Chapter 2

Literature Review

2.1 RSA Encryption Algorithm

RSA encryption algorithm was invented by [RSA1978] extending the concept of "public key cryptosystem" proposed by Diffie and Hellman [DH1976]. RSA encryption algorithm is a simple algorithm which can be understood easily, implement and analyze with minimum knowledge of number theory.

RSA cryptosystem is an asymmetry cryptosystem, which involve a pair of keys, namely public key and private key, to perform encryption and decryption. To generate a key pair, one must first find two large random prime numbers $p$ and $q$ of approximately equal size. Hence, the modulus for the key, $N$, is defined as product of $p$ and $q$.

$$N = pq$$  \hspace{2cm} (2.1)

$N$ is made public, anyone may know the $N$ value. But since it is hard to recover the prime factors from $N$, therefore these two factors is effectively hidden from everyone.

An encryption exponent, $e$, which is relatively prime to $\varphi(N) = (p-1)(q-1)$ is calculated. In other words, the greatest common divisor, g.c.d($e, \varphi(N)$) is equal to 1. Finally, to calculate the private key/decryption exponent, $d$, is then defined as

$$d = e^{-1} \mod(\varphi(N))$$  \hspace{2cm} (2.2)
The encryption key is thus the pair of positive integer \((e, N)\). Similarly the decryption key is \((d, N)\).

Let assume \(M\) represent a plain text message with value in the range of 0 and \(N-1\). Any message can be represented in numeric form. To perform encryption on plain text message block, \(M\), raise \(M\) to \(e\)-th power modulo \(N\) to get the cipher, \(c\), as shown below:

\[
c = M^e \mod N \tag{2.3}
\]

and the decryption on \(c\) can be performed by

\[
M \equiv c^d \mod N \tag{2.4}
\]

**Example 2.1:** By using two prime numbers, \(p=23\) and \(q=11\), generates a pair of keys and perform encryption and decryption.

Get the modulus, \(N\),

\[
N=23 \times 11 = 253
\]

Then calculate,

\[
\varphi(N) = (23-1)(11-1) = 220
\]

and, public exponent \(e=3\) where g.c.d\((3, 220)\) is equal to 1. Finally, the private exponent, \(d\), where \(d=147\).

To encrypt a message, \(M\), with a value \(M=5\),

Cipher, \(c = 5^3 \mod 253\)

\[
c = 125
\]

Cipher \(c\) will be decrypted to \(M\) as below,

\[
M \equiv 125^{147} \mod 253
\]

\[
M = 5
\]
2.1.1 Choice of $p$ and $q$

In order to make the factorization of RSA modulus infeasible, its prime factors $p$ and $q$ must be chosen appropriately. Given the strength of current factoring algorithms, both $p$ and $q$ should be almost equal in length and at least in binary length of 512.

2.1.2 Choice of $e$ and $d$

The public key $e$ is chosen to be as small as possible to make the encryption efficient. The least possible encryption exponent is $e=3$. However, using small encryption exponent such as $e=3$ is vulnerable to low-exponent attack [JAB1981].

2.1.3 Advantages

- The primary advantage of public-key cryptography is increased security and convenience: private keys never need to be transmitted or revealed to anyone. In a secret-key system, by contrast, the secret keys must be transmitted (either manually or through a communication channel) since the same key is used for encryption and decryption. A serious concern is that there may be a chance that an enemy can discover the secret key during transmission.

- Public-key systems can provide digital signatures that cannot be repudiated. Authentication via secret-key systems requires the sharing of some secret and sometimes requires trust of a third party as well. As a result, a sender can repudiate a previously authenticated message by claiming the shared secret was somehow compromised by one of the parties sharing the secret. For example, the Kerberos secret-key authentication system involves a central database that keeps copies of the secret keys of all users; an attack on the database would allow
widespread forgery. Public-key authentication, on the other hand, prevents this type of repudiation; each user has sole responsibility for protecting his or her private key. This property of public-key authentication is often called non-repudiation.

- Public-key cryptography is not meant to replace secret-key (symmetry key) cryptography, but rather to supplement it, to make it more secure. The first use of public-key techniques was for secure key establishment in a secret-key system; this is still one of its primary functions. Secret-key cryptography remains extremely important and is the subject of much ongoing study and research.

2.1.4 Disadvantages

- One of the disadvantages of using public-key cryptography for encryption is the speed. There are many secret-key encryption methods that are significantly faster than any currently available public-key encryption method. Nevertheless, public-key cryptography can be used with secret-key cryptography to get the best of both worlds. For encryption, the best solution is to combine public and secret-key systems in order to get both the security advantages of public-key systems and the speed advantages of secret-key systems. Such a protocol is called a digital envelope.

- Public-key cryptography may be vulnerable to impersonation, even if users' private keys are not available. A successful attack on a certification authority will allow an adversary to impersonate whomever he or she chooses by using a
public-key certificate from the compromised authority to bind a key of the adversary's choice to the name of another user.

- Public-key cryptography is usually not necessary in a single-user environment. For example, if you want to keep your personal files encrypted, you can do so with any secret key encryption algorithm using, say, your personal password as the secret key. In general, public-key cryptography is best suited for an open multi-user environment.

2.2 Fundamental Theorem of Arithmetic

Every positive integer \( N \) can be written as a product of primes, this factorization is unique except the order of the prime factors.

\[
N = p_1^{n_1} p_2^{n_2} \ldots p_s^{n_s} = \prod_{i=1}^{s} p_i^{n_i}
\]  

(2.5)

Table 2.1 shows some samples of prime factorization.

