2009 International Conference on Machine Learning and Computing
IPCSIT vol.3 (2011) © (2011) IACSIT Press, Singapore

A Secured Cryptographic Messaging System

Hassan Mathkour, Ghazy Assassa, A. Al-Muharib, A. Juma'h

Department of Computer Science, College of Information and Computer Science, King Saud University, Saudi Arabia

Abstract. In this paper, we describe a system for securing SMS messages during and after its transmission over mobile network. The system can also be used for protecting personal data for users who desire to keep their confidential data stored on their mobile phones. The system has been implemented employing the AES encryption algorithm and using symmetric-key encryption. The system has been tested using various devices. The testing and validation of the system is also reported.

Keywords: SMS, Encryption, Decryption, Mobile Transmission

1. Introduction

Mobile phone users desire more secure and private communication in their daily usage of their mobiles. This is especially important in communications of secret nature such as that in military and governmental communication. Securing voice calls is a difficult task as calls may be tabbed in transmission through various means. On the other hand, securing communication through the popularly used means, namely text messages, can be helpful and useful in many cases.

We describe a secured text messages communication environment via SMS. For this purpose, we develop a mobile-based application named Secret Short Message Service (SSMS). It encrypts a text message before sending it and decrypts the message in the receiver's side. In this way, the message is unreadable while transmitted even if it is intercepted while transmitting it over the network. The proposed system can send encrypted messages via SMS and allow users to encrypt/decrypt messages for personal usage without sending them. The latter feature is desirable for those who want to ensure the privacy of their own information.

SSMS employs symmetric-key encryption. The same secret key is used for both encryption and decryption. Therefore, the secret key must be known by the sender and the receiver of the message. Key distribution remains a problem when using symmetric-key encryption, but we found that it is the best solution when considering time complexity, efficiency, and costs.

SSMS depends on secret key embedding, where the message's secret key is distributed inside the cipher text after message encryption process. Secret key embedding is used for checking the correctness of a decryption key which is entered by the user. This schema saves time and space as there is no need for a database to store the secret key related to each message.

2. Background and Related Work

Many governmental and civilian applications benefit from encryption. Such applications include secure communication, e-commerce, mobile telephone networks, e-banking, and digital right management [6]. Although encryption protects the confidentially of messages, other techniques are required to verify the integrity and authenticity of the messages; for example, a message authentication code (MAC) or digital signatures. Standards and computer programs to perform encryption are widely available, but successful employment of encryption to ensure security is a challenging problem. A single blunder in system design or

execution may allow dangerous attacks. In some cases, an adversary can obtain valuable information without directly undoing the encryption [6].

A conventional encryption scheme has five major parts: Plaintext, Encryption Algorithm, Secret Key, Cipher text, and Decryption Algorithm [5]. In such a scheme, it is essential for secure communication that the sender and receiver have a way to exchange secret keys in a secure manner. Symmetric key encryption is faster than public key encryption since public key encryption places heavier computational load than symmetric key encryption [12]. Examples of commonly used symmetric-key encryption algorithms are: DES (Data Encryption Standard), TripleDES/3-DES, AES/Rijndeal. Table 1 compares these algorithms where AES seems to be the better algorithm. AES supports key sizes of 128 bits, 192 bits, and 256 bits, in contrast to the 56-bit keys offered by DES, a predecessor of AES [4, 5].

Table 1: Symmetric-key algorithms.

Algorithm	Length (bits)	Known Vulnerabilities
DES	64	yes
AES/Rijndael	128, 192, 256	no
TripleDES	128, 192	no

A fundamental primitive in modern cryptography is the cryptographic hash function. To be of cryptographic use, a hash function h is typically chosen such that it is computationally infeasible to find two distinct inputs which hash to a common value (i.e., two colliding inputs x and y such that h(x) = h(y)), and that given a specific hash-value y, it is computationally infeasible to find an input (pre-image) x such that h(x) = y [1]. Several hash functions techniques are available. This includes CRC, SHA, SHA-1, SHA-256, SHA-384, SHA-512, MD2, MD4,MD5, RIPEMD-128, RIPEMD-160, RIPEMD-256, RIPEMD-320, HMAC, HAVAL, UHASH [9]. Table 2 presents a comparison among such techniques.

Table 2: Hash algorithms comparison

Algorithm	Output size	Internal state	Block size	Length size	Word size	Collision	Security	Speed
HAVAL	256/224/192/160/128	256	1024	64	32	Yes	Medium	Low
MD2	128	384	128	No	8	Almost		
MD4	128	128	512	64	32	Yes	Low	Medium
MD5	128	128	512	64	32	Yes	Moderate	Medium
RIPEMD-128/256	128/256	128/256	512	64	32	No	Moderate	Slow
RIPEMD-160/320	160/320	160/320	512	64	32	No	High	Slow
SHA	160	160	512	64	32	Yes	Low	Fast
SHA-1	160	160	512	64	32	With flaws	Moderate	Medium
SHA-256/224	256/224	256	512	64	32	No	High	Slow
SHA-512/384	512/384	512	1024	128	64	No	Extreme	Slow
CRC32	32	32	512	32	16	Yes	Low	Fast

3. The Proposed System

Our proposed system (SSMS) uses symmetric-key encryption where messages are encrypted, and decrypted using a single secret key. We have selected AES (Rijndeal) algorithm to be employed in SSMS. 128-bits length is suitable for our purpose for practical message length and processing time resulting in a reasonable cost this is incurred by a service provider.

