A Survey on RSA Security Enhancement

Mr. Hardik J Prajapati[1], Prof. Tushar J Raval[2], Prof. Karishma A Chaudhary[3]

[1] *Lecturer ,Information Technology Department, G.P.Ahmedabad, Gujarat, India*
[2] *Associate Professor, Computer Engineering Department, L D College of Engineering, Gujarat, India*
[3] *Assistant Professor, Computer Engineering Department, L D College of Engineering, Gujarat, India*

ABSTRACT

The most common public key algorithm is RSA cryptosystem used for encryption and decryption. It is the first public key algorithm which provides security to transfer and saving of data over the network. In RSA cryptosystem there is less security and time of computation is still lengthy. This paper suggest a new algorithm concept to presents the modified form of RSA algorithm in order to speed up the implementation of RSA algorithm during data exchange across the network. This includes the architectural design and enhanced form of RSA algorithm through the use of third prime number in order to make a modulus n which is not easily decomposable by intruders. A database system is used to store the key parameters of RSA cryptosystem before it starts the algorithm. The proposed RSA method is compared with the original RSA method by some theoretical aspects. Comparative results provide better security with proposed algorithm.

Keyword: - *RSA Algorithm , Encryption , Decryption ,MD5*

1. Public-key cryptosystem and RSA

Public-key cryptography is also known as *asymmetric-key* cryptography; to distinguish it from the *symmetric-key* cryptography we have studied thus far. Encryption and decryption are carried out using two different keys. The two keys in such a key pair are referred to as the public key and the private key. With public key cryptography, all parties interested in secure communications publish their public keys. Party A, if wanting to communicate confidentially with party B, can encrypt a message using B's publicly available key. Such a communication would only be decipherable by B as only B would have access to the corresponding private key. Party A, if wanting to send an authenticated message to party B, would encrypt the message with A's own private key. Since this message would only be decipherable with A's public key, that would establish the authenticity of the message meaning that A was indeed the source of the message. This is illustrated by the middle communication link in Figure 1. The communication link at the bottom of Figure 1 shows how public-key encryption can be used to provide both confidentiality and authentication at the same time. Note again that confidentiality means that we want to protect a message from eavesdroppers and authentication means that the recipient needs a guarantee as to the identity of the sender. A's public and private keys are designated PUA and PRA. B's public and private keys are designated PUB and PRB. let's say that A wants to send a message M to B with both authentication and confidentiality. The processing steps undertaken by A to convert M into its encrypted form C that can be placed on the wire are:

C = E (PUB, E (PRA, M)) where E() stands for encryption.

The processing steps undertaken by B to recover M from C are
M = D(PUA, D(PRB, C)) where D() stands for decryption.

Public key Cryptography is developed to address two key issues:

key distribution – how to have secure communications in general without having to trust a KDC with your key.
digital signatures – how to verify a message comes intact from the claimed sender.

RSA cryptosystem is one of the famous security algorithm which is composed of three phases- key generation, encryption process and decryption process. Let's consider the procedure how keys are generated in RSA cryptosystem

2. Review of RSA Algorithm:

The RSA public key cryptosystem was invented by R. Rivest, A. Shamir and L. Adleman. The RSA cryptosystem is based on the dramatic difference between the ease of finding large primes and the difficulty of factoring the product of two large prime numbers (the integer factorization problem). This section gives a brief overview of the RSA algorithm for encrypting and decrypting messages.

Key generation

For the RSA cryptosystem, we first start off by generating two large prime numbers, 'p' and 'q', of about the same size in bits. Next, compute 'n' where n = pq, and 'x' such that, x = (p -1)(q-1). We select a small odd integer less than x, which is relatively prime to it i.e. gcd(e,x) = 1. Finally we find out the unique multiplicative inverse of e modulo x, and name it 'd'. In other words, ed = 1 (mod x), and of course, 1 < d < x. Now, the public key is the pair (e,n) and the private key is d.

RSA Encryption

Suppose Bob wishes to send a message (say 'm') to Alice. To encrypt the message using the RSA encryption scheme, Bob must obtain Alice's public key pair (e,n). The message to send must now be encrypted using this pair (e,n). However, the message 'm' must be represented as an integer in the interval [0,n-1]. To encrypt it, Bob simply computes the number 'c' where c = m ^ e mod n. Bob sends the ciphertext c to Alice.