<table>
<thead>
<tr>
<th>Table 2.1 Prime factors for positive integer</th>
</tr>
</thead>
<tbody>
<tr>
<td>3000=2^3 x 3 x 5^3</td>
</tr>
<tr>
<td>3001=prime</td>
</tr>
<tr>
<td>3002=2 x 19 x 79</td>
</tr>
<tr>
<td>3003=3 x 7 x 11 x 13</td>
</tr>
<tr>
<td>3004=2^2 x 751</td>
</tr>
</tbody>
</table>

An integer \( N>1 \) is said to be a prime number if the only divisor is \( \pm 1 \) and \( \pm N \). If \( N>1 \) and not a prime number, then \( N \) is said to be composite.
2.3 Factoring Algorithm

The strength of RSA [RSA1978] cryptosystem is related to the difficulty of factoring a positive integer into primes. But there is no proof to conclude whether the integer factoring problem is in fact difficult to solve. For the last 20 years, many efficient factoring algorithms have been invented and this had threaten the RSA cryptosystem to increase its modulus from 512-bit to 1024-bit for better security as 512-bit key is recently been broken.

There are several efficient factoring algorithms that can be used to decompose an integer $N$. Each has its advantages and disadvantages. Factoring algorithms fall into two categories, special purpose and general purpose; the efficiency of the former depends on the unknown factors, whereas the efficiency of the latter depends on the number to be factored. Special-purpose algorithms are best for factoring numbers with small factors, but the numbers used for the modulus in the RSA cryptosystem do not have any small factors. Therefore, general-purpose factoring algorithms are the important ones in the context of cryptographic systems and their security.

A "general number" is one with no special form that might make it easier to factor; moduli used in the RSA cryptosystem are created to be general numbers. A "special number" means generally that there is an easy way of expressing it. For example, the number might be a Fermat number, which means that it is equal to $2^{2n} + 1$ for some integer $n$.

Here are several useful algorithms which can be categorized into two main classes mentioned above:
a. **Running time depends mainly on the size of $N$ (General Purpose)**

- Continued Fraction algorithm [RIE1985] and the widely used Quadratic Sieve algorithm (QS) [SIL1987], which under a plausible assumption of running time $O(\exp(\sqrt{c \ln N \ln \ln N}))$, where $c$ is a constant.

- The Number Field Sieve algorithm (NFS) [CLAM2001 and POM1982] has expected run time of $O(\exp(c(\sqrt{\ln N \ln \ln N})^2))$.

b. **Running time depends on the size of $f$, the factor found. (Special Purpose)**

- Trial division algorithm, which has discouraging run time of $O(f \times (\log N)^2)$.

- Pollard $\rho$-algorithm [RIE1985], which under plausible assumptions has expected run time $O(f^{\frac{1}{2}} \times (\log N)^2)$.

- Lenstra’s Elliptic Curve (ECM) algorithm, run time of $O(\exp(\sqrt{c \ln f \ln \ln f}) \times (\log N)^2)$.

### 2.3.1 Trial Division

Trial division is a straightforward factorization algorithm. If $N$ is a composite number then one of the prime factors is at most $\sqrt{N}$. To factor $N$, trial division successively divide primes 2, 3, 5, .... up to $\lceil \sqrt{N} \rceil$. Trial division is typically the fastest for factoring integer below $10^9$. The parallel implementation is also straightforward. You may gain linear speedup with $n$ processor to carry out $n$ trial division in parallel.
2.3.2 Pollard - $\rho$ algorithm

Pollard - $\rho$ [RIE1985 and JAB1981] is excellent in finding prime factors that are too big for trial division. It is efficient for composite number within the range of $10^{10}-10^{13}$. The idea of this method is described below.

Pick two random numbers $a$ and $b$ in the set $\{1, \ldots, N-1\}$ and a congruence where $a \equiv b \mod p$ can be found, where $p$ is the factor of $N$. If $a \equiv b \mod p$, then it means $(a-b)$ is multiple of $d$ and this also true that $N$ is also multiple of $d$, $(a - b)=kd$ and $N=k'd$.

From this establishment, one of the prime factor can be retrieved from g.c.d $((a-b),N) > 1$.

To pick a random number, Pollard-$\rho$ method suggests that a sequence of numbers from an equation $x_{k+1} = (x_k^2 + a)$ or $x_{k+1} = f(x_k)$ need to be built. Where $a$ is a constant and not congruent to 0 or -2 modulo $N$. The $x$ will be picked from the range of 0 up to (but not include) $N$. Therefore, $x$ can be built from $(x_k^2 + a) \mod N$, e.g.: start with some $x_1$ then subsequent $x$ will be gathered from $x_{k+1} = (x_k^2 + a) \mod N$.

Let assume if at some point, $k$, where $x_k \equiv x_j \mod p$ with $k < j$ or $x_k \neq x_j$. Then, because of the way that modulo arithmetic works, $f(x_k)$ will be congruent to $f(x_j)$ modulo $p$. So, once reach upon $x_k$ and $x_j$, each element in the sequence starting with $x_k$ will be congruent $\mod p$ to the corresponding element in the sequence starting at $x_j$.

Thus, once the sequence gets to $x_k$ it has looped back upon itself to match up with $x_j$ (when considering them modulo $p$).
This looping is what gives the “Rho” method its name. Because the method only considers a number modulo $p$, this makes a finite set of numbers. The loop back characteristic can be depicted as following.

![Figure 2.1 Pollard-$\rho$](image)

2.3.3 Quadratic Sieve Algorithm

Before discussing about the Quadratic Sieve algorithm (hereinafter refer as QS), knowing the fundamental idea of general factoring algorithm is essential which is the predecessor of QS introduced by Carl Pomerance [POM1996]. This section will start by the introduction of Fermat’s difference-of-square technique which is the basic of QS algorithm. Subsequent subsections will further elaborate essential components and methods of QS algorithm that make it the efficient algorithm to factor large integer.

