논문 17-42-01-31 The Journal of Korean Institute of Communications and Information Sciences '17-01 Vol.42 No.01
https://doi.org/10.7840/kics.2017.42.1.250

위·변조에서 사용되는 암호알고리즘 성능 비교에 대한 연구

이 준 영◆, 장 남 수°

A Study on the Cryptography Algorithm Performance Comparison Used in Modulation and Forgery

Jun Yeong Lee◆, Nam Su Chang°

요 약

최근 다양한 서비스를 제공하기 위해 모바일 기기의 활용도가 증가하고, 이에 따라 안드로이드 플랫폼에서의 앱 위·변조 공격이 급증하고 있다. 이에 대응하기 위해 국내의 금융 분야에서는 암호 알고리즘을 기반으로 한 앱 위·변조 방지 솔루션을 도입하고 있다. 그러나 스마트폰에 설치되는 앱의 용량이 지속적으로 증가하고, 웨어러블이나 IoT 등 제한된 자원을 가진 환경이 확산되면서 앱 위·변조 방지 솔루션의 처리 속도의 한계점을 가진다. 본 논문에서는 고속경량 암호 LEA와 LSH를 사용한 앱 위·변조 방지 솔루션을 제안한다. 또한 이 기법을 구현한 시뮬레이션 프로그램의 실험결과를 제시하고 기존 암호 알고리즘이 적용된 앱 위·변조 방지 솔루션과의 성능을 비교한다.

Key Words : APP anti-tamper, LEA, ARIA, SEED, LSH, SHA

ABSTRACT

Recently, the use of mobile devices has increased in order to provide a variety of services, and thus there has been a surge in the number of application malicious attacks on the Android platform. To resolve the problem, the domestic financial sector has been introducing the app anti-tamper solution based on cryptographic algorithms. However, since the capacity of apps installed in smartphones continues to increase and environments with limited resources as wearables and IoTs spread, there are limitations to the processing speed of the anti-tamper solutions. In this paper, we propose a novel anti-tamper solution by using lightweight hash function LEA and LSH. We also present the test results of a simulation program that implements this method and compare the performance with anti-tamper solutions based on the previous cryptographic algorithms.

Ⅰ. 서 론

최근 Android 플랫폼을 대상으로 한 악성 코드의 수가 증가하고 있고, 그와 더불어 위·변조 애플리케이션의 수요가 증가하고 있다. 대부분의 애플리케이션은 실행과 동시에 개인정보를 탈취하거나, 안전하지 않은 상태의 시스템에서 사용자의 금융 정보를 탈취하고 있다. 이러한 형태의 공격으로부터 사용자의 정보를 안전하게 지키기 위해 앱 위·변조 방지 솔루션이 도입되었다.[1] 일반적으로 위·변조 방지 기술은 암호 알고

◆ First Author : Sejong Cyber University Graduate School of Information Security, jylee315.kr@gmail.com, 정회원
° Corresponding Author : Sejong Cyber University Department of Information Security, nschang@sjcu.ac.kr, 정회원
논문번호 : KICS2016-11-355, Received November 17, 2016; Revised December 5, 2016; Accepted Jaunary 13, 2017

리즘을 많이 사용하고 있고, 스마트폰의 사양이 좋아지면서 앱의 용량이 갈수록 커지고 있다. 더불어 웨어러블이나 IoT 제품들이 증가하면서 앱 위·변조 방지 솔루션이 적용된 앱의 처리 속도가 갈수록 느려져 고속 암호 알고리즘의 필요성이 크게 대두되었다.[2]

따라서 본 연구에서는 앱 위·변조 방지 솔루션에서 사용되는 암호 알고리즘들과 최근 국내에서 개발된 고속 경량 암호 알고리즘들의 성능 비교 결과를 제시한다. 위·변조를 방지하기 위한 방법으로는 크게 두 가지로 분류할 수 있다.

- 앱 위·변조를 사전에 막는 방법
- 앱 무결성을 검사하는 방법

위·변조를 사전에 막는 방법은 앱이 바이너리 난독화, 동적 분석 방지, 해킹툴 탐지 등을 이용하여 적법한 권한을 가지고 있지 않은 제 3자가 앱을 수정하는 것을 미리 차단하는 방법이다.[3,4] 이 방법에서 서버와 데이터를 주고받거나 중요 데이터를 노출시키지 않게 하기 위해 암호 알고리즘을 사용한다. 본 논문에서는 실제 사용되는 DxShield라는 솔루션에 적용되어 있는 대칭키 알고리즘 SEED[5]와 다른 국내외 대칭키 알고리즘인 AES[6], ARIA[7], LEA[8]의 성능을 비교한다.