A message's secret key is embedded into the cipher text by random positioning using a suitable hash function. This operation is a very helpful and powerful operation. It helps in checking the correctness of an entered secret key by comparing it to the message's real secret key after extracting it from the cipher (where the key is embedded). In addition, it is very powerful and space saving. It eliminates the need for a database to keep the secret key that is related to each message.

Message decryption is performed without checking the entered secret key. This will save time. For correctly entered secret keys, such a process will be fast. But it might cause some problems with incorrect entries. If a wrong key is entered the cipher will be decrypted displaying a non understandable text. Wrong keys may cause problems (i.e. long time which may lead to a hanging)

To ensure the protection of the embedded secret key in the message, the secret key will be encrypted before encrypting the text. Encryption as well as decryption will be done using the encrypted secret key. This encryption is performed using a non complicated hash function which results in a suitable encrypted key. When a secret key is entered for decryption, the entered secret key will be automatically encrypted using the same hash function, and then it is compared to the message's secret key. Fig. 1 depicts the architecture of the proposed system. The business model is depicted in Fig. 2.

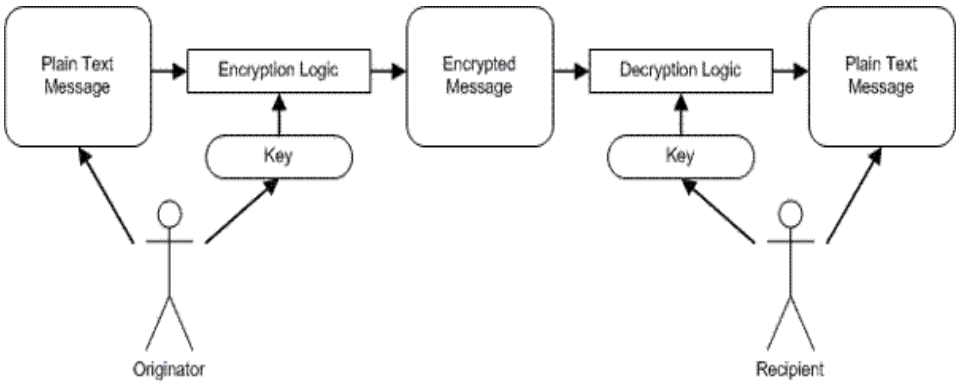

Fig.1: system architecture

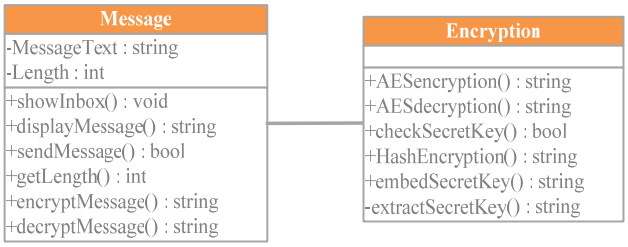

Fig. 2 class diagram (business model)

The Bouncy castle j2me cryptography library was used for AES 128-bit encryption/decryption and hashing using SHA-128 [22]. SSMS is developed using NetBeans IDE including the Mobility pack, which is an additional tool (Plug-In) that is designed for mobile application development, along with Sun wireless toolkit for CDLC.

4. Validation and Testing

The validation and testing process was performed in four phases:

- Testing the system on a PC under any operating system using a suitable emulator to validate the system and ensure that it is working properly.
- Installing the system on multiple mobile phone devices which are running under different environments, operating systems, and CPU capabilities. To make sure it is able to run properly on all of them.

- Performing encryption/decryption operations via the system to make sure it comes up with the correct results.
- Producing a beta version of SSMS, and distributing it on a group of five or more persons (from multiple majors including security) with different mobile phones, and wait for them to come with the results.

Table 3 presents an encryption/decryption comparison among different devices with different message lengths. Table 4 presents details of tested devices. Fig. 3 depicts Encryption speed comparison.

Table 3: An encryption/decryption comparison

Message Length	PC		Nokia N73		Nokia 3250		Nokia N76		Motorola V3	
	E	D	E	D	E	D	E	D	E	D
1	5	2	182	177	275	267	361	355	744	735
32	9	8	240	235	393	386	402	394	821	814
128	15	35	263	259	421	414	466	461	1038	1023
256	28	44	277	271	438	430	517	510	X	X
512	195	180	309	302	459	448	564	558	X	X

*The speed is measured by milliseconds (ms).
* X means unsupported

Table 4: Details of the tested devices

Device	Processor Speed	RAM
PC	Intel Centrino 1.6 GHz	1.24 GB
Nokia N73	330 MHz	26 MB
Nokia 3250	220 MHz	21 MB
Nokia N76	200 MHz	20 MB
Motorola V3	66.1 MHz	1 MB

