RSA ALGORITHM PERFORMANCE IN SHORT MESSAGING SYSTEM EXCHANGE ENVIRONMENT

Hatim Mohamad Tahir[1], Tamer N. N. Madi, Mohd Zabidin Husin”[2], Nurnasran Puteh[3]

[1,2,3]*University Utara Malaysia, {hatim, zabidin,nasran}@uum.edu.my*

ABSTRACT. Short Message Service (SMS) is a widely service for brief communication. With the rise of mobile usage it has become a popular tool for transmitting sensitive information. This sensitive information should be totally secure and reliable to exchange. This urgent need for secure SMS, led to drive for RSA implementation, which is considered one of the strongest algorithms in security since we are going to bring big security into small device. Our main goal in this project is to design an experimental test-bed application in order to use this application in evaluating the performance of RSA. This report explains and documents the process of implementing an RSA in Experimental SMS Exchange Environment using J2ME language which is available in several mobile devices on the market today.

Keywords: short message service (SMS), RSA algorithm, J2ME

INTRODUCTION

Most mobile operators encrypt all mobile communication data, including SMS messages but sometimes this is not the case. Even when encrypted, the data is readable for the operator. Although Global System for Mobile communications (GSM) traffic is usually encrypted, there is little or no security in some cases where the device is lost, stolen or otherwise accessed by an adversary. Among others these needs give rise for the need to develop additional encryption for SMS messages so that only accredited parties are able to engage communication (Hassinen 2003)(Peersman 2000).

Our approach to this problem is to develop a secure application that can be used in mobile devices to encrypt messages that are about to be sent. Naturally decryption for encrypted messages is also provided. The encryption and decryption are characterized by secret keys that all legal parties have to process. This application will be use in testing the performance of RSA algorithm in SMS exchange environment(Ratshinanga 2004). Several mobile device manufacturers have adopted Java as their platform offered for software developers. To certain extent Java applications are portable between devices of different vendors [3,4]. Some mobile device manufacturers provide an application programming interface (API) for SMS services. These facts make Java a natural choice for our application.

There are some security aspects related to secure SMS such as confidentiality, integrity and availability. By default, there is no encryption applied for SMS messages during transmission. Cyclic redundancy check is provided for SMS information passing across the signaling channel to ensure that the short message does not get corrupted. Forward error protection is incorporated using conventional encoding. Short message is assigned a lifetime or validity period. Failure to deliver the message within the period causes it to be marked for purge. Some applications would allow secondary action to be taken when the lifetime expires (Stallings 2006)(Hwu 2006).

RELATED WORKS

RSA is an algorithm for public-key cryptography (Stallings 2006). It was the first algorithm known to be suitable for signing as well as encryption, and one of the first great advances in public key cryptography. RSA is widely used in electronic commerce protocols, and is believed to be secure given sufficiently long keys and the use of up-to-date implementations (Ratshinanga 2005). Figure 1 demonstrate the main three processes and their steps in RSA: key generation, encryption and decryption (Wagner 2001).

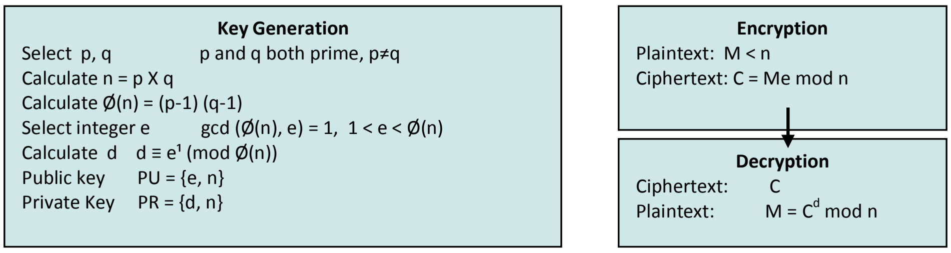

Key Generation
Select p, q   p and q both prime, p≠q
Calculate n = p X q
Calculate Ø(n) = (p-1) (q-1)
Select integer e   gcd (Ø(n), e) = 1, 1 < e < Ø(n)
Calculate d   d ≡ e¹ (mod Ø(n))
Public key   PU = {e, n}
Private Key   PR = {d, n}
Encryption
Plaintext: M < n
Ciphertext: C = Me mod n
Decryption
Ciphertext: C
Plaintext: M = C^d mod n

