Analysis of the RSA Encryption Algorithm

Betty Huang

June 16, 2010

Abstract

The RSA encryption algorithm is commonly used in public security due to the asymmetric nature of the cipher. The procedure is deceptively simple, though; given two random (large) prime numbers p and q, of which n = pq, and message m, the encrypted text is defined as c = me (mod n). E is some number that is coprime to the totient(n). The public key (n, e), however, makes it difficult for the user to find the private key (n, d), due to the fact that given only n, it is extremely difficult to find the prime factors p and q. The fastest methods currently have O(sqrt(n)) complexity, but require expensive resources and technology (Kaliski). The aim of this paper is to analyze various factorization methods.

1 Introduction

My project aims to research and compare the various factorization methods available, including a proposal towards improving the current methods already available. There is an industry demand towards improving the speed of the RSA cryptosystem, as it is widely used by agencies that require secure transfer of information between clientele and administrators. Since the security of the algorithm is depended on the mathematical difficulty and computationally "hard" problem of factoring, it is important to recognize potential exploits of factorization.

1.1 Background

The field of public key cryptography is rapidly advancing, due to the growth of internet connections around the world. There is a great demand for proper distribution of public keys and ability to secure data across greater networks. Prior to the 1970s, encryption and decryption was accomplished through the same key. However, the decryption (or private) key should ideally be accessible only to the regulators and not the general public. Diffie and Hellman, two researchers from Stanford University, proposed the idea of using different keys for encryption and decryption, which allowed for private access to the decryption key (Hellman). Therein lays the heart of asymmetric key ciphers, leading to a revolutionary overhaul of securing data online. No longer did suppliers have to worry about allowing unauthorized access to the decryption key, which increased overall security of sensitive data transfers on the web. The RSA encryption algorithm was developed by Ronald Rivest, Adi Shamir, and Leonard Adleman at MIT, and falls into the class of asymmetric ciphers known as "trap door ciphers." Simply put, trap door, or one way ciphers allow for simple encryption, but decryption remains in the realm of nearly impossible without the decryption key. The fundamental building blocks of RSA remain in number theory, modular exponentiation, integer factorization ("expensive"), and prime generation. At the core, RSA relies on two facts:

1. multiplying numbers together is easy, and

2. factoring numbers is hard (Kaliski).

Current methods put factorization in square root order, which means that factoring 100000 requires approximately 367 time steps. With numbers that exceed 100 digits, the square root of such a number is around 50 digits, which means that that there are 10^{25} possibilities. If a computer were able to calculate one million factorizations in a second, then over the span of the universe, it could try 10^{24} different combinations total (Davis 03). Thus, to find the factors of a number with 100 digits, one computer would take over the span of the universe. Since the beginning of human existence, mankind has wrestled with the issue of factorization. The first prominent breakthrough was accredited to Fermat, who coined the "difference of two squares" algorithm. First, a smallest perfect square is found that is greater than the number in question, and then checking to see if the difference between the number and the square is another perfect square. If it is not, then find the next smallest perfect square that is greater than the first perfect square. This method exploits the difference of squares property $(a^2-b^2=(a-b)(a+b))$, and is better than brute force. There have been other methods since developed, including Lenstra's elliptic curve technique, and Pollard's probabilistic algorithm. The fastest process currently is an expansion of Fermat's technique, known as the Quadratic Sieve (QS). There are several permutations to the Quadratic Sieve, including processes that lend itself to parallelized interpretations.

For a long period of time, factoring numbers has been a purely academic problem with no practical applications. It was known to most humans and mathematicians that factoring large numbers, however, is a difficult and expensive problem due to the nature of finding factors. It was not until Rivest, Shamir, and Adleman came along and developed the RSA encryption algorithm that factoring numbers gave way to a practical use. The core of the RSA is the simplicity in which numbers can be multiplied, but the difficulty in factoring numbers (Davis). Essentially, the RSA function chooses two random prime numbers p and q, and then multiplies these two numbers together (to guarantee that there are two factors, and only two factors).

1.2 Scope of Study

The field of study will be in encryption and cryptanalysis in C.

