Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.29, NO.5, Oct. 2019 ISSN 2288-2715(Online)
https://doi.org/10.13089/JKIISC.2019.29.5.985

FIPS-OpenSSL 코드 분석을 통한 암호모듈 자가시험 보안요구사항 분석*

서 석 충†‡
국민대학교

Study on Selftest Requirements in Cryptographic Module Validation Program with FIPS-OpenSSL Source Code Analysis*

Seog Chung Seo†‡
Kookmin University

요 약

본 논문에서는 미국에서 암호모듈검증을 획득한 대표적인 소프트웨어 형태의 암호모듈인 FIPS-OpenSSL의 소스코드를 분석하여 암호모듈 검증 및 시험기준에서 요구하는 보안요구사항이 소프트웨어적으로 어떻게 구현되어있는지 분석한다. 특히, 암호모듈이 반드시 탑재하고 있어야 하는 자가시험기능 (동작 전 자가시험, 조건부 자가시험) 관점으로 소스코드를 분석한다. 비록 미국 암호모듈 검증제도에서는 FIPS 140-2를 검증기준으로 삼고 있지만, FIPS 140-2는 국내 암호모듈 검증제도에서 암호모듈 검증 및 시험기준으로 삼고 있는 암호모듈 국제표준인 ISO/IEC 19790과 24759의 근간이 되었기 때문에 많은 유사함이 존재한다. 본 논문의 분석을 통하여 향후 암호모듈 개발업체에서 자가시험기능을 정확하고 안전하게 구현할 수 있을 것으로 기대한다.

ABSTRACT

This paper analyzes the source code of FIPS-OpenSSL cryptographic module approved as FIPS cryptographic module in USA and shows how the selftest requirements are implemented as software cryptographic library with respect to pre-operational test and conditional tests. Even though FIPS-OpenSSL follows FIPS 140-2 standard, lots of security requirements are similar between FIPS 140-2 and Korean cryptographic module validation standards. Therefore, analysis from this paper contributes to help Korean cryptographic module vendors develop correct and secure selftest functions on their own cryptographic modules, which results in reducing the test period.

Keywords: Cryptographic Modules (CMs), Cryptographic Module Validation Program (CMVP), SelfTest, Conditional SelfTest, Power-on SelfTest

I. 서 론

미국은 1995년부터 연방정부에서 소통되는 중요 데이터의 보호를 위해 암호모듈을 사용하도록 제한하고 있으며 암호모듈의 안전성을 검증하기 위해 암호모듈검증제도(CMVP: Cryptographic Module Validation Program)을 운영하고 있다 [1]. 국내에서는 2005년부터 KCMVP (Korean CMVP) 제도를 운영하여 국가 및 공공기관에서 사용되는 암호모듈의 안전성을 검증하고 있다 [6]. 미국의 경우에는 현재 FIPS 140-2 기준을 이용하여 암호모듈이 갖추어야 할 보안요구사항을 정의하고 있으며 국

Received(07. 22. 2019), Accepted(09. 03. 2019)
* 이 논문은 2019년도 정부의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(No.2019R1F1A1058494)
† 주저자, scseo@kookmin.ac.kr
‡ 교신저자, scseo@kookmin.ac.kr(Corresponding author)

내의 경우에는 암호모듈 국제표준인 ISO/IEC 19790과 24759를 바탕으로 작성된 KS X ISO/IEC 19790:2015와 24759:2015를 암호모듈 검증기준과 시험기준으로 삼고 있다 [2, 3, 4, 5]. 실제로 암호모듈 국제표준인 ISO/IEC 19790과 24759는 FIPS 140-2를 바탕으로 작성되었기 때문에 미국과 국내의 검증기준 및 시험기준 상에 많은 유사성이 존재한다. 뿐만 아니라, 2021년 10월부터 미국 암호모듈 검증기준으로 사용될 FIPS 140-3 기준은 국제표준인 ISO/IEC 19790:2012와 ISO/IEC 24759:2017을 개정하여 개발될 것이기 때문에 국내 암호모듈 기준과 그 유사성이 더해질 것으로 예측된다.

