Encryption & Decryption using Secure RSA

Amninder Singh Narota
Department of Computer Science
Central Michigan University, U.S.A
Email: narot1a@cmich.edu

Abstract—The Secure RSA algorithm [25] is generally 6 step algorithm and the security is based on the randomly selected 2 pairs of prime numbers on the assumption that it is easy to find the multiply to prime numbers together, but it is extremely difficult to factor their product. In today's world speed of processing is decreasing at an exponential rate which makes these numbers easily crackable. To overcome this limitation we need to look at the bigger aspects for the application and even larger prime number. Working with even larger numbers are somewhat limitation to computer science since we are limited to at most 64-bit integers.

This paper introduces the concept and implementation of RSA algorithm for security purpose, Mersenne Twister for generating pseudorandom number with large key space and faster primality test to generate prime number. We will enhance the performance of the system by adding one more prime number to the algorithm and implement external libraries with low computation time to work with large numbers. As the result, it will be faster to encrypt and more secure to be decrypted by crypto analyst.

Index Terms—Cryptography, RSA, MT19937, Mersenne Twister, Miller-Rabin, Prime Numbers, Pseudorandom Numbers, Karatsuba Multiplication

1 Introduction

Cryptography makes secure web sites and electronic safe transmissions possible. For a web site to be secure all of the data transmitted between the computers where the daw is kept and where it is received must be encrypted. This allows people to do online banking, online trading and make online purchases with their credit cards, without worrying that any of their account information is being compromised. Cryptography is very important to the continued growth of the internet and electronic commerce.

E-commerce is increasing at a very rapid rate. By the turn of the century, commercial transactions on the internet are expected to total hundreds of billions of dollars a year. This level of activity could not be supported without cryptographic security. It has been said that one is safer using a credit card over the internet than within a store or restaurant. It requires much more work to seize credit card numbers over computer networks than it does to simply walk by a table in a restaurant and lay hold of a credit card receipt. These levels of security, though not yet widely used, gibe the means to strengthen the foundation with which e-commerce can grow.

People use email to conduct personal and business matters on a daily basis. E-mail has no physical form and may exist electronically in more than one place at a time. This poses a potential problem as it increases the opportunity for an eavesdropper to get a hold of the transmission. Encryption protects email by rendering it very difficult to read by any unintended party. Digital signatures can also be used to authenticate the origin and the content of an e-mail message.

Cryptography is also used to regulate access to satellite and cable tv. Cable tv is set up so

people can watch only the channels they pay for. Since there is a direct line from cable company to each individual subscriber's home, the cable company will only send those channels that are paid for. Many companies offer pay-per-view channels to their subscribers. Pay-per-view cable allows cable subscribers to "rent" a movie directly through the cable box. What the cable box does is decode the incoming movie, but not until the movie has been rented. If a person wants to watch a per-per-view movie, he calls the cable company and requests it. In return, the cable company sends out a signal to subscriber's cable box, which unscrambles (decrypts) the requested movie.

To achieve security there are two ways in which we can achieve.

- Encrypted file transfer.
- Strong secure protocol for transmission

2 Background & Related Work

There are two types of cryptographic algorithm to accomplish these goals: Symmetric and Asymmetric cryptography. The initial unencrypted dat is referred as normal text. RSA is (*Rivest, Shamir & Adleman*) is asymmetric cryptographic algorithm developed in 1977. It generated two keys: public key for encryption and private key to decrypt message. RSA encrypt and decrypt data, second phase is encryption, where actual process of conversion of plain text to cipher text is being carried out and third phase is decryption, where encrypted text is converted in to plain text at the other side.

Secure RSA prevents files from hackers and help safe transmission of files from one end to other [25]. The algorithm introduced in this report is a modification to the existing RSA algorithm. This algorithm eliminates the need to send product of two random prime numbers in the public key. Further, this algorithm replaces the role of n in encryption and decryption by an integer.

Cryptography is a process which is associated with scrambling plaintext into cipher text, then back again to plain text. The key feature of asymmetric cryptography system is encryption and decryption procedure are done with two different keys - public key and private key. Private Key can not be derived with hep of public key that provides much strength to security of cryptography.

This is one main difference between symmetric and asymmetric cryptography, but that difference makes whole process different. This difference is small but it is enough that it has implications throughout the security. Mainly, symmetric cryptography is seen as faster, more lightweight and better suited for applications that have a lot of data to transfer, while at the same time, it is known to be less secure and more open to wider areas of attacks because of maintenance for a private key required. This drawback is removed by asymmetric cryptographic algorithm discussed in following section.

2.1 Elliptic Curve Cryptosystem (ECC) [17]

Over the past 30 years, public key cryptography has become a mainstay for secure communications over the Internet and throughout many other forms of communications. It provides the foundation for both key management and digital signatures. In key management, public key cryptography is used to distribute the secret keys used in other cryptographic algorithms (e.g. DES). For digital signatures, public key cryptography is used to authenticate the origin of data and protect the integrity of that data. For the past 20 years, Internet communications have been secured by the first generation of public key cryptographic algorithms developed in the mid-1970's. Notably, they form the basis for key management and authentication for IP encryption (IKE/IPSEC), web traffic (SSL/TLS) and secure electronic mail.

The majority of public key systems in use today use 1024-bit parameters for RSA and

Diffie-Hellman. The US National Institute for Standards and Technology has recommended that these 1024-bit systems are sufficient for use until 2010. After that, NIST recommends that they be upgraded to something providing more security. The question is what should these systems be changed to? One option is to simply increase the public key parameter size to a level appropriate for another decade of use. Another option is to take advantage of the past 30 years of public key research and analysis and move from first generation public key algorithms and on to elliptic curves.

Elliptic Curve Cryptography provides greater security and more efficient performance than the first generation public key techniques (RSA and Diffie-Hellman) now in use. As vendors look to upgrade their systems they should seriously consider the elliptic curve alternative for the computational and bandwidth advantages they offer at comparable security. It consists of both encryption and signature algorithms.

2.2 ELGamal System

The ElGamal System [9] provides an alternative to rya for public key encryption.

1) Security of RSA depends on the presumed difficulty of factoring large integers.

2) Security of ElGamal algorithm depends on the difficulty of computing discrete logs in a large prime modulus.