RSA Decryption

To decrypt the ciphertext c, Alice needs to use her own private key d (the decryption exponent) and the modulus n. Simply computing the value of c ^ d mod n yields back the decrypted message (m). Any article treating the RSA algorithm in considerable depth proves the correctness of the decryption algorithm. And such texts also offer considerable insights into the various security issues related to the scheme. Our primary focus is on a simple yet flexible implementation of the RSA cryptosystem that may be of practical value.

2.1 Basic Working

RSA cryptosystem is one of the famous security algorithm which is composed of three phases- key generation, encryption process and decryption process. Let's consider the procedure how keys are generated in RSA cryptosystem:

A. Key Generation

(1) Select p and q both prime number, p is not equal to q.
(2) Calculate n = p x q.
(3) Calculate ø (n) = (p -1) x (q-1).
(4) Select integer e whose gcd (Ø (n), e) = 1; 1 < e <Ø (n).
(5) Calculate private key d = e-1 (mod Ø (n)).
(6) Public key PU = {e, n}.
(7) Private Key PR = {d, n}.

B. Encryption Procedure

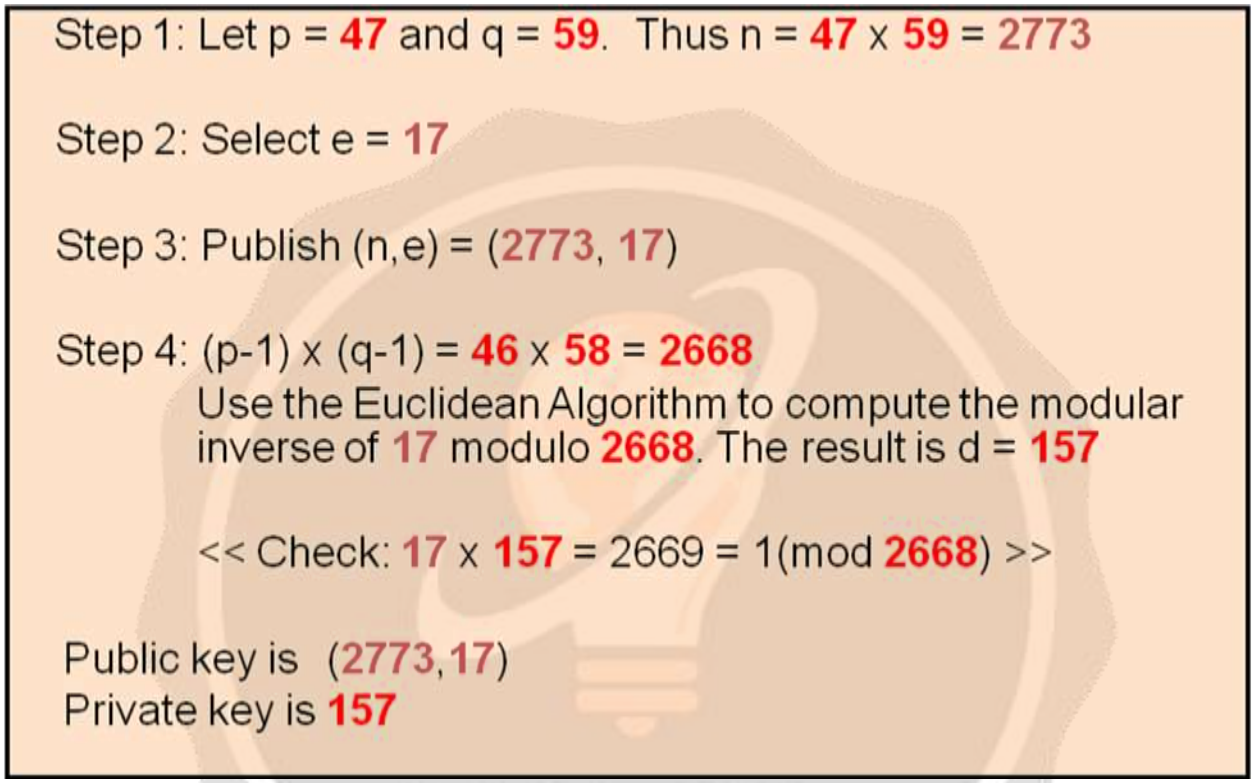
Plaintext- Message (M)
Cipher text- C = M e mod n.

C. Decryption Procedure

Cipher text- C
Plaintext- M = C d mod n.

Where, M is message, p and q are prime numbers, N is common modulus, e and d are public and private keys.