2 Review of Literature

Rivest's original paper on the RC5 algorithm has been helpful in my research and understanding of the encryption mechanism in order to approach it from an angle of attack. Since I wanted to experience the thought process on breaking a cipher, I did not review other, widely available literature on RC5 cryptanalysis. After that, I studied two papers that detailed two different methods in breaking the RC5. The first paper, titled "On Differential and Linear Cryptanalysis of the RC5 Encryption Algorithm," offered the first approach taken towards attacking the RC5. Authors Burton S. Kaliski Jr. and Yiqun Lisa Yin focused efforts on recovering the expanded S table, rather than the input (thus, their results are key-length independent). Figure two explains the algorithm procedure that Yin et al. used. Refer to Appendix A for the version that I used for RC5.

In conclusion Yin et al suggest using a 12 round implementation of the RC5, which helps prevent against most common differential/linear cryptanalysis attacks. However, this paper had been published in 1995, and I expect that recent innovations would make it possible to use attacks against the RC5, so I continued looking for more recent papers. The most recent update was in 1998, also authored by Yin and Kaliski, which further expanded on their research.

There are several methods in dealing with block ciphers:

1. The exhaustive search - this is the most intuitive and brute-force type of attack. Simply put, the attacker finds one plaintext/ciphertext combination, and then essentially runs through all possible combinations until a match is found.

2. Statistical searches - the attacker analyzes patterns and matches between plaintext/ciphertext combinations

3. Differential Cryptanalysis - the attacker choosese two plaintexts, P1 and P2, with altercations (”difference”) P’ between the two. After P1 and P2 are encrypted, their ciphertexts C1 and C2 are compared, and the difference between the two is identified as C’. The goal of differential cryptanalysis is to find a a pair of (P’, C’) that occur with more than normal probaility.

4. Linear Cryptanalysis - the idea here is to find items (plaintexts, ciphertexts, etc.) that occur over several iterations with probaility != 1/2 (Kaliski).

This research has led me since to Rivest’s RSA Encryption Algorithm; The first paper I encountered was published by B. Kaliski, which described the mathematical properties that functioned as the backbone of the RSA encryption algorithm. He details several mathematical properties that are mentioned in my literature review of the paper. In summary, the core aspect of the security of the RSA is that factorization is difficult, or expensive. In terms of time spent/resources required, factoring large numbers has gained notoriety in the mathematic community since the advent of these usage. The RSA exploits this inherent property in order to increase security, but is considerably slower than other competing asymmetric ciphers (ex. DES).

3 Development

For the first part of the quarter, I refined the code implementation of the RSA cryptosystem in C and reading the mathematical concepts behind the code.

In addition, I read several papers outlining the core concepts and foundations that the algorithm rests upon. When choosing large prime numbers, it's often expensive for computer systems to process 200+ digits, which increases the difficulty (ie. memory and time spent) of the problem. Upon further examination, I found that the fastest method available was the Quadratic Sieve, explained in Appendix B. The QS lends itself to parallel implementation, and reigns as champion in terms of speed for factorization fewer than 110 digits. Numbers that exceed this bound are better approximated by General Number Field Sieve (Gerver). As of now, I have tested code implementations of several various prominent prime factorization techniques in order to determine the fastest on several conditions. Although many authors (Gerver, Pearson, Pomerance) agree that the QS is the fastest algorithm up to 110 digits, it is not as successful for "medium large" numbers.

Following the implementation of the RSA encryption algorithm, I focused on computational techniques on factoring. The majority of the security that follows from the RSA algorithm comes from the age-old problem of factoring. There are various prime factorization methods available, and as time passes, they become more sophisticated in nature and computational power. Following the discussion of the significance of these factorization techniques, I analyzed several algorithms that were within the computational confines of an average processor.