암호모듈 검증기준과 시험기준은 오랜 기간 전문가들에 의해 개발되어 왔다. 하지만, 보안요구사항과 시험요구사항의 단순 설명으로 구성된 표준문서의 한계로 인하여 암호모듈 개발업체들이 보안요구사항을 실제 소프트웨어 또는 하드웨어 형태로 구현하는데 많은 어려움을 호소하고 있다. 미국과 국내의 대부분의 암호모듈 개발업체는 검증받은 암호모듈이 해당 업체들의 자산이기 때문에 소스코드를 공개하지 않고 있다. 하지만, 대표적인 보안통신 암호라이브러리인 OpenSSL의 경우에 2015년 미국 암호모듈검증인 FIPS 인증을 획득한 후 FIPS 버전의 OpenSSL인 FIPS-OpenSSL의 소스코드를 공개하여 다른 암호모듈 개발업체에서도 암호모듈 개발 시 참고하도록 하고 있다 [7]. 특히, FIPS-OpenSSL은 2015년 이후 버전을 업그레이드하면서 윈도우, 리눅스, AIX, 모바일 환경 등 다양한 운영환경에서 테스트되면서 대표적인 암호모듈로 자리잡았다. 뿐만 아니라, 미국의 경우에는 FIPS-OpenSSL를 기반으로 많은 수의 암호모듈이 개발되어 인증받고 있는 추세이다.

본 논문에서는 FIPS-OpenSSL의 소스코드를 분석하여 암호모듈 검증 및 시험기준에서 요구하는 보안요구사항이 소프트웨어적으로 어떻게 구현되어있는지 분석한다. 특히, 암호모듈이 반드시 탑재하고 있어야 하는 자가시험기능(동작 전 자가시험, 조건부 자가시험)에 대해 소스코드를 분석한다. 국내의 경우에는 공개되어있는 검증필 암호모듈의 소스코드가 없기 때문에 개발업체에서 검증기준과 시험기준을 바탕으로 암호모듈을 직접 개발해야한다. 따라서, 기준을 잘못 이해하여 구현하는 경우가 많으며 이로 인해 암호모듈 시험기간이 크게 늘어나는 경우가 매우 빈번하다. 본 논문의 분석을 통하여 향후 암호모듈 개발업체들이 자가시험기능을 정확하고 안전하게 구현할 수 있을 것으로 기대한다.

본 논문은 다음과 같이 구성된다. 2장과 3장에서는 각각 FIPS-OpenSSL 분석을 위한 방법론과 FIPS-OpenSSL의 전반적인 구조에 대해 설명한다. 4장에서는 자가시험 보안요구사항들이 소프트웨어로 어떻게 구현되어있는지를 분석한다. 5장에서는 FIPS 140-2의 자가시험과 KS X ISO/IEC 19790 자가시험내용을 비교하고 6장에서 결론을 맺는다.

II. FIPS-OpenSSL 분석 방법론

OpenSSL은 소스코드가 방대하고 EVP, 함수포인터, 심볼 정의들이 매우 복잡하기 때문에 단순 소스코드 분석만으로는 실행흐름을 파악하기 어렵다. 따라서, 본 논문에서는 FIPS-OpenSSL의 실행흐름을 디버깅하여 내부 동작을 파악한다. 암호모듈의 내부 동작을 디버깅하기 위해 윈도우즈 운영환경에서 FIPS-OpenSSL v2.0.16 소스코드를 디버깅하도록 빌드하였으며 내부변숫값 확인을 위하여 최적화 옵션을 생략하였다. 구체적인 환경 설정 방법은 다음과 같다.

2.1 설치 요구 사항

FIPS-OpenSSL 암호모듈을 디버깅하기 위해 필요한 설치 요구사항은 [Table 1]과 같다.

먼저 개발환경인 visual studio 2008을 설치한 후, nasm 어셈블러, perl 스크립터 순서로 설치를 진행한다. 위의 사항들을 설치한 후 패스를 등록해야 한다. 패스는 윈도우 환경에서 내컴퓨터→속성→고급

Table 1. Required installation files

Type	File version
FIPS-OpenSSL	openssl-fips-2.0.16.tar.gz
nasm assembler [8]	nasm-2.13.03-installer-x86.exe 또는 nasm-2.13.03-installer-x64.exe
perl scriptor [9]	padre-on-strawberry-5.12.3.0-v5.exe
develop tool	visual studio 2008

→사용자변수 및 시스템변수에 등록 가능하다. 먼저, visual studio에서 제공하는 nmake의 패스인 "C:\Program Files\Microsoft Visual Studio 9.0\VC\bin;"을 등록한다. 다음으로 nasm의 패스인 "C:\Program Files\NASM;"를 등록한다. 마지막으로 Perl의 패스인 "C:\Strawberry\perl\bin;C:\Strawberry\perl\site\bin;C:\Strawberry\c\bin;"를 등록한다.

