Implementations of Block Cipher SEED on Smartphone Operating Systems

HwanJin Lee, DongHoon Shin, and Hyun-Chul Jung
Security R&D Team
Korea Internet & Security Agency (KISA)
Seoul, Korea
{lhj79, dhshin, hcjung}@kisa.or.kr

***Abstract*—As more and more people are using smartphones these days, a great deal of important information, such as personal information and the important documents of corporations among other things, are being saved on smartphones. Unlike a PC, people can access another person's smartphone without great difficulty, and there is a high possibility of losing one's smartphone. If smartphone is lost without encryption, important information can be exploited. In addition, the open cryptographic library for PCs cannot be used due to the limited performance of the smartphone. This paper introduces the optimization implementation technique for the smartphone OS and the results of using that technique. In addition, the results of a speed comparison with the open cryptographic library will be presented. According to the results of comparing the one-time encryption implementation time with the open cryptographic library, the performance time was improved by 12% for Windows Mobile, 8.57% for iOS, and 39.62% for Android.**

Keywords-SEED; Windows mobile; iOS; Android; implementation; blockcipher.

I. INTRODUCTION

Use of the smartphone is increasing due to the rapid development of IT technologies. Now, we can make a call or use various functions such as e-mailing, web surfacing, and Office programs simply with a small smartphone. However, the risk of loss or theft is also increasing due to the device's small size and light weight. Due to their inconvenient portability, around 200,000 smartphones are lost or stolen every month on average.

Loss of a smartphone can lead to a serious leak of an personal information, as smartphones contain a large amount of such information (call details, received messages, phone numbers, schedules, location information, financial transaction information, etc.). And smartphones are also used for business and sales purposes. So, secondary damage can be caused if a smartphone containing a corporation's sensitive information is lost or stolen.

Data encryption is very important to protect the various types of personal information and confidential information stored in the smartphone. SEED is block cipher that can be used for data encryption. Because that SEED is Korean and International Standard, the usage of SEED has been covered the security service applications in Korea. So, application of smartphone which needs security service must be implemented SEED in Korea. However, a smartphone has limited power and offers inferior performance compared to a PC. Therefore, it is difficult to use an open cryptographic library such as OpenSSL, which is designed for the PC environment, in a smartphone. We need to study on the way for the effective use of SEED in smartphone.

This paper presents the results of implementing the block cipher SEED to a smartphone. The results of a comparison with open cryptographic libraries (OpenSSL, BouncyCastle) will also be presented. The SEED is a block cipher established as an international standard ISO/IEC and the Korean standard. Section 2 introduces the SEED and open cryptographic libraries; Section 3 introduces smartphone operating systems; Section 4 presents the implementation method; Section 5 presents the implementation and comparison results; and Section 6 presents the conclusion.

II. SEED AND OPEN CRYPTOGRAPHIC LIBRARIES

A. SEED

SEED is a 128-bit symmetric key block cipher that had been developed by KISA (Korea Internet & Security Agency) and a group of experts since 1998. SEED has been adopted by most of the security systems in Korea. SEED is designed to utilize the S-boxes and permutations that balance with the current computing technology. The input/output block size and key length of SEED is 128-bits. SEED has the 16-round Feistel structure. A 128-bit input is divided into two 64-bit blocks and the right 64-bit block is an input to the round function, with a 64-bit sub-key generated from the key scheduling [1].

SEED has been adopted as an industrial association standard of Korea (TTA, Telecommunication Technology Association) at 1999 and ISO/IEC and IETF International Standard at 2005 [2, 3].

Classification	**Number and Title**
Korean Standard	TTAS.KO-12.0004 : 128-bit Symmetric Block Cipher(SEED)
International Standard	Standard ISO/IEC 18033-3 : Information technology - Security techniques - Encryption algorithms - Part 3 : Block ciphers.
	IETF RFC 4269 : The SEED Encryption Algorithm ※ RFC4269 obsoletes RFC 4009.