The first algorithm is also the one that follows most intuitively for individuals that are presented with a number to factor, which is to divide every number less than the number in question until prime factors are obtained. This is known as trial division, and is often the method of choice for numbers that are less than 5 digits. For example, given the composite–which a number that is divisible– 225 (a ?powerful number"), we recognize that 5 divides evenly into 225, leaving us with a remainder of 45, which is divisible by 5, leaving 9. From there, we see that the prime factors of 225 are 3, 3, 5, and 5. The unique property of prime factorization is that there is only one such combination of factors that form a particular composite (the RSA algorithm exploits this property by multiplying two prime numbers together, which means that two prime numbers p and q are the only two possible factors of the composite n).The trial division algorithm is relatively simple, which is why it is the most popular approach for most people that do math on a daily basis. $(a^2 - b^2 = (a - b)(a + b))$ Expanding on this property, the Fermat theorem is less intuitive, but exploits a property with numbers and their squares. If a number n can be expressed into $(a^2 - b^2)$, thus the number can be factored into (a+b)(a-b). Coding this algorithm was fairly simple; I would attempt to find an a such that $a^2 - N = b^2$. This algorithm is only efficient for certain numbers, such as 8051 = (8100 - 49).

The pseudocode:

```
Given n = ab, find a and b. to do this, we try to represent n as the difference of two perfect squares. for all$i > 0$, x = sqrt(n) + i. calculate $x^2 - n$for all i until $x^2 - n = y^2$, a perfect square. Now, $x^2 - n = y^2$, thus
$n = x^2 - y^2$
n = (x - y)(x + y)
a = x - y
b = x + y
```

A worked example, as demonstrated in Pomerance's paper, is the integer 8051. The simplest way would be to try as many numbers as possible until factors were found. Fermat's solution is much more elegant; instead, the number is re-expressed as 8100 - 49.
$8100 = 90^2$
$49 = 7^2$
so the factors are 90 and 7.

I considered the possibility of improving on these algorithms. For trial division, there are obvious improvements that could be made in terms of the numbers checked. For example, if the numbers 2 and 3 were checked, there would be no point in checking 6, 12, 18, 24, etc... and this could dramatically reduce the number of cycles that brute force would require. Another improvement could be checking only prime numbers, but that might be difficult because that would require the user to either a) have a list of the prime numbers up to an arbitrary number available, and b) find all the prime numbers, which would be an impossible problem. The Fermat theorem has obvious improvements as well, in the case of an ?bad? number to factor. Having an exit case after a certain number of tries would avoid the possibility of running into a case that is less effective than brute force itself.

4 Results

The second part of my development during this quarter was the implementation and testing of the RSA algorithm in C. The code is included in Appendix B, and the results are as follows: see Figure 1.

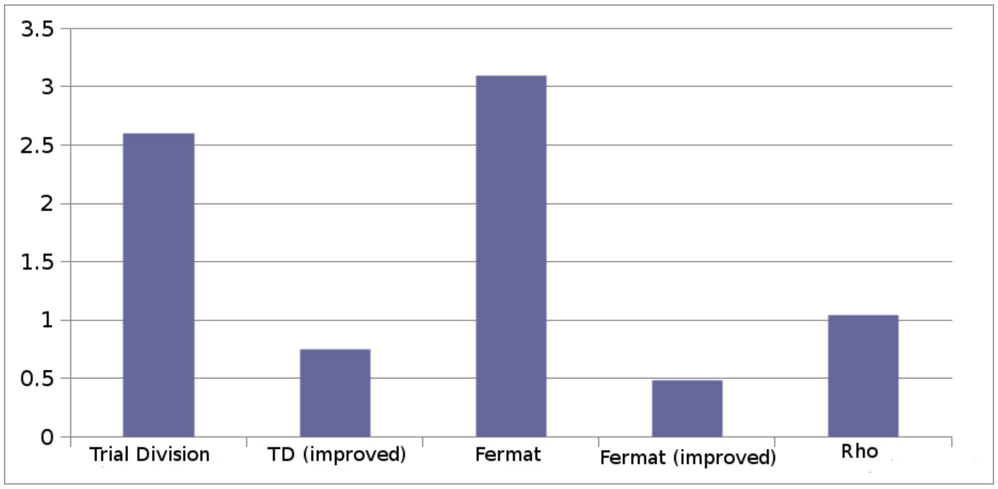

Figure 1: comparison of the various methods of integer factorization.

