime then one has to find another e.The public and private keys can now be calculated as usual.

```
30
31      return ((e, n1,int(n2)),(d,p1,q1,p2,int(q2)))
```

In [9]:

```
1  print("RSA Encryption/Decryption")
2  p1,q1,p2,x1,x2,y1,y2=get_nos(ne=3,n=15)
3  print("Generating your public/private keypairs now . . .")
4  public,private = generate_keypair(3,p1,q1,p2,x1,x2,y1,y2)
5  print(public,private)
```

```
RSA Encryption/Decryption
Generating your public/private keypairs now . . .
(7, 2255, 1887) (1543, 41, 55, 37, 51)
```

For small d implementation, change inputs to (nd,n) and replace (e,d, ne) with (d,e, nd). Thus, we can use the same python code to calculate small d keys as well.

The code of this algorithm can be found in the following link:

https://pastebin.com/GfVPaA72

6. Contribution

The following contributions have been made apart from what was mentioned in the paper

- Checking if a number is prime for every number is a time consuming process. Instead of the traditional approach to check prime numbers, we used the sieve of eratosthenes algorithm to check if a number is prime or not. In this algorithm, we create an array of all the numbers beforehand and mark those numbers that are prime. In this algorithm, the time complexity to check whether a number is prime or not is O(1) which is way better compared to O(n) time complexity of the original algorithm.
- In the encryption and decryption process, the value of exponents can be very high. This did not only increased the complexity and runtime of the algorithm but also caused overflow of integer data type in most cases. To tackle this problem we used the modular arithmetics method and decomposed the power into smaller powers. To further reduce the running time we used the dynamics programming approach to calculate the power. We decomposed the exponents into powers of two. At every point we used the result of the previous step to get the next value. Eg if we calculate x^{23} we decomposed it into $x^0 * x^1 * x^2 * x^4$. At every step, to calculate x^{i+1}, we square the value of x^i and check if it added it to the sum if i existed in the decomposition.

7. Approach

The team has decided to read all the references in the proposed paper to understand various aspects of the RSA algorithm. After going through the papers it was decided to

implement the RSA algorithm and its various variants using python and look for the time taken to implement each algorithm with huge primes and the security aspects.

8. Security Analysis

Small exponent RSA

The small-d RSA algorithm can be easily broken down, as long as the value of d, the private key exponent is smaller than $N^{1/4}/3$, where N is the public moduli which satisfies, $N = pq$, where p, q are 2 prime numbers. Michael J. Wiener, found an attack against the small-d RSA algorithm, which employs continuous fractions. As we know that totient of N is equal to $(p-1)(q-1)$ and that $ed \equiv 1 \ mod(p-1)(q-1)$.Then there exists an integer k such that k/d is a continuous approximant of e/N. As we know that both e, the public exponent, and N are public, and as d is small, then by employing continuous fractions we can easily find the value of private exponent d.

The small-e RSA is safe for even small values such as 3, if and only if we use it to encode the plain text only once. If we send the same plaintext with the same public exponent, using different public moduli, then the small-e RSA algorithm would be highly susceptible to attacks, as it can be broken down using the Chinese Remainder Theorem and the Hastad's attack.

CRT Decryption

In the CRT-RSA decryption we break the private exponent d into dp and dq such that $dp = d \ mod(p-1)$ and $dq = d \ mod(q-1)$, therefore decryption steps also breaks into two parts - $Cp = C^{dp} \ mod \ p$ and $Cq = C^{dq} \ mod \ q$, these can now be used to compute

the original message. But if a fault is introduced in any one of the two signatures, which can be caused by a plethora of ways, such as variation in supply voltage, temperature changes, X-rays, white light etc. then the CRT-RSA algorithm is successfully broken and it would become impossible to recover the message. These types of attacks are known as faults injection attacks, the attacker then can perform side channel analysis to recover some of the sensitive information. Although CRT-RSA is faster and safer than the small exponent RSA algorithm, it is still very much susceptible to these fault attacks.

Twin RSA

According to Arjen K. Lenstra and Benjamin M. M. de Weger the people credited for the discovery of the twin RSA algorithm, twin RSA is as secure as the normal RSA algorithm, it is susceptible to the same attacks as the original RSA algorithm. Twin RSA is used wherever 2 instances of RSA is required. In twin RSA, instead of storing two different moduli for the instances, we store one moduli and, a fixed number, the difference between the two moduli, which is generally a small integer e.g. 2. So the only different way to attack the twin RSA algorithm is to somehow use this fixed difference to factor out the two moduli. But according to Cunningham factoring project calculating the factors of a number of type $b^k \pm 1$ does not in any way help in calculating the factors of a number of type $b^k \mp 1$. Thus attacks on Twin RSA are the same as normal RSA, like the continuous fractions attack and the Hastad's attack, implying this algorithm is as much as secure as the normal RSA and has a better storage management.

Dual RSA

There can be various attacks on small-e dual RSA, one such attack focuses uses the special structure of the dual RSA's moduli. But first the parameters k1 and k2 of the key generation algorithm must be found. The parameters k1 and k2 are found using lattice based method with an algorithm known as exhaustive searching with a parameter L. But this exhaustive searching algorithm will fail if we ensure that the size of the public

exponent is greater than the sum of N/4 and L/2, $e > N/4 + L/2$, where N is the public moduli and L is the parameter of exhaustive searching algorithm.