Furthermore, several standards (Cryptographic Message Syntax, Cipher Suites to Transport Layer Security, IPsec etc.) have been adopted as Korean standard and International standards [4~6].

Classification	Number and Title
Korean Standard	TTAS.KO-12.0025 : Modes of Operation for The Block Cipher SEED
International Standard	IETF RFC 4010 : Use of the SEED Encryption Algorithm in Cryptographic Message Syntax (CMS)
	IETF RFC 4162 : Addition of SEED Cipher Suites to Transport Layer Security(TLS)
	IETF RFC 4196 : The SEED Cipher Algorithm and Its Use with IPsec

The usage of SEED has been covered the security service applications such as, e-Commerce, e-mail, dedicated receiver with Broadcasting, financial service, data storage, electronic toll collection, VPN, Digital Right Management, etc.

In particular, under the auspices of the Bank of Korea, eleven banks and one credit card company has launched a pilot service of K-cash for about 600 franchisees in Seoul since July of the year 2000. SEED has been used to protect the privacy of the users and the transaction data in this service.

B. *Cryptographic Libraries*

1) *OpenSSL*

OpenSSL [7] is an open cryptographic library written in C language. OpenSSL implements most of the encryption algorithms we use in our daily life, such as symmetric-key ciphers, hash functions, public-key ciphers, message authentication codes, and SSL/TLS. OpenSSL complies with and can be used in various platforms such as Unix, Linux, and Windows.

2) *BouncyCastle*

BouncyCastle [8] is an open cryptographic library written in Java and C# language. BouncyCastle can be implemented with J2ME, JDK 1.6, and C# API. Like OpenSSL, most encryption algorithms used in our daily life have been implemented.

III. SMARTPHONE OPERATING SYSTEM

A. *Windows Mobile*

Windows Mobile [9] is a mobile operating system developed by Microsoft that was used in smartphones and mobile devices. It is used in a variety of devices such as smartphone, vehicle on board and portable media devices etc. The current and last version is "Windows Mobile 6.5".

Windows Mobile can be classified as Application, Operating System and Cryptographic Service Provider (CSP). And it includes the Cryptography API set (CryptoAPI), which provides services that enable application developers to add encryption and decryption of data.

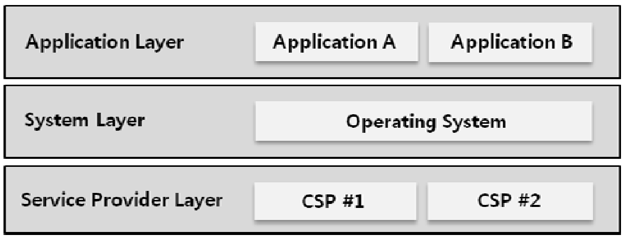


Figure 1. Architecture overview of Windows Mobile

Windows mobile provides service encryption, hashing and digital signature, etc. Windows mobile supports many block ciphers such that DES, 3DES, IDEA, CAST, RC5 and AES-128/192/256. Supporting hash functions are MD2, MD4, MD5, SHA1/256/384/512 and HMAC. And digital signatures are RSA, DSS and ECDSA. But it does not support SEED. So, application developer must program SEED or port OpenSSL for using SEED in Windows mobile.

B. *iOS*

iOS [10] is the operating system that runs on iPhone, iPod touch, and iPad devices. Although it shares a common heritage and many underlying technologies with Mac OS X, iOS was designed to meet the needs of a mobile environment, where users' needs are slightly different.

The iOS security APIs are located in the Core Services layer of the operating system and are based on services in the Core OS (kernel) layer of the operating system. Core Services layer includes key chain service, certificate, key, trust service and randomization service and supports library for a symmetric-key Cipher, digital signature etc. The iOS security APIs are based on services in the Core Services layer, including the Common Crypto library in the libSystem dynamic library. Common Crypto library supports DES, 3DES, AES-128/256 block cipher. And it supports MD2/4/5, SHA-1/224/256/384/512 hash functions. But it does not support SEED as windows mobile. So, SEED be programmed or ported Openssl in application of iOS.