앱 무결성을 검사하는 방법은 해시 알고리즘을 이용하여 원본 앱의 해시 값과 실제 사용되는 앱의 해시 값을 비교하여 앱이 변경되었는지 판별하는 방법이다.[9,10] 본 논문에서는 실제 무결성 검사를 위해 사용되는APP Protect라는 솔루션에 적용되어 있는 SHA[11]와 국내에서 개발된 LSH[12]의 성능을 비교한다.

Ⅱ. 앱 위·변조 차단 솔루션에서의 성능 비교

본 절에서는 테스트 대상인 해킹 방어 솔루션(DxShield)에 대하여 간략히 기술하고 솔루션이 동작하는 과정에서 사용되는 블록 암호 알고리즘 SEED와 국내외 다른 블록 암호 알고리즘인 ARIA와 AES, LEA(국내 고속 경량 암호 알고리즘)의 성능을 비교하고자 한다.

DxShield는 안티 디버그, 난독화, 치팅앱 탐지, 저장파일 변조 차단, E2E 암/복호화를 지원한다.

표 1. DxShield에서 사용되는 암호 알고리즘
Table 1. cryptography algorithm in DxShield

Division	Contents
ECDH[13]	E2E-Transmission interval key exchange
SEED	E2E-Transmission interval En/Decryption

DxShield에서 사용되는 암호 알고리즘은 다음과 같다.

그림 1은 DxShield의 전체 흐름도를 보여준다. 그림에서 보라색으로 표시된 구간은 블록 알고리즘을 이용하여 클라이언트와 서버가 데이터를 암/복호화를 수행하여 안전하게 전달하는 구간이다. 이 부분에 SEED와 ARIA, LEA를 적용하여 성능을 비교 분석한다.

그림 1 DxShield 솔루션 전체 흐름도를 상세 기술하면 다음과 같다.

① 클라이언트 키 쌍 생성 및 클라이언트에서 서버로 초기 데이터(공개키)를 요청
② 서버 키 쌍 생성 및 초기 데이터 생성 후 전송
③ 클라이언트에서 서버로 저장 파일 요청
④ 서버의 개인키와 클라이언트 공개키로 비밀키를 생성하고 블록 암호 알고리즘으로 저장 파일 암호화 및 전송
⑤ 필요시 클라이언트의 개인키와 서버의 공개키로 비밀키를 생성하고 블록 암호 알고리즘으로 저장 파일 복호화
⑥ 저장 파일 정보 수정 후 암호화하여 보관

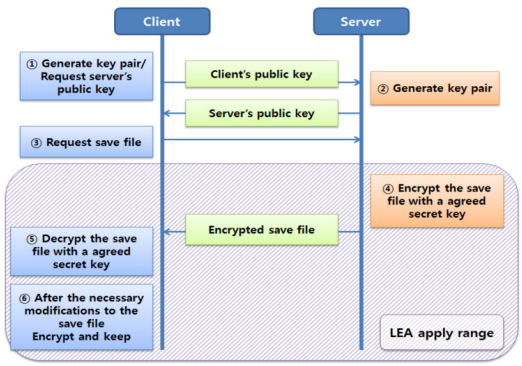


그림 1. DxShield 전체 흐름도
Fig. 1. DxShield full flow chart

2.1 구현 환경

구현물은 라이브러리 형태로 제공되는 솔루션이기 때문에 DxShield가 실행되는 것을 확인하기 위해 그림 2와 같이 안드로이드용 샘플 앱을 만들어 테스트

표 2. 스마트폰 구현 환경
Table 2. Smartphone implementation environment

CPU/ Opration System	Samsung Exynos 4210 1.2GHz Dual-Core/ Android 4.0.4
RAM	1GB
Language	C, Java(JNI)
Compiler	GCC 4.8.2

표 3. 서버 구현 환경
Table 3. Server implementation environment

CPU/ Opration System	AuthenticAMD Common KVM processor / Linux dev01 2.6.32-50-generic #112-Ubuntu SMP UTC 2013 x86_64 GNU/Linux
RAM	8GB
Language	C, Java(JNI)
Compiler	Visual Studio 2008