Figure 1. RSA Algorithm Mechanisms

Java 2 Micro Edition (J2ME) is a runtime environment designed for devices with very limited resources such as mobile phones or handheld computers. J2ME is comprised of CLDC (Connected Limited Device Configuration) and Mobile Information Device Profile (MIDP)(Harkey 2002). A program developed for J2ME is called a MIDlet. MIDlets use classes defined in Application programming interface (APIs) of CLDC and MIDP. There is no straight interaction between a MIDlet and the device itself, since MIDlets are run by the Java virtual machine (JVM) (Helal 2002)(Kolsi 2004). The architecture of J2ME is depicted in Figure 2. This architecture limits the functionality a MIDlet (Liu 2003) can have into those provided by the runtime environment.

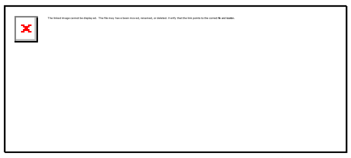


Figure 2. J2ME Architecture Structure

J2ME do not have the crypto API of J2ME. Bouncy castle has Java implementations of cryptographic algorithms. Bouncy castle also has a package designed for J2ME (Piroumian 2002). So it can be used in this application because it has many cryptographic algorithms but the total size of an application will be very big. By Obfuscation way, Java applications are compiled into byte code and can be decompiled into Java source. Obfuscation "scrambles" the source code so that it is more difficult to decompile (Chun 1999). In obfuscation, classes and variables are renamed (a, b, c, d ...). Unnecessary classes are removed eg. Bouncy castle and the size of the MIDlet application decreases. Therefore, the total size of the application will be small and compatible with the memory of mobile device (Lindquist 2004)(Chat 2003).

According to (Hassinen 2003) in their paper an application for sending encrypted SMS messages using cryptographic methods based on theory of quasigroups is proposed. The encryption algorithm is characterized by a secret key. The research on cryptographic strength of quasigroup encryption is still in early stages. The cryptosystem has not yet undergone much scrutiny from the cryptographic community. Several widely used cryptosystems today are conjectured to be safe. Hence, after extensive study by cryptographers they seem to be safe. This research has not yet been done to satisfactory extent on quasigroups. The application design itself doesn't restrict using any suitable encryption algorithm. Quasigroup encryption seems to be well suited for applications such as SMS encryption. The algorithm is compact and needs quite a small amount of memory which is an important aspect on mobile devices.

Another study was done by (Hassinen 2005). It showed how to send, receive, and store text messages securely with a mobile phone without any additional hardware. It also shows how to authenticate the sender of a message and how to ensure that the message has not been tampered with. The choice of Blowfish and Quasigroup encryption methods was motivated by his research interest (Hassinen 2005). His goal was not to find the fastest or the most secure algorithm, since additional algorithms can be implemented, if necessary, later on. One topic for future research is to implement and test other possibly suitable algorithms. According to our knowledge, several papers handle usage of SMS messages in different applications for industry, health care and personal communication but none of the articles address the security issues.

We can say that our research will be adding an improved application in SMS exchange field using the RSA algorithm knowing that most previous studies of this field conducted using symmetric algorithms and not asymmetric algorithm such as RSA. (Hassinen 2005) mentioned that there is no paper talking about the security issues. This was an incentive for us to develop a new application for SMS security as well as contribute on this field.

METHODOLOGY

The methodology adopted from that the development process of this application is an experimental process. The methodology basically consists of five phases; i) Preparing test bed ii) Coding iii) Compiling and Running iv) Testing and v) Verifying and Validating. In the preparing test-bed phase, the blueprint of the test-bed is devised. This is followed by the coding phase whereby the Java files and classes are coded. When the classes are ready we will compile it in the compiling and running phase. Then we run our new application through the emulator. Upon that, the application now is ready to test its performance. Finally, in the verifying and validating phase, where at this phase all the necessary configurations has been done by checking if the application fulfills the requirements which we need or not.

