Factorizing RSA Key Using Prime Database

[1]Habibollah Lotfi, [2]Smita Bedekar, [3]Abdullah Ansari
[1,2,3]Interdisciplinary School of Scientific Computing, University of Pune, Pune, India

Abstract
Nowadays prime numbers have significant role in security fields. In security, factorizing a key is a tough challenge and various methods like ECPP, NFS and etc has been developed and tested theoretically as well as practically. The method that we have considered for factorization uses a database of prime numbers for generated keys which applied in RSA [1].

Keywords
Prime Numbers; Network; Security; Factorization; ECPP; NFS

I. Introduction

Security relies on having a reliable connection to transfer data safely. In the Network Security we have security protocols for different layers. Each day there are enormous efforts to check these protocols reliability, strength and weakness. Security depends on two very important subjects: first one is Hardware and second one is developing Algorithms. By developing Hardware, a number of operations, that can be done per time unit, increase exponentially. Thus for example if 5 years ago, time consumption for factorization of a public key was 1 year with using current Hardware, now for the same key it takes much less time, maybe around 1 month.
There are various kinds of attack to factor a public key; one of them is Integer Factorization Attack [2]. Factorization Algorithms and fast calculation are possible and sometimes there is less advantage of using of sophisticated Hardware.
Clearly there are two solutions to have a secure system:

- Changing Security Algorithms.
- Increasing Length of public keys.

Indeed, number of Security Algorithms is rare that is why it is not possible to keep changing the algorithms. Additionally sometimes adapting and updating old systems are not easy or possible. On the other hand, an easy way is to keep on increasing the length of the public key which is not favorable regarding time and memory.
Some new algorithms such as Number Field Sieving (NFS) [5], Elliptic Curve (EC) [2] Factorization, Modified Pollard Rho-1, etc were able to factorize a key in a much efficient manner. All of them rely on mathematics; rather than computational part. But in our effort we are more interested in computational concept.

II. Our Approach

Idea of this method is to generate a Database of primes which provides all possible public-keys and save them in a hash table. To factor a public key, one should look up in this table to find the public-key then get its factors through the key and its corresponding prime multipliers.

A. Providing Prime Numbers

First task is providing a Database of Primes. There are number of algorithms which are able to test Primality of a Number. But the output of all those algorithms is not completely reliable. Here the critical steps are:

1. Primality Testing [3]
2. Pseudo Primality Testing Algorithms [5]

Pseudo Primality Testing Algorithms are using to generate prime numbers but they are not able to cover or generate all prime numbers. Thus if we use these kind of algorithm we will miss some prime numbers in our database of prime numbers, therefore the generated Prime Database is incomplete.
To solve these problems we decide to generate primes in an assured manner. The time consumption of this method is high but it guarantees to obtain whole prime numbers database. We applied Brute Force [2] method, divide a number to all primes below/equal its square root. If a number has passed this condition obviously it's a prime number then we will go to the next step. Once we generate this database of prime, we can use it for several times, so the cost of (time and resources for) providing this reliable database of prime is acceptable.

B. Making Keys

Provided with Primes Database, we need to calculate all possible public keys afterword by multiplying a prime number with the rest of prime numbers in database.

C. Making Keys and Primes Database

In this stage we store each public key considering its relation with prime factors in a table. Making relation between key tables and prime number tables is the main issue that should be handled very carefully.

III. Applying Key Factorization

Now we are able to factorize a public key, given as an input number to our Database. First of all we search in key tables to find the mach. Then we look for its pair of prime factors.

IV. In Practical and Real World

A. Table and Some Useful Information

In practice we come to know the reality of this method. So as usual, time consuming and memory are two basic factors which we should be aware of. Therefore time consuming to generate prime numbers and memory requirement (Hard Disk) to save them are shown in the table below:

Table 1: Experimental Result

	Experimental Results Table		
	Time (s)	Memory	Number Counting
10	0.0	---	4
10^2	0.0	63 B	(21) 215
10^3	0.001795	572 B	(143) 168
10^4	0.01618	5.2 KB	(1061) 1229
10^5	0.18514	49 KB	(8363) 9592
10^6	1.87577	471 KB	(68906) 78498

	Time (s)	Memory	Number Counting
10^7	28.7873	4.5 MB	(586081) 664576
10^8	514.3245	43.7 MB	(5096876) 7761455
10^9	7423	430 MB	(45086079) 50847554
10^{10}	184709	4.1 GB	(404204977) 455052511

Details of computer configuration, Inter Core i3 3.2 GHz processor i386, 2 GB Main
Memory (RAM), Operating System: Fedora 14 (Laughlin)

Obviously with a quick view over the table, one can easily observe:

1. Time Consuming
The time consuming to generate primes is increasing exponentially with increasing length of the number.

2. Memory Requirement
Moreover the memory requirement is also raising exponentially as well.

B. Calculations to Estimate Memory Requirement and Time Consuming

1. Memory Requirement
A computer needs one byte to store one digit. When the length of a number increases, it needs more memory to store it. If we want to store a number in memory we need:

$$L = n+1$$

where n is length of number in digits. For example if:

$$N = 308457624821$$

$$L= 13 * 1 = 13 \text{ bytes } +1 \text{ byte}$$

(To store a number, it needs the number of digits bytes plus one. Because last bit of each number is
NULL and system is automatically going to perform it.) To store all 13 digits prime numbers we need:

3.6470 TB

and then to store all 24 digits we need a storage of the size:

15016011617.028559
= 14664073 PB
= 16 ZB (2^70)

This is a very huge Memory requirement. In 2009, entire World Wide Web (WWW) approximately had 500 EB, which is just half ZB.

2. Time Consuming
To achieve better time consuming easily we can do almost all of the computational part in parallel.

V. Conclusion

A. Very Less Time Consuming
In the mentioned method, to provide a Prime Database, by using Distributed Systems and Parallel Computing we can speed up the algorithm more efficiently.

B. Not Enough Resources in the World
To store the result of above method we do not have enough resources in the world even for small primes, i.e. a number with length of 30 digits!

C. Length of Keys in Real World
Nowadays, considering minimum length of key in Crypto Systems, i.e. RSA with 309 digits (1024 bits), it is not possible to think about breaking keys using Primes Database.

D. Key Tables
Providing such enormous Database, taking back up, Database Management, etc, won't be easy and somehow is impossible. Dealing with this Database, making relation between entities is rigorous. Worse situation will occur in performing multiplication which needs huge memory (Hard Disk) requirement.

VI. Further Research
Considering the mentioned method in this paper, we observe that there is possibility to factor RSA keys with high efficiency. If we find a method to store all possible keys including prime numbers table we will be able to factor the public-key in RSA security algorithm.

VII. Acknowledgment
I express my profound sense of gratitude and thanks to Dr. B. Gore, Assistant Professor at the Centre of Modeling and Simulation, University of Pune for his spiritual support during this project.

References
[1] Rivest, Ronald L., Adi Shamir, Len Adleman,"A method for obtaining digital signatures and public-key cryptosystems", Communications of the ACM, Vol. 21, No. 2, 1978, pp. 120-126.
[2] Yan, Song Y.,"Cryptanalytic attacks on RSA", Springer, 2007.
[3] Atkin, A. O. L., Morain, F.,"Elliptic curves and primality proving", Mathematics of computation, Vol. 61, No. 203, 1993, pp. 29-68.
[4] Prime Number Theorem (2016) [Online] Available: http://en.wikipedia.org/wiki/Prime_number_theorem
[5] Cohen, Henri, et al., eds.,"Handbook of elliptic and hyperelliptic curve cryptography". CRC press, 2005.