The ElGamal signature algorithm is similar to encryption algorithm in that the public key and private key have the same form; however encryption is not the same as signature verification, nor is decryption the same as signature creation. Signature creation depends on the ElGamal signature algorithm. The main disadvantage of ElGamal is the need for randomness and it's slower speed. ElGamal has the disadvantage that the cipher text is twice as long as the plain text. ElGamal is not semantically secure.

2.3 DSS (Digital Signature Standard)

A digital signature is represented in a computer as a string of binary digits [18]. A digital signature is computed using a set of rules and a set of parameters such that the identity of the signatory and integrity of the data can be verified. An algorithm provides capability to generate and verify signatures. Signature generation [18] makes use of a private key to generate a digital signature. Signature verification [18] makes use of a public key which corresponds to, but is not the same as, the private key. Each user possesses a private and public key pair. Public key are assumed to be known to the public in general. Private keys are never shared. Anyone can verify the signature of a user by employing the user's public key. Signature generation can be performed only by the possessor of the private key.

The advantages of this system are

- The length of signature is shorter.
- The key generation is faster.
- The processing time code is less.

Drawbacks of DSS are

- DSS and RSA are not compatible.
- The verification process is slower than RSA

2.4 Diffie-Hellman key agreement protocol

Although Diffie-Hellman [11] key agreement itself is an anonymous (non-authenticated) key-agreement protocol, it provides the basis for a variety of authenticated protocols, and is used to provide perfect forward secrecy in Transport Layer Security's ephemeral modes (referred to as EDH or DHE depending on the cipher suite). In the original description papers, the Diffie-Hellman exchange by itself does not provide authentication of the communicating parties and is thus susceptible to a man-in- the-middle attack. An attacking person in the middle may establish two different Diffie-Hellman key exchanges, with the two members of the party "A" and "B", appearing as "A" to "B", and vice versa, allowing the attacker to decrypt [11]

(and read or store) then re-encrypt the messages passed between them. [11] A method to authenticate the communicating parties to each other is generally needed to prevent this type of attack. In the original description papers, the Diffie-Hellman exchange by itself does not provide authentication of the communicating parties and is thus susceptible to a man-in- the-middle attack. An attacking person in the middle may establish two different Diffie-Hellman key exchanges, with the two members of the party "A" and "B", appearing as "A" to "B", and vice versa, allowing the attacker to decrypt [11] (and read or store) then re-encrypt the messages passed between them. A method to authenticate the communicating parties to each other is generally needed to prevent this type of attack. Secure Sockets Layer (SSL)/Transport Layer Security (TLS), Diffie-Hellman protocol is used in Secure Shell (SSH), Internet Protocol Security (IPSec), Public Key Infrastructure (PKI).

2.5 The Sieve of Eratosthenes

Eratosthenes gave a method to generate all prime numbers between 1 and n. [24] The algorithm is as follows:

STEP 1: Write down all numbers from 2 to n.

STEP 2: Take the first uncrossed number, say P and cross all multiple of P, except P.

STEP 3: Repeat STEP 2 until no number crosses out.

Since the time complexity is very large for this algorithm, this method was not reasonable for this project.

The algorithm [19] is defined in Algorithm1

The complexity of the algorithm is $O(n(\log n)(\log\log n))$ [24] bit operations with a memory requirement of $O(n)$.Time complexity in RAM machine model is $O(n \log\log n)$ operations; this is a direct consequence of the fact that the prime harmonic series asymptotically approaches $\frac{1}{(ln(\ln(N)))}$. The segmented version of the sieve of Eratosthenes, with basic optimizations, uses $O(n)$ operations and $O(n^{\frac{1}{2}} \log\log n/\log n)$ bits of memory.

Algorithm 1 Sieve of Eratosthenes

Input: n
Output: list of prime numbers
  for $i := 2$ to n **do**
    $a\{i\} \leftarrow 1$
  end for
  $p \leftarrow 2$
  while $p^2 < n$ **do**
    $j \leftarrow p^2$
    while $j < n$ **do**
      $a\{j\} \leftarrow$
      $j \leftarrow j + p$
    end while
    repeat
      $p \leftarrow p + 1$
    until $a\{p\} \leftarrow 1$
  end while
  return a

2.6 Fermat's Little Theorem [20]

"*Let p be a prime which does not divide the integer a, then $a^{p-1} = 1 \mod p$*" [20]. The result is trivial (both sides are zero) if p divides a. If p does not divide a, then we need only multiply the congruence in Fermat's Little Theorem by a to complete the proof. This theorem satisfies only the subset of prime numbers since there are few composite numbers which also satisfy this property. Those numbers were discovered by Robert Carmichael. The smallest carmichael is 561 ($3 \times 11 \times 17$). In 1994 it was proved that there are infinite carmaichael numbers.

2.7 Gauss Theorem

Carl Friedrich Gauss considered the question of prime-counting function that gives the number of primes less than or equal to x, for any real number x which is as follows:

$$\frac{\pi(x)\log(x)}{x} \to 1; \text{ as } x \to \infty \qquad (1)$$

$$\pi(x) \sim \frac{x}{\log(x)} \tag{2}$$

Equation (1) equates to 1 as x approaches to infinity. For example:

$$\pi(100) = 25$$
$$\frac{100}{\log(100)} \approx 22$$

$$\pi(1000000000) = 50847534$$
$$\frac{1000000000}{\log(1000000000)} \approx 48254942$$

40 years later Gauss came up with even better approximation for $\pi(x)$ which is as follows:

$$\pi(x) \sim Li(x)Li(x) = \int_2^x \frac{1}{\log(t)}\partial(t) \tag{3}$$

for example:

$$Li(1000000000) = 50849234$$

Error difference is just 1700

x	$\pi(x)$	$\frac{x}{\log x}$	$Li(x)$
10^4	1,229	1085.7	1246.1
10^8	5,761,455	5.42×10^6	5.762×10^6
10^{12}	37,607,912,018	3.61×10^{10}	3.760×10^{10}
10^{16}	279,238,341,033,925	2.71×10^{14}	2.792×10^{14}
10^{18}	24,739,954,287,740,860	2.41×10^{16}	2.473×10^{16}

TABLE 1
Prime numbers

The prime counts in the TABLE 1 are taken from Edwards [8]

3 METHOD

3.1 Using 2 pairs of prime numbers

Secure RSA [7] is a asymmetric-key crypto system, meaning that for communication, two keys are required: a public key and a private key. furthermore, unlike RSA, it is one way, the public key is used only for encryption, and the private key is used only for decryption.

