American Journal of Engineering Research (AJER)
e-ISSN: 2320-0847 p-ISSN : 2320-0936
Volume-5, Issue-8, pp-266-275
www.ajer.org
Research Paper Open Access

A Cryptographic Application for Secure Information Transfer in a Linux Network Environment

Asif Karim
University of East London, United Kingdom

ABSTRACT: *Information has always been the raw factor for assessing any given situation and a cardinal element of cementing any decision, of both positive and negative consequences. So the vitality of valid information is of immense significance to respective group or individuals for analyzing the situation and act upon it. The aim of this research was to develop an application that can be deployed among networked PCs and will be able to transfer messages, files and commands among the PCs intended. The sent items will be secured using RSA (Ron Rivest, Adi Shamir, and Leonard Adleman) Encryption algorithm and wrapped in Transport Control Protocol (TCP). The primary goal had been to develop and deploy the application in Linux environment, as there are significant paucity of products in Linux that can provide regular users a secure medium of communication within a single networked environment.*

Keywords: *RSA, Fedora, Linux, Cryptography, Public Private Key, TCP, Anjuta.*

I. INTRODUCTION

One of the crux objectives of this project is to develop a user friendly product in Linux platform that can be used to carry out information transfer in a secure manner. If one looks at Windows platform, the user will see that there are plenty of software which can do that, but for Linux environment, there is basically not much at all. So, keeping in mind the growth of Linux, it is presumable that this is an important project to undertake. In the later section, more light will be shed on this important decision making factor.

The developed program accepts a line of text from user, encrypts the message and then directs the encrypted message to other networked PC, wrapped up in TCP packets. And in the destination, the packets are dissected, the message is disinterred (Stevens, 1993) and the decryption process finally reveals the original line of text, an encrypted reply can also be sent to the sender. Same principle applies for sending files and commands. The program presents a GUI based on GNOME desktop environment of Linux and in fact has been written for Linux boxes. The programming language is ‘C’ as it offers the maximum flexibility regarding Socket programming (Yocom, Turner, Davis, 2004). The user need to provide the target using IP address. The packets will then be sent into the target only. Various outside tools have been applied in order to develop the application. The encryption algorithm as discussed above, is the RSA public key algorithm.

Also to get the corresponding character out from a single integer, representing a block of three characters, an innovative technique has been used which will be construed in an appropriate heading later in this paper.

The overall goal of the project is to demonstrate how RSA algorithm can be used to secure communication channel among various PCs in a networked environment. The transport protocol will primarily be TCP, but user will have the option of choosing UDP as well in the later version of the product. The user interface has been designed and developed using Anjuta, an Integrated Development Environment specifically erected for Linux operating systems. GLADE IDE is the primary building blocks used the GUI development.

II. RESEARCH METHODOLOGY

Research methodology applied for this research were to look for similar journals in the Internet, browse through books especially on cryptology and questionnaire the people involved in the user base. RSA is such a vast topic and could be used for many kinds of application for information encryption purposes. So going through journals and consulting reputed books based upon public key encryption have cemented the theory behind the scene. Questionnaire was also an integral part that had been used to evaluate how the project would fetch in the eyes of the user and how it can be improved.

III. LITERATURE REVIEW

This section will furnish an overview of public key algorithm and will comprehensively discuss the algorithm behind RSA.

3.1) Overview of Cryptography

In this contemporary era of digital supremacy, Information has molded into the ultimate play-card of the highest stratum of many media giants and other exploiters of consumer market. As the theme circles around, "Accurate Information is Knowledge, and Knowledge is Power". So, the safeguard of Information is a concern for many that promotes the growth and culture of dissecting information in order to fortify it, make it intractable for outsiders to abuse it. The study has been termed as Cryptography (Pell, Oliver; http://www.ridex.co.uk/cryptology/).

There are mainly two major types of cryptology branches: Symmetric-key cryptography and Public-key cryptography. In cryptography, a key is a piece of information (a parameter) that controls the operation of a cryptographic algorithm. In encryption, a key specifies the particular transformation of plaintext into ciphertext (The encrypted text), or vice versa during decryption. (http://en.wikipedia.org/wiki/Cryptography)

Symmetric-key cryptography refers to the encryption methods in which both the sender and receiver share the same key (or, less commonly, in which their keys are dissimilar, but related in an easily computable way). This was the sole encryption publicly known until 1976.

Symmetric-key cryptosystems typically puts forward the same key for encryption and decryption. However, the message or group of messages may have a different key than others. A considerable disadvantage of symmetric ciphers is the key management necessary to apply them securely. As the number of keys augments exponentially; that does become an intrusive bottleneck after a certain while. Another problem is that the common key will be known by all whoever wants to decrypt the messages. (Oppliger, Rolf; 2005)

In a groundbreaking 1976 paper, Whitfield Diffie and Martin Hellman proposed the notion of Public-Key cryptography in which two different but mathematically oriented keys are used — a public key and a private key. A public key system is so constructed that calculation of one key (the 'private key') is computationally infeasible from the other (the 'public key'), even though they are necessarily interrelated. Instead, both keys are generated secretly, as an interrelated pair. In public-key cryptosystems, the public key may be freely distributed, while its paired private key must remain covert. The public key is typically used for encryption, while the private key is utilized for decryption. RSA is an example of this kind of cryptography. (Schneier; 1996).