2.2 FIPS-OpenSSL 빌드 방법

다운받은 openssl-fips-2.0.16.tar.gz 파일의 압축을 풀면 〔Fig. 1〕의 결과가 생성된다. ms 폴더에는 운영환경별 빌드 스크립트가 존재하고 빌드 결과는 out32dll 폴더에 생성된다. 또한, crypto와 fips 폴더에는 암호 프리미티브와 암호모듈 관련 소스코드들이 들어있다. ms 폴더 안에 들어있는 do_fips.bat 파일은 FIPS-OpenSSL을 빌드하는 스크립트로서 기본적으로 최적화 옵션이 들어가 있기 때문에 디버깅 시 변수의 값을 확인할 수 없다. 따라서, do_fips.bat 스크립트를 수정하여 최적화 옵션을 해제해야 한다. do_fips.bat 스크립트를 분석해 보면 66번째 라인에서 "perlutil\mk1mf.pl dll%ASM% %TARGET% 〉ms\%MFILE%"의 실행문이 있으며 다시 mk1mf.pl 스크립트를 분석하면 208번째 라인에서 "VC-32.pl"을 호출하고 있다. VC-32.pl 스크립트를 분석하면 145번째 라인에서 운영환경에 대한 최적화 옵션이 "$opt_cflags=$f.' /Ox /O2 /Ob2';"와 같이 적용되어 있다. 따라서, 암호모듈 내부 값 및 변수에 대한 디버깅을 수행하기 위해서는 "$opt_cflags=$f.'';"로 최적화 옵션을 적용하지 않아야 한다.

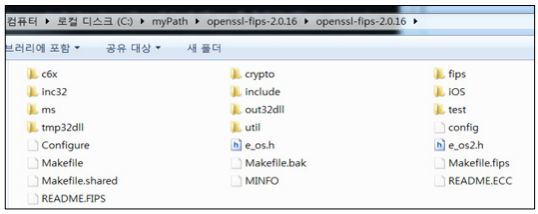

Fig. 1. Result of decompression

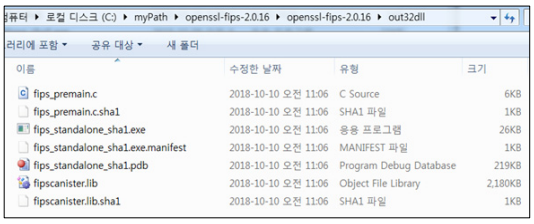

Fig. 2. Result of FIPS-OpenSSL Build

do_fips.bat 스크립트에서 최적화 옵션을 제거한 후에는 윈도우 컴파일 환경에서 스크립트를 실행하여 FIPS-OpenSSL 암호모듈을 빌드해야 한다. 32-bit 버전의 경우 vcvar32.bat를 이용하고, 64-bit 버전의 경우 vcvar64.bat를 이용한다 (C:\Program Files\Microsoft Visual Studio 9.0\VC\bin에 위치함). 32-bit 환경에서 빌드하기 위해서는 CMD 창에서 vcvar32.bat를 실행한 후, fips-openssl 폴더의 ms 폴더로 이동한 후, "do_fips.bat debug"를 실행한다. 빌드 스크립트가 정상적으로 동작하면 〔Fig. 2〕와 같은 파일들이 out32dll 폴더에 생성된다. fips_premain.c 파일은 향후 암호모듈을 사용하는 어플리케이션에서 암호모듈의 무결성 검증을 위해 함께 포함하여 사용해야 하는 파일이며 fipscanister.lib 파일이 실제 빌드 과정을 통해 생성된 FIPS-OpenSSL 암호모듈 파일이다.