Secure RSA can be summarized as follows:

STEP 1: *Choose four large prime numbers p, q, r and s randomly and independently of each other. All primes should be of equivalent length."*

STEP 2: *Compute $n = p \times s, m = r \times s, \phi = (p-1)\times(q-1)$ and $\lambda = (r-1)\times(s-1)$*

STEP 3: *Choose an integer $e, 1 < e < \phi$, such that GCD$(e, \phi) = 1$*

STEP 4: *Compute the secret exponent $d, 1 < d < \phi$, such that $e \times d(\mod \phi) = 1$*

STEP 5: *Select an integer $g = m + 1$*

STEP 6: *Compute the modular multiplicative inverse: $\mu = \lambda^{-1}(\mod m)$*

The public key is (n, m, g, e) and private key is (d, λ, μ).

Let F be a file to be encrypted where the contents of file are taken into string S. Select random number r, where $r < m$ Compute cipher text as:

$$c = g^{s^e \mod n} \times r^m \mod m^2$$

Compute original message:

$$S = (\frac{c^\lambda \mod m^2 - 1}{m} \times \mu \mod m)^d \mod n$$

3.2 Using 3rd prime number

For STEP 3 of algorithm of Section 3.1, we found that there are only two possibilities:

1) First, e has to be prime.
2) Second, e should not be multiple of ϕ

This reduced the time complexity of generation of e since we need to consider only two conditions.

Also, STEP 4 of algorithm of Section 3.1 can be further enhanced by introducing one more random prime number x.

STEP 4 of Section 3.1 which is $e \times d \mod \phi = 1$, can be written as:

$$e \times d = 1 + (x \times \phi) \tag{4}$$

$$d = \frac{1 + (x \times \phi)}{e}$$

since $d < \phi$

$$\frac{1 + (x \times \phi)}{e} < \phi$$
$$(x \times \phi) < (e \times \phi) - 1$$
$$x < \frac{(e \times \phi) - 1}{\phi}$$

since ϕ is very large $\frac{1}{\phi}$ can be ignored and value of x will be:

$$x < e \tag{5}$$

STEP 4 of the algorithm can be rewritten as: *Compute the secret exponent* d

$$d = \frac{1 + (x \times \phi)}{e};\ \text{where } x < e \tag{6}$$

3.3 Karatsuba Multiplication

Secure RSA as defined in Section 3.1 needs large multiplicative computation for large prime numbers which will take $O(n^2)$ digit operation thus limiting the performance. To overcome this we needed faster algorithm for multiplication which we are going to explain in the this section.

Karatsuba's algorithm for fast multiplication was first published in "*Multiplication of Many-Digital Numbers by Automatic Computers*" [13], Proceedings of the USSR Academy of Sciences. The algorithm space for this algorithm is surprisingly rich. There are many methods of integer multiplication beyond what we learnt called conventional long multiplication and this is one of them.

3.3.1 Mathematical Derivation

To illustrate the algorithm, we let X and Y be two $2k$-bit unsigned integer and split them both in half

$$X = 2^k X_1 + X_0$$
$$Y = 2^k Y_1 + Y_0 \tag{7}$$

In conventional long multiplication, the product XY is computed with four k-bit multiplications and three additions as shown in Equation 8

$$XY = 2^{2k} z_2 + 2^k z_1 + z_0 \tag{8}$$

$$z_2 = X_1 Y_1, z_1 = X_0 Y_1 + X_1 Y_0, z_0 = X_0 Y_0 \tag{9}$$

As shown in Equation 9, Karatsuba [13] noticed that the middle term z_1 can be computed reusing the terms z_2 and z_0. Reusing these terms allow us to replace two of the multiplications and one addition with four additions/-subtractions and one multiplication

$$\begin{aligned} z_1 &= X_0 Y_1 + X_1 Y_0 \\ &= X_1 Y_1 + X_0 Y_0 - (X_1 - X_0)(Y_1 - Y_0) \\ XY &= T_1 - T_2 \\ T_1 &= 2^{2k} z_2 + z_0 + 2^k (z_2 + z_0) \\ T_2 &= 2^{2k} (X_1 - X_0)((Y_1 - Y_0) \end{aligned} \tag{10}$$

To elaborate, let's take an example of $x = 5678$ and $y = 1234$. Now we will split first number x into two halves as $a = 56$ and $b = 78$ similarly for y, $c = 12$ and $d = 34$

STEP 1: Compute $a \times c = 672$

STEP 2: Compute $b \times d = 2652$

STEP 3: Compute sum of $(a + b) \times (c + d) = 134 \times 46 = 6164$

STEP 4: Compute STEP 3 - STEP 2 - STEP 1 = 2840

STEP 5: Pad result of STEP1 with 4 zeros, pad 2 zeros for result from STEP 4

and add them along with result from STEP 2 and resultant is the required result which is 7006652.

Also, Xin and Ziaofei [21] analysed the critical techniques that may be combined in the design of fast hardware for RSA cryptography: Chinese remainders, star chains, Hensel's odd division (also known as Montgomery modular reduction), carry-save representation, quotient pipelining, and asynchronous carry completion adders.

Assuming that we replace two of the multiplications with only one makes the program faster. The question is how fast? Using Karatsuba, we improved the multiplication process by replacing the initial complexity of $O(n^2)$ by $O(n^{\log 3})$, which can be seen in the Figure 1 for the performance of Karatsuba Multiplication over conventional multiplication for big n.