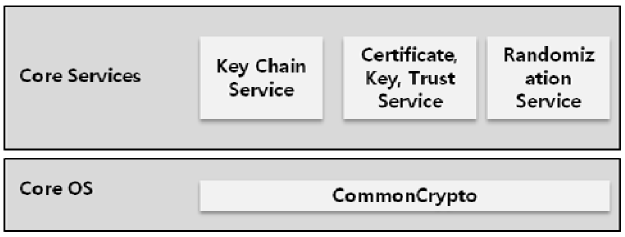


Figure 2. Architecture overview of iOS

C. Android

Android [11] is an open-source software stack for mobile devices that includes an operating system, middleware and key applications. Google Inc. purchased the initial developer of the software, Android Inc., in 2005.

Android can be classified as system layer, crypto library and crypto class. Android developer can use java.security package and Javax.crypto package in crypto library. java.security package provides all the classes and interfaces that constitute the Java security framework. java.security package supports certificate and signature. javax.crypto package provides the classes and interfaces for cryptographic applications implementing algorithms for encryption, decryption, or key agreement. javax.crypto package supports stream cipher, block cipher, hash function and MAC. Android does not support SEED. To use SEED, android developer must program SEED or port BouncyCastle.

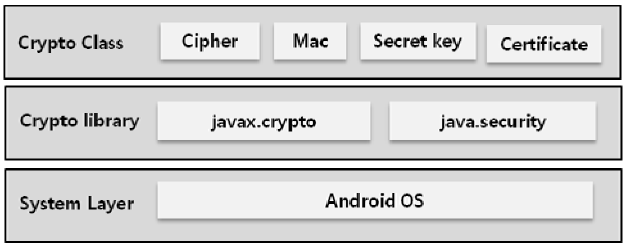


Figure 3. Architecture overview of Android

IV. 3. Optimization on Smartphone OS

A. General

1) *32-bit processing*

- ARM core version 6x version runs on 32bit. Therefore, data processed by the algorithm is implemented in 4 bytes.

2) *Little endian*

- For the ARM core environment, an algorithm was designed and implemented, based on the little endian.

3) *Memory management*

- The embedded system has a limited and inefficient memory allocation system. The memory space for temporary variables was allocated in advance and re-used.

4) *Loop optimization*

- Unnecessary loops were reduced and repetition was removed. For example, the ARM7 and ARM9 processors require one cycle for subtraction processing, and three cycles for selection control processing. That is, if subtraction is configured in a loop, four cycles are required to process one loop. In addition, we reduce the number of iteration as much as possible, when “for loop” is used.

5) *Variable declaration and operation*

- The unsigned type variable was used. Regular processes handle “unsigned integer” operation much faster than “signed” operation.

B. Android

Android application development codes are compiled in a machine-independent byte code, and executed by a Dalvik virtual machine in the Android device.

That is, as the Java code is executed in the Java Virtual Machine, the Android app is executed in the Dalvik Virtual Machine. Therefore, Android’s processing speed is slower than that of native codes.

To improve the processing speed, Java provides JNI (Java Native Interface), which accesses the source coded in other languages, and executes the source code.

The operation of memory copy and XOR (exclusive OR), which are most frequently used in actual encryption algorithms, were compared. It takes about 380ms on average when the System.arraycopy Java method is used for 4096 bytes of data. However, it takes 266ms on average when implementing memory copy using the memcpy C function in the JNI. As a result, we can see that the performance of memory copy was improved by 140%. For XOP operation, Java takes 830ms and JNI takes 161ms, which implies that the performance of XOR operation was improved by 500% or more.

Therefore, if the algorithm is implemented appropriately in the Android environment using the JNI, a very efficient encryption algorithm can be implemented.