2.3 디버깅을 위한 Visual Studio 2008 설정

Visual Studio 2008을 이용하여 FIPS-OpenSSL을 디버깅하기 위해서는 헤더파일과 라이브러리에 대한 설정을 해야한다. 다음과 같이 헤더파일 경로와 라이브러리 종속성을 설정해야 한다. 헤더파일 경로는 "openssl-fips-2.0.16\include"

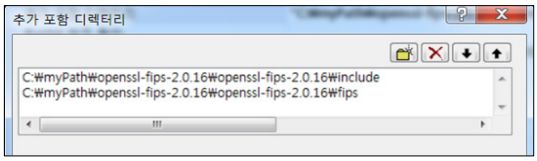

Fig. 3. Header file configuration

Fig. 4. Library file configuration

와 "openssl-fips-2.0.16\fips"이며, 참조하는 라이브러리 이름은 fipscanister.lib이다 (각각 〔Fig. 3.〕과 〔Fig. 4.〕 참조). 또한, FIPS-OpenSSL 암호모듈을 이용하는 응용프로그램은 〔Fig. 2〕의 fips_premain.c 파일을 함께 빌드해야 암호모듈에 대한 무결성 검증이 올바르게 수행된다. 즉, fips_premain.c 파일의 FINGERPRINT_premain(void) 함수내부에서는 main 함수 시작 전 암호모듈에 대한 서명값을 비교하는 암호모듈무결성 검증을 수행한다.

III. FIPS-OpenSSL 명세

3.1 암호모듈 보안수준 및 암호모듈 경계

FIPS-OpenSSL은 기존의 OpenSSL 라이브러리에 포함된 crypto 부분에 대해 FIPS 보안수준 1을 만족하도록 FIPS 기능을 추가한 것으로서 윈도우즈, 리눅스, AIX 등을 포함한 대부분의 운영환경에서 암호기능을 제공한다 ("역할, 서비스 및 인증". "설계보증" 영역은 각각 보안수준 2와 3을 만족한다). 〔Fig. 5.〕는 FIPS-OpenSSL 암호모듈의 경계에 대한 도식이다 〔6〕. 암호모듈은 외부 API 인터페이스를 통해 응용 프로그램에게 암호서비스를 제공하며, 응용 프로그램은 암호모듈을 이용하여 키와 같은 핵심보안매개변수를 생성하여 유지할 수 있다. 다만, 암호모듈 내부에서 임시로 생성되는 보안관련 정보들은 사용이 완료된 후 제로화 된다.

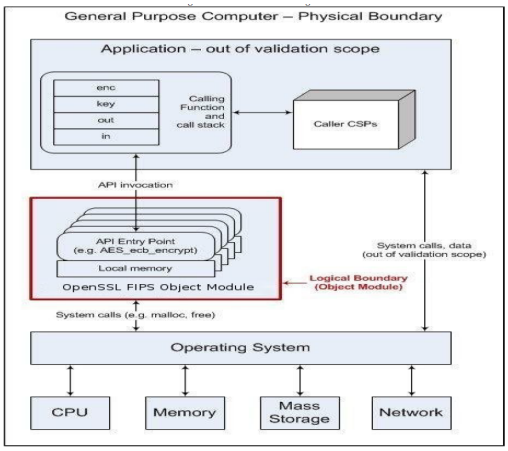

Fig. 5. Cryptographic Boundary 〔7〕

3.2 탑재 알고리즘

〔Table 2〕는 FIPS-OpenSSL에 탑재된 FIPS 승인 암호알고리즘 목록이다. KCMVP 검증대상 암호알고리즘 목록과 달리, SHA-1을 승인 암호알고리즘 목록에 포함하고 있으며, 공개키 기반 알고리즘의 경우, RSA는 4096 비트까지의 키 길이를 지원한다. 또한, DSA 역시 3072 비트까지의 키 길이를 지원하고 있다(국내 KCMVP에서는 RSA와 KCDSA의 경우 각각 3072 비트, 2048 비트 키 길이 까지만 지원함). 또한, 타원곡선 기반 전자서명 및 키 설정 알고리즘의 경우, 국내 KCMVP에서 지원하지 않는 P-384/512, K-409/571, B-409/571까지 지원하고 있다. 대칭키 암호알고리즘의 경우 KCMVP에서 지원하지 않는 파일시스템 암호화 운영모드인 XTS 모드를 지원한다.