3.3.2 Karatsuba Algorithm Implementation

The algorithm we followed to generate result for Karatsuba multiplication is explained in Algorithm 2

We measured the program and found the results of multiplication for pairs of random *n*-digit numbers which is in TABLE 2 and Figure 1. All times are in milliseconds and shows results we obtained.

digits	Karatsuba	Long Conventional Multiplication
8	0.059923	0.063902
16	0.006230	0.006141
32	0.016622	0.017437
64	0.048354	0.058411
128	0.140272	0.212314
256	0.437445	0.808407
512	1.317523	3.164557

TABLE 2
Karatsube VS Long Conventional Multiplication Time

3.4 Pseudorandom Number Generation (MT19937)

To increase the security of encryption we need pseudorandom number generator with large

Algorithm 2 Karatsuba Multiplication

Input: two large numbers n_1 and n_2
Output: product of n_1 and n_2
  if $n_1 < 10$ **or** $n_1 < 10$ **then**
    return $n_1 \times n_2$
  end if
  if $n_1 > n_2$ **then**
    $m = n_1$
  else
    $m = n_2$
  end if
  $m_2 = \frac{m}{2}$
  {split the digit sequences about the middle}
  $high_1, low_1 = split\_at(n_1, m_2)$
  $high_2, low_2 = split\_at(n_2, m_2)$
  $z_0 = karatsuba(low_1, low_2)$
  $z_1 = karatsuba((low_1 + high_1), (low_2 + high_2))$
  $z_0 = karatsuba(high_1, high_2)$
  return $(z_2 \times 10^{2\times m_2}) + ((z_1 - z_2 - z_0) \times 10^{m_2}) + (z_0)$

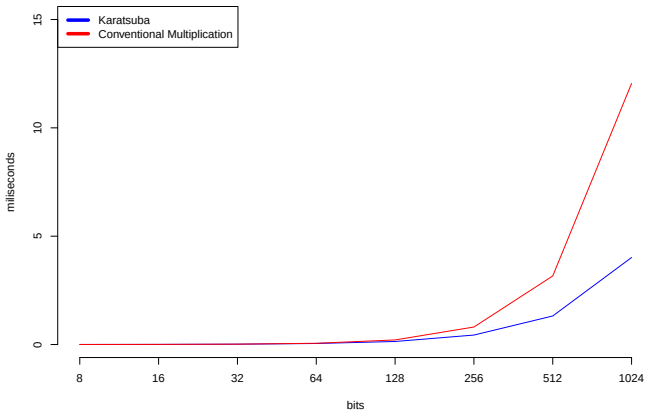

Fig. 1. Graph of results for Karatsuba and Grade School Multiplication

period. To achieve so Linear Congruential Generator (LCG) [15] can be used which is fast and require minimal memory (typically 32 or 64 bits) to retain state. This makes them valuable for simulating multiple independent streams but it is not recommended where high-quality randomness is critical which is highly prioritized in cryptography. If a linear congruential generator is seeded with a character and then iterated once, the result is simple classical cipher called an affine cipher [2]; this cipher is

easily broken by standard frequency analysis. Also, LCG tend to exhibit some severe defects. for instance, if a LCG is used to choose points in n-dimensional space, the points will lie on, at most $m^{\frac{1}{m}}$ hyperplanes [4]. For this research, we introduced Mersenne Twister (MT19937) pseudorandom number generator which we will explain in this section.

The Mersenne twister [16] is pseudorandom number generator. It is, by far, the most widely used Pseudorandom number generator. It's name derives from the fact that it's period length is chosen to be 24^{th} Mersenne Prime though the "guarantee" isn't there anymore. MT19937 is a variant of the twisted generalised feedback shift-register algorithm. It has passed the DIEHARD statistical tests. MT19937 uses 624 words of state per generator and is comparable in speed to the other generators. The original generator used a default seed of 4357 and "choosing **s** equal to zero". [16]

It is quite possible to have a random generator that produces the exactly same values for two different seeds. This isn't an issue unless we depend on some behavior of the randomness. The main point of course being cryptography is crypto graphical random number generators try very hard to be very random even if we run 10 generators in parallel. However, this might defeats the purpose of repeatability. The authors claim speeds 1.5 to 2 times faster than *Advanced Encryption Standard in counter mode* [22].

The MT19937 generates sequence of word vectors, which are considered uniform pseudorandom integers between 0 and $(2^w - 1)$ [16]. Dividing by $2^w - 1$, each word vector can be a real number in $[0, 1]$. With the restriction that $2^{nw-r} - 1$ is a Mersenne prime. For a word x with w bit width, it is expressed as the recurrence relation:

$$x_{k+n} = x_{k+m} \oplus (x_k^u | x_{k+1}^l)A; k = 0, 1, 2..... \quad (11)$$

In Equation 11 | represent as the bitwise OR and $\oplus$ represent as the bitwise exclusive or XOR, x_w, x_l being x with upper and lower bit masks applied. We choose a form of a matrix

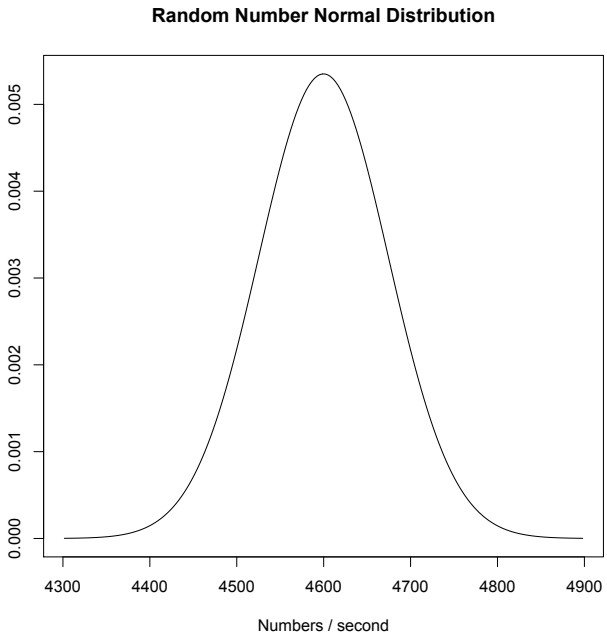

Fig. 2. Normal Distribution of Random Number Generator

A so that multiplication by A is very fast:

$$A = R = \begin{pmatrix} 0 & I_{w-1} \\ a_{w-1} & (a_{w-2}, ...a_0) \end{pmatrix} \quad (12)$$

With I_{n-1} as the $(n-1) \times (n-1)$ identity matrix (and in contrast to normal matrix multiplication, bitwise XOR replaces addition). The rational normal form has the benefit that it can be efficiently expressed as:

$$xA = \begin{cases} ShiftRight(x), & \text{if } x_0 = 0. \\ ShifrRight(x) \oplus a, & \text{if } x_0 = 1. \end{cases} \quad (13)$$