2.2 Example



2.3 Importance of RSA Method

RSA algorithm is one of the famous security cryptosystem based on number theory. RSA method ensures that information is confidential and authenticated thus it provides secure communication over the network. Its security is based on the difficulty in factoring very large numbers. Based on this principle, the RSA encryption uses prime factorization as the trapdoor for encryption. It uses public key encryption in which anyone use public key to encrypt the data and send over the network. It provides authentication and security over the network in order to provide private key to decrypt the information therefore only indented receiver can decrypt the information. RSA algorithm is used for both data encryption and digital signature.

2.4 Limitation in RSA Method

The limitation of using public key cryptography for encryption and decryption is speed. Its computation takes time to compute the mathematical operation of RSA algorithm. Public key used for encryption should be authenticated. If hacker know the factors of a large prime number, then this break the security of algorithm, because the values of public key and private keys are known with the help of factors. Loss of private key may leak the information in the communication network. It provides communication secure but still there are many problem with RSA cryptography which are states below:-

Speed - RSA cryptography takes time to compute its operation for encryption and decryption of data. Therefore its calculations are lengthy and take lot of time. To reduce the complexity of RSA algorithm we need to modify it.

Public Key Must Be authenticated - In RSA cryptography public key is used by the sender to encrypt the message. Thus only authenticated user can participate in encryption procedure.

Computational Cost - RSA algorithm refers to an asymmetric cryptography in which two different keys are used for encryption and decryption, therefore its computational cost is high as compare to symmetric cryptography because in symmetric cryptography same secret key is for both encryption and decryption of message.

Loss Of Private Key May Break The Security – RSA security based on private key. During decryption process private is used for decrypt the message. If any unauthorized person knows the value of private key then whole security of RSA algorithm is break. *Attacks On RSA* - There are various attacks in RSA cryptosystem such as factorization problem, low decryption exponent, common modulus, short message, cyclic attack etc. These attacks can break the security of RSA

3. Basic Survey

Shilpi Gupta and Jaya Sharma proposed a hybrid encryption algorithm based on RSA algorithm and diffie hellman algorithm. They proposed an algorithm by combining the two most popular algorithm RSA algorithm and diffie hellman algorithm in order to achieve higher security. RSA algorithm can be used for both public key encryption and digital signature. Diffie hellman algorithm is used to exchange the secret key between two parties and is also used for providing more secure cipher text. RSA keys were taken as input to diffie hellman algorithm. A GUI developed using java applet provides options to the input user message and to upload file. Thus it provides the better efficiency in terms of time complexity. A limitation of this paper is that the key size of this algorithm is large which is modified further by many authors. Ashutosh Kumar Dubey et. al proposed a novel method called cloud-user security based on RSA and MD5 algorithm for resource attestation and sharing in java environment. A new secure cloud computing environment is established by using both RSA and MD5 algorithm. According to them, cloud-user security method contains two parts. First part is controlled by user which gets permission by the cloud. Second part shows a secure trusted computing for the cloud. If admin want to read and update the data from cloud it take permission from the client environment. In this way it provides a way to hide the data from normal user. Thus it protects their data from cloud provider. When the user uploads the data in the cloud, the data is encrypted by using RSA encryption algorithm. Cloud admin decrypt the data by its private key. If admin wants to update the data it needs secret key provided by a user through message digest tag which is generated by MD5 algorithm. This paper present the two most secure algorithm used for data gathering and data sharing in the cloud computing environment. The limitation of this algorithm is it is helpful for today's requirement. A novel approach is proposed by Wuling Ren and Zhiqian Miao for RSA key generation. A hybrid encryption algorithm is implemented which is based on DES algorithm and RSA algorithm in Bluetooth communication . DES encryption is used for the transmission of data because of its higher efficiency in block encryption. RSA algorithm is used for the encryption of keys of DES algorithm because of its better management of keys. Thus it provides dual protection in Bluetooth communication network. In Bluetooth network there are vulnerable attacks happened, thus DES and RSA hybrid algorithm are more secure and easier to achieve. It provides secure data transmission between the Bluetooth devices. As compared to the fixed Bluetooth network it is more vulnerable to be attacked.

3.1 Problem Definition

RSA algorithm works slowly and provides less security over the network. To increase the speed of computation of RSA algorithm and to increase the security we need to modify the RSA algorithm which can be done by third prime number and offline storage method.