There are three main classes in this application with adding to another helping classes. The three main classes are: Send, Receive and RSAEncrypt. By using Send and Receive classes we can send SMS between sender and receiver without encryption. For encrypting the SMS we included the third class to the code which is RSAEncrypt. We can compile the classes by going to the next phase which is the compiling and running.

Java Application Descriptor (JAD) file will also be created. After the JAD has been created we can adjust the setting. This setting is also the same settings of the application. Figure 3. explain these settings of JAD file.

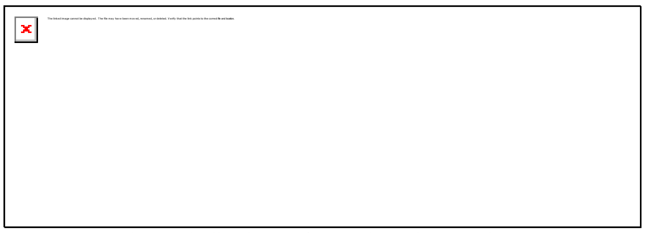

The linked image cannot be displayed. The file may have been moved, renamed, or deleted. Verify that the link points to the correct file and location.

Figure 3. Settings of the JAD File (Project)

After adjusting the settings of the application, it now becomes ready to compile and run. The three main processes which are to enable bouncy castle and obfuscation, code compiling and application. We should notice that Sun's obfuscator has problems with projects that have more than 26 classes, so the following changes are necessary to ktool.properties file which is exist in this path: C:\WTK22\wtklib\Windows. Figure 4 illustrate the changes needed to enable obfuscation:

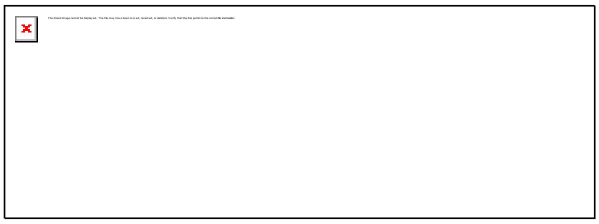

The linked image cannot be displayed. The file may have been moved, renamed, or deleted. Verify that the link points to the correct file and location.

Figure 4. Changes needed to Enable Obfuscation

We compile the Java code classes by using J2ME Wireless Toolkit (WTK). There is a button which is called Build to compile and pre-verified the code. Figure 5 explain the output of code compiling process in the normal situation:

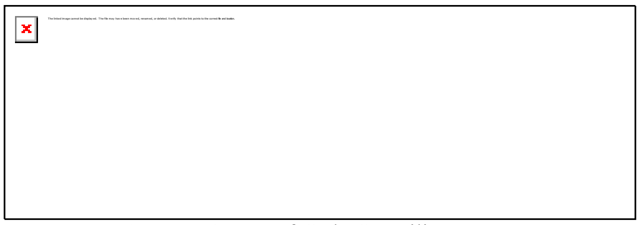

The linked image cannot be displayed. The file may have been moved, renamed, or deleted. Verify that the link points to the correct file and location.

Figure 5. Output of Code Compiling Process

After successfully compiling, we need to run the application on the emulator device. Upon successfully running the emulator, we can launch our application to use it in sending and receiving encrypted SMS. Figure 6 illustrate how the application works:

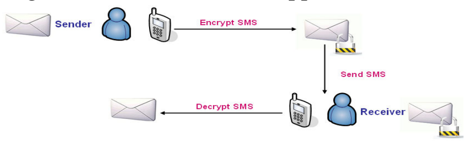


Figure 6. How MIDlet Works

ANALYSIS AND FINDINGS

We measure the effectiveness and the performance of RSA algorithm in the experimental SMS exchange environment. Our approach is to test Execution Time and Memory Usage. J2ME Wireless Toolkit (WTK) program support performance testing of the MIDlet application. We verify and validate our application by checking if the application fulfills the requirements.

The objective of our testing is to measure the efficiency and speed of RSA algorithm by using Application Performance Testing. Application Performance Testing (RSA Testing) is the process of verifying that an implementation performs in accordance with a particular standard, specification, and environment. Execution Time, Memory Usage and Network Traffic are tested in order to find advantages of RSA. It is not intended to be exhaustive and successfully passed test suite does not imply a 100 percent guarantee of RSA. However, it does insure with a reasonable degree of confidence that the RSA is consistent with its strength and speed.