Where

$$x = (x_k^u | x_{k+1}^l); k = 0, 1, 2, ... \quad (14)$$

and

$$a = (a_{w-1}, a_{w-2}, ..., a_0)$$

To improve k-distribution to v-bit accuracy, we multiply each generated word by a suitable $w \times w$ invertible matrix T from the right. For the tempering $r \to z = xT$, we chose the following

successive transformations:

$$y = x \oplus (x >> u) \tag{15}$$
$$y = x \oplus (x << s) \text{ AND } b \tag{16}$$
$$y = x \oplus (y << t) \text{ AND } c \tag{17}$$
$$y = x \oplus (x >> l) \tag{18}$$

With $<<$ and $>>$ as the bitwise left and right shifts and $\&$ as the bitwise AND. The first and last transforms are added in order to improve lower bit equidistribution. From the property of TGFSR, $s+t \geq \lfloor \frac{w}{2} \rfloor - 1$ is required to reach the upper bound of equidistribution for the upper bits.
According to MT19937 [16] The coefficients are:

$$\begin{aligned}(w, n, m, r) &= (32, 624, 397, 31)\\ a &= 9908B0DF_{16}\\ u &= 11\\ (s, b) &= (7, 9D2C5680_{16})\\ (t, c) &= (15, EFC60000_{16})\\ l &= 18\end{aligned} \tag{19}$$

The algorithm for Equation 11 through Equation 19 can be summarized in Algorithm 3 which is derived from "*Mersenne Twister: A 623-dimensionally equidistributed uniform pseudorndom number generator*" [16] The values for parameters $w, n, m, r, a, u, s, b, t, c$ and l are taken from *Table II Parameters and k-distribution of Mersenne Twisters* [16]. .

Also, the authors of MT19937 [16] mentioned for Equation 19 that "*If $r = 0$, then this recurrence reduces to the previous TGFSR*" which was the reason we included this version of algorithm in our work and change the value of r to user's need.

Mersenne Twister implementations cannot be parallelized across parallel computing cores simply through changing the initial seed for each core as this does not provide uncorrelated sequences on each generator sharing identical parameters. To solve this problem and enable Mersenne twister parallel implementations, the authors of MT19937 [16] developed a library for the dynamic creation of Mersenne Twister parameters. This library receives user's specification such as word length, period, size of working area and a process ID, so that ID number is encoded in the characteristic polynomial of Mersenne Twister.

Algorithm 3 Mersenne-Twister

Step0:
$u \leftarrow \underbrace{1...1}_{w-r}\underbrace{0...0}_{r}$; (bitmask of upper w-r)
$ll \leftarrow \underbrace{1...1}_{w-r}\underbrace{0...0}_{r}$; (bitmask of lower r bits)
$a \leftarrow a_{w-1}a_{w-2}a_{w-3}...a_2a_1$
Step1:
$i \leftarrow 0$
$x[0], x[1], x[2].....x[n-1] \leftarrow$ "any non-zero initial values"
Step2:
$y \leftarrow (x[i] \text{ AND } u) \text{ OR } (x[(i+1) \bmod n] \text{ AND } ll)$
Step3: $x[i] \leftarrow x[(i+m) \bmod n] \text{ XOR } (y >> 1)$

$$\text{XOR} \begin{cases} 0, & \text{if least significant bit of y=0.} \\ a, & \text{if least significant bit of y = 1.} \end{cases}$$

Step4: calculate $x[i]T$

$$\begin{aligned} y &\leftarrow x[i]\\ y &\leftarrow y \text{ XOR } (y >> u)\\ y &\leftarrow y \text{ XOR } ((y << s) \text{ AND } b)\\ y &\leftarrow y \text{ XOR } ((y << t) \text{ AND } c)\\ y &\leftarrow y \text{ XOR } (y >> l) \end{aligned}$$

Step5:

$$i \leftarrow (i+1) \bmod n$$

Step6: Goto **Step2**

Figure 3 shows the Cummulative Density Function (which is defined by $F(x) = P(X \leq x)$) graph for the random number generator with mean of 4600 and standard deviation of almost 74.

MT19937 we worked with showed the following statistical property which makes it preferable over other pseudorandom number generators. [3]

- **Probability Density Function:** $0.00535134e^{-0.0000899652(x-4599.84)^2}$

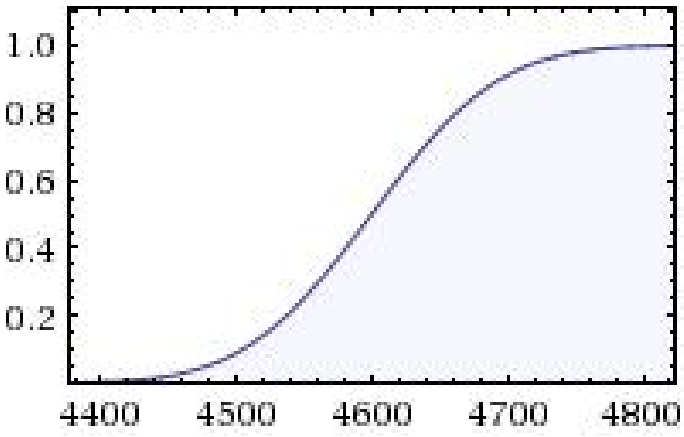

Fig. 3. Cumulative Density Function

- **Cumulative Distributive Function (CDF):** $\frac{1}{2}erfc(0.009485(4599.84 - x))$

For faster primality test for random number generated by MT19937 we used Miller-Rabin Primality test which is by far fastest performing algorithm. We are going to explain the implementation in next section.