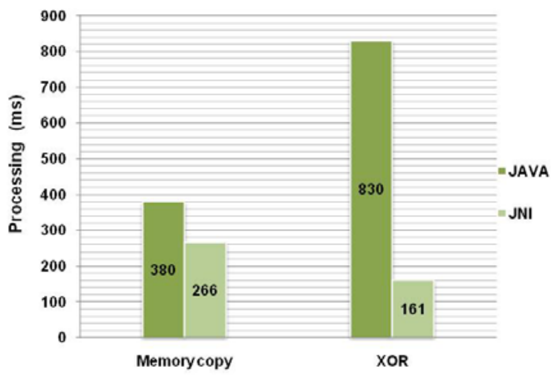
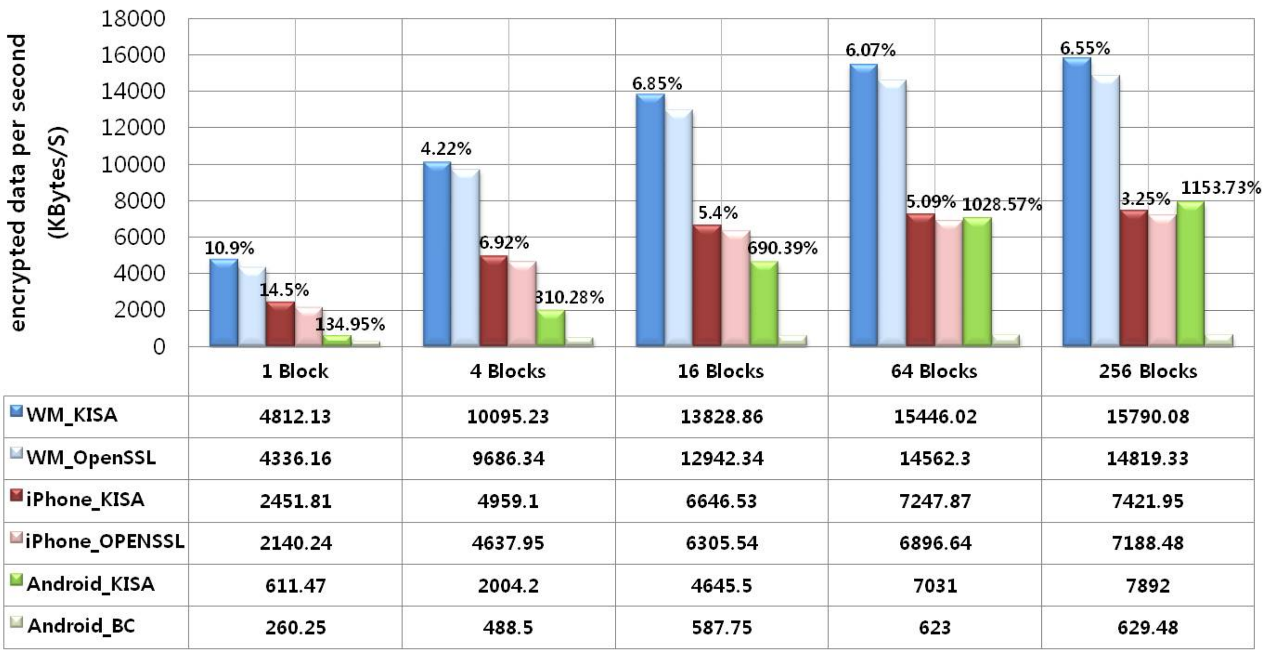


Figure 4. Processing time of JAVA and JNI

V. Implementation Results

This paper presented the results of the SEED encryption speed and power consumption for each smartphone OS. In addition, the results of the comparison with open cryptographic libraries (OpenSSL, BouncyCastle) are also presented. Speed was compared with the amount of encrypted data per second and the one-time encryption time. Power consumption was compared, based on the time used to consume 1% of the battery. The algorithm test was conducted in the following environment.