2.3.3.1 Fermat’s difference-of-square Technique

To factor a composite number 8051 we can write this number as $8100 - 49$, which is $90^2 - 7^2$. Here we use $a$ and $b$ to represent the factors,

\[
8051 = ab = u^2 - v^2 \quad (2.6)
\]

\[
= 90^2 - 7^2
\]

\[
= (90+7)(90-7)
\]
Therefore the factors for 8051 are 97 and 83. Every odd composite can be factored as a difference of squares by using the identity below,
\[
ab = \left(\frac{1}{2} (a+b)\right)^2 - \left(\frac{1}{2} (a - b)\right)^2
\]
(2.7)

2.3.3.2 Enhancement of difference-of-square Technique

Maurice Kraitchik [POM1996] came up with an enhancement of Fermat’s difference-of-square, which had been used until today as the basic of most modern factoring algorithm.

Instead of finding integer \(u\) and \(v\) with \(u^2 - v^2\) equal to \(N\) (number to be factored), it is reasoned that \(u^2 - v^2\) might equal to a multiple of \(N\) (where \(u^2 - v^2 = kN\), that is \(u^2 \equiv v^2 \mod N\). This also means \(u^2 - v^2\) is divisible by \(N\), where \(N|(u^2 - v^2)\) or \(N|(u+v)(u-v)\).

An interesting solution is \(u \equiv \pm v \mod N\). If \(u \equiv \pm v \mod N\), then \(N\) will not have any factors. From the interesting solution of \(u \neq \pm v \mod N\), the non-trivial factor of \(N\) can be found by calculating the greatest common divisor (g.c.d) of \(u - v\) and \(N\), which denoted as \(g.c.d(u-v,N)\) in order to find one of the factor.
\[
a = g.c.d(u-v,N)
\]
\[
b = g.c.d(u+v,N)
\]

Let see an example below for finding \(u\) and \(v\). It shows how a problem can be broken into small group of problems.
Example 2.2: To factor $N=2041$.

From,

\[ u^2 - v^2 = N \]
\[ v^2 = u^2 - N \]

$v^2$ can be represented as $Q(x)$,

\[ Q(x) = x^2 - N \] \hspace{1cm} (2.8)

or in modular arithmetic form,

\[ x^2 = Q(x) \mod N \] \hspace{1cm} (2.9)

The first square above $N$ is $46^2=2116$. So, for $x > N^{1/4}$, $x=46,47,\ldots$. For sake of simplicity use $x=46,47,48,49,50$ and $51$. Therefore $Q(x_i)$ can be represented as follow:

\[
\begin{align*}
Q(46) &= 46^2-2041=75 \\
Q(47) &= 47^2-2041=168 \\
Q(48) &= 48^2-2041=263 \\
Q(49) &= 49^2-2041=360 \\
Q(50) &= 50^2-2041=459 \\
Q(51) &= 51^2-2041=560
\end{align*}
\]

From equation 2.8, we noticed that $Q(x)$ is a square, therefore it is necessary to select from the relations above for squares. Unfortunately, there isn’t any perfect square from the result above. But there is another option, instead of finding square from $x$, multiplying $Q(x_i)$s might able get a product with square.

\[ v^2 = Q(x_1) \cdot Q(x_2) \cdot \ldots \cdot Q(x_i) \text{ and } u^2 = x_1^2 \cdot \ldots \cdot x_i^2 \]

where, $u^2 = x_1^2 \cdot \ldots \cdot x_i^2 = Q(x_1) \cdot \ldots \cdot Q(x_i) = v^2 \mod N$

It is noticed that some of the $Q(x)$s above can be factored easily,

\[
\begin{align*}
Q(46) &= 75 = 3 \cdot 5^2 \\
Q(47) &= 168 = 2^3 \cdot 3 \cdot 7 \\
Q(48) &= 263 = a \text{ prime number} \\
Q(49) &= 360 = 2^3 \cdot 3^2 \cdot 5 \\
Q(50) &= 459 = 3^3 \cdot 17 \\
Q(51) &= 560 = 2^4 \cdot 5 \cdot 7
\end{align*}
\]
By multiplying the selected Q(x)s, namely Q(46)•Q(47)•Q(49)•Q(51), \(2^{10} \times 3^4 \times 5^5 \times 7^2 = (2^5 \times 3^2 \times 5^2 \times 7)^2\), which is a square. Thus,

\[v^2 = (2^5 \times 3^2 \times 5^2 \times 7)^2 \equiv (46 \times 47 \times 49 \times 51)^2 \mod N\]

Finally, we get \(v = 2^5 \times 3^2 \times 5^2 \times 7 = 50400\) and \(u = 46 \times 47 \times 49 \times 51 = 540283\)

Once value of \(u\) and \(v\) have been identified g.c.d\((u-v, n)\) or g.c.d\((u+v, n)\) need to be calculated in order to get the factor of \(N\). Greatest common divisor can be done efficiently using Euclid algorithm [RIE1985, KNU1981].

### 2.3.3.3 Smooth Number and Complexity

A number is called B-Smooth if it has no prime factors that exceeding a fixed boundary of B. Here, B is the largest prime factor.

The probability that a random positive integer up to \(X\) is B-Smooth can be denoted as \(\Psi(X, B)/[X] \approx \Psi(X, B)/X\). Where \(\Psi(X, B)\) is the number of B-Smooth numbers in the interval of \([1, X]\). Thus, the expected number of random numbers that must be examined to find just one that is B-Smooth is \(X/ \Psi(X, B)\). Iterative operation about \(\pi(B)\) times for such B-Smooth number to be found. Here \(\pi(B)\) represent the number of primes up to B. Therefore total expected random numbers that must be examined is about \(\pi(B) \times X/ \Psi(X, B)\).