3.5 Miller-Rabin Primality Test [14]

We know of two ways to prove that a number n is composite:

1) Exhibit a factorization $n = ab$, where $a, b > 1$
2) Exhibit a Fermat [20] witness for n, *i.e* a number x satisfying $x^{n-1} \not\equiv 1(\mod n)$

The Miller-Rabin test is based on a third way to prove that a number is composite

3) Exhibit a "fake square root of $1 \mod n$" *i.e* a number x satisfying $x^2 \equiv 1(\mod n)$ but $x \not\equiv \pm 1(\mod n)$

If x, n are positive integers such that $x^2 \equiv 1(\mod n)$ but not $x \not\equiv \pm 1(\mod n)$ then n is composite. Generally this can be proved that for any nonzero polynomial

$$P(x) = a_0 + a_1 x + ... + a_k x^k$$

The numder of $x \in 1, 2, 3, ..., p-1$ satisfying $P(x) \equiv 0(\mod p)$ is at most k. The proof is by induction on k, the base case $k = 0$ being trivial. Otherwise, suppose a satisfies $P(a) \equiv 0(\mod p)$. We may write $P(x) = (x-a)Q(x) + c$ where $Q(x)$ is a polynomial of degree $k-1$ with integer coefficients. The congruence $P(a) \equiv 0(\mod p)$ implies that c is divisible by p. If b satisfies $P(b) \equiv 0(modp)$ but $Q(b) \not\equiv 0(\mod p)$ then p is a divisio of $(b-a)Q(b)$ but not $Q(b)$, hence $b \equiv a(modp)$. It follows that every $b \in 1, 2, 3, ..., p-1$ satisfying $P(b) \equiv 0(\mod p)$ satisfies either $b = a$ or $Q(b) \equiv 0(\mod p)$. By the induction hypothesis, at most $k-1$ elements of $1, 2, ..., p-1$ satisfy the second congruence [14]

The idea of test is to pick a random x in $1, 2, 3, ..., n-1$ and use it to try finding either a Fermat witness [20] or a fake square root of $1 \mod n$. Why Miller-Rabin test works? We have seen that when n is prime, the test always outputs "probably prime". When n is composite but is not a *Carmichael* number, we have seen that it outputs "composite" with probability at least $\frac{1}{2}$. The working on Miller-Rabin test in this case best be understood in terms of *Chinese Remainder Theorem* [10] which is;

"Let $n_1, n_2, ..., n_k$ be numbers, no two of which have a common factor. For any numbers $a_1, a_2, a_3, ..., a_k$, the system of congruences

$$\begin{aligned} x &\equiv a_1(\mod n_1) \\ x &\equiv a_2(\mod n_2) \\ &\cdot \\ &\cdot \\ x &\equiv a_k(\mod n_k) \end{aligned}$$

has a solution. Any two solutions x_1, x_2 are congruent $\mod n_1 n_2 ... n_k$". If n is a *Carmichael* number, then Miller-Rabin outputs "composite" with probability at-least $\frac{3}{4}$.

The Miller-Rabin can be summarized that the number to be tested say p and y be a non trivial square root of $1(\mod p)$. [14] Then we must have that $y^2 = 1(\mod p)$ and so $(y-1)(y+1) = 0(\mod p)$. This implies that either $y = 1 \mod p$ or $y = -1 \mod p$, which implies that y is a trivial square root.

Thus, if there is a non trivial square root of $1 \mod p$, then p has to composite. For an example

Algorithm 4 Miller-Rabin Primality Algorithm

Input: $n > 2$, an odd integer to be tested for primality; k, a parameter that determines the accuracy of the test

Output: composite if n is composite, otherwise probably prime.

```
write n − 1 as 2^s d with d odd by factoring powers of 2 from n − 1
repeat
  x ← a^d mod n
  if x = 1 or x = n − 1 then
    continue
  end if
  for r = 1tos − 1 do
    x ← x^2 mod n
    if x = 1 then
      return composite
    end if
    if x = n − 1 then
      do next loop
    end if
  end for
  return composite
until k times
return probably prime
```

of a non trivial square root of a composite, consider $p = 15$. We have that $4^2 = 16 = 1(\mod 15)$, thus 15 is composite. The fact about non trivial square roots can be used to prove that if p is prime, then for any a relatively prime to p, some power of a from a given set of powers must be -1 or a specific odd power of a must be 1.

If for some a none of the above set of powers is -1 and the specific odd power is not 1, then it must be the fact that p is composite. It can also be shown that for composite p, the chances of finding such a is at least $\frac{3}{4}$. This a is the witness in the primality test and is not necessarily a non-trivial square root of $1 \mod p$. The algorithm is defined in Algorithm 4

4 RESULTS

4.1 Using 2 pairs of prime number

We found that that time consuming step in the algorithm as mentioned in Section 3.1 is STEP 4. Since ϕ is very large which is why it makes it time consuming to compute the value of d. We ran independent test to find the average time (in milliseconds) taken to encrypt and decrypt using d and found the following results:

bits	milliseconds
33	78.310144
96	320.994816
160	963.48288
220	10155.14112
273	33831.1593

TABLE 3
Time taken to encrypt and decrypt

4.2 Using 3rd prime number

Performance wise introduction of new method of finding the value of d reduced computation time as compared to the previous approach. We ran tests for encryption and decryption using new method and it reduced time consumption. The values are mentioned in Table 4:

bits	milliseconds
33	27.386368
96	33.817088
160	160.193792
220	2901.262848
273	4219.404032

TABLE 4
Time taken to encrypt and decrypt with new method

4.3 Analysis & Discussion

By comparing the result of old method of encryption and the method we developed we are in state to advocate that our method is working better than old method. The algorithm have many important parameters affecting it's level of security and speed. By increasing the modulus length it is caused of increasing the complexity of decomposing it into it's factors.

This also increases the length of private key and hence difficult to detect the key. Another parameter is modular multiplicative inverse μ, which is new factor of private key, so it will be more difficult to choose μ by trying all possible private keys (brute force attack) hence the security also increases as well as difficulty of detecting the private key.

4.4 Pseudorandom Number Generation (MT19937)

Our method for doing simple benchmarking and Mersenne Twister implementation generates numbers at a rate steadily over 4600 pseudo-random numbers per second which is very large. The test code reports normal distribution formulated from mean and standard deviation of performance of MT19937 is shown in Figure 4. [3]. Also, comparing our results with results mentioned in "Testing Random Number Generators" [5] we found that as the number range increase the probability density increases exponentially (Figure 3) as compared to the results from paper [5] which is linear and then remains constant. From this we can imply that as the key space of number increases probability density increases.

Also, we tested to generate large prime number and our system showed significant performance to generate 34^{th} mersenne prime with negligible load on the system. Fig. 5 shows the performance graph of the test conducted to compute time (in nano seconds) to generate large primes. To find the normal distribution for prime numbers generated using MT19937 we generated 31 sets of 100, 200, 300, 400, 500, 600 and 700 numbers. We found that on average our system is generating somewhat 216 random prime numbers per second with the standard of almost 14.