	1 Block	4 Blocks	16 Blocks	64 Blocks	256 Blocks
WM_KISA	4812.13	10095.23	13828.86	15446.02	15790.08
WM_OpenSSL	4336.16	9686.34	12942.34	14562.3	14819.33
iPhone_KISA	2451.81	4959.1	6646.53	7247.87	7421.95
iPhone_OPENSSL	2140.24	4637.95	6305.54	6896.64	7188.48
Android_KISA	611.47	2004.2	4645.5	7031	7892
Android_BC	260.25	488.5	587.75	623	629.48

Figure 5. The amount of encrypted data per second

TABLE I. TEST DEVICE SPECIFICATIONS

Platform	Device	OS	H/W specification	Open cryptographic libraries
Windows Mobile	HTC HD2	Windows Mobile 6.5	- CPU : Qualcomm, Snapdragon 1Ghz - RAM : 448MB	OpenSSL
iPhone	Apple iPhone 3GS	iOS 4.0	- CPU : ARM, Cortex A8 600Mhz - RAM : 256MB	OpenSSL
Android	HTC Nexus One	Android 2.2 (Froyo)	- CPU : Qualcomm, Snapdragon 1Ghz - RAM : 512MB	BouncyCastle

A. Results of speed comparison

The data processing amount per second refers to the amount of data be encrypted by SEED for one second. CBC (Cipher-block chaining) was used as a mode of operation. The length of the input plaintext was set to 1, 4, 16, 64, and 256 blocks respectively (1 block = 128 bits). The length of the input plaintext is 4 means input of SEED-CBC is 4 blocks plaintext and operating number of CBC is 4. The length of the input plaintext is considered for size of information in storage of smartphone such as phone numbers, Social Security number etc. To reduce the possibility of error, we repeat more than 1000 seconds. And results were divided by process time. For example, in case 4 blocks, SEED-CBC only operates 4 blocks plaintext until time is more than 1000 seconds.

According to the results of the comparison of the data encrypting amount per second, Windows Mobile was improved by 4.22~10.98%, whereas iOS and Android were improved by 3.25~14.56% and 134.95~1153.73%, respectively. As a result, windows mobile and iOS did not show a great improvement. But Android showed high improvement rate for using JNI. We think that increasing of encrypted data per second depending on the length of the input plaintext because memory input/output. With smaller block, it is decreasing number of memory access. 1 block access memory 256 times when 256 blocks access memory once. For the cases of Windows Mobile and iOS, the improvement rate decreases while increasing the size of input data to encrypt. We think that the reasons are way of optimization. It's not optimization of algorithm structure but optimization of algorithm implementation. It is depend on time of access memory and smartphone OS. For the case of Android, the improvement rate increases continuously, because that BouncyCastle is very slow. But it will reach the limit.

The one-time encryption time refers to the time period required to execute an encryption algorithm once. To reduce the possibility of error, the average of the results of 80,000 repeated executions was calculated. According to the results of the comparison, Windows Mobile improved by 12%, whereas iOS improved by 8.57% and Android by 39.62%. The results are different from results of encrypted data per second. We think that the reasons are difference of calculation methods. Results of encrypted data per second include checking time whether more than 1000 seconds or not.

TABLE II. ONE-TIME ENCRYPTION TIME

(ms/1call)	Windows Mobile	iOS	Android
Ours	0.0044	0.0064	0.032
OpenSSL (BouncyCastle *)	0.0050	0.0070	*0.053
Improvement rate (%)	12%	8.57%	39.62%

B. *Electricity consumption*

The time taken to use 1% of the battery was measured, when executing an encryption algorithm. The results of the comparison of electricity consumption show a power saving of 19.9% for Windows Mobile, and of 14.85% and 12.36% for iOS and Android respectively.

TABLE III. ELECTRICITY CONSUMPTION COMPARISON

sec	Windows Mobile	iOS	Android
Ours	120.21	125.3	112.45
OpenSSL (BouncyCastle *)	100.2	109.1	100.08
Improvement rate (%)	19.97%	14.85%	12.36%

VI. CONCLUSION

It is essential to protect the information stored in smartphones owing to their increasing popularity and various functions. However, a method of optimization other than a PC is required due to the limited performance of the smartphone. This paper presents the results of the optimal application of the block cipher SEED, which was selected as both the international standard ISO/IEC and the Korean standard, to the smartphone. Also, the results of comparing the open cryptographic library OpenSSL (including the SEED) with BouncyCastle were presented.