3.2 Proposed Method

In proposed method we developed an algorithm which is based on modified RSA cryptosystem Considering these assumptions for algorithm , q, and r are prime numbers. n is common modulus. e is public key. d is private key. M is message.

RSA Proposed Method

1) Select the random values p, q, and r.
2) Calculate n=p*q*r.
3) Calculate Ø (n) = (p-1) (q-1) (r-1).
4) Calculate e such that gcd (e, Ø(n))=1 and
1<e<Ø(n).
5) Encrypt the message M where M<n and encrypt with public key e such that C=M e mod n.
6) Calculate private key d = e-1 (mod Ø (n)).
7) Decrypt the message M such that M=C d mod n

Steps of Proposed RSA Algorithm-

RSA algorithm refers to public key cryptography method. It is an asymmetric technique, based on two different key pair's public and private keys. In our proposed method we use key indexes instead of actual values of private and public keys at the time of communication between sender and receiver. In a secure communication using public key cryptography (RSA algorithm) following procedure are taken:-
1) Firstly the sender fetches the value of public key indexes from the database.
2) It encrypt the message with public key indexes KU= (e1, n1) and send the message to the receiver.
3) When the message received by the receiver it fetches the value of private key indexes which is placed along with sender public key indexes in the database table.
4) It decrypt the message with (d1, n1).In this way secure communication are provided with this proposed method.

