**An Implimentation of RSA algorithm**

by

**Maziar Kaveh**

**Electrical and Computer Engineering Dep. University of Stavanger (UiS)**

[m.kaveh@stud.uis.no](mailto:r.asadollahi@stud.uis.no)

**Abstract**

**RSA** is an [algorithm](http://en.wikipedia.org/wiki/Algorithm) for [public-key cryptography](http://en.wikipedia.org/wiki/Public-key_cryptography) that is based on the presumed difficulty of [factoring](http://en.wikipedia.org/wiki/Factorization) [large integers](http://en.wikipedia.org/wiki/Integer), the [factoring problem](http://en.wikipedia.org/wiki/Factoring_problem).

This paper presents an implementation of RSA algorithm. The application works fine with p and q in any sizes however since all algorithms developed. Block size will be created dynamically in the application (number od bits -1). has been implemented using Java programming language for this project. The project services get some plain text or cipher and a key and outputs cipher or plain text respectively.

**Keywords**: RSA algorithm, Public key, Asymmetric encryption, Deterministic Algorithm, Prime numbers, GDC, Plaintext, block, Ciphertext.

## Introduction

Public key cryptography was invented in 1976 by Whitfield Diffie and Martin Hellman. And for this reason it is sometime called Diffie-Hellman encryption. The development of publi-key cryptography is the greatest and perhaps the only true revolution in the entire history of cryptography.All other cryptographic systems have been based on the elementry tools of substitution and premutation but public key algorithms are based on mathematical functions rather than on substuitution and premutation.Public key algorithms are asymmetric and two separate key are used to encryption and decryption. A public key known to everyone and a private or secret key known only to the recipient of the message.

The most widely used public key algorithm is RSA. The difficulty of attacking RSA is based on the difficulty of finding the prime factors of a composite number. RSA stands for [Ron Rivest](http://en.wikipedia.org/wiki/Ron_Rivest), [Adi Shamir](http://en.wikipedia.org/wiki/Adi_Shamir) and[Leonard Adleman](http://en.wikipedia.org/wiki/Leonard_Adleman), who first publicly described it in 1977. The prime factors must be kept secret. Anyone can use the public key to encrypt a message, but with currently published methods, if the public key is large enough, only someone with knowledge of the prime factors can feasibly decode the message.

## Design and Implementation

In order to implement the RSA algorithm we need to choose the following design features that depends on your choice for each parameter it affected security and speed of encryption and decryption specially bit length of the prime numbers.

Design overview:



1. Choose two large primes p and q and denote :
2. Choose a large e that :

Compute d as:

1. Public key: n, e and private key: n, d.
2. Encryption For Plain text M : 
3. Decryption for cipher C : 

**P and Q bits length:** Because the value of n=pq will be known to everyone as part of public key, in order to prevent the discory p and q by exhustive methods ,these primes must be large numbers.In other hand the method that is used to generate these numbers must be resonably efficent. A RSA key length of 1024 bits is sufficient for many medium-security purposes such as web site logins but for high-security applications or for data that needs to remain confidential for more than a few years, we should use at least a 2048-bit key.In the implementation to find prims p and q we use a random generate method and we use Fermat primalitry test several times.The default random numbers that pick to do **Fermat** test is 10 but the user can change it to reach a sufficient number of tests. It seems **Miller-Rabin** algorithm is more efficient and popular than Fermat but in most algorithms tests are probabilistic. This implimentaion support almost 30 bits and even more (depends on the CPU) length for p and q. The buttle neck of supporting larger number is finding d that by increasing sizes of ø (phi) and e it getting harder and harder to find. In order to improve the implementation we need to implement the Extended **Euclidean Algoithm** for finding **modular multiplicative devisor** (MMD).

**Block Size:** RSA is a block cipher and plaintext is encrypted in blocks, with each block having a binary value less than n, So the block size is i bits, where  . In the application we use a fix 8 bits block size and the blocking has not been implemented due to the lack of time but it is possible to have different block size depends on the n. The default block size 8 bits are used in application to support ASCII chars without padding. And because of the size of the block (8 bits) then n should be greater than  as mentioned above . However the implimentation always check to be sure that.

**Finding e:** We also need to choose an integer e that 1<e<and greatest common devisor (GDC) of. In other hand e and are coprime and then e is released as the public key exponent. Having a short bit-length e results in more efficient encryption but small values of e such as 3 seems less secure especially for small plaintext value, ciphertext can be easily decrypted by taking root of the cipher text. To find out e we use Euclid’s Algorithm to find GCD of two integers (a random number that supposed to be e and) and then if GCDthen we have e.

**Finding d:** d is the multiplicative inverse of e mod.It means. This is equivalent to .In order to find d we can use Extended Euliden Algorithm that seems needed to do Euliden Algorithm and save result in the table and replace them step by step in a recursive method but I couldn’t implement it completely so to find d I just used the equation but using incremental integer to find d and when and e getting larger and larger, finding d will take more and more time and the algorith become almost insufficient. In RSA the private key consists of n and the private exponent d which must be kept secret. p, q and  must also be kept secret because they can be used to calculate d.