We can examine the method execution time with the Profiler utility. The Profiler collects data from an emulator during runtime. By seeing how much time the method of RSA or any another methods take to execute, we can see what potential problems might exist in the application.

Profiling Data Display

In the Call Graph tree, we see folders for top-level methods. Opening a method's folder displays the methods called by it. By selecting a method in the tree shows the profiling information for it and all the methods called by it. For each method, we can see the following information: *Name:* The fully qualified name of the method, *Count:* The number of times the method was called during execution, *Cycles:* shows the amount of processor time spent in the method itself, *%Cycles:* is the percentage of the total execution time that is spent in the method itself, *Cycles with Children (*CWC*):* is the amount of time spent in the method *and* its called methods and *%Cycles with Children (%*CWC*):* shows the time spent in the method and its called methods as compared to the total execution time.

In this kind of testing we choose four methods in our application and examined the average time of execution for each of these methods: Encrypt Method, SendMessage Method, Decrypt Method and ReceiveMessage Method.

Table 1. Encrypt Method in Sender Side

Count	Cycles (ms)	%Cycles	CWC (ms)	%CWC
1 1 1	23.15	0.6	61.49	1.6
1	22.76	0.5	104.8	2.5
1	22.24	0.8	61.09	2.3
1	22.59	0.7	66.46	2.3
1	21.54	0.8	62.73	2.5
1	21.45	1	64.33	3.2
1	21.36	0.9	61.8	2.6
1	21.5	0.8	63.59	2.4
1	21.49	0.7	65.15	2.3
1	22.66	0.8	63.78	2.4
Avg Count	Avg Cycles (ms)	Avg %Cycles	Avg CWC (ms)	Avg %CWC
1	**22.07**	**0.76**	**67.52**	**2.41**

Table 2. SendMessage Method in Sender Side

Count	Cycles (ms)	%Cycles	CWC (ms)	%CWC
1	88.9	2.4	152.4	4.1
1	89.96	2.1	196.8	4.7
1	88.59	3.4	151.6	5.9
1	93.99	3.3	162.5	5.7
1	88.6	3.5	153.6	6.1
1	90.34	4.5	156.9	7.8
1	88.72	3.8	152.7	6.6
1	97.48	3.8	163.2	6.3
1	93.6	3.3	161	5.7
1	92.41	3.6	158.3	6.1
Avg Count	Avg Cycles (ms)	Avg %Cycles	Avg CWC (ms)	Avg %CWC
1	**91.26**	**3.37**	**160.9**	**5.9**

Tables 1 and 2, we can conclude that the number of times that Encrypt Method and SendMessage Method were called during execution is same and equal 1. The average execution times in seconds for Encrypt Method and SendMessage Method without children methods were 22.07 ms and 91.26 ms respectively. Also the percentage of time spent on a method's execution in respect to the time the entire program ran without children methods for Encrypt Method and SendMessage Method were 0.76% and 3.37% respectively. The average execution time in seconds for Encrypt Method and SendMessage Method with children methods were 67.52 ms and 160.9 ms respectively. Also the percentage of time spent on a method's execution in respect to the time the entire program ran with children methods for Encrypt Method and SendMessage Method were 2.41% and 5.9% respectively.

Table 3. Decrypt Method in Receiver Side

Count	Cycles	%Cycles	CWC	%CWC
1	21.07	1	63.82	3.1
1	21.22	1.3	67.09	4.2
1	21.01	1.4	63.73	4.5
1	22.18	1.6	66.02	4.9
1	31.93	2.5	80.59	6.4
1	22.16	1.6	65.78	5

Table 4. ReceiveMessage in Receiver Side

Count	Cycles	%Cycles	CWC	%CWC
1	0.766	0	179.8	8.7
1	0.767	0	241.8	15.4
1	0.757	0	97.31	6.9
1	0.776	0	17.41	1.3
1	0.756	0	18.51	1.4
1	1.102	0	23.27	1.7