When B is in reasonable size and X is very large, the probability for smooth relation is very, very small. So one after another, more time need to be spent in each number, only to find out almost always that the number is not B-Smooth and thus a number
that will be discarded. QS algorithm that capable to search smooth relations in an
efficient way will be discussed in the following section.

2.3.3.4 The Quadratic Sieve

Carl Pomerance [POM1996] suggested a sieving algorithm that can be performed
extremely fast which allows trial division of numbers without really dividing them.
He introduced a quadratic polynomial in the form of,

\[ Q(x) = (x + \left\lfloor \sqrt{N} \right\rfloor)^2 - N \]  \hspace{1cm} (2.10)

where,

\[ \left\lfloor \sqrt{N} \right\rfloor \] is the smallest integer which is larger than the square root of \( N \).

\( x = \pm 1, \pm 2, \ldots \)

The following relation is satisfied,

\[ (x + \left\lfloor \sqrt{N} \right\rfloor)^2 \equiv Q(x) \mod N \]  \hspace{1cm} (2.11)

As seen from the example 2.2 in section 2.3.3.2, a relation is called smooth when it is
completely factor over a set of prime numbers, \( p \). This set of prime numbers is call
Factor Base or FB for short. There are two issues:

1. How to select \( x \) so that \( Q(x) \) is smooth?

2. How to determine a set of prime numbers so that there won't be any unsuccessful
   trial division of \( p \) over \( Q(x) \)?

By answering above questions, a refinement of Kraitchik's methods (section 2.3.3.2)
for better factorization speed can be produced - Quadratic Sieve.
2.3.3.5 Solving Congruence

For a smooth relation, $Q(x)$, that is divisible by a prime number $p$, the following relation is satisfied:

$$p | Q(x) \quad \text{or} \quad p \bigg| \left( x + \left\lfloor \sqrt{N} \right\rfloor \right)^2 - N$$

Then, the modulus in the form of

$$(x + \left\lfloor \sqrt{N} \right\rfloor)^2 = N \mod p$$

Let $t = (x + \left\lfloor \sqrt{N} \right\rfloor)$,

$$t^2 \equiv N \mod p \quad \text{(2.12)}$$

If $N$ is a non-zero square modulo $p$, or more precisely if $N$ is the quadratic residue of $p$ then there are two residues $a$ and $b$ such that $Q(x) \equiv 0 \mod p$. If $N$ is not a square modulo $p$ then $Q(x)$ is never divisible by $p$ and therefore no further computation with $p$ will be done.

**Example 2.3:** Solve equation (2.12) where $N=87463$, $\left\lfloor \sqrt{N} \right\rfloor = 296$ and $p=13$

Congruence (2.12) has 2 solutions namely $+t$ and $-t$.

$$\pm t = (x + \left\lfloor \sqrt{N} \right\rfloor) \mod p \quad \text{(2.13)}$$

$$x_1 = t - \left\lfloor \sqrt{N} \right\rfloor \mod p \quad \text{(2.14)}$$

$$x_2 = -t - \left\lfloor \sqrt{N} \right\rfloor \mod p \quad \text{(2.15)}$$

Solving congruence $t^2 \equiv 87463 \mod 13$, we get

$$t^2 \equiv 12 \mod 13$$

$$t = 5 \quad \text{and} \quad t = -5$$

Therefore,
\[ x_1 \equiv 5 - 296 \mod 13 \]
\[ = -5 \mod 13 \]
\[ \text{or} \quad = 8 \mod 13 \]

\[ x_2 \equiv -5 - 296 \mod 13 \]
\[ = -2 \mod 13 \]
\[ \text{or} \quad = 11 \mod 13 \]

Two x values that Q(x) are divisible by 13 can be obtained. This is also true for 
\[ Q(x + p), Q(x + 2p), \ldots, Q(x + kp) \] . By solving congruence (2.12) for each prime
number in the factor base, we are able to construct a list of x values (or candidate list
of x) that are divisible by the prime numbers and avoid any unsuccessful trial division.

Although there won't be any unsuccessful trial division of primes but certainly waste
of time if Q(x) not entirely decomposed after division by some primes contained in
FB.

### 2.3.3.6 Building Factor Base

Notice that \( p | Q(x) \) or if \( (x + \sqrt{N})^2 \equiv N \mod p \) then N must be a quadratic residue of
p. In order to build a set of prime numbers, test on this congruence for quadratic
residue must be done using Legendre Symbol, \( \left[ \frac{N}{p} \right] = 1 \) (see [RIE1985] pp. 278-288).

Hence, the factor base should only consists of primes such that this is the case and
FB = \{ \( 2, 3, 5, \ldots, B \} \cup \{-1\} \).

Legendre symbol denoted that \( \left[ \frac{N}{p} \right] = \begin{cases} +1 & \text{If N is quadratic residue} \\ -1 & \text{otherwise} \end{cases} \)

The requirements for the factor base are:

- The primes should be small enough to increase the possibility that Q(x) factors
  completely over them.
• The size of the factor base should be small so we don’t need to find too many relations, but this will be very rare and hard to find.

• The size of the factor base should be large so we have a fair chance of finding value of $x$ such that $Q(x)$ can be factored over the factor base, but more of them are needed.

Based on [LAN2001], the optimum value of factor base can be obtained roughly by:

$$B = \left( e^{\sqrt{\ln(n) \ln\ln(n)}} \right)^{1/2}$$

(2.16)

2.3.3.7 Sieving

Sieving is the time consuming stage in Quadratic Sieve algorithm where $Q(x)$ is repeatedly tested for complete factorization over the FB. This section will discuss how candidate $x$s chosen from section 2.3.3.5 can be short listed and left only $x$s that have the higher probability to be factored completely over the FB.