Table 2. Approved Algorithms in FIPS-OpenSSL 〔7〕

Service	Algorithm	Option
Random Bit Generator	Hash_DRBG, HMAC_DRBG, CTR_DRBG (AES)	Supporting Predicttion
Symmetric-Key Algorithm	DES	3-key TDES TECB, TCBC, TCFB, TOFB, CMAC
	AES	ECB, CBC, OFB, CTR, XTS, CCM, GCM, CMAC
Hash function	SHA-1, SHA-2	
Keyed Hash	HMAC	SHA-1, SHA-2
Public-Key Algorithm	RSA	2048/3072/4096 with SHA-2
Digital Signature	RSA	2048/3072/4096 with SHA-2
	DSA	1024/2048/3072 with SHA-2
	ECDSA	P-224/256/384/512 K-233/283/409/571 B-233/283/409/571
Key Establishment	ECDH	P-224/256/384/512 K-233/283/409/571 B-233/283/409/571

3.3 FIPS 모드 구동

FIPS-OpenSSL을 FIPS 모드로 구동(국내 암호모듈 검증기준의 검증대상 동작모드와 동일)하기 위해서는 "FIPS_module_mode_set(int onoff, const char* auth)" 함수를 호출해야하며, onoff="1", auth="Default FIPS Crypto Officer Password" 또는 "Default FIPS Crypto User Password"를 입력해야한다. FIPS-OpenSSL은 "역할, 서비스 및 인증" 보안요구사항에 대해 보안수준 2를 만족하고 있기 때문에 사용자 인증 메커니즘을 제공한다. 따라서, 암호모듈 관리자 또는 사용자로서 암호모듈을 구동하고자 할 경우, "FIPS_module_mode_set()" 함수 호출 시, 사전에 정의된 패스워드를 입력해야 한다. "FIPS_module_mode_set()"의 주요동작은 "fips_check_auth()"를 통한 사용자 인증과 "FIPS_selftest()"를 통한 전원인가 자가시험이다. [Fig. 6.]은 암호모듈 내부에 정의된 운영자 별 패스워드와 "fips_check_auth()"에서 수행되는 패스워드 체크 메커니즘이다. 암호모듈 내부에서 사전에 정의된 패스워드를 사용하여 인증을 수행하고 있음을 확인할 수 있다.

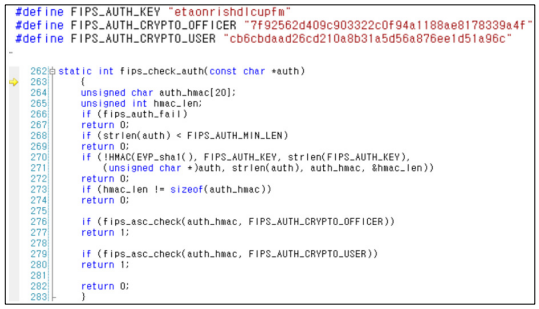

Fig. 6. FIPS-OpenSSL Authentication Mechanism

3.4 암호모듈 상태관리

FIPS-OpenSSL 암호모듈은 내부 전역변수를 이용하여 FIPS 모드와 암호모듈에 대한 동작을 관리한다. 즉, [Fig. 7.]과 같이 암호모듈의 상태를 관리하는 변수가 fips.c 파일에 static 변수로서 정의되어 있다. "fips_auth_fail" 변수는 인증 수행에 대한 성공/실패 여부 정보를 관리하며 "fips_mode" 변수는 암호모듈이 현재 FIPS 모드로 동작 중인지의 정보를 관리한다.

"fips_selftest_fail" 변수는 자가시험 수행에 대한 성공/실패 여부 정보를 관리하며, 암호모듈은 FIPS로 시작하는 외부 암호서비스 API의 초반에 자가시험 성공여부를 확인하여 자가시험이 실패인 경우 서비스를 제공하지 않는다. 즉, [Fig. 8.]과 같이 FIPS 외부 API에서는 "FIPS_selftest_failed()"를 호출하여 서비스 수행 전 자가시험 수행에 대한 성공여부를 확인한다.

FIPS-OpenSSL 암호모듈에서는 상태관리를 통하여 암호모듈이 정상적으로 동작하는지의 여부를 확인하고 있으며 정상동작 상태에서만 응용 프로그램에게 암호서비스를 제공한다.