3.2) Principles of Public Key Cryptology

This section illustrates a detail working mechanism of Public Key cryptology and will be further elaborated with example in the following sections attributed to RSA.

Generally **Two** keys are involved:

- Public Key - One of the keys allocated to each person is called the "public key", and is published in an open directory somewhere where anyone can easily look it up, for example by email addresses.
- Private Key - Each person keeps their other key secret, which is then called their "private key". (Stewart, William; http://www.livinginternet.com/i/is_crypt_pkc_work.htm)

As an example, consider John wants to email Mary. If John wants to propel an encrypted email to Mary, he encrypts his message with Mary's public key, and then sends it to her. He need not to worry about interception or eavesdropping since the only person that can decipher the message is Mary herself, because she is the only one that has the corresponding private key that can decrypt it. The advantages are: (Stewart, William; http://www.livinginternet.com/i/is_crypt_pkc_work.htm)

- The sender and the recipient no longer need to meet or use some other potentially insecure method to exchange a common secret key. Since everyone has their own set of keys, then anyone can securely communicate with anyone else by first looking up their public key and using that to encrypt the message.
- The disclosure of a key doesn't compromise all of the communications on a network, since disclosure of public keys is intended, and only messages sent to one person are affected by the disclosure of a private key.

This Encryption method is considered as one of the concrete shielding mechanism contrived till now.

3.3) Computations involved in RSA Algorithm

The algorithm used for RSA involves some rules from number theory. Below is an easy-to-understand illustration: (Litterio, Francis; http://world.std.com/~franl/crypto/rsa-example.html)

Find P and Q, two large (e.g., 1024-bit) prime numbers.

1. Choose E such that E is greater than 1, E is less than PQ, and E and (P-1)(Q-1) are relatively prime, meaning they have no prime factors in common. E does not have to be prime, but it must be odd. (P-1)(Q-1) can't be prime because it is an even number.
2. Compute D such that (DE - 1) is evenly divisible by (P-1)(Q-1). Mathematicians write this as DE = 1 (mod (P-1)(Q-1)), and they call D the multiplicative inverse of E. This is easy to do -- simply find an integer X which causes D = (X(P-1)(Q-1) + 1)/E to be an integer, then use that value of D.
3. The encryption function is C = (T^E) mod PQ, where C is the ciphertext (a positive integer), T is the plaintext (a positive integer), and ^ indicates exponentiation. The message being encrypted, T, must be less than the modulus, PQ.
4. The decryption function is T = (C^D) mod PQ, where C is the ciphertext (a positive integer), T is the plaintext (a positive integer), and ^ indicates exponentiation.

The public key is the pair **(PQ, E)**. The private key is the number **D** (is not revealed). The product PQ is the modulus (often called N in the literature). E is the public exponent. D is the secret exponent.

One can publish ones public key freely, because there are no known easy methods of calculating D, P, or Q given only (PQ, E) (the public key). If P and Q are each 1024 bits long, the sun will burn out before the most powerful computers presently in existence can factor the modulus into P and Q. .

3.4) An Example of RSA Algorithm

Below is a practical look at RSA Algorithm: (Litterio, Francis; http://world.std.com/~franl/crypto/rsa-example.html)

P = 61	# first prime number (destroyed after computing E and D)
Q = 53	# second prime number (destroyed after computing E and D)
PQ = 3233	# modulus (given this to others)
E = 17	# public exponent (given this to others)
D = 2753	# private exponent (kept secret!)

Public key is (E,PQ).
Private key is D.
The encryption function is:

```
encrypt(T) = (T ^ E) mod PQ
           = (T ^ 17) mod 3233
```

The decryption function is:

```
decrypt(C) = (C ^ D) mod PQ
           = (C ^ 2753) mod 3233
```

To encrypt the plaintext value 123, do this:

```
encrypt(123) = (123 ^ 17) mod 3233
             = 337587917446653715596592958817679803 mod 3233
             = 855
```

To decrypt the ciphertext value 855, do this:

```
decrypt(855) = (855 ^ 2753) mod 3233 = 123
```

Under Linux, the utility called “**bc**” can be used to calculate the value of (855 ^ 2753) mod 3233. .

3.5) Security Issues

RSA has been considered as one of the most formidable of encryption techniques for years now. However, as time passes, RSA do became susceptible to well-structured attacks. But its intransigence still couldn’t be fully challenged and conquered. Reasons are stated below: (http://www.woodmann.com/crackz/Tutorials/Rsa.htm)

Let **n** is the modulus equal to **p x q**. d is the exponent which is inverse to **e**. **e** is prime to **(p-1)(q-1)**; **p** and **q** are two large primes around 200 digits each.

When the factorization of modulus **n** is known, that is, when p and q are known, the Euclidean algorithm can be used to quickly find an exponent **d** inverse to e modulo **(p-1)(q-1).** This permits the decryption of messages sent using that individual’s public key.

However no method is known to decrypt messages that are not based on finding a factorization of **n**. The most efficient factorization methods in 1995 required billions of years to factor 400-digit integers. Consequently when **p** and **q** are 200-digit primes, messages encrypted using **n = pq** as the modulus cannot be found in a reasonable time unless the primes p and q are known. This problem commonly knows as Integer Factorization.