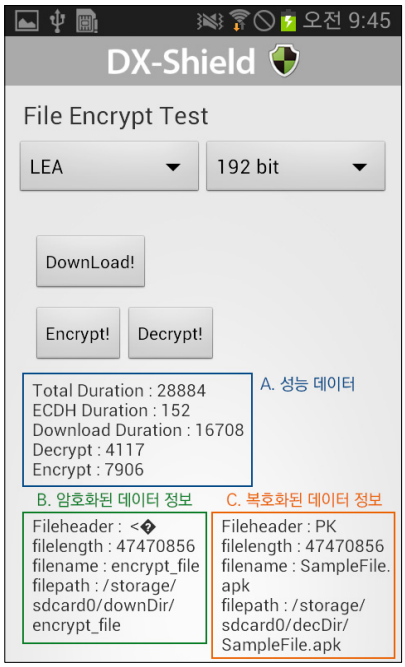


그림 2. DxShield의 구현 화면
Fig. 2. Implementing screen of DxShield

를 수행한다. 성능 비교를 위해 필요한 구현 환경과 소스코드 출처는 표 2, 3과 같다.

2.2 구현 결과

본 항에서는 LEA와 ARIA, SEED의 성능을 비교한다. 상용화되어있는 솔루션 DxShield에 각각의 블록 암호 알고리즘을 적용하여 동작시간을 측정하는 방법으로 테스트를 진행한다. 알고리즘 종류 이외의 데이터는 동일한 조건으로 다음과 같이 설정한다.

- 구현 언어: C, Java(JNI)
- 암호모드: CTR[14]
- 저장 파일 크기: 47MB

DxShield 클라이언트가 키를 생성하고 서버에서 암호화하여 보낸 저장 파일을 복호화하고 다시 암호화하는 동작시간과 암호화와 복호화 각각의 동작시간을 비교하면 표 4와 같다.

표 4. 클라이언트에서의 소요시간(단위:초)
Table 4. The time required by the client(unit: second)

Algorithm		Solution operation time	En/Decryption function operation time	
			Encryption	Decryption
LEA	128	35.89(1)	2.81(1)	2.69(1)
	192	36.86	3.16	2.97
	256	35.39	3.20	3.11
AES	128	44.54(1.24)	6.82(2.42)	6.76(2.5)
	192	45.55	7.73	7.54
	256	46.83	8.60	8.61
ARIA	128	58.74(1.63)	13.85(4.92)	13.80(5.11)
	192	62.53	15.02	14.94
	256	68.71	17.68	17.66
SEED	128	45.08(1.25)	6.64(2.36)	6.52(2.41)

서버가 클라이언트에서 서버의 공개키를 요청받은 시점부터 키 쌍을 생성하고 저장파일을 암호화하여 클라이언트로 전달하는 동작시간과 암호화 함수의 동작시간을 비교하면 표 5와 같다.

LEA를 적용시킨 DxSheild는 SEED를 적용시켰을 때와 비교하여 1.2배의 속도향상을 가져왔고 AES와는 1.2배, ARIA와는 1.6배 높은 성능을 보여주었다. 암/복호화를 하는 테스트에서 암호화일 경우, SEED와는 2.3배, AES와는 2.4배, ARIA와는 4.9배의 성능을 보였고 복호화일 경우, SEED와는 2.4배, AES와는 2.5배, ARIA와는 5.1배의 성능을 보였다.

표 5. 서버에서의 소요시간(단위:초)
Table 5. The time required by the client(unit: second)

Algorithm		Solution operation time	Encryption Function Operation time
LEA	128	28.25(1)	27.97(1)
	192	28.43	28.11
	256	28.48	28.15
AES	128	28.31(1.00)	28.14(1.00)
	192	28.85	28.63
	256	29.09	28.81
ARIA	128	30.62(1.08)	29.88(1.06)
	192	32.78	31.96
	256	32.95	32.07
SEED	128	**31.35(1.10)**	**30.82(1.10)**

서버에서의 암호화 함수 동작시간은 서버의 솔루션 동작 시간과 유사하게 측정되었다. 그 이유는 안드로이드에 파일을 전송할 때 한 번에 큰 용량을 보낼 수가 없어서 저장 파일을 10MB씩 잘라서 암호화하여 전송하였다. 그 결과 암호화 함수 동작시간 안에 파일을 클라이언트로 전송하는 시간이 함께 포함되어 위와 같은 결과가 나왔다. 그러므로 클라이언트의 암/복호화 성능비교를 참조하는 것이 정확한 비교가 될 것으로 보인다.