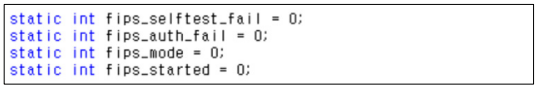

Fig. 7. State control variables in fips.c file

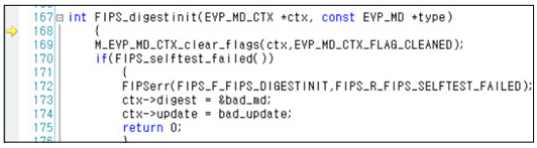

Fig. 8. State Check before executing cryptographic service (In case of Message Digest Function)

IV. FIPS-OpenSSL 자가시험 기능 분석

본 절에서는 FIPS-OpenSSL이 수행하는 자가시험에 대해 분석한다. 암호모듈 검증기준에 따르면 암호모듈은 암호서비스를 수행하기 전에 암호모듈 자체가 변조되었는지 확인해야하고 또한, 각 암호 서비스가 정상적으로 동작하고 있는지를 스스로 검사할 수 있어야 한다. FIPS 140-2에서 요구하는 자가시험은 크게 전원인가 자가시험 (Power-on SelfTests)와 조건부 자가시험 (Conditional Tests)으로 구성되며 FIPS-OpenSSL에서는 암호모듈이 초기화될 때 전원인가 자가시험을 수행하고, 공개키 암호 키 쌍 생성 또는 난수발생기를 통한 난수 생성 시 조건부 자가시험을 수행하고 있다.

4.1 전원인가 자가시험

암호모듈이 초기화될 때 수행되는 전원인가 자가시험에서는 암호모듈에 대한 무결성을 확인하는 소프트웨어 무결성 시험과 암호모듈에 탑재된 각 승인 알고리즘들에 대한 KAT (Known Answer Test) 또는 PCT (Pairwise Consistency Test)시험이 수행된다. [Table 3]은 FIPS-OpenSSL에서 수행되는 전원인가 자가시험의 목록이다.

Table 3. Power-on SelfTests executed by FIPS-OpenSSL (KAT: Known Answer Test, PCT: Pairwise consistency test) [7]

Algorithm	Type	Methods
Software Integrity	KAT	Verifying integrity value of Cryptographic module with HMAC-SHA1
AES	KAT	Separate Encrypt/Decrypt KAT with 128-bit key length
T-DES	KAT	Separate Encrypt/Decrypt KAT with 3-key
AES CMAC	KAT	Sign and verify CBC mode with 128/192/256 key lengths
TDES CMAC	KAT	CMAC generation and verify with CBC mode 3-key
AES CCM/ GCM	KAT	Separate Encrypt/Decrypt KAT with 192-bit key length(CCM) and 256-bit key length(GCM)
XTS-AES	KAT	XTS-AES-128 and XTS-AES-256
HMAC	KAT	One KAT per SHA1, SHA224, SHA256, SHA384, and SHA512
RSA	KAT	Sign and verify with 2048-bit key and SHA-256
DSA	PCT	Sign and verify using 2048-bit key, SHA384
DRBG	KAT	CTR_DRBG: AES-256(with/without derivation function) HASH_DRBG: SHA-256 HMAC_DRBG: SHA-256
ECDSA	PCT	Keygen, sign, and verify with P-224, K-233 and SHA-512
ECC CDH	KAT	Shared secret computation

소프트웨어 무결성(Software Integrity) 시험은 사전에 HMAC-SHA1을 이용하여 계산된 무결성 값과 현재 암호모듈의 무결성값을 비교하여 암호모듈의 무결성이 훼손되었는지를 검증한다. DSA, ECDSA를 제외하고는 KAT (Known Answer Test) 시험이 수행되며 이는 암호모듈에 하드코딩되어 정의되어 있는 입력(평문, 키)에 대해 암호 연산의 수행결과가 사전 계산되어 저장된 결과와 일치하는지를 확인한다. 대칭키 암호, 공개키 암호와 같이 역함수가 존재하는 경우에는 암호화와 복호화 각각에 대한 KAT 시험을 수행하고, 역함수가 존재하지 않는 HMAC, DRBG 등의 경우에는 생성함수에 대한 KAT 시험만을 수행한다. DSA, ECDSA에 대해서는 PCT (Pairwise Consistency Test)가 수행되며, 이는 특정 메시지에 대해 서명을 생성한 후, 생성된 서명에 대해 검증이 올바르게 동작하는지를 확인하는 형태로 수행된다. RSA는 KAT 시험을 수행하는데 사전에 정의된 메시지에 대해 생성된 서명이 이미 계산되어 저장된 값과 일치하는지를 확인하고 다시 서명 검증을 수행하여 성공하는지를 확인한다.