Scientists claim that the next generation quantum computing will be sufficient enough to destabilize RSA stronghold with ease. But the development of quantum computers itself is in an embryonic stage and also hasn't been theoretically completely solidified yet. (Aaronson, Scott; http://www.scottaaronson.com/writings/highschool.html ; 2002)

So, this leads to factorizing and finding each Primes, and the estimated completion time is really out of any reasonable attempt. This makes RSA one durable algorithm ever devised till this day.

Below is an example demonstrating the difficulty in breaking through RSA:

The security of RSA relies on the computational intractability of the Integer Factorization Problem (IFP), for which, no efficient (i.e., polynomial-time) algorithm is known. To get an idea how redoubtable the integer factorization is, consider the following 2048 bits (617 digits) composite number, known as RSA-2048 (Yan, S.Y, 2008)
25195908475657893494027183240048398571429282126204032027777137836043662020707595556264018525880784406918290641249515082189298559149176184502808489120072844992687392807287776735971418347270261896375014971824691165077613379859095700097330459748808428401797429100642458691817195118746121515172654632282216869987549182422433637259085141865462043576798423387184774447920739934236584823824281198163815010674810451660377306056201619676256133844143603833904414952634432190114657544454178424020924616515723350778707749817125772467962926386356373289912154831438167899885040445364023527381951378636564391212010397122822120720357.

It is a product of two prime numbers. The basic idea of RSA encryption and decryption is,
Encryption, **C** = **Me** (mod N);
Decryption, **M** = **Cd** (mod N);

Where **N** = **pq** with p and q prime,
M = Plaintext
C = Ciphertext,
e and d encryption exponent and decryption exponent, respectively.

Let, for example, **e** = 65537, **N** be the above mentioned number RSA-2048, and **C** the following number:
21859805614455549302401938962917715975381114472854342292150049925418121103256208767902225983106799128610119089769511935775476540852269795663824292287063708323169440487394769407843277578199861497994206436166946261408885274160021723305205957488066846353603028794423582262770813499706106470077169306460071262980916541699844999292531337428138732590332878186320959546870156074276759915720731486943230589265183618950810376467872168336018311899427370639870779548080069850187887587515053212373800623567195852763946133986860441037844981838391305986458712839620011281598913455842775066742715153760973671204647757116059031684587.

To recover **M** from **C** one requires to find **d**; to find **d** one needs to calculate **Ø(N);** to calculate **Ø(N)** one needs to factor **N**. But unfortunately, factorizing **N** is thorny when **N** is large (in the present case, N is a 2048-bit number, which is far beyond the computing power of any factoring algorithm on any computer at present); no polynomial-time factoring algorithm is known so far. Thus, RSA is secure and **C** is safe since it is difficult to recover **M** from **C** without factoring **N**. This is essentially the whole idea of RSA! One can try to decrypt the above given RSA ciphertext **C** or try to factor the number RSA-2048 in order to get an idea how difficult it is to break RSA or to factor a large number.

IV. MARKET EVALUATION

Under this section, some of the products in the market will be reviewed and will be compared with the product that will be developed. The critical shortage of this kind of application in the field of Linux operating system will also be highlighted.

There has been a huge discrepancy in the availability of general purpose networked cryptographic Software between Windows and Linux platform. For obvious reasons, Windows has a lot more such products while in Linux it is nearly absent. The reason may be the misunderstanding of Open-source concept or the ordeal of development. Furthermore, it is also quite tough to find a product that can do both encryption and transfer of data at the same time, especially in Linux. However, there are some inbuilt facilities in Linux such as **scp**, which can transfer file to a remote PC securely. In the later part of this section a difference between **Encryptor (**the application that has been developed) and **scp** will be drawn up. Another key difference is that in many cases, Windows based products are known for its easy user interface but in Linux the UI is not always that intuitive. Our product also addresses this issue and thus the User Interface design will be kept to the most simple of its kind. (Levelsqe, Michelle; Fundamental Issues with open source Software development).

4.1) DeepCoder (Windows)

DeepCoder is a program to encrypt, decrypt and/or digitally sign e-mails and files. It is a relatively easy to use program. It is mainly Windows and supports Secret Key Encryption as well as public Key encryption.

One of the biggest shortcoming of DeepCpder is that it is *not* networked and it can't carry over commands from user to be executed on the remote PC, though it has a variety of encryption algorithm to choose from, but it is only limited into a single PC and in incapable of carrying out command based tasks.

4.2) Primasoft (Windows)

It is a software that can be used to encrypt files and folders. It uses both RSA and AES. Its user interface is quite simple and it has the ability to Zip the file after encryption.

The core difference between Primasoft and Encryptor is that this software is only able to encrypt/decrypt files of folders, not instant messages nor commands. Furthermore, like the previous one, it is also not networked thus cannot be used in a networked environment. But it is quite easy to use.