1	21.92	1.8	65.12	5.6
1	21.08	1.9	70.86	6.4
1	21.9	1.8	66.16	5.4
1	22.1	1.7	68.41	5.4
Avg Count	Avg Cycles	Avg %Cycles	Avg CWC	Avg %CWC
1	22.66	1.66	67.76	5.09

1	0.75	0	16.66	1.4
1	0.759	0	18.25	1.6
1	0.758	0	21.2	1.7
1	0.76	0	17.76	1.4
Avg Count	Avg Cycles	Avg %Cycles	Avg CWC	Avg %CWC
1	0.795	0	65.20	4.15

Tables 3 and 4, we can conclude that the number of times that Decrypt Method and ReceiveMessage Method were called during execution is same and equal 1. And the average execution time in seconds, for Decrypt Method and ReceiveMessage Method without children methods were 22.66 ms and 0.795 ms respectively. Also the percentage of time spent on a method's execution in respect to the time the entire program ran without children methods for Decrypt Method and ReceiveMessage Method were 1.66% and 0% respectively. The average execution time in seconds for Decrypt Method and ReceiveMessage Method with children methods were 67.76 ms and 65.20 ms respectively. Also the percentage of time spent on a method's execution in respect to the time the entire program ran with children methods for Decrypt Method and ReceiveMessage Method were 5.09% and 4.15% respectively.

Time of Encryption and Decryption

Another test conducted was the time of Encryption and Decryption. We fix the size of SMS and then take 10 readings for the encryption and decryption time. We calculate the average time of encryption and decryption to use it in table 4.5, then we fix another size and so on, we start from size 12 until 84 and take 10 readings of average time of encryption and decryption. From able 5 we can calculate the encryption average time (EAT) which is equal 19ms and the decryption average time (DAT) which is equal 21ms.

Table 5. Encryption and Decryption Average Times

SMS Size (Byte)	Encryption Avg Time (ms)	Decryption Avg Time (ms)
12	28.1	28.2
20	7.8	7.7
28	17.4	17.3
36	9.4	12.5
44	12.6	9.4
52	6.2	15.5
60	21.9	26.6
68	23.1	25
76	31.1	33.1
84	34.1	38.2
Avg	**19.17 ms**	**21.35 ms**

Memory Usage Testing

We also conducted test for optimization of memory usage. The Memory Monitor Extension feature enables us to see how much memory is used by application during runtime and to see a breakdown of the amount of memory usage per object. The Memory Monitor displays usage information in Graph. The Memory Usage graph displays the following information namely the amount of memory used, the amount of unused memory available and the total amount of memory available at startup.

In Tables 6 and Table 7, we took five readings for the used and free memory at the sender and the receiver sides. After that we calculate the percentage of used and free memory to identify how much the application used the memory.

Table 6. Memory Usage in Sender Side

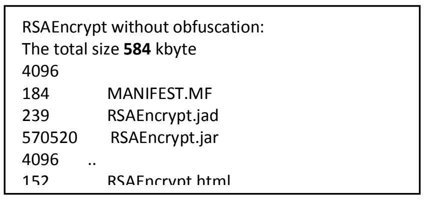
Used (byte)	Free (byte)	Total (byte)
48444	451556	500000
46336	453664	500000
47528	452472	500000
46272	453728	500000
46336	453664	500000
Avg Used (byte)	**Avg Free (byte)**	**Avg Total (byte)**
46983.2	**453016.8**	**500000**
% Used	% Free	
9.39664	90.60336	

Table 7. Memory Usage in Receiver Side

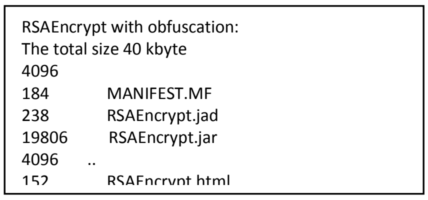
Used (byte)	Free (byte)	Total (byte)
42020	457980	500000
43244	456756	500000
44340	455660	500000
43244	456756	500000
43244	456756	500000
Avg Used (byte)	**Avg Free (byte)**	**Avg Total (byte)**
43218.4	**456781.6**	**500000**
% Used	% Free	
8.64368	91.35632	