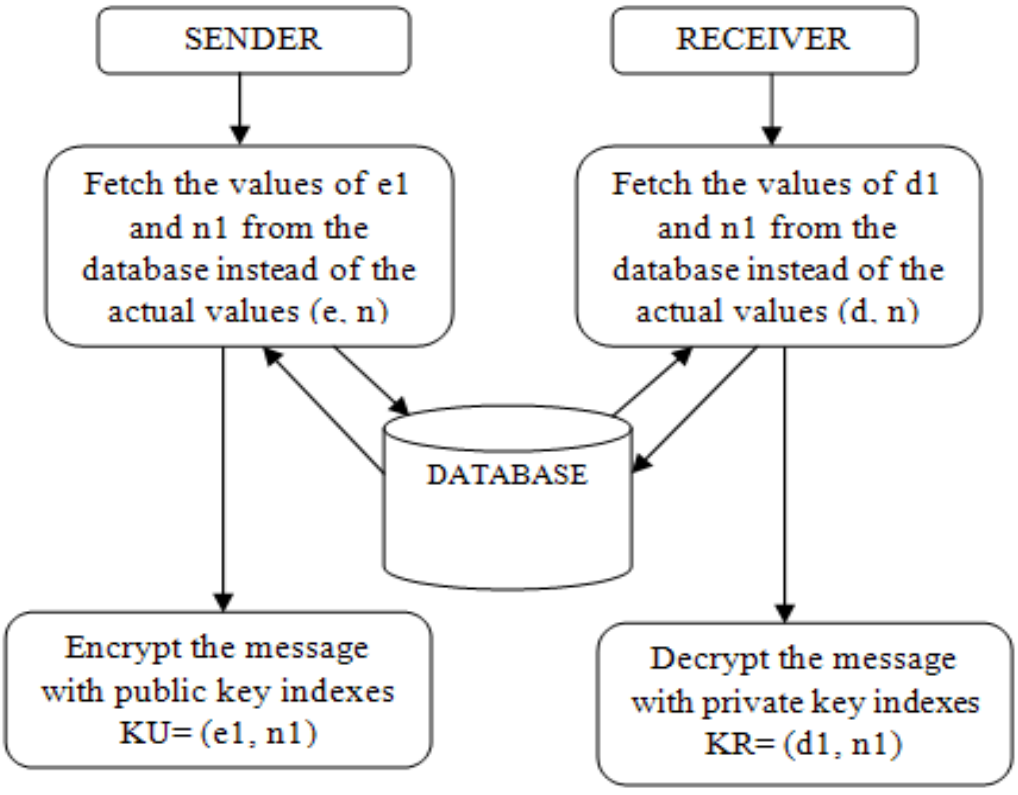


Fig -1: Architecture of Proposed RSA Method Working

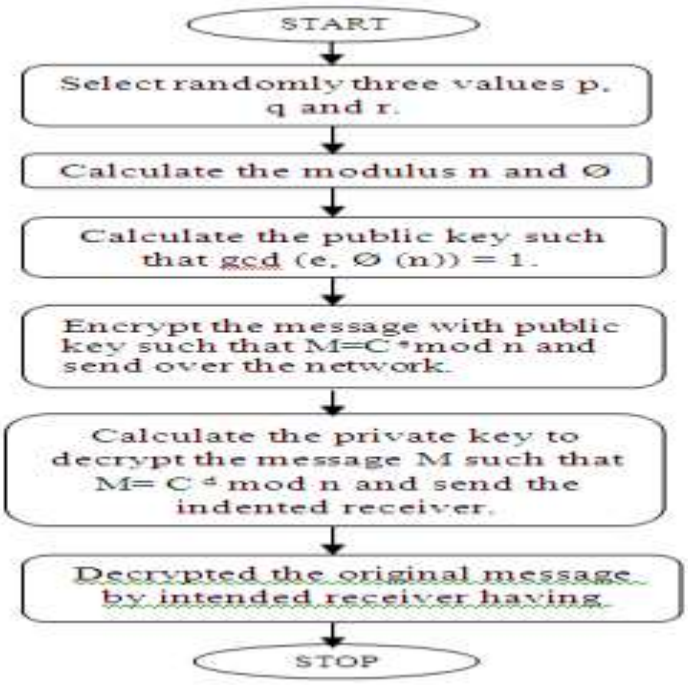


Fig -2: Flowchart of Modified RSA

3.3 COMPARISION ANALYSIS BETWEEN PROPOSED RSA ALGORITHM (PRSA) VS RSA ALGORITHM

Table -1: Comparison Between Proposed RSA Algorithm(PRSA) VS RSA

RSA	Proposed RSA
In this we select p and q to calculate common modulus n.	In this we calculate p ,q and r to calculate common modulus n
The strength of large prime number depend on two variables p and q.	The strength of large prime number depend on three variables p, q and r. It is difficult to break the large prime number into three as compare in existing RSA algorithm.
Here the p, q, e and d are calculated at the time of data transmission.	p, q, d and e are stored in two database tables before algorithm starts.
We use the real values of (e, n) at the time of encryption and (d, n) at the time of decryption the data.	We take the index value correspond the values of e and d from the database table and exchange at the time of encryption and decryption rather than original key (e, d).

4. IMPLEMENTATION METHODOLOGY

4.1 Basic Introduction for Implementation

We have implemented the RSA cryptosystem in two forms : a console mode implementation, as well as a user friendly GUI implementation. We focus on the console mode implementation here, and leave the GUI implementation for a later section of this report. The console application uses a 1024 bit modulus RSA implementation, which is adequate for non-critical applications. By a simple modification of the source code, higher bit-strengths may be easily achieved, albeit with a slight performance hit.

4.2 Handling large integers and the GMP library

Any practical implementation of the RSA cryptosystem would involve working with large integers (in our case, of 1024 bits or more in size). One way of dealing with this requirement would be to write our own library that handles all the required functions. While this would indeed make our application independent of any other third-party library, we refrained from doing so due to mainly two considerations. First, the speed of our implementation would not match the speed of the libraries available for such purposes. Second, it would probably be not as secure as some available open-source libraries are. There were several libraries to consider for our application. We narrowed the choice to three libraries: the BigInteger library (Java), the GNU MP Arbitrary Precision library (C/C++), and the OpenSSL crypto library (C/C++). Of these, the GMP library (i.e. the GNU MP library) seemed to suit our needs the most. The GMP library aims to provide the fastest possible arithmetic for applications that need a higher precision than the ones directly supported under C/C++ highly optimized assembly code. Further the GMP library is a cross-platform library, implying that our application should work across platforms with minimal modifications, provided it is linked with the GMP library for the appropriate platform. We have used the facilities offered by the GMP library heavily throughout our application. The key generation, encryption and decryption routines all use the integer handling functions offered by this library.

4.3 Application overview

In this subsection, we present the basic structure of the console mode RSA implementation. The program is meant for use on a per user basis where each user's home directory stores files containing the private and public keys for the particular user. The application stores the private and public keys for a user in the files $HOME/.rsaprivate and $HOME/.rsapublic respectively.
At the very beginning the program searches for the existence of the aforementioned files and reads in the values of the private and public keys. If they are not present (as when the application is run for the first time), the program proceeds to generate the keys and writes them to the files.
Following this, the user is presented with a menu, asking him whether he would like to encrypt a file, decrypt an encrypted file, or quit the application. If the user chooses to encrypt a file, he is asked to enter the path to the file containing the recipient's public keys as well as the number of characters to encrypt at a time (this is justified later). If decryption is chosen, the path to the encrypted file is requested and the program subsequently decrypts the file to the standard output.