암호알고리즘에 대한 KAT는 [Fig. 9.]과 같이 세 단계로 구성된다. 첫 번째는 알고리즘 별 테스트벡터 타입에 대한 정의로서 알고리즘 KAT를 수행하는데 필요한 데이터들의 타입을 정의한다. 예를 들어, AES 알고리즘의 경우, 평문, 암호문, 키에 대한 데이터 자료형 정의가 필요하다. 두 번째는 실제 테스트벡터 값들에 대한 정의이다. 즉, 첫 번째 단계에서 선언한 자료형에 대해 실제 데이터 값을 정의하는 부분이다. 세 번째는 앞선 두 단계에서 정의한 데이

```
// (1) testvector type declaration
typedef struct {
    // testvector elements
}algorithmname_KAT;

// (2) testvector value definition
static const algorithmname_KAT vector[] = {
    // testvector values
};

// (3) KAT procedure
int FIPS_selftest_algorithmname()
{
    // verifying the result of KAT
    for(n = 0; n < numOfTVs; n++){
        if(fips_algorithmname_test()){
            goto err;
        }
    }
}
```

Fig. 9. KAT structure in FIPS-OpenSSL

터 타입과 값을 이용하여 실제 KAT 과정을 수행하는 부분이다. 정의되어있는 테스트벡터 개수에 대해 for 루프를 구동하여 KAT 시험을 수행하고 오류가 발생한 경우 자가시험 실패 오류코드를 반환한다.

다음에서 주요 알고리즘들에 대한 세부 자가시험을 확인한다.

4.1.1 AES KAT 시험

FIPS-OpenSSL에서는 AES 블록암호에 대해 ECB, CBC, CTR을 비롯한 다양한 운영모드를 탑재하고 있으나 ECB 모드만을 이용하여 KAT 시험을 수행한다. [Fig. 10.]는 fips_aes_selftest.c 파일에 정의된 AES ECB 모드 KAT에 대한 테스트벡터 정의이다. [Fig. 11.]은 AES에 ECB 모드에 대해 정의된 테스트벡터 수 만큼 KAT 시험을 수행하는 코드이다. fips_cipher_test() 함수([Fig. 12.])를 호출하여 실제 KAT 시험을 수행한다. [Fig. 12.]은 fips_cipher_test() 함수의 내부코드이며, 암호화 연산에 대한 KAT 시험과 복호화 연산에 대한 KAT 시험을 수행하는 것을 확인할 수 있다.

```
static const struct
    {
    const unsigned char key[16];
    const unsigned char plaintext[16];
    const unsigned char ciphertext[16];
    } tests[]=
    {
    {
    { 0x00,0x01,0x02,0x03,0x04,0x05,0x06,0x07,
      0x08,0x09,0x0A,0x0B,0x0C,0x0D,0x0E,0x0F },
    { 0x00,0x11,0x22,0x33,0x44,0x55,0x66,0x77,
      0x88,0x99,0xAA,0xBB,0xCC,0xDD,0xEE,0xFF },
    { 0x69,0xC4,0xE0,0xD8,0x6A,0x7B,0x04,0x30,
      0xD8,0xCD,0xB7,0x80,0x70,0xB4,0xC5,0x5A },
    },
    };
```

Fig. 10. Testvector definition for AES KAT in fips_aes_selftest.c

```
int FIPS_selftest_aes()
    {
    int n;
    int ret = 0;
    EVP_CIPHER_CTX ctx;
    FIPS_cipher_ctx_init(&ctx);

    for(n=0 ; n < 1 ; ++n)
    {
    if (fips_cipher_test(FIPS_TEST_CIPHER, &ctx, EVP_aes_128_ecb(),
                tests[n].key, NULL,
                tests[n].plaintext,
                tests[n].ciphertext,
                16) <= 0)
        goto err;
    }
    ret = 1;
    err:
    FIPS_cipher_ctx_cleanup(&ctx);
    if (ret == 0)
        FIPSerr(FIPS_F_FIPS_SELFTEST_AES,FIPS_R_SELFTEST_FAILED);
    return ret;
    }
```