Obfuscation Testing

In this test, two cases were experiment, the first one if we apply the obfuscation the total size of 40 bytes and the second case if we do not apply the obfuscation the total size of 584 bytes. Figure 7 illustrate the effects of obfuscation.



```
RSAEncrypt without obfuscation:
The total size 584 kbyte
4096
184         MANIFEST.MF
239         RSAEncrypt.jad
570520      RSAEncrypt.jar
4096     ..
152         RSAEncrypt.html
```



```
RSAEncrypt with obfuscation:
The total size 40 kbyte
4096
184         MANIFEST.MF
238         RSAEncrypt.jad
19806       RSAEncrypt.jar
4096     ..
152         RSAEncrypt.html
```

Figure 7. i) without Obfuscation **ii) with Obfuscation**

CONCLUSION

We have conducted several simple testing to make sure that RSA algorithm is operational in our experimental SMS exchange environment. The purpose of performance testing is to evaluate the RSA algorithm performance. The main factor that affect the performance of the testing is using of bouncy castle and obfuscation which allow RSA to be more efficient in this environment. We have successfully delivered a Secure SMS MIDlet application using RSA algorithm. All the designing, configuration and testing of the application with RSA algorithm have proven that it is possible to implement RSA in experimental SMS exchange environment. The process of designing, configuration, and testing described in the report shows the implementation of experimental RSA in Secure SMS application. This research can be further expanded in the future to incorporate other public key algorithms and comparing with them to identify the best public key algorithm for secure SMS.

REFERENCES

Hassinen M. and S. Markovski (2003). *Secure SMS messaging using Quasigroup encryption and Java SMS API*. 187 - 200.

Peersman G., P. Griffiths, H. Spear, S. Cvetkovic, and C. Smythe (2000). A tutorial overview of the short message service within GSM. *Computing & Control Engineering Journal*, vol. 11, 79-89.

Harkey D., S. Appajodu, and M. Larkin (2002). *Wireless Java Programming for Enterprise Applications: Mobile Devices Go Corporate*. John Wiley.

Hassinen M. (2005). SafeSMS-end-to-end encryption for SMS.. *ConTEL 2005- Proceedings of the 8th International Conference on Telecommunications*, vol. 2.

Ratshinanga H., J. Lo, and J. Bishop (2004). A Security Mechanism for Secure SMS Communication. *Proceedings of SAICSIT.* pp. 1-6.

Li G., Y. Liu, X. Cai, C. Wang, and D. Zhou (2003). *A Distributed and Adaptive Data flow System for SMS*. vol. 2, pp. 1350 -1355.

Stallings W., (2006). *Cryptography and Network Security*. 4th Ed: Prentice Hall.

Hwu J.S., S.F. Hsu, Y.B. Lin, and R.J. Chen (2006). *End-to-end Security Mechanisms for SMS*. Department of Computer Science & Information Engineering, National Chiao Tung University,.

Ratshinanga H., J. Lo, and J. Bishop (2005), *A Security Mechanism for Secure SMS Communication*. Computer Science Department, University of Pretoria, South Africa.

Wagner N.,(2001). *The Laws of Cryptography: The RSA Cryptosystem*.

Helal S., (2002). Pervasive Java. *Pervasive Computing, IEEE*. vol. 1, pp. 82-85.

Kolsi O. and T. Virtanen (2004). MIDP 2.0 security enhancements. *Proceedings of the 37th Annual Hawaii International Conference on System Sciences*, pp. 287-294.

Piroumian V., (2002). *Wireless J2ME Platform Programming*.

Lindquist T. E., M. Diarra, and B. R. Millard (2004). A Java Cryptography Service Provider Implementing One-Time Pad. *Proceedings of the 37th Annual Hawaii International Conference on System Sciences*, pp. 189-194.

Chat A., (2003), *MIDlet Example Using the Wireless Messaging API and the Nokia SMS API: Chat*. Forum Nokia. Retrieved from http://www.nokia.com

Chun H. L. W., (1999). *Interworking Of SMS Between GSM Based GMPCS System And IS-41 Based Cellular System Using I-SMC.* vol. 3, pp. 1432 - 1436.

Jiang H., (1998). *Reliability, Costs and Delay Performance of Sending Short Message Service in Wireless.* vol. 2, pp. 1073 – 1077.