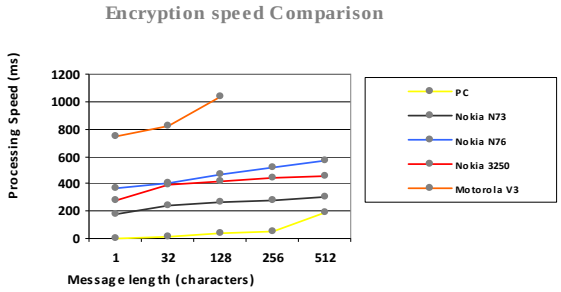

Fig. 3: Encryption speed comparison

5. Conclusion and future remarks

Testing SSMS has been conducted on different environments, where we observe:

- SSMS works properly on all the mobiles which support Java, with some changes in the CDLC Version and MIDP Version, because some devices has different Java system versions. No changes need be made in the source code of the application.
- The encryption/decryption speed is very convenient except in cases where the RAM is less than 1 MB and the message length is more than 256 characters. In such a case, the device cannot perform encryption or decryption.
- AES algorithm encryption with 128-bit key length is sufficient for mobile devices.

Embedding the secret key into the message after encryption and extracting it while decryption was not implemented as it requires more processing time. This is because it includes hashing and random positioning of each character.

In the next version of SSMS, the following will be considered:

- Uniting the secret key for each user so that the user does not have to remember the secret key that he decided to use to communicate with other users. This can be done in many ways, one of them is by creating a password for each user by asking him to enter it when the application is installed on his device and store the key in the database. When a user (A) sends a new message to another user (B), the secret key of A will be embedded into the cipher text and the timestamp will also be embedded. When B receives the message, will be asked to enter his secret key and then the secret key of A and the timestamp will be extracted from the cipher text. If the secret key of B is correct and the timestamp is earlier than the date and time when the application was installed, the cipher text will be decrypted using the secret key of A, and the original message will be displayed.
- Integrating SSMS with the device so that the user can select the receiver from his contact list directly.
- Providing a way to receive the sent messages directly from SSMS instead of copying each message manually to the application, or accessing the inbox directly through the application.
- Modifying the application to support non-Unicode characters such as Arabic characters.

6. Acknowledgement

This work is partially supported by the research center of the college of computer and information sciences in King Saud University, Riyadh, Saudi Arabia.

7. References

[1] A. Menezes, P. van Oorschot, and S. Vanstone, 1996. Handbook of Applied Cryptography, *CRC Press, Canada.*

[2] Ian Curry, March 2001. An introduction to cryptography and digital signatures, *Entrust-Securing digital identities and information*, pp1-6

[3] Monica Pawlan, Cryptography: The ancient art of secret messages, February 1998 http://www.pawlan.com/Monica/crypto/.

[4] H.X Mel and Doris Baker (2001), Cryptography decrypted, Addison-Wesley Professional, *Pearson Technology Group, Canada*, ISBN-10: 0201616475 | ISBN-13: 9780201616477.

[5] Joan Daemen, Vincent Rijmen, September 1999, AES Proposal: Rijndael, Belgium pp 1-45 csrc.nist.gov/archive/aes/rijndael/Rijndael-ammended.pdf

[6] Rick Smith and Rick Wanner, Mobile encryption, Group discussion and project, CDI east 2006, pp 1-14 www.sans.edu/resources/student_projects/200612_001.pdf.

[7] Eric Olson and Woojin Yu, Encryption for mobile computing, 2003. bwrc.eecs.berkeley.edu/classes/cs252/Projects/Reports/yu_olson.pdf

[8] Helger Lipmaa and Shiho Moriai. Efficient Algorithms for Computing Differential Properties of Addition. In Mitsuru Matsui, editor, *Fast Software Encryption: 8th International Workshop, FSE 2001*, volume 2355 of Lecture Notes in Computer Science, pages 336--350, Yokohama, Japan, April 2--4, 2001. Springer-Verlag. ISBN 3-540-43869-6.

[9] M. Hassinen, SafeSMS - end-to-end encryption for SMS, *Proceedings of the 8th International Conference on Telecommunications Volume 2*, ConTEL 2005, ISBN: 953-184-081-4

[10] RSA laboratories, http://www.rsa.com .

[11] MyCrypto, http://www.mycrypto.net .

[12] Wikipedia, http://www.wekipedia.org.

[13] Encryption algorithms, http://www.networksorcery.com .

[14] VeriSign, http://www.verisign.com .

[15] Steve Fridel, An illustrated guide to cryptographic hashes, http://www.unixwiz.net .

[16] Cryptography and encryption, http://www.cryptoForge.com .

[17] Combining symmetric and asymmetric encryption, codeproject, http://www.codeproject.com .

[18] Basic components of modern cryptography, Microsoft, http://www.microsoft.com/technet

[19] Privacy and encryption, http://www.pgpi.org .

[20] SMS Special 007 group, http://www.sms007.cz .

[21] Michel J. Yuan and Ju Long , Build database-powered mobile applications on the Java platform, JavaWorld.com, January 2002.

[22] The Legion of the Bouncy Castle, http://www.bouncycastle.org.