As this attack tries to manipulate the special structure of the dual RSA's moduli, it finds a system equation consisting of 2 equations and 5 variables. Out of these 5 as k1 and k2 are already known, that just leave 3 variables and 2 equations. Depending upon the value of the public exponent, an exhaustive search or brute force search could be used to guess the value of one variable and the system of equations can be solved to find the other 2 variables. To defend against this attack a value of public exponent should be chosen such that it makes the exhaustive search infeasible. It's found that if the value of public exponent is in between the range L1 and N/2-L1, both included, the search becomes infeasible. Here L1 is chosen such that the exhaustive search of the space 2^L1 becomes impossible.

In case of small-d dual RSA all the attacks that apply to small-d RSA also work for dual RSA, therefore the private exponent should satisfy $d > 7N/6 - 1/3\sqrt{(N^2 + 6Ne)}$, apart from this the condition limiting the value public exponent in small-e dual RSA also apply here, in that equation we substitute e with d - $d > N/4 + L/2$, here N is the public moduli and L is the parameter of exhaustive searching algorithm. There is one more attack to which small-d dual RSA is vulnerable, namely the lattice based attack, which takes the help of Lenstra-Lenstra-Lovasz lattice basis reduction algorithm to solve complex equations of d,e k1,k2,N1,N2 and concludes that in order for attack to be successful the value of private exponent should be smaller than N/3.

9. Comparison between original RSA and dual RSA

In this section we will compare the performance of original rsa small e and dual rsa small e.

Sr.	Aspect	RSA	Dual RSA
1	**Key Pairs**	This algorithm has a single key pair.	Dual RSA has 2 key pairs, and hence used where two instances of RSA are required.
2	**Security**	RSA is vulnerable to many attacks depending upon its type.	The security boundary of the different types of RSA, namely small-e, small-d,Rebalanced RSA, etc, is raised when dual RSA is applied.
3	**Computational Cost**	The key generation algorithm is less computationally expensive.	The key generation algorithm of dual RSA is significantly more computationally expensive, but can be handled by modern computers effectively.
4	**Time consumed for Encryption/Decryption**	RSA's time consumption is moderate.	Dual RSA consumes a little more time when compared with RSA, but it is a good trade off for the additional features and security.

Apart from the above dual RSA helps us to implement Blind Signatures successfully, which was a security risk in case of original RSA.

10. Applications

Authentication & Secrecy

Dual RSA solves the problem of relocking which is encountered in the standard RSA algorithm which arises when a person first signs a message to maintain authenticity and then encrypts it. If the user for which the message is intended, as the value of modulus smaller than the value of modulus of the person who sent it, then the receiver would not be able to recover the message. This can be solved by Dual RSA. In Dual RSA there are 2 instances of RSA with some key pairs. We define a value called threshold, h, and each user's key pair will then be created such that the modulus value of one instance of RSA is less than the defined threshold and the other is greater. So now if the sender first signs the message with the instance of RSA with modulus value smaller than the threshold and encrypts it with the other instance with modulus value greater than the threshold then it could be successfully decrypted by the receiver.

Blind Signatures

Blind signature basically refers to the concept of a message generated by one user to be signed by a different user without revealing any details of that message to the user signing it. This has various applications like in e-cash, electronic voting system, untraceable electronic mail etc. A problem with this is that if the signing user uses the same key pairs for both signing blind signatures and encrypting/decrypting operations then it becomes easy to perform a man in the middle attack. Dual RSA solves this problem as it has two different instances of RSA and then one of the instances can be exclusively used for signing blind signatures.

11. References

- D. Boneh and H. Shacham, “Fast variants of RSA,” *Crypto Bytes*, vol. 5, no. 1, pp. 1–9, 2002.

- J.-J. Quisquater and C. Couvreur, “Fast decipherment algorithm for RSA public-key cryptosystem,” *Electron. Lett,* vol. 18, no. 21, pp. 905–907, Oct. 1982.

- H.-M. Sun and M.-E.Wu, An approach towards Rebalanced RSA-CRT with short public exponent Cryptology ePrint Archive, Report 2005/ 053, 2005.

- M. J. Hinek, “Another look at small RSA exponents,” in *Topics in Cryptology-CT-RSA 2006*, ser. Lecture Notes in Computer Science, D. Pointcheval, Ed. New York: Springer, 2006, vol. 3860, pp. 82–98.

- A. K. Lenstra, B. M. M. de Weger, T. RSA, E. Dawson, and S. Vaudenay, Progress in Cryptology—Mycrypt 2005, ser. Lecture Notes in Computer Science. New York: Springer, 2005, vol. 3715, pp. 222–228

12. Appendix

Given below are the links to the codes implemented

- Small e RSA: https://pastebin.com/1vdLLAD5
- Small d RSA: https://pastebin.com/ftMMHfpU
- CRT Decryption: https://pastebin.com/y4MZEdq8
- Twin RSA: https://pastebin.com/W2pk9Gpq
- Dual RSA: https://pastebin.com/GfVPaA72