4.3) SecureCRT (Windows)

SecureCRT combines strong terminal emulation with the strong encryption, broad range of authentication options, and data integrity of the Secure Shell protocol. It has different convenient feature such as Menu tabs, Button bar, Easy management of environment variables etc. It can do all kind of terminal related tasks with encryption and can communicate in a networked environment. It is based on OpenSSh. From File Transfer to Command Transfer all can be done in an encrypted environment.

This software is really useful as it is almost capable of doing everything that our product supposed to do, except it has a lot more feature and thus may sometimes confuse the user, however, one function missing from it is the capacity of one to one instant message transfer in an encrypted form.

4.4) PuTTY (Linux)

It is a free software SSH, Telnet, rlogin, and raw TCP client. It was originally available only for Windows, but now recently Linux version is also available.

This is the only attractive and notable networking software in Linux that closely resembles Encryptor. It offers both File Transmission and Remote Command Execution, however, it doesn't offer any one to one message encryption and transfer instantly. It is the highest resembling open source product available.

V. LIFE CYCLE OF EACH OF THE MODULES OF ENCRYPTOR

A Statechart diagram (STD) describes the possible states of a single object and the events that cause state transitions. They are useful for showing the life cycle of the module (Fowler, 2003).

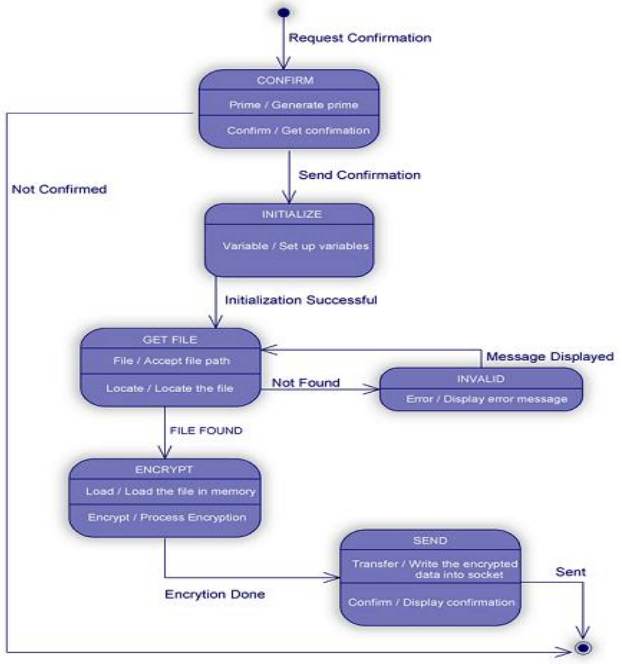

Request Confirmation
CONFIRM
Prime / Generate prime
Confirm / Get confirmation
Not Confirmed
Send Confirmation
INITIALIZE
Variable / Set up variables
Initialization Successful
GET FILE
File / Accept file path
Locate / Locate the file
Message Displayed
Not Found
INVALID
Error / Display error message
FILE FOUND
ENCRYPT
Load / Load the file in memory
Encrypt / Process Encryption
SEND
Transfer / Write the encrypted data into socket
Confirm / Display confirmation
Sent
Encrytion Done

Figure 1: Statechart diagram of File Encryption module

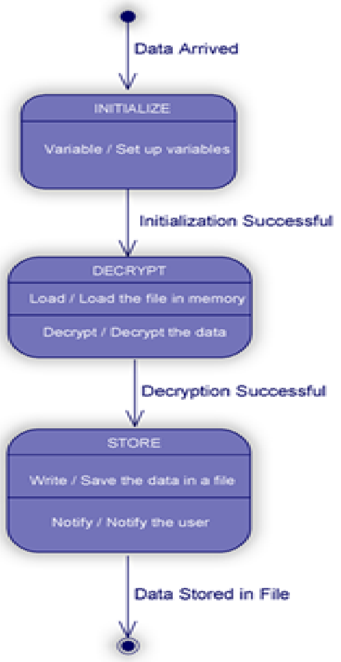

Data Arrived
INITIALIZE
Variable / Set up variables
Initialization Successful
DECRYPT
Load / Load the file in memory
Decrypt / Decrypt the data
Decryption Successful
STORE
Write / Save the data in a file
Notify / Notify the user
Data Stored in File