4.4 RSA key generation

The generation of the RSA keys is of paramount importance to the whole application. The application maintains a constant named 'BITSTRENGTH' which is the size of the RSA modulus (n) in bits. According to this setting, two character arrays to contain the digits of the primes p and q are declared. A simple loop through all the digits of this array initializes the array with a random string of bits. We have used C's inbuilt random number generation routines to generate the bits of the string. The random generator is seeded using the srand() routine by the return value of the function time(), which returns the time since the epoch (00:00:00 UTC, January 1, 1970), measured in seconds. Key generation at the same return value of time() is avoided by sleep(), which delays the execution by one second, thus ensuring that the random numbers are never repeated. At the end of this process, we have strings containing binary representations of
the numbers p and q, but they are not prime yet. To achieve that, two gmp integers are first initialized with the contents of these strings. Then, the function mpz_nextprime() is called, which changes p and q to the next possible primes. This function uses a probabilistic algorithm to identify primes. According to the gmp manual, it is adequate

for practical purposes and the chance of a composite passing is extremely small. Now that we have the two 512-bit primes p and q, calculating the values of n (=pq) and x (=(p- 1)*(q-1)) is a simple matter of invoking mpz_mul() with the proper arguments. Next, to determine the value of 'e', we started with a value of 65537, and incremented it by two each time until the condition gcd(e,x) = 1 is satisfied (which, incidentally, is almost always satisfied by the number 65537 itself). Now there exists a procedure in the gmp library with the prototype int mpz_invert(mpz_t ROP, mpz_t OP1, mpz_t OP2) which computes the multiplicative inverse of OP1 modulo OP2 and puts the result in ROP. Using this function helps us avoid writing our own routine based on the Extended Euclidean Algorithm (as this function executes extremely fast). In this way, we obtain the value of d, which completes our quest for the RSA keys. Finally the keys (d,e,n) are stored in two files .rsapublic and .rsaprivate, both in the user's home directory. It is to be noted that the entire application can use a higher (or lower) bitstrength of the RSA modulus n by simply changing the constant BITSTRENGTH at the very beginning of the source-code.

4.5 RSA encryption

As the application is executed, the existence of the key files are checked and if they do not exist, the RSA keys are generated. Following this, the user is presented with a menu, which enables him to choose to encrypt, decrypt or quit. First, the path to the file containing the public key of the recipient is requested from the user. However the user is also given the option of using his own public keys (in which case, only he can decrypt the message). Using the values of e and n read from the relevant file containing the public keys, the message is encrypted. A critical requirement for the proper functioning of the RSA algorithm is that the message m must be represented as an integer in the range [0,n-1] (where n is, as usual, the RSA modulus). Our application converts text messages to integers by using a simple mapping of every character to its ASCII code. But encrypting only one character at a time is not only expensive in terms of the time required to encrypt and decrypt, but also in terms of security. This is because the encrypted integers would then only be from a small finite set (containing a maximum of as many integers as the number of ASCII characters). Hence, we ask the user the number of characters to encrypt at a time. For an RSA encryption scheme with the modulus size of 1024 bits, we have seen that about 100 characters can be encrypted at once. Lesser number of characters cause encryption and
(especially) decryption to take significantly longer, whereas higher number of characters often violate the condition that the message m must lie in the interval [0,n-1]. The entire file to encrypt is processed as a group of strings each containing the specified number of characters (except possibly the last such string). Each character in such a string is converted to its 3- character wide ASCII code, and the entire resulting numeric string is our message m. Encrypting it is achieved by computing m ^ e mod n. There is a gmp routine specifically for such a computation having the prototype void mpz_powm (mpz_t ROP, mpz_t BASE, mpz_t EXP, mpz_t MOD) which sets ROP to (BASE raised to EXP) modulo MOD. Thus invoking moz_powm(c,m,e,n) stores the encrypted partial message in the integer c. This is written each time to the file to encrypt
to until the whole message has been processed. This completes the encryption process.