An array (sieve array, log $p[x]$) of $x$ length will be initialize to hold the log($p$). For each odd prime number in FB, solve $t^2 \equiv N \mod p$ and obtain two $x$ values which its $Q(x)$ is divisible by $p$. For each $x$ as the solution for the congruence, $x + p, x + 2p, \ldots, x + kp$ are also the solutions. With all the necessary data in hand, the sieve array can be marked for which $Q(x)$ is divisible by $p$ by adding log($p$) to location $x$ in the sieve array. A value of $Q(x)$ that factors completely over the FB should theoretically have its log ($Q(x)$) reduced to zero by this procedure. However this is not always the case because a prime in the factor base might be a factor of $Q(x)$.
more than once. This is why probability elements need to be considered. Silverman [SIL.1987] suggested that an integer should be considered as possible being able to factor completely if the accumulated log(p) value in sieve array is approximately, 
\[ \log p[x] \geq \frac{1}{2} \ln(n) + \ln M - c \ln(p_{\max}) \]. Where, \( M \) = number of integers to be sieved over and \( p_{\max} \) = the largest prime in the FB. \( c \) has been suggested as following values for specific digit length,

<table>
<thead>
<tr>
<th>Table 2.2 c value</th>
<th>Digit</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.5</td>
<td>30</td>
</tr>
<tr>
<td>2.0</td>
<td>45</td>
</tr>
<tr>
<td>2.6</td>
<td>66</td>
</tr>
</tbody>
</table>

All sieve array at location \( x \) that doesn’t satisfy above threshold will be discarded. The sieving interval can be made between \( \left\lfloor \sqrt{N} \right\rfloor - M \leq \left\lfloor \sqrt{N} \right\rfloor + M \), where \( M \) is set to a bound value. If the numbers of smooth relations was not yet sufficient, then another \( 2M \) integers will be sieved again. This will continue until desired amount of smooth relations are found.

### 2.3.3.8 Choosing Appropriate Congruence

After sufficient smooth relations had been collected, the relations such that the product of the following congruence’s right hand side which is a square need to be chosen.

\[
(t_1 \times t_2 \times \ldots \times t_i)^2 = Q(x_1) \times Q(x_2) \times \ldots \times Q(x_i) \mod N
\]

\[
= \left( \prod_{i=1}^{FB} p_i^{e_i} \right)^2 \mod N
\]

(2.17)

**Example 2.4:** Suppose smooth relations for \( N=7429 \) and \( FB=\{2, 3, 5, 7\} \) are obtained by solving \( x_i^2 \equiv Q(x_i) \mod 7429 \) for \( x = -3, 1, 2 \)

22
\[ Q(-3) = 83^2 - 7429 = -540 = -1 \times 2^2 \times 3^3 \times 5 = 83^2 \mod 7429 \]
\[ Q(1) = 87^2 - 7429 = 140 = 2^2 \times 5 \times 7 = 87^2 \mod 7429 \]
\[ Q(2) = 88^2 - 7429 = 315 = 3^2 \times 5 \times 7 = 88^2 \mod 7429 \]

By solving a linear algebra system, the appropriate congruence for the above example can be filtered and the one such that satisfy 2.17 can be obtained. The selection process is controlled by coefficient \( \lambda_i \in \{0,1\}, 1 \leq i \leq 3 \). If \( \lambda_i = 1 \), then the congruence \( i \) is chosen; otherwise it is not. Let multiply the above relations,

\[ (-1 \times 2^2 \times 3^3 \times 5^4) \times (2^2 \times 5 \times 7)^{\xi} \times (3^2 \times 5 \times 7)^{\xi} = (-1)^{\xi} \times (3^{1+2\xi}) \times (5^{4+\xi}) \times (7^{\xi}) \]

If the above equation is to be square then the exponent of -1 and all prime numbers are to be even. This led us to the following linear system:

\[
\begin{align*}
\lambda_1 &= 0 \mod 2 \\
2\lambda_1 + 2\lambda_2 &= 0 \mod 2 \\
3\lambda_1 + 2\lambda_3 &= 0 \mod 2 \\
\lambda_1 + \lambda_2 + \lambda_3 &= 0 \mod 2 \\
\lambda_2 + \lambda_3 &= 0 \mod 2
\end{align*}
\]

The coefficients of \( \lambda_i \) are reduced mod 2, we obtain the simplified linear system

\[
\begin{align*}
\lambda_1 &= 0 \mod 2 \\
\lambda_1 + \lambda_2 + \lambda_3 &= 0 \mod 2 \\
\lambda_2 + \lambda_3 &= 0 \mod 2
\end{align*}
\]

The solution is

\[ \lambda_1 = 0, \quad \lambda_2 = \lambda_3 = 1 \]

This shows that the product of the right-hand sides of the second and third congruence is a square. Therefore, second and third congruence will be chosen. Gaussian elimination can be used to solve this linear system in GF[2]. The matrix has column size of the FB and the row size of FB+10 smooth solution. The row size must has the value >FB+10 to have a greater probability of finding a solution.
2.3.3.9 Variants of Quadratic Sieve

The basic quadratic Sieve algorithm described above is using one polynomial to search smooth value. This happens to increase the value of \( Q(x) \) when \( x \) gets bigger and directly affect the possibility of smooth relations [SIL1987, MIN1995 and POM1985]. Thus, as the algorithm progress, smooth relations become less frequent. To overcome this problem, multiple polynomial QS or MPQS; the variation of QS need to be used. MPQS is using a family of polynomial rather than just one polynomial used in basic QS. It is said that MPQS runs about 17 times faster than the basic QS method.