Figure 2: Statechart diagram of File Decryption module

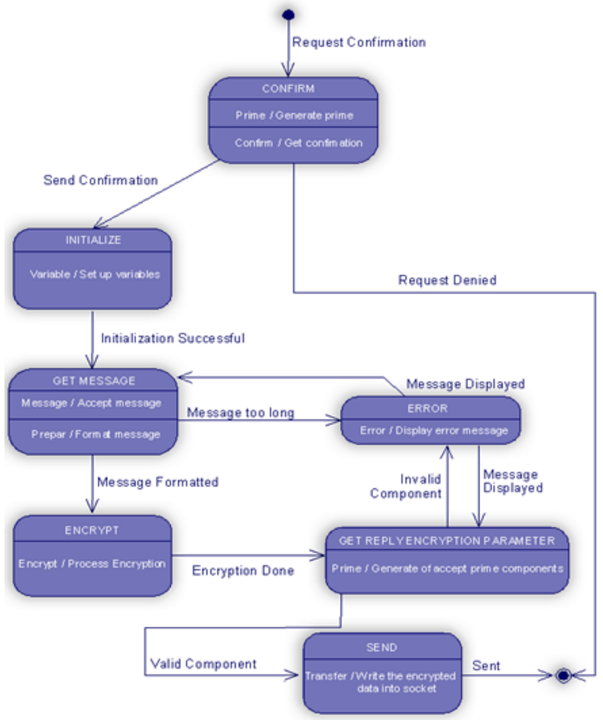


Figure 3: Statechart diagram of Message Encryption module

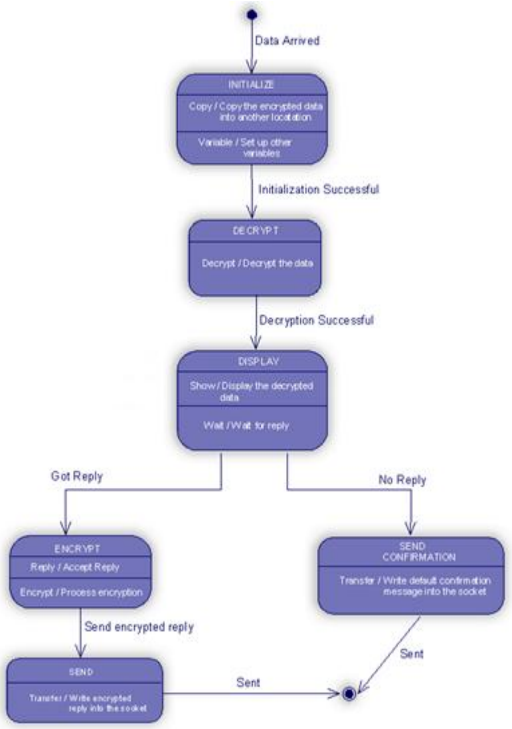

Data Arrived
INITIALIZE
Copy / Copy the encrypted data into another location
Variable / Set up other variables
Initialization Successful
DECRYPT
Decrypt / Decrypt the data
Decryption Successful
DISPLAY
Show / Display the decrypted data
Wait / Wait for reply
Got Reply
No Reply
ENCRYPT
Reply / Accept Reply
Encrypt / Process encryption
SEND CONFIRMATION
Transfer / Write default confirmation message into the socket
Send encrypted reply
Sent
SEND
Transfer / Write encrypted reply into the socket
Sent

Figure 4: Statechart diagram of Message Decryption module

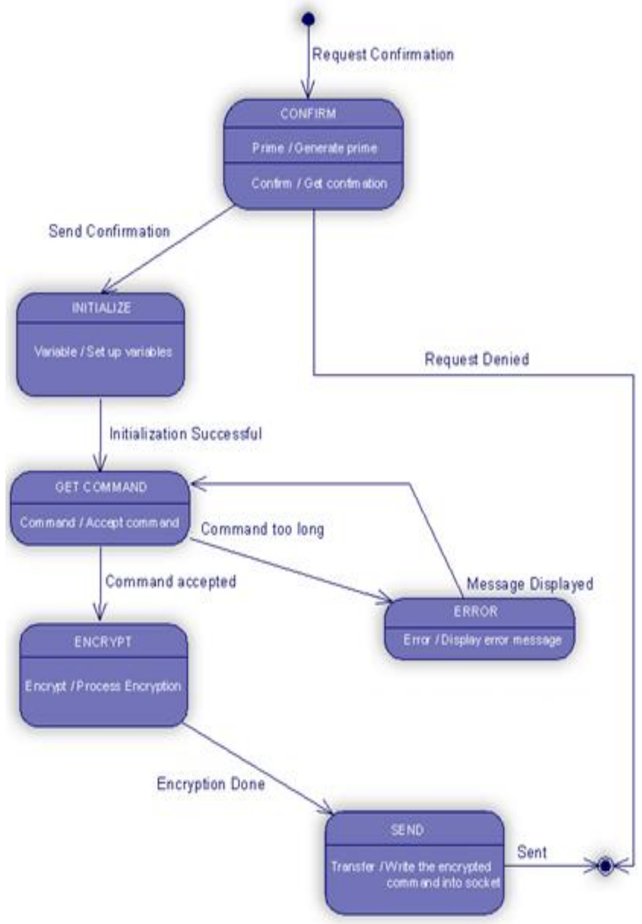


Figure 5: Statechart diagram of Command Encryption module

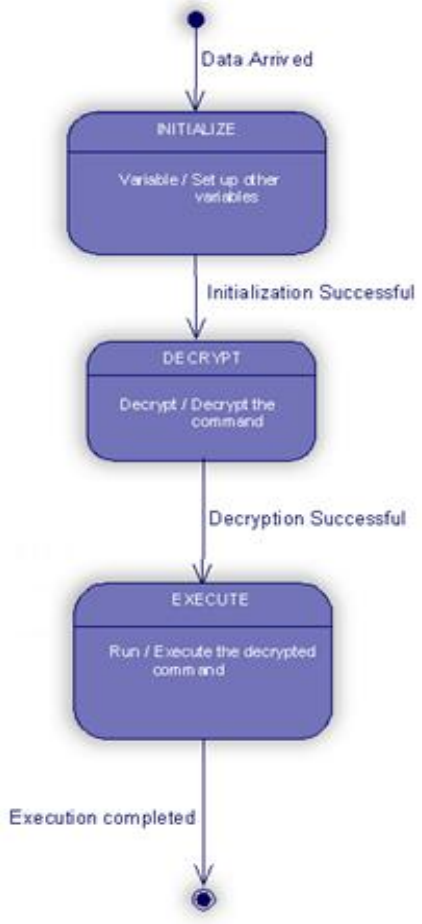

Data Arrived
INITIALIZE
Variable / Set up other variables
Initialization Successful
DECRYPT
Decrypt / Decrypt the command
Decryption Successful
EXECUTE
Run / Execute the decrypted command
Execution completed