Ⅲ. 앱 무결성 검사 솔루션에서의 성능 비교

본 절에서는 테스트 대상인 앱 위·변조 방지 솔루션(APP Protect)에 대하여 간략히 기술하고 솔루션이 동작하는 과정에서 사용되는 해시 알고리즘 SHA와 국내에서 최근 개발된 고속 경량 해쉬 알고리즘 LSH의 성능을 비교하고자 한다.

APP Protect는 시스템 위·변조 탐지, APP 위·변조 탐지, 실시간 대응 시스템 연계를 지원한다. APP Protect에 사용된 암호 알고리즘은 다음과 같다.

그림 3은 APP Protect의 전체 흐름도를 보여준다. 그림에서 보라색으로 표시된 구간은 해시함수를 이용하여 무결성 검증값을 생성하는 구간이다. 이 부분에 SHA2와 LSH를 적용하여 성능을 비교 분석한다.

그림 3 APP Protect 솔루션 전체 흐름도를 상세 기술하면 다음과 같다.

① 클라이언트에서 서버로 초기 데이터(공개키)를 요청
② 서버 키 쌍 생성 및 초기 데이터 생성
③ 클라이언트로 초기 데이터(공개키)를 전송
④ 클라이언트 키 쌍 생성
⑤ 해시 알고리즘을 이용한 검증 대상 앱의 Dynamic Hash 추출
⑥ 검증 데이터(시스템 검증 데이터 + Dynamic Hash) 암호화
⑦ 암호화된 검증 데이터를 서버에 전송
⑧ 검증데이터 복호화 후 앱 위•변조 검사(클라이언트에서 생성한 Dynamic Hash와 서버에서 생성한 Dynamic Hash와 비교) 진행
⑨ 검증 결과 암호화
⑩ 암호화된 검증 결과를 클라이언트에 전송
⑪ 암호화된 검증 결과 복호화 후 검증 결과 확인

표 6. APP Protect에서 사용되는 암호 알고리즘
Table 6. cryptography algorithm in APP Protect

Division	Contents
ECDH	E2E-Transmission interval key exchange
SHA-256	Application integrity verification
SEED	E2E-Transmission interval En/Decryption

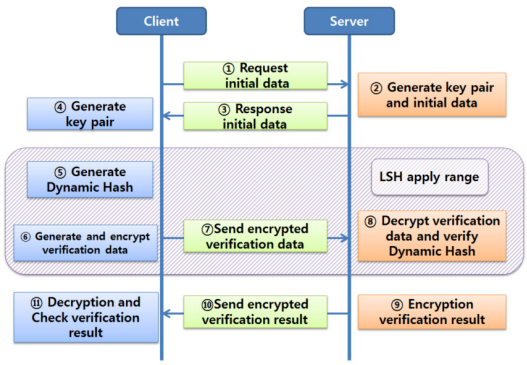


그림 3. APP Protect 전체 흐름도
Fig. 3. APP Protect full flow chart

3.1 구현 환경

구현물은 라이브러리 형태로 제공되는 솔루션이기 때문에 APP Protect가 실행되는 것을 확인하기 위해 그림 4와 같이 안드로이드용 샘플 앱을 만들어 테스트를 수행한다. 성능 비교를 위해 필요한 구현 환경과

그림 4. APP Protect의 구현 화면 (스마트폰, 스마트워치)
Fig. 4. Implementing screen of APP Protect (Smartphone, Startwatch)

표 7. 스마트폰 구현 환경
Table 7. Smartphone implementation environment

CPU/ Opration System	Samsung GALAXY Note3 Samsung Exynos 5 Octa 5420 2.3GHz / Android Kitkat 4.4.2
RAM	3GB
Language	C(ARM NEON), Java(JNI)
Compiler	GCC 4.8.2

소스코드 출처는 표 7~10과 같다.