According to the results of optimizing and implementing the SEED in smartphones, SEED provided better performance than the open cryptographic library in all areas (i.e., data processing amount per second, one-time encryption execution time, and electric consumption). In particular, Android showed a remarkably enhanced performance than other OS when optimized with the JNI.

Recently, the smartphone has been attracting ever more attention among the general public, and the number of service environments that capitalize on this increasing attention is also rising continuously. Accordingly, the number of environments that require a high level of security is also increasing, such as smart office and mobile cloud. To create a safe smartphone use environment, methods of optimizing the various encryption algorithms are needed. Consequently, research on optimizing the algorithms, such as public key cipher and SSL/TLS, is needed.

ACKNOWLEDGMENT

This research was supported by the ICT Standardization program(2011-PM10-18) of MKE(The Ministry of Knowledge Economy)

REFERENCES

[1] Korea Internet and Security Agency, Block Cipher Algorithm SEED, Available from http://seed.kisa.or.kr/eng/about/about.jsp. [Accessed: June 5, 2011]

[2] ISO/IEC 18033-3, Information Technology–Security Techniques–Encryption Algorithms–Part 3: Block Ciphers, ISO, 2005.

[3] Lee, H., Lee, S., Yoon, J., Cheon, D., and J. Lee, "The SEED Encryption Algorithm", RFC 4269, December 2005.

[4] J. Park, S. Lee, J. Kim, and J. Lee, "Use of the SEED Encryption Algorithm in Cryptographic Message Syntax (CMS)", RFC 4010, February 2005.

[5] H.J. Lee, J.H. Yoon, and J. Lee, "Addition of SEED Cipher Suites to Transport Layer Security (TLS)", RFC 4162, August 2005.

[6] H.J. Lee, J.H. Yoon, S.L. Lee, and J. Lee, "The SEED Cipher Algorithm and Its Use with IPsec", RFC 4196, October 2005.

[7] OpenSSL, Available from http://www.openssl.org/. [Accessed: June 5, 2011]

[8] BouncyCastle, Available from http://www.bouncycastle.org/index.html. [Accessed: June 5, 2011]

[9] Windows Mobile 6.5 MSDN, Available from http://msdn.microsoft.com/en-us/library/bb158486.aspx. [Accessed: June 5, 2011]

[10] iOS Refernece Library, Available from http://developer.apple.com/library/ios/navigation/. [Accessed: June 5, 2011]

[11] Android, Available from http://www.android.com/. [Accessed: June 5, 2011]

[12] C. R. Mulliner, “Security of Smartphones”, Master’s Thesis submitted in University of California, Santa Barbara, 2006.

[13] Lisonek, D. and Drahansky, M., "SMS Encryption for Mobile Communication", the 2008 International Conference on Security Technology, 2008, pp. 198-201.

[14] F. Fitzek and F. Reichert, "Mobile Phone Programming and its Application to Wireless Networking", Springer, 2007.

[15] L. Shurui, L. Jie, Z. Ru, and W. Cong, "A Modified AES Algorithm for the Platform of Smartphone", 2010 International Conference on Computational Aspects of Social Networks, 2010, pp. 749-752.

[16] F. M. Heikkila, "Encryption: Security Considerations for Portable Media Devices," IEEE Security and Privacy, vol. 5, no. 4, 2007, pp. 22-27,

[17] JLC. Lo, "A framework for cryptography algorithms on mobile devices", Master's Thesis, 2007.

[18] S. M. Habib and S. Zubair, "Security Evaluation of the Windows Mobile Operating System", Master's Thesis, 2009.

[19] Asghar, M.T., Riaz, M., Ahmad, J. and Safdar, S., "Security model for the protection of sensitive and confidential data on mobile devices", International Symposium on Biometrics & Security Technologies, 2008, pp. 1-5.

[20] A. Visoiu and S. Trif, "Open Source Security Components for Mobile Applications," Open Source Science Journal, vol. 2, no. 2, 2010. pp. 155-166.