Figure 6: Statechart diagram of Command Decryption module

VI. DEVELOPMENT TOOLS

This section apart from default C compiler, will outline major tools that have been used to build the overall product. The platform was Linux Fedora 8 and the product, (Encryptor) should work in all versions of Fedora and other Linux Distributions.

Glade:

One of the primary reason to build this project is to enable the features in a GUI mode for ease of use. And for this regard Glade came into consideration. Glade is a program designed to enable the efficient building of graphical user interfaces for Linux based applications. As long as the gtk+ and/or gnome libraries (libraries for programming GUI) are installed, Glade should work fine (Krause, 2007).

GMP:

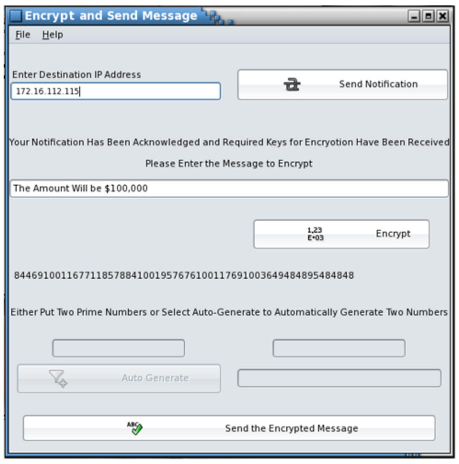
As the calculation part can get real convoluted, some extra packages must be used. The gmp package contains GNU MP, a library for arbitrary precision arithmetic which operates on signed integers, rational numbers and floating point numbers. GNU MP is designed for speed, for both small and very large operands. it generally emphasizes speed over simplicity/elegance in its operations.

VII. RESULTS

This section will carefully outline the testing steps that have been taken into consideration and will also report the result.

7.1) Message Transfer

The *Test Case* would be "The amount will be $100,000". The user shall enter the destination IP Address, and after sending a notification to the destination, shall receive the keys for message encryption. User then enters the desired message, encrypts it, and send it to the destination as shown in Figure 7. Once the Encrypt button is pressed, the encrypted form of the message is shown (in Figure 7, it is 844691……)


Encrypt and Send Message

File Help

Enter Destination IP Address: 172.16.112.115

Send Notification

Your Notification Has Been Acknowledged and Required Keys for Encryotion Have Been Received

Please Enter the Message to Encrypt

The Amount Will be $100,000

Encrypt

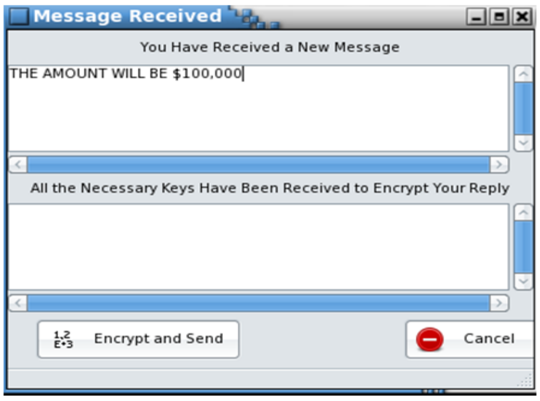
```
8446910011677118578841001957676100117691003649484895484848
```

Either Put Two Prime Numbers or Select Auto-Generate to Automatically Generate Two Numbers

Auto Generate

Send the Encrypted Message

Figure 7: Encrypt and transfer of message in other networked PC


Message Received

You Have Received a New Message

THE AMOUNT WILL BE $100,000

All the Necessary Keys Have Been Received to Encrypt Your Reply

Encrypt and Send

Cancel

Figure 8: Received message decrypted at the destination networked PC, receiver can then may choose to send an encrypted reply to the sender.

The module has been found to be working properly, infact the reply of the received message also works according to the specification.