표 8. 스마트워치 구현 환경
Table 8. Smartwatch implementation environment

CPU/ Opration System	Sony Smartwatch3 ARM Cortex-A7 MPCore 1.2GHz/ Android Lolipop 5.1.1
RAM	512MB
Language	C(ARM NEON), Java(JNI)
Compiler	GCC 4.8.2

표 9. 서버 구현 환경
Table 9. Server implementation environment

CPU/ Opration System	Intel i5-3470 3.20GHz / Windows 7 Ultimate K
RAM	8GB
Language	C(SSSE3), Java(JNI)
Compiler	Visual Studio 2013

표 10. 소스코드 출처
Table 10. code source

Source Code	Source
LSH-224/256/384/512	http://kcryptoforum.or.kr/ Reference Code, Fast Implementation Code
SHA-224/256/384/512	OpenSSL-1.0.2

3.2 구현 결과

본 항에서는 LSH와 SHA2의 성능을 비교한다. 성능 비교는 APP Protect에 각각의 해시 알고리즘을 적용하여 무결성을 검증하는 동작시간을 측정하는 방법으로 테스트 하였다.

알고리즘 종류 이외의 데이터는 동일한 조건으로 다음과 같이 설정하였다.

- 구현 언어: C, Java(JNI)
- 앱 파일 크기: 20MB
- 서버 최적화 수준: 레퍼런스, 범용, SSSE3
- 서버 최적화 수준: 레퍼런스, 범용, NEON

APP Protect 클라이언트가 키를 생성하고 무결성 검증 값을 보내는데 소요된 시간과 해시함수의 동작시간을 비교하면 표 11과 같다.

서버가 클라이언트의 초기 데이터 요청을 받은 시점부터 마지막 결과를 클라이언트에 전달되는 시점까지의 시간과 해시함수의 동작시간을 비교하면 표 12와 같다.

LSH를 적용시킨 APP Protect 전체 속도는 SHA2와 비교하여 LSH 범용 소스코드의 경우 1.4배, LSH

표 11. 클라이언트에서의 소요시간(단위:초)
Table 11. The time required by the client(unit: second)

Algorithm		Solution operation time		Hash function operation time	
		Smartphone	Smartwatch	Smartphone	Smartwatch
SHA	224	0.81	1.94	0.79	1.87
	256	**0.80 (2.66)**	**1.98 (2.17)**	**0.78 (2.68)**	**1.91 (2.48)**
	384	1.26	3.28	1.25	3.19
	512	1.27	3.37	1.26	3.28
LSH common	224	0.70	2.13	0.69	2.08
	256	0.71 (2,36)	2.18 (2.39)	0.70 (2.41)	2.09 (2.71)
	384	1.29	3.91	1.27	3.81
	512	1.26	3.94	1.24	3.85
LSH simd_opt	224	-	-	-	-
	256	**0.3 (1)**	**0.91 (1)**	**0.29 (1)**	**0.77 (1)**
	384	-	-	-	-
	512	0.37	1.17	0.36	0.99

표 12. 서버에서의 소요시간(단위:초)
Table 12. The time required by the server(unit: second)

Algorithm		Smart phone solution	Smart watch solution	Hash function operation time
SHA	224	1.25	2.32	0.17
	256	**1.28 (2.37)**	**2.32 (2.03)**	**0.17 (3.4)**
	384	1.81	3.71	0.25
	512	1.88	3.80	0.25
LSH common	224	0.99	2.43	0.12
	256	1.00 (1.85)	2.48 (2.17)	0.12 (2.4)
	384	1.64	4.25	0.16
	512	1.58	4.29	0.16
LSH simd_opt	224	-	-	-
	256	**0.54(1)**	**1.14(1)**	**0.05(1)**
	384	-	-	-
	512	0.61	1.42	0.06

SIMD 적용 소스코드의 경우 2.4~3배 빠른 성능을 보인다. APP Protect에 적용한 경우, 클라이언트 해시 알고리즘 테스트에서 LSH 범용 소스코드는 SHA보다 0.85~1.15배, LSH SIMD 적용 소스코드는 2.5~3.5배 빠른 성능을 보였다. APP Protect에 적용한 경우, 서버 해시 알고리즘 테스트에서 LSH 범용 소스코드는 SHA보다 1.4~1.5배, LSH SIMD 적용 소스코드는 3.4~4.1배 빠른 성능을 보였다.

Ⅳ. 결 론

본 논문에서는 앱 위·변조 방지 솔루션에서 사용되는 암호 알고리즘들과 최근 국내에서 개발된 고속 경량 암호 알고리즘들 LEA와 LSH의 성능을 비교하였다. LEA를 적용시킨 DxShield 솔루션의 동작 시간은 AES를 적용시켰을 때와 비교하여 1.2배, ARIA와는 1.6배, SEED와는 1.2.배의 높은 성능을 보여주었다. LSH를 적용시킨 APP Protect 솔루션의 동작 시간은 SHA를 적용시켰을 때와 비교하여 LSH 범용 소스코드의 경우 1.4배, LSH SIMD 적용 소스코드의 경우 2.4~3배 빠른 성능을 보여주었다.