4.6 RSA decryption

If, in the menu, the user chooses to decrypt an encrypted file, the decryption routine is invoked. The operation of this routine is really quite straightforward. From the file to decrypt (the path to which is input from the user), the function processes each encrypted integer. It does so by computing the value of c ^ d mod n by invoking gmp_powm(m,c,d,n) and stores the decrypted part in m. Of course, the values of d and n are read in beforehand. Here m however contains the integer representation of the message i.e. it is a string of numbers where each 3 character sequence signifies the ASCII code of a particular character. An inverse mapping to the relevant character is carried out and the message, now as was in the original file is displayed on the standard output. The decryption process is over once all the integers (ciphertext) in the encrypted file are processed in the described manner. This completes a discussion of our console mode implementation of the RSA algorithm. This public key infrastructure has been tested in a multi-user scenario successfully.

4.7 GUI Implementation

This section describes the GUI version of our implementation of the RSA algorithm. While the concepts and libraries used are essentially the same, we briefly describe the implementation with special regard to the considerations that were unique to the graphical version. Finally we present the reader some screenshots of the application. The GUI application was developed using the KDE/Qt libraries on Red-Hat Linux version 8.0. We used

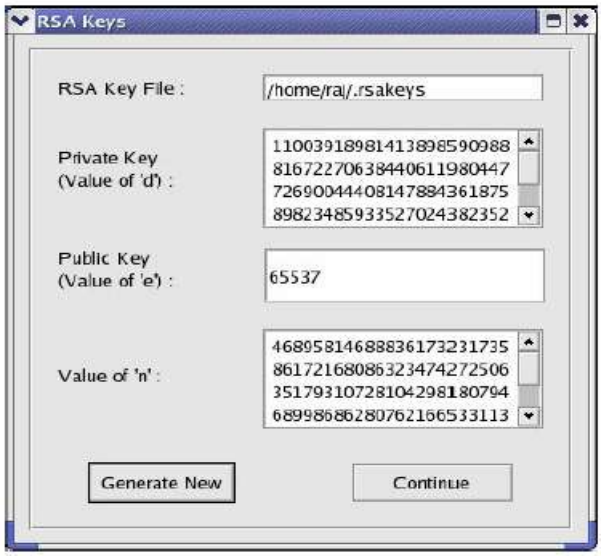
KDevelop 2.1 as our integrated development environment. Our application consists of three C++ classes, of which the class named RSA is the most important. It provides slots (signal handlers) for encrypting files, decrypting files, mailing the encrypted file to another user, loading the values of the RSA keys from the key-files, saving encrypted/decrypted files and so on. One notable difference as far as
features are concerned is that the GUI application does not ask the user the number of characters to enccrypt together. The performance of the GUI version is similar to the console mode, since the underlying algorithms are the same. But as any program running on an X-server does require considerable amount of memory, the speed might be somewhat slower than the console version on a system without adequate memory.

4.8 Screenshots of the application.

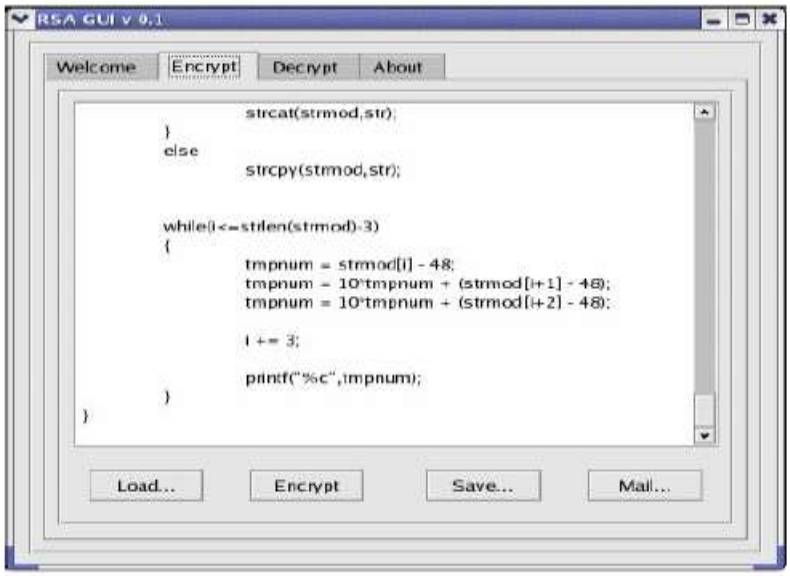
The following screen shows the main window of our RSA GUI:

When the user clicks on the "Check RSA Key Pairs" button, if the key files do not exist (as in the first time), the following message is shown:

The user is taken to the key generation dialog, whose functions are pretty selfexplanatory.
This dialog is shown next:

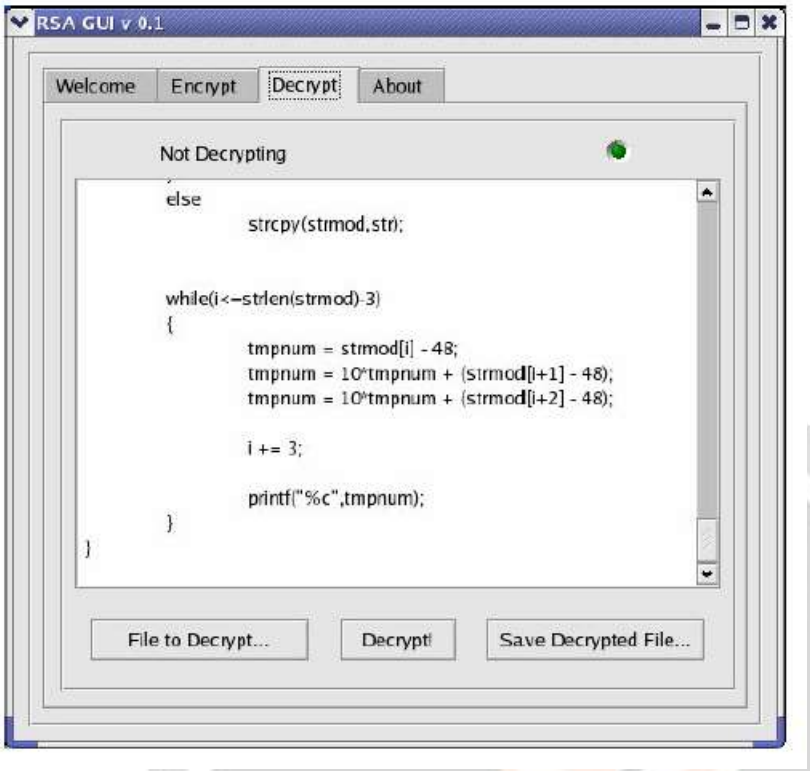


The following diagram shows the "Encrypt" tab of the main window:



The encrypted file can be mailed to another user by clicking on the "Mail…" button:

And finally, here is the similar looking "Decrypt" tab:



The decrypted file may be saved using the "Save Decrypted File…" button, which opens up a file - dialog for saving the decrypted file.

5. CONCLUSIONS

In proposed method keys are stored offline before the process start. Thus, the speed of process increased as compared to original RSA method. If an unauthorized person wants to know the value of p, q and r from the two database tables, it is difficult to guess the value of parameters p, q and r simultaneously from the database. This method will provide more security and it is reliable to use in networks and cloud computing environment. We worked on security and speed by providing offline storage method with the use of three prime numbers instead of two prime numbers as in RSA algorithm. In future some security concepts can be applied in the existing RSA algorithm for providing more efficiency and security.

5. REFERENCES

[1]. Cryptography and Network Security by William Stalling
[2]. http://en.wikipedia.org/wiki/RSA
[3]. Ms. Ritu Patidar and Mrs. Rupali Bhartiya,"Modified RSA Cryptosystem Based on Offline Storage and Prime Number", 2013 IEEE International Conference on Computational Intelligence and Computing Research, 978-1-4799-1597-2/13/$31.00 ©2013 IEEE.
[4]. Chandra segar T and Vijayaragavan R,"Pell's RSA key generation and its security analysis", 2013 IEEE International Conference on Computational Intelligence and Computing Research, IEEE – 31661,4th ICCCNT – 2013 July 4 - 6, 2013, Tiruchengode, India.
[5]. Prof.Dr.Alaa Hussein Al-Hamami and Ibrahem Abdallah Aldariseh, "Enhanced Method for RSA Cryptosystem Algorithm", 2012 International Conference on Advanced Computer Science Applications and Technologies, 978-0-7695-4959-0/13 $25.00 © 2013 IEEE DOI 10.1109/ACSAT.2012.102.
[6]. Xuewen Tan and Yunfei Li,"Parallel Analysis of an Improved RSA Algorithm", 2012 International Conference on Computer Science and Electronics Engineering, DOI 10.1109/ICCSEE.2012.286.

[7].Qing Liu and Yunfei Li,Lin Hao,Hua Peng,"Two Efficient Variants of the RSA Cryptosystem", 20 10 International Conference On Computer Design And Appliations (ICCDA 2010) 978-1-4244-7164-5/$26.00 © 2010 IEEE,Vol-5.