For downloading the code you can go to

<https://code.google.com/p/uis-security-asignments/>

Or checkout the source with subversion using this URL

svn checkout ***http***://uis-security-asignments.googlecode.com/svn/trunk/ uis-security-asignments-read-only

You need to go to [Oracle website](http://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1880260.html) to download Java JDK for compiling the source code.

After downloading and installing JDK you may set JAVA\_HOME environment variable to the path that JDK has been installed.

Use

java –version

To check your Java installed correctly.

For building the project you need to [download](http://maven.apache.org/download.cgi) and install in your hard drive apache Maven project.

Again you can use

mvn -version .

To install the project you just need to go to the root of source code which pom.xml located and type

mvn install

Whole source codes will be compiled and 28 unit tests will run after compiling the jar files will be copied into target folder.

By running run.sh in UNIX environment or run.bat in windows the command user interface will come and you can input your data.

In this project tried to used Object Oriented concepts, Dependency injection using [Spring](http://www.springsource.org/about) framework, [JUnit](http://junit.org/) and Spring test framework were used for unit and integration testing of the project which was tried to developed base on TDD, Finally [SLF4J](http://www.slf4j.org/) over [LOG4j](http://logging.apache.org/log4j/2.x/) was used for logging.

RSAKeys

Keys are stored in this model for sharing among users.

UnsignedBigNumber

the model that has array of bits as value and uses LogicalUtils class for mathematic operation on values.

RandomGenerator

next()

generate a random number

next(int n)

generate n bits random number

EncryptionService

encrypt()

This is what a client wants to work with just getting text or byte array ,private or public key and modulus as inputs and cypher code returns as return function.

decrypt()

Does same operation like encrypt() gets cypher code and returns array of bytes.

For each service class there is a test class which tests unit and integration of frame work RSAEncryptionServiceTest, RSAEncryptionServiceTest, are test classes.

## Test results:

mvn test

Runs all tests

All dependency libraries were added using maven to the project.

**Discussion:** We start the test from the lowest prime size that can generate n greater than 256 (anyway because we use ASCII and the max ASCII value is 127 we can check n to be less than 128 ) but to support the block size generally we always check n to be grater than . The first prime size was 5 and in some cases it took a lot of time to find p,q that pq > 256. By choosing prime size 6 as the test result shows in some cases the value of e could be very =small (5) and it make it easy to attack for small value of plaintext, by finding root of ciphertext. But when we increase the prime size as the test result shows the ciphers value for blocks increased very well and for prime size 30 the cipher value for the plaintext block (8 bits) has 16 to 17 digits (56 bits) . As in RSA confusion and diffusion are not used so we don’t need to change the plaintext and so on to see the changes. In fact RSA is **deterministic algorithms** that make it vulnerable to another attach type called **Chosen plain text attack.** By looking at test results we can find out that this implementation are totally vulnerable against this type of attack. And the only reason is that the block value is as same as the char value so an attacker can make cipher for all ASCII chars and nothing else is needed to decipher the encryption ! Even the key is not needed because in fact what we have done is converting value of chars from ASCII to another value. However having small block size make the algorithm very vulnerable to Chosen plain text attack. For example if we choose 16 bits for block size then for each 2 chars this happens. So we need a big block size to make it secure against this attack and problem.

**Conclusion:**  In this assignment I implemented the RSA public key algorithm for Asymmetric encryption. Generally the algorithm has been implemented without using any library or function and all functions have been implemented using available algorithms. The only part that has not been implemented is block size and in fact as the tests result show it is needed to be implemented to make the result secure and using fix block size as same as char values make it unsecure. But the test results shows that we could generate p and q with 10 digits(32 bits) or even more if you be patient and by increasing the size of the primes(p,q) the cipher values are increased dramatically and make it secure against attacks for example mathematic attacks. So as the results show generally implementation sounds promising and the algorithm has been implemented and works fine. Apart from block size that should be implemented there are some weakness that have been mentioned in design and implementation part like we can use Miller-Rabin algorithm for Primality test or using Extended Euclidean to find MMD.

In application the plaintext should be ASCII and the cipher text is shown in Base64String to be readable and also both plaintext and ciphertext are shown in binary or decimal. In order to use the application first you need to set the prime size (16 bits by default) and then generate keys. You also can see all generated parameters(p,q,n,phi,e and d).In some cases specially when prime size is big or too small it maybe take time to generate parameters. The minimum prime size set to 6 to generate n greater than 256 and support the 8 bits block .You can decrypt the cipher text in main page or by pushing Decryption Form button a Decryption form is shown and by entering the ciphertext and you can see the deciphered text. The private key(n and d) are also shown in the Decryption form.

### References:

1. -William-Stallings “Cryptography and Network Security”.
2. <http://www.webopedia.com/TERM/P/public_key_cryptography.html>
3. <http://www.di-mgt.com.au/cryptopad.html>
4. <http://www.javamex.com/tutorials/cryptography/rsa_key_length.shtml>
5. <http://www.jsums.edu/cms/tues/docs/Cryptography/Module-NumberTheoryBasics.pdf>
6. <http://www.di-mgt.com.au/rsa_alg.html>
7. <http://en.wikipedia.org/wiki/Public-key_cryptography>
8. <http://en.wikipedia.org/wiki/Padding_(cryptography)>
9. <http://en.wikipedia.org/wiki/RSA_Factoring_Challenge>