7.2) Secure File Transfer

Any text file can be encrypted and sent securely using the following interface:

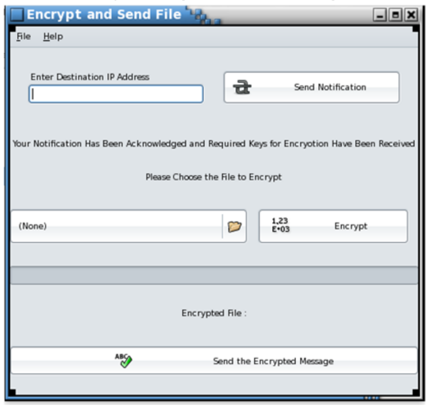
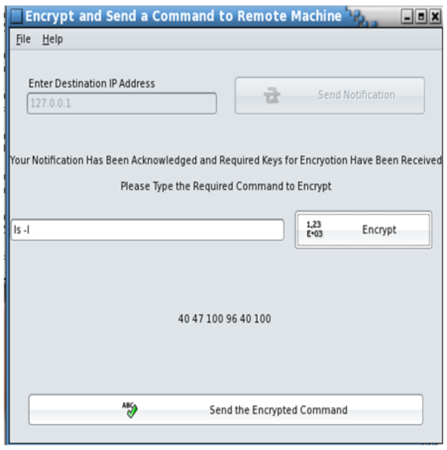

Encrypt and Send File
File Help
Enter Destination IP Address
Send Notification
Your Notification Has Been Acknowledged and Required Keys for Encryption Have Been Received
Please Choose the File to Encrypt
(None)
1.23 E+03
Encrypt
Encrypted File :
Send the Encrypted Message

Figure 9: Encrypted File Transfer facility

This module has also been found to be operating according to the specification.

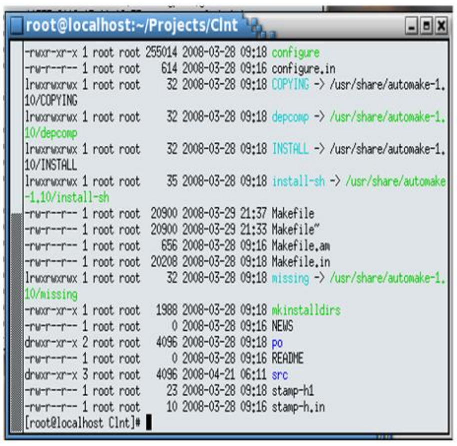
7.3) Command Transfer and Execution at the Destination

This last segment supposed to encrypt a command and then transfer this encrypted command to the destination, decrypt it there and execute at that system. We will try sending ***"ls –l"***, and observe how the system handles the operation.


Encrypt and Send a Command to Remote Machine
File Help
Enter Destination IP Address
Send Notification
Your Notification Has Been Acknowledged and Required Keys for Encryption Have Been Received
Please Type the Required Command to Encrypt
1.23 E+03
Encrypt
Send the Encrypted Command

```
127.0.0.1
ls -l
40 47 100 96 40 100
```

Figure 10: Encrypted Transfer of Linux Commands



```
root@localhost:~/Projects/Clnt
-rwxr-xr-x 1 root root 255014 2008-03-28 09:18 configure
-rw-r--r-- 1 root root    614 2008-03-28 09:16 configure.in
lrwxrwxrwx 1 root root     32 2008-03-28 09:18 COPYING -> /usr/share/automake-1.
10/COPYING
lrwxrwxrwx 1 root root     32 2008-03-28 09:18 depcomp -> /usr/share/automake-1.
10/depcomp
lrwxrwxrwx 1 root root     32 2008-03-28 09:18 INSTALL -> /usr/share/automake-1.
10/INSTALL
lrwxrwxrwx 1 root root     35 2008-03-28 09:18 install-sh -> /usr/share/automake
-1.10/install-sh
-rw-r--r-- 1 root root  20900 2008-03-29 21:37 Makefile
-rw-r--r-- 1 root root  20900 2008-03-29 21:33 Makefile~
-rw-r--r-- 1 root root    656 2008-03-28 09:16 Makefile.am
-rw-r--r-- 1 root root  20208 2008-03-28 09:18 Makefile.in
lrwxrwxrwx 1 root root     32 2008-03-28 09:18 missing -> /usr/share/automake-1.
10/missing
-rwxr-xr-x 1 root root   1988 2008-03-28 09:18 mkinstalldirs
-rw-r--r-- 1 root root      0 2008-03-28 09:16 NEWS
drwxr-xr-x 2 root root   4096 2008-03-28 09:18 po
-rw-r--r-- 1 root root      0 2008-03-28 09:16 README
drwxr-xr-x 3 root root   4096 2008-04-21 06:11 src
-rw-r--r-- 1 root root     23 2008-03-28 09:18 stamp-h1
-rw-r--r-- 1 root root     10 2008-03-28 09:16 stamp-h.in
[root@localhost Clnt]# █
```

Figure 11: Command has been decrypted and executed at the destination

This module has also been found to be operating according to the specification.
Below is the initial startup screen:

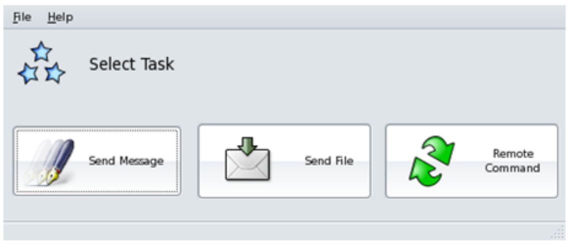


Figure 12: Opening Screen

VIII. DEPLOYMENT

To run the product, user can simply activate a shell script which will in turn auto-run all the commands to startup the program. Same for both the source and destination PC. The shell scripts will be shipped with the program itself. The complete program with instructions can be obtained from *http://tinyurl.com/Download-Encryptor*.