<table>
<thead>
<tr>
<th># decimals</th>
<th>date</th>
<th>algorithm</th>
<th>effort (MIPS years)</th>
</tr>
</thead>
<tbody>
<tr>
<td>39</td>
<td>Sep 13, 1970</td>
<td>CF</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>1983</td>
<td>CF</td>
<td></td>
</tr>
<tr>
<td>55–71</td>
<td>1983–1984</td>
<td>QS</td>
<td></td>
</tr>
<tr>
<td>45–81</td>
<td>1986</td>
<td>QS</td>
<td></td>
</tr>
<tr>
<td>78–90</td>
<td>1987–1988</td>
<td>QS</td>
<td></td>
</tr>
<tr>
<td>87–92</td>
<td>1988</td>
<td>QS</td>
<td></td>
</tr>
<tr>
<td>93–102</td>
<td>1989</td>
<td>QS</td>
<td></td>
</tr>
<tr>
<td>107–116</td>
<td>1990</td>
<td>QS</td>
<td>275 for C116</td>
</tr>
<tr>
<td>RSA–100</td>
<td>Apr 1991</td>
<td>QS</td>
<td>7</td>
</tr>
<tr>
<td>RSA–110</td>
<td>Apr 1992</td>
<td>QS</td>
<td>75</td>
</tr>
<tr>
<td>RSA–120</td>
<td>Jun 1993</td>
<td>QS</td>
<td>835</td>
</tr>
<tr>
<td>RSA–129</td>
<td>Apr 1994</td>
<td>QS</td>
<td>5000</td>
</tr>
<tr>
<td>RSA–130</td>
<td>Apr 1996</td>
<td>NFS</td>
<td>1000</td>
</tr>
<tr>
<td>RSA–140</td>
<td>Feb 1999</td>
<td>NFS</td>
<td>2000</td>
</tr>
</tbody>
</table>

2.4 Distributed and Parallel Computing

A distributed system is one which components are distributed across a network of computers and coordinates their action by message passing in order to achieve a
common goal [GJT2001]. Sharing of resources is the main motivation for constructing a distributed system. There are many compelling reasons for building an application this way. You may find several organizations in the world such as SETI@Home and Genome@Home which applying massive distributed computing by utilizing networked computers in the internet to help them in solving huge computation problems.

The following subsection will briefly explain the beneficial of using distributed computing and it challenges. Distributed system can be in many forms and section 2.4.3 will introduce the concept of Tuple Space (discussed in section 2.4.3) employed by this thesis.

2.4.1 Benefits of Distributed Computing

Performance: When a person satisfy with his current computer performance for running an application, it doesn’t mean that the same performance will be produced for running a new version of that application. However, it is possible to achieve more performance by adding another computer. To do this, we may think of decomposing a problem into smaller ones and distribute them over one or more computers to be computed in parallel. Parallel mechanism discussed here is suitable for those problems where communication to computation ratio is low.

Scalability: When designing an application based on this system model, performance improvement not only can be seen; the application also capable to scale based on the problem domain: distributed system that can grow or shrink to match the size of the problem without redesign the application.
common goal [GJT2001]. Sharing of resources is the main motivation for constructing a distributed system. There are many compelling reasons for building an application this way. You may find several organizations in the world such as SETI@Home and Genome@Home which applying massive distributed computing by utilizing networked computers in the internet to help them in solving huge computation problems.

The following subsection will briefly explain the beneficial of using distributed computing and it challenges. Distributed system can be in many forms and section 2.4.3 will introduce the concept of Tuple Space (discussed in section 2.4.3) employed by this thesis.

2.4.1 Benefits of Distributed Computing

Performance: When a person satisfy with his current computer performance for running an application, it doesn’t mean that the same performance will be produced for running a new version of that application. However, it is possible to achieve more performance by adding another computer. To do this, we may think of decomposing a problem into smaller ones and distribute them over one or more computers to be computed in parallel. Parallel mechanism discussed here is suitable for those problems where communication to computation ratio is low.

Scalability: When designing an application based on this system model, performance improvement not only can be seen; the application also capable to scale based on the problem domain: distributed system that can grow or shrink to match the size of the problem without redesign the application.
Fault tolerance and availability: Standalone system typically have little tolerance for failure; once system fail the whole process will be stopped. On the other hand, if one of the computers in a distributed system crashed others are still capable to perform the task. A carefully designed distributed system can reduce down time and maximize its availability.

2.4.2 Challenges

Designing a distributed system can be quite difficult as it involves a more complex blueprint, hard to build and debug as compared to standalone application. The system can getting more complex when designing distributed system for various hardware architecture and operating system platform. However, platform independent programming language such as JAVA which is specially design for networking is able to overcome most of the heterogeneity problems.

Nevertheless, this is not the only difficulty that one has to face when designing and implementing distributed applications: Processes need to communicate with each other in order to work together. Since the communication is over the networks, network latency and reliability is always taking into account when designing an efficient distributed application. Actions taken by processes need to be synchronized (wait for their turn) when processes are cooperate to complete a task. Finally, if the application is fault tolerant it should be able to resist partial failures in a manner that it maintains a consistent global state.
2.4.3 Tuple Space

Tuple Space was first described by David Gelernter in the context of a programming language called Linda [CAR1984]. Linda is an implementation of Tuple Space that provides functionalities for parallel and distributed programming environment.

A tuple space is a globally shared persistent memory (provided by Linda as Virtual Share Memory (VSM)) in which processes in Linda parallel program execute simultaneously and exchanging data (known as tuple) by generating, reading and consuming tuples. A tuple is simply a vector of typed valued called fields. It acts as logical persistent storage unit in tuple space. For example, a tuple could consist of ("odi", 9.4, 1975). To update a tuple, the tuple is removed from the tuple space, modified and returned back to tuple space.

Linda parallelism model is based on generative communication [CAR1984] whenever two processes need to communicate. They don’t exactly exchange messages or share a variable; instead, process will generate a data object (tuple) and deposits it in the tuple space. After this, the receiver may now access the tuple in the tuple space. A process that wants to create a second, concurrently executed process generate a “live tuple” – via eval(expression), and sets it adrift in the tuple space. The “live tuple” carries out some specific computation at it own, independent of the process that generates it, and then produces the result in the form of ordinary data object tuple to be collected by the process.