비교표에서 알 수 있듯이 암호 알고리즘의 성능은 솔루션 전체 동작시간에 대비해서 얼마나 큰 시간적 비중을 차지 하냐에 따라서 솔루션 전체 동작시간에 영향을 준다. 특히 앱 무결성을 검사하는 솔루션 APP Protect에서는 해시함수에 소요되는 시간이 매우 높은 비율을 차지한다. 테스트 결과 고속 경량 암호 알고리즘의 적용은 클라이언트의 처리속도를 빠르게 하여 사용자의 편의성을 높여주고 서버에서의 부하를 감소시켜 전체적으로 비용감소에도 도움이 될 것으로 기대된다.

References

[1] K. Y. Park and J. H. Seo, “Research on information security technology trends smartphone,” in *Proc. KICS Int. Conf. Commun.*, pp. 305-306, Nov. 2014.

[2] D. Kim, S. Yoon, and Y. Lee “Security for IoT services,” *KICS Inf. and Commun. Mag.*, vol. 20, no. 8, pp. 53-59, Jul. 2013.

[3] B. Choi, H. J. Shim, C. H. Lee, S. Cho, and S. Cho, “An APK overwrite scheme for preventing modification of android applications,” *J. KICS*, vol. 39, no. 8, pp. 309-316, May 2014.

[4] J.-W. Kim and I. Kim, “The counterfeit & modulation checks on the android applications,” in *Proc. KITS Int. Conf. Commun.*, pp. 460-464, 2012.

[5] TTAS, *128-bit Block Cipher SEED*, TTAS.KO-12.0004/R1, Dec. 2005.

[6] NIST, *Announcing the Advanced Encryption Standard(AES)*, FIPS PUB-197, Nov. 2002.

[7] KS, *128 bit block encryption algorithm ARIA*, KS X 1213:2004, Dec. 2004.

[8] D. Hong, J.-K. Lee, D.-C. Kim, D. Kwon, K. H. Ryu, and D.-G. Lee, “LEA: A 128-bit block cipher for fast encryption on common processors,” in *Proc. WISA 2013, LNCS*, vol. 8269, 2014.

[9] J.-Y. Lee, D.-E. Cho, and J.-Y. Lee, “An integrity verification method for secure application on the smartphone,” *KIIT*, vol. 9, no. 10, pp. 223-228, Oct. 2011.

[10] S. Kim, S. Kim, and D. H. Lee, “A study on the vulnerability of integrity verification functions of android-based smartphone banking applications,” *JKIISC*, vol. 23, no. 4, pp. 743-755, Aug. 2013.

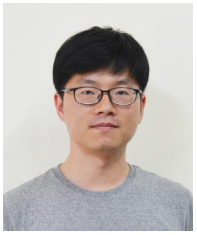
[11] NIST, *Secure Hash Standard*(SHS), FIPS 180-4, Aug. 2015.

[12] D.-C. Kim, D. Hong, J.-K. Lee, W.-H. Kim, and D. Kwon, “LSH: A new fast secure hash function family,” *Inf. Security and Cryptology, ICISC 2014*, pp. 286-313, Jan. 2015.

[13] Daniel R. L. Brown, *Standards for efficient cryptography, SEC 1: Elliptic curve cryptography*, Certicom Res., May 2009.

[14] NIST, *Recommendation for block cipher modes of operation: Methods and techniques*, SP 800-38A, 2001.

이 준 영 (Jun Yoeng Lee)



2008년 2월 : 명지대학교 컴퓨터소프트웨어학과 졸업

2014년 2월~현재 : 세종사이버대학교 정보호호대학원 석사과정

<관심분야> 모바일보안, 산업보안, 암호프로토콜

장 남 수 (Nam Su Chang)

2002년 2월 : 서울시립대학교 수학과 학사 졸업

2004년 8월 : 고려대학교 정보보호대학원 석사 졸업

2010년 2월 : 고려대학교 정보경영공학전문대학원 박사 졸업

2010년 7월~현재 : 세종사이버대학교 정보보호학과 조교수

<관심분야> 공개키 암호 알고리즘, 공개키 암호 암호분석, 암호칩 설계 기술, 부채널 공격