IX. CONVERSION OF RAW DATA INTO DIGITS FOR THE PURPOSE OF ENCRYPTION

The software accepts a line of text at a time and encrypts the whole of it before sending to the other end over network using TCP packets. This section describes the procedure by which individual characters are extracted out and are made encryption-able.

The line of text will be broken down into a block of three characters each, and if the last block contains less than three characters, it will be padded up with ‘space’ character. Then the individual blocks will be converted to their ASCII value, which will then be reduced by 50 (excluding the ‘space’ character). This needs to be reduced because for the ease of calculation and to keep the third principle of section 3.3 intact.

After the encryption the decryption formula will be executed at the other end point to decipher back the message. The following coding realizes the aforementioned illustration:

```
int main (void)
{
   int len;

   register int k , y , i , o ;

   char w[80] = {'\0'};
   char w1[4];

   char ** x = NULL;
   int ** xi = NULL;

   i = k = y = o = 0;
   strupr(gets(w));

   len = strlen(w);

   if (len % 3) ((len % 3) == 1) ? strcat(w,"  ") : strcat(w," ");
   len = strlen(w);

   x = (char **)malloc(len / 3 * sizeof (* x));
   for (; i < len ; i += 3 , k++)
      x[k] = (char *)calloc(sizeof (char) * 4,'\0');

   k = i = 0;
   xi = (int **)malloc(len / 3 * sizeof (* xi));

   for (; i < len ; i += 3 , k++)
      xi[k] = (int *)calloc(sizeof (int) * 3,'\0');
```

```
    k = i = 0;
    do{
        y = i + 3;

        for (o=0 ; i < y; i++)
            w1[o++] = w[i];

        w1[3] = '\0';
        strncpy(x[k],w1,4);

    }while( k++ < ( len / 3 ) - 1 );

K = 0; y = 0;
do{
    for (i=0;i<3;i++){
    y = toascii(*((x[k] + i))) ;
    if  (y == 0x20) y -= 50;
    *((xi[k] + i)) = y;
    }
}while (k++ < len / 3 - 1);
  y = 0;
    while (free(x[y++]) , y < len / 3);

    free(x);
    return (0);
}
```

So, in the above program, the Arrays pointed to by ***xi**, will be holding the each blocks of three characters as numbers, and then this block will be sent as encrypted text.

X. CONCLUSION AND FUTURE ASPIRATIONS

This paper sheds light on one of the most important public key encryption of modern era. The product this research paper describes can be extremely useful in an intranet and can be securely used. Encryptor is an important steps in my view as it addresses a key shortage area of products under Linux environment. As with any software products, continuous improvement is a cardinal aspect and as such adding the capability to recognize destination through Hostname, encrypting and sending multiple files in one go, addition of log-in feature and capacity to use encrypted voice chat are some of the future options on hand.

ACKNOWLEDGEMENT

Major appreciation goes for the group of participants for partaking in the requirement analysis phase through interview and questionnaire, their valuable remarks and suggestions have been quite helpful. Further, the test and deployment team also deserves credit for their meticulous efforts.

REFERENCES

[1]. Pell, Oliver (2007). Cryptology. Ridex. Retrieve on 22 September 2007 at http://www.ridex.co.uk/cryptology/
[2]. Rolf, Oppliger (2005). Contemporary Cryptography, First Edition, Artech House Publishers, pp. 45-79
[3]. Stewart,William(2008). Public Key Cryptography. LivingInternet. Retrieve on 3 January 2008 at http://www.livinginternet.com/i/is_crypt_pkc_work.htm
[4]. Stewart,William(2008). Public Key Cryptography. LivingInternet. Retrieve on 3 January 2008 at http://www.livinginternet.com/i/is_crypt_pkc_inv.htm
[5]. Litterio, Francis (n.d.). RSA. STD. Retrieve on 3 February 2008 at http://world.std.com/~franl/crypto/rsa-example.html
[6]. Schneier, Bruce (1996). Applied Cryptography: Protocols, Algorithms, and Source Code in C, Second Edition, Wiley, pp. 461-466
[7]. Aaronson, Scott (2002). Quantum Computing. Scottaaronson. Retrieve on 25 November 2007 at http://www.scottaaronson.com/writings/highschool.html
[8]. Yan, S.Y. (2008). Cryptanalytic attack on RSA, First Edition, Springer, pp. 55-88
[9]. Krause, Andrew (2007). Foundations of GTK+ Development, Third Edition (Corrected), Apress, pp. 15-156, 229-330
[10]. Stevens, Richard (1993). TCP/IP Illustrated, Volume 1: The Protocols, First Edition, Addison-Wesley, pp. 595-682
[11]. Levelsqe, Michelle (2004). Fundamental Issues with open source Software development. FirstMonday. Retrieve on 25 February 2008 at www.firstmonday.org/issue/issue9_4/levesque/
[12]. N. Yocom, W. Turner, K. Davis (2004). The Definitive Guide to Linux Network Programming, First Edition, Apress, pp. 41-173
[13]. Gay, Warren (2000). Linux Socket Programming by Example, First Edition, Que Publishing, pp. 53-190, 203-395
[14]. Fowler, Martin (2003). UML Distilled: A Brief Guide to the Standard Object Modeling Language, Third Edition, Addison-Wesley, pp. 99-115