Tuple space provides a simple scheme which has a series of important implication. First of all, communication and process creation are two facets of the same operation.
To create processes, “live tuple” are generated, which will then turn into data object tuple as result of the process; to communicate, data object tuple are created directly when new object tuple is added into the tuple space, any interesting party may access it. Secondly, the data is exchanged in the form of persistent object; therefore, the receiver may read or modified the tuple any time it wants. The fact that senders in Linda environment needn’t know anything about receiver and vice versa promote a communication approach called loosely coupled communication.

The `eval()` mechanism in Linda for creating new process or “live tuple” in tuple space shows that Linda model capable to provide the simplest and fine grained approached to parallelism.

The following are the four basic operations of Linda:

a. `rd(tuple)` - copies data from the tuple space without removing the corresponding data object.

b. `in(tuple)` - retrieves data from the tuple space, simultaneously removing the corresponding data object.

c. `out(tuple)` - writes a tuple into tuple space.

d. `eval(expression)` - writes a tuple to tuple space after arguments in the `expression` are evaluated by creating new processes which perform their tasks independently.

A variety of access routes to this tuple are possible, e.g., any of the following operations suffices:

\[
rd(\text{"a string"}, ?fval, ?ival, ?strval12)
\]

\[
rd(?strval, 9.4, ?ival, \text{"another string"})
\]

28
The "?" operator designates a "wildcard" value returned from a matching tuple. Fields marked by the operator do not participate in the (associative memory) matching process. Any of the three examples, rd() operations results in a non-destructive reading of the original tuple. If the operation were an in(), the tuple would be removed from tuple-space. Figure 2.2 [PAT1993] depict the model and basic operation of Linda.

Figure 2.2 Linda operation environments
Linda's principles are now widely apply by many large software companies such as Sun Microsystems and IBM. An implementation of tuple space called JavaSpaces in Java programming language has been introduced into Sun's JINI technology family in order to provide functionality for building distributed system. IBM has a similar project, called TSpaces (visit http://www.almaden.ibm.com/cs/TSpaces/index.html).

2.4.4 JavaSpaces

JavaSpaces is the implementation of tuple space concept in Java language [JSPACE]. Java is popular for its portability and the combination of Java language plus tuple space has yielded interest among developer's community for designing powerful parallel and distributed application. JINI [EDW1999] on the other hand, is a set of specification that enables services in the network to interact seamlessly and flexibly. JavaSpaces is part of the member/service in JINI. A detailed discussion of JINI infrastructure can be reviewed in section 2.4.6.

Most of the features and definitions describe in tuple space can also apply to JavaSpaces with little differences. In JINI specification, JavaSpaces is consider a service which act as a group of shared, network accessible repository for objects that can be utilized by a group of JINI clients and services. It is a high level coordination tool for gluing processes together into a distributed application.

Objects resided in JavaSpaces are called Entries and unlike tuple space all Entries are passive. A passive Entry can be "activated" to provide useful functions by running the code inside it when explicitly retrieve from JavaSpaces. For more details review of
JavaSpaces, please refer to [FHA1999]. Subsection below will explain the basic operations of JavaSpaces may perform.

2.4.4.1 JavaSpaces Operations

Figure 2.3 depicts the three basic operations that facilitate JavaSpaces interaction. Objects/Entries are written into and read or taken from a space, which are analog to basic operations in word processing application: paste, copy and cut.

Additional functionalities include Entry notification mechanism and snap-shooting technique designed to reduce the performance impact of respected template matching when doing associative lookup.

![Diagram of JavaSpaces operations: write, take and read](image)

Figure 2.3 An overview of JavaSpaces operations: write, take and read

2.4.4.2 Associative Lookup

Associative lookup is the mechanism by which objects/entries are retrieved from JavaSpaces. This is done by first create a template to describe the desired object to be
retrieved based on the object's content. A template is actually an object with some or all of its fields set to specific values and the others are left as null to act as wildcards.

2.4.4.3 Atomic Transaction

JavaSpaces also implement a transaction model to ensure that an operation in a space is atomic. Each transaction is supported for single operation on a single space involving one or more object/entry. Transactions are important way to deal with partial failure.

2.4.5 Space-based Communication and Parallel Computing

Space-based communication is slightly different from message-passing or remote method invocation (RMI). Space-based communication loosens the ties between information of the senders and receivers therefore promote a loosely coupled communication style in which senders and receivers don’t interact directly, but indirectly through a space [FHA1999]. This eliminates the worries about multithreaded server implementation, synchronization issue, network latency which always is the concern when designing distributed system. In the case of designing parallel computing, these characteristics offer two main advantages: load balancing and scalability. Such distributed computing that exploits networked resources and load balancing the workload lead to adaptive computing [CGKW1993].

JavaSpaces helps in creating reliable and fault tolerance system. The system model is simple and clean, hence, programmer can concentrate more on the problems.
2.4.6 JINI (Java Intelligent Network Infrastructure)

JINI is a set of Java APIs and network protocols that facilitate the developer to build and deploy distributed systems that are organized as federation of services [VEN]. Service or Service provider is described as anything that connected to the network and is ready to perform a useful function. Therefore, JINI technology infrastructure provides a mechanism allowing formation of a federation of services or a set of services which enable them to access each other seamlessly in order to accomplish the same goal. The main goals of JINI system are,

- Enabling users to share services and resources over a network
- Providing easy access to resources anywhere on the network while allowing the network location of the services/resources to change.
- Simplifying the task of building, maintaining and altering the network of devices, software and user.