Appendix A. A Simple Example of the RSA Encryption Algorithm

The algorithm:
$c = M^e (mod\ \mathrm{n})$
$M = c^d (mod\ \mathrm{n})$
Public key: (n, e)
Private key: (n, d)
Where C represents the ciphertext, and M represents the plaintext message in numeric format.
Choose two large primes (100 digits or more) p and q. For simplicity, small primes will be chosen so the math is easier to follow: p = 11 and q = 7. Multiply p*q, and set N equal to the result. This N is part of the public key.
N = p*q = 11 * 7 = 77
Choose a number e that is coprime (shares no common factors) with
k=(p - 1)(q - 1).
k=(p - 1)(q - 1) = (10)(6) = 60.
We choose e to be 13, which is also part of the public key. This information is sufficient to encode the message.
For the decryption, one must find d such that ed = 1 (mod k). Since the number k is not released to the public, it is extremely difficult for someone to find d. Since we do know what k is, we find that:

13d = 1 (mod 60), or the inverse modular function
d = 37
Suppose a friend passes the message m = 53 (note that $1 < m <$ N). In order to encrypt the message,
$C = m^e(mod\ \mathrm{N})$
$C = 53^{13}(mod\ 77)$

The subsequent decryption:

$M = c^d(\mathrm{mod\ N})$

M = 53

Appendix B. Comparison table of various integer factorization methods

1.55E+45 Time Success Resultant

Brute Force 2.6 no 2 * 3 * 607 * 7669 * 55330323753934231552903581534602334349
Brent (p-1) 0.748 yes 2 * 3 * 607 * 7669 * 20947 * 413551 * 51083807 * 370770947 * 337227786373
Pollard 3.099 no 2 * 3 * 4655083 * 55330323753934231552903581534602334349
Williams (p+1) 0.483 yes 2 * 3 * 607 * 7669 * 20947 * 413551 * 51083807 * 370770947 * 337227786373
Lenstra (Elliptic Curve) 1.044 yes 2 * 3 * 607 * 7669 * 20947 * 413551 * 51083807 * 370770947 * 337227786373

Appendix C. Code

```

#include <stdio.h>
#include <math.h>

 int phi,M,n,e,d,C,FLAG,m,D;

int gcd(int a,int b) //greatest common divisor
{
  int c;

  if(a<b)
  {
    c = a;
    a = b;
    b = c;
  }

  while(1)
  {
    c = a%b;
    if(c==0)
      return b;
    a = b;
    b = c;
  }
}

// int privatekey(int val)
// {
//   int d;
//   d= e

}

int totient(int X) // calculates how many numbers between 1 and
    N - 1 which are relatively prime to
N.
{
  int i;
        phi = 1;
        for (i = 2 ; i < X ; ++i)
             if (gcd(i, X) == 1)
                 ++phi;
      return phi;
int encrypt(int message)
{

  int result;
  result=message;
  int x;
  for (x=0;x<e;x++)
  {

    result=result*message;

  }

  result=result%n;
  return result;

}

int decrypt(int ciphertext)
{
  int x;

  int plainres=plainres;
  for (x=0;x<d;x++)
  {
    plainres=plainres*ciphertext;
  }
  plainres=plainres%n;
  return plainres;
}

}
```

References

[1] Landquist, E., The quadratic sieve factoring algorithm. Math, 448, 2?6.

[2] H. W. Lenstra Jr, Factoring integers with elliptic curves, Annals of mathematics 126, no. 3 (1987): 649-673.

[3] D. Pearson, A parallel implementation of RSA, Cornell University (July 1996).

[4] Montgomery, P.L., 1994. A survey of modern integer factorization algorithms. CWI Quarterly, 7(4), 337?365.

[5] C. Pomerance, Analysis and comparison of some integer factoring algorithms, Mathematics Centrum Computational Methods in Number Theory, Pt. 1 p 89-139(SEE N 84-17990 08-67) (1982).

[6] C. Pomerance, 2008. A tale of two sieves. Biscuits of Number Theory, 85.

[7] Factoring large numbers with a quadratic sieve, Mathematics of Computation 41, no.163 (1983): 287-294.

[8] T. Denny et al., On the factorization of RSA-120, in Advances in Cryptology(CRYPTO)93, 166-174.

[9] Parallel implementation of the RSA public-key cryptosystem, International Journal of Computer Mathematics 48, no. 3 (1993): 153-155.

[10] Kaliski, The Mathematics of the RSA Public-Key Cryptosystem, RSA Laboratories. April 9 (2006).