4.5 Large Multiplication (Karatsuba Multiplication)

Karatsuba Multiplication [6] algorithm mentioned in section 3.3 was implemented to enhance the performance of multiplication also

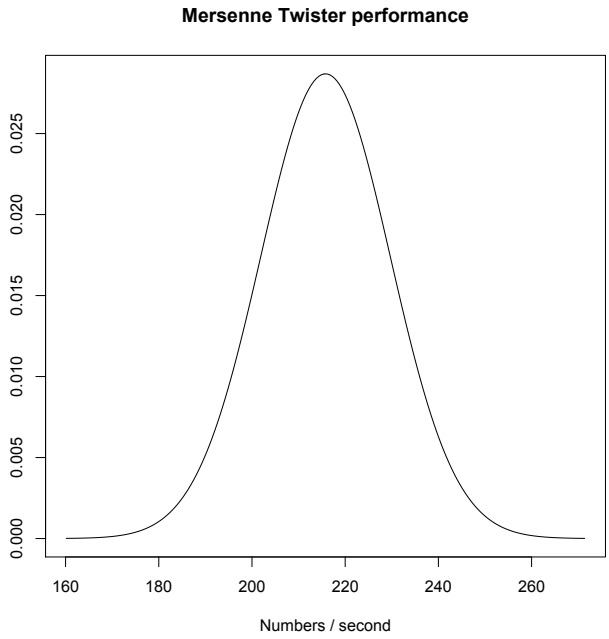

Fig. 4. Mersenne Twister 19937 Performance

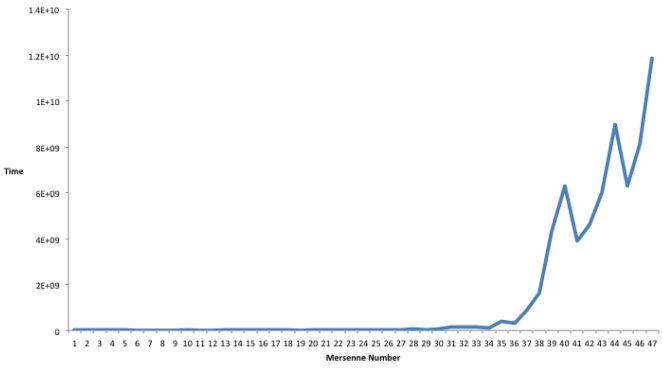

Fig. 5. Time VS Mersenne Prime Number Generation

had significant performance as shown in Figure 1. The multipliers have significantly lower area-delay products compared with previous designs [6]. For key generation we worked to generate keys of size upto 512-bits to 1000-bits long keys for significant performance based on time of computation.

To further boost the multiplicative performance we used third party library for mathematical computation called GMPY2 [1] which

is a Python module that provides access to the GNU Multiple Precision (GMP) library. GMP provides very fast and highly optimized routines for working with arbitrary precision integers, rationals, and floating point numbers.

4.6 Comparision

bits	Secure RSA	Our result
33	78.310144	27.386368
96	320.994816	33.817088
160	963.48288	160.193792
220	10155.14112	2901.262848
273	33831.1593	4219.404032

TABLE 5
Time comparison of Secure RSA and Our Result

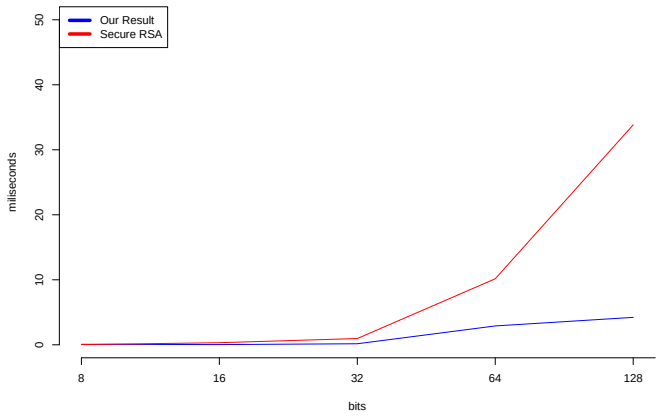

Fig. 6. Time comparison of performance

To test with new value of d for Secure RSA, we collected average time taken for 10 sets of 33, 96, 160, 220, 273 *bits* of data for encryption and decryption and found that the change had significant affect on the performance. The values of comparison are provided in TABLE 5 and performance graph is Figure 6.

Though, introduction to one more random prime number slowed the performance but using Karatsuba Multiplication as mentioned in Algorithm 2 over came the limitation. To prove this hypothesis we ran test of generation of 60 encryptions and decryptions (10 sets of file encryption and decryption with 32, 65, 94, 127, 160, 320 *bits* size random prime numbers) and found the performance to be significant. The result is shown in Figure 7. We performed Normal Distribution Function and found that on average probability of encryption and decryption of 500 files is very high as shown in Figure 8.