The Java application environment provides a good computing platform for distributed computing because of it portability and platform independency. JINI view of the network doesn't involve a central controlling authority. Instead of a central authority, JINI's runtime infrastructure provides a way for clients and services to locate each other easily. This is done with the help of a lookup service which stores a directory of currently available and registered services. Once services are identified and located, clients or services will operate independently of JINI runtime infrastructure. If the JINI's lookup service crashes, any distributed system brought together by the lookup service before hence can still continue to operate and communicate with each other. JINI also provides a method that enable clients to locate and communicate with
services in the absence of lookup service. Understanding of how JINI architecture works is described below.

2.4.6.1 JINI Service Architecture

JINI technology infrastructure provides mechanism for devices, services and users to join and detach from a network in a natural and dynamic manner [JINIARC]. Central organizing mechanism for JINI based system lies on a service called Lookup Service. Whenever new services become available on network, registration with lookup service is mandatory in order for other services or clients to query and locate the resources that required.

The JINI service infrastructure uses one network level protocol called discovery and two object-level protocol called join and lookup. Here, discovery enables clients and services to locate lookup services. join enables a service to register itself with lookup service that was found. Finally, lookup is performed when a client needs to query and invoke a service with lookup service in order to help the client to accomplish its goals.

a. Discovery Process

A lookup service is found by dropping a multicast request packet destined to a well known port. If the lookup service detected the presence of the request packet, it will contact the sender directly by making unicast connection. A mobile object called service registrar will be received by the sender via RMI. This service registrar is an object that will be used by the sender to conduct further communication with the lookup service.
b. Join Process

Once service registrar is obtained, service provider can perform a \textit{join} operation with the lookup service in order to become part of the federation of services that are registered in the lookup service. To do a \textit{join}, the service provider invoke service registrar's \texttt{register()} method, passing a parameter called a \textit{service item} which describe services that can be provided. This service item will be kept by the lookup service and will be given to other services or clients upon request. The service item contains several objects including an on object call \textit{service object} which provides all the necessary communication functionalities.

c. Lookup Process

Client will invoke the \texttt{lookup()} method on the service registrar object to perform query of services. What the client needs to do is to create a \textit{service template} and pass it as argument. The service template contains a reference of an array of Class objects. The Class object defines the Java type of the service object that desired by the client. Whenever a service was found in lookup service, a service object for that desired service will be returned.

Service object implement a well-known interface and provides all the necessary implementations in order for client to communicate with the service provider. The service object act as a proxy to remote server and client doesn’t need to know about the underlying network protocol to do the job; all it needs to do is to use and call the methods defined by the well-known interface. Figure 2.4 shows how service object works.
Sequence diagram below shows the overall processes that happen between lookup service, client and service.

1. Discovery (port 4160)
2. Reply with service registrar object
3. Call service registrar's register() method to register
4. passing a service item object
5. Discovery (port 4160)
6. Reply with service registrar object
7. Call service registrar's lookup() method to locate JavaSpaces service
8. Returning JavaSpaces's service
9. Client communicate with JavaSpace via the service object

Figure 2.5 JINI's Discovery, Join and Lookup

2.5 Differences between LINDA and JavaSpaces

Although JavaSpaces is the successor of LINDA, they are dissimilar in many aspects (further details can be found in [JSPACE]):
JavaSpaces entries are typed where type is used together with the field values for entry matching.

The JavaSpaces support transactions which can span multiple spaces.

Entries deposited in space are leased which needed for garbage collection.

JavaSpaces model allows clients to be notified when a matching entry is written to the space through remote events.

JavaSpaces model has no equivalent to LINDA's "eval".

2.6 Current Research

Tuple Space model has opened up a new door for building distributed and parallel system in much simplicity methods than others. By combining the power of Java programming language with the flexibility of the JINI network technology, JavaSpaces has become the most simple and expressive tool for building distributed system which take care of the inherent problems of distributed computing on behalf of the developers.

Several empirical studies on JavaSpaces performance have been conducted by Noble et al. [NOB2001]. Several tests had been carried out to determine the usability of JavaSpaces implementation for parallel and distributed system, they are: prime number generator, Monte Carlo algorithm (particle shielding and light propagation), low-level I/O rate and etc. They conclude that JavaSpaces has a drawback in communication latency which is unsuitable for several algorithm but others may significant benefit from it in term of computation speedup.
Arnold et al. [AKR2002] use the combination of JINI network and JavaSpaces object repository to design and implement an efficient and scalable persistent JavaSpaces service which called RDBSpace. The system is supported by relational database back-end. Their implementation has successfully overcome the limitation of Sun's persistent JavaSpaces on using structured query language (SQL) for object retrieval and storage.

Meanwhile, a research project was conducted by Zorman et al. [ZKR2002] to implement distributed genetic algorithms (GAs) that attempt to solve the Knapsack problem. They found that JINI and JavaSpaces are able to reflect and simulate the real scenario of their problem domains and successful of decreasing the execution time of a genetic algorithm which process in a parallel fashion.

2.7 Chapter Summary

RSA algorithm and several factoring algorithms are reviewed in this chapter. RSA algorithm has been accepted by public and widely uses in today applications like email, online transaction, and etc. Its security strength has caught attention and raise concern of public as newer and faster factoring algorithm are being discovered. Luckily, there is no new advance algorithm discovered yet to replace NFS [POM1982] algorithm to factor current uses of public key size. However, the trend of key size is growing as time goes along with the increase of computer speed.

Solving a complex mathematical problem may require the power of multiple processors. Coordination of these resources requires advance protocols and expensive equipments. Tuple space has brought new concept of how parallelism can be achieved
in low cost manner. This chapter has briefly explained the JINI architecture and its tuple space implementation called JavaSpaces.

Scientific communities are employing JINI and JavaSpaces in their research field by utilizing the networked resources. An ongoing investigation is crucial to develop a more efficient system on top of this new technology.