Fig. 11. KAT test procedure

```
if (!fips_post_started(id, subid, NULL))
    return 1;
if (FIPS_cipherinit(ctx, cipher, key, iv, 1) <= 0)
    goto error;
if (!FIPS_cipher(ctx, citmp, plaintext, len))
    goto error;
if (memcmp(citmp, ciphertext, len))
    goto error;
if (!fips_post_corrupt(id, subid, NULL))
    citmp[0] ^= 0x1;
if (FIPS_cipherinit(ctx, cipher, key, iv, 0) <= 0)
    goto error;
FIPS_cipher(ctx, pltmp, citmp, len);
if (memcmp(pltmp, plaintext, len))
    goto error;
rv = 1;
```

Fig. 12. Internal codes of fips_cipher_test() function

4.1.2 HMAC KAT 시험

HMAC 알고리즘은 역함수가 존재하지 않기 때문에 HMAC 생성에 대한 KAT 시험만을 수행한다. [Fig. 13.]는 fips_hmac_selftest.c 파일에 정의된 HMAC KAT를 위한 테스트벡터이다. HMAC에 대한 KAT 시험을 수행하는 FIPS_selftest_hmac() 함수는 AES KAT 시험 방법과 유사하게 정해진 수의 테스트벡터에 대해 for 루프를 구동하면서 메시지를 입력으로 넣어 출력된 HMAC 값을 사전에 정의된 HMAC 값과 비교하여 시험을 수행한다.

나머지 대칭키 기반 알고리즘들인 CMAC, AES-CCM, AES-GCM, AES-XTS, DES에 대해서도 [Fig. 9.]에서 명시된 방법을 준수하여 KAT를 수행한다. 각 알고리즘에 대한 KAT 시험

```
static const HMAC_KAT vector[] = {
    {   NID_sha1,
    { 0x09,0x22,0xd3,0x40,0x5f,0xaa,0x3d,0x19,
      0x4f,0x82,0xa4,0x58,0x30,0x73,0x7d,0x5c,
      0xc6,0xc7,0x5d,0x24 }
    },
    {   NID_sha224,
    { 0xdd,0xef,0x0a,0x40,0xcb,0x7d,0x50,0xfb,
      0x6e,0xe6,0xce,0xa1,0x20,0xba,0x26,0xaa,
      0x08,0xf3,0x07,0x75,0x87,0xb8,0xad,0x1b,
      0x8c,0x8d,0x12,0xc7 }
    },
    {   NID_sha256,
    { 0xb8,0xf2,0x0d,0xb5,0x41,0xea,0x43,0x09,
      0xca,0x4e,0xa9,0x38,0x0c,0xd0,0xe8,0x34,
      0xf7,0x1f,0xbe,0x91,0x74,0xa2,0x61,0x38,
      0x0d,0xc1,0x7e,0xae,0x6a,0x34,0x51,0xd9 }
    },
    {   NID_sha384,
    { 0x08,0xbc,0xb0,0xda,0x49,0x1e,0x87,0xad,
      0x9a,0x1d,0x6a,0xce,0x23,0xc5,0x0b,0xf6,
      0xb7,0x18,0x06,0xa5,0x77,0xcd,0x49,0x04,
      0x89,0xf1,0xe6,0x23,0x44,0x51,0x51,0x9f,
      0x85,0x56,0x80,0x79,0x0c,0xbd,0x4d,0x50,
      0xa4,0x5f,0x29,0xe3,0x93,0xf0,0xe8,0x7f }
    },
```

Fig. 13. Testvector definition for HMAC KAT in fips_hmac_selftest.c

소스코드는 openssl-fips-2.0.16\fips\알고리즘이름\fips_알고리즘이름_selftest.c 파일에서 참고할 수 있다.

4.1.3 RSA KAT 시험

RSA 암호에 대해서는 RSA-PSS 전자서명 방법을 이용하여 KAT 시험을 수행한다. FIPS-OpenSSL은 2048/3072/4096 비트 키에 대해 RSA 암호 기능을 제공하나, 가장 짧은 키 길이인 2048 비트 키에 대해서만 KAT를 수행한다. RSA 암호에 대한 자가시험 테스트벡터 (RSA 키 파라미터 및 사전에 계산된 서명값) 및 소스코드는 fips_