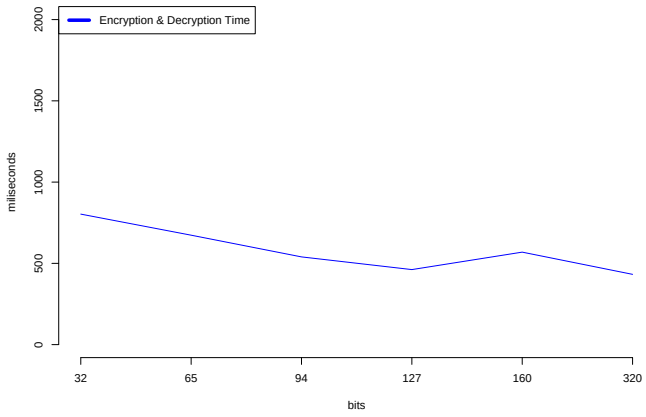

Fig. 7. Encryption & Decryption of file

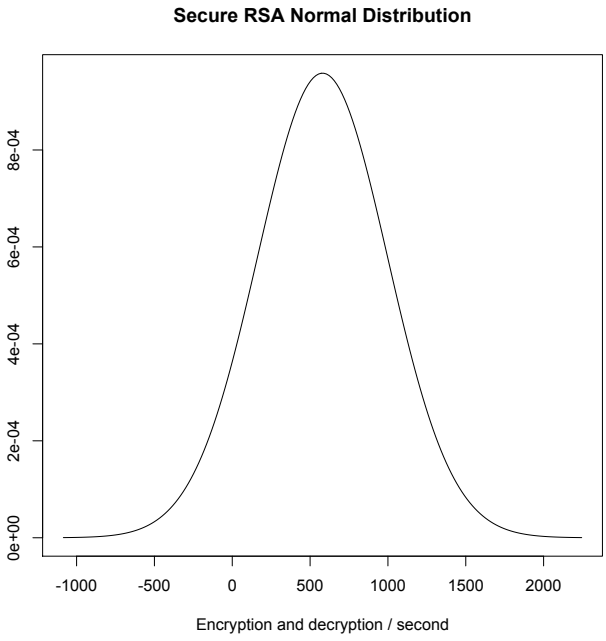

Fig. 8. Normal distribution of Encryption & Decryption of file

5 Conclusion of using 2 pair of prime number VS introducing 3rd prime number

Comparing this results with the "*File Encryption and Decryption using Secure RSA*" [12], we found that time taken for 32 bits-number reduced exponentially as we increase key space. on average our system performed 15 times more than old method.

The digital world of cryptography is getting bigger and bigger, not smaller and our method have showed a new approach to make secure by increasing the bit size of the key generated with not compromising with good computational performance up to 512-bit long keys and comparatively significant performance up to 1900-bit long key for encryption and decryption. Our method indicated we can reduce the time complexity of computation to generate private key using newly introduced method.

Secure RSA algorithm is used to encrypt files and transmit encrypted files to other end where it is decrypted. Our system works on the large numbers. It has broad development prospects. The system application was designed to take the efficiency and reusability into account. Great level of security is achieved using this algorithm. Secure RSA algorithm for file transmission algorithm can be used where high security file transmission required in public forums.

With the increasing computing power available to even casual users, the security conscious have had to move on to increasingly robust encryption, lest they find their information vulnerable to brute-force attacks. The latest milestone to fall is 768-bit RSA. In "*Factorization of a 768-bit RSA modulus*" [23] announced that they factored one of these keys.

Most modern cryptography relies on single large number that are the product of two primes. By knowing the numbers, it is relatively easy to encrypt and decrypt data, otherwise finding by brute force is big computational challenge. "*There is nothing new to be reported for the square root step, except for the resulting factorization of RSA-768. Nevertheless, and for the record, we present some of the details*" [23] they wrote.

As future work multiple file encryption and decryption can be possible. It has broad development prospects.

References

[1] Gmpy2.0 [gmpy2.readthedocs.org], May 2014.

[2] G. Aithal, K. Bhat, and U. Sripathi. Implementation of stream cipher system based on representation of integers in residue number system. In *Advance Computing Conference (IACC), 2010 IEEE 2nd International*, pages 210–217, Feb 2010.

[3] W. Alpha. Statistical performance of mersenne twister mt19937.

[4] E. Bach. Efficient prediction of marsaglia-zaman random number generators. *IEEE Transactions on Information Theory*, 44(3):1253–1257, 1998.

[5] D. Biebighauser. Testing random number generators, 2000.

[6] G. CESARI and R. MAEDER. Performance analysis of the parallel karatsuba multiplication algorithm for distributed memory architectures. *Journal of Symbolic Computation*, 21(4–6):467 – 473, 1996.

[7] R. Dhakar, A. Gupta, and P. Sharma. Modified rsa encryption algorithm (mrea). In *Advanced Computing Communication Technologies (ACCT), 2012 Second International Conference on*, pages 426–429, Jan 2012.

[8] H. M. Edwards. *Riemann's zeta function*. Dover Ed, June 2001.

[9] K. Huang and R. Tso. A commutative encryption scheme based on elgamal encryption. In *Information Security and Intelligence Control (ISIC), 2012 International Conference on*, pages 156–159, Aug 2012.

[10] S. Iftene. General secret sharing based on the chinese remainder theorem with applications in e-voting. *Electron. Notes Theor. Comput. Sci.*, 186:67–84, July 2007.

[11] E. R. R. Inc. Diffie-hellman key agreement method. *The Internet Society*, 1999.

[12] R. S. Jamgekar and G. S. Joshi. File encryption and decryption using secure rsa. In *International Journal of Emergine Science and Engineering*, pages 426–429, February 2013.

[13] A. Karatsuba and Y. Ofman. Multiplication of many-digital numbers by automatic computers. *Proceedings of USSR Academy of Sciences*, 145(7):293–294, 1962.

[14] B. Kleinberg. Introduction to algorithms (cs 482). *The Miller-Rabin Randomized Primality Test*, 05 2010.

[15] C.-C. Li and B. Sun. Using linear congruential generators for cryptographic purposes. In G. Hu, editor, *Computers and Their Applications*, pages 13–19. ISCA, 2005.

[16] M. Matsumoto and T. Nishimura. Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Trans. Model. Comput. Simul.*, 8(1):3–30, Jan. 1998.

[17] J. W. B. J. H. N. H. J. M. M. Naehrig. Elliptic curve cryptography in practice.

[18] U. D. of Commerce. Digital signature standard. *Federal Information Processing Standard Publications*, pages 186–2, January 2000.

[19] U. of Tennessee. Sieve of eratosthenes.

[20] J. Robinson. Elaboration on history of fermat's theorem and implications of euler's generalization by means of totien ttheorem. Master's thesis, University of Arizona, 2011.

[21] M. Shand and J. Vuillemin. Fast implementations of rsa cryptography. In *Computer Arithmetic, 1993. Proceedings., 11th Symposium on*, pages 252–259, Jun 1993.

[22] R. Tirtea and G. Deconinck. Specifications overview for counter mode of operation. security aspects in case of faults. In *Electrotechnical Conference, 2004. MELECON 2004. Proceedings of the 12th IEEE Mediterranean*, volume 2, pages 769–773 Vol.2, May 2004.

[23] T. K. K. A. J. F. A. K. L. E. T. J. W. B. P. G. A. K. P. L. M. D. A. O. H. to Riele; Andrey Timofeev and P. Zimmermann, editors. *Factorization of a 768-bit RSA modulus*, volume 1.4, February 2010.

[24] P. Univeristy. Sieve of eratosthenes.

[25] X. Zhou and X. Tang. Research and implementation of rsa algorithm for encryption and decryption. In *Strategic Technology (IFOST), 2011 6th International Forum on*, volume 2, pages 1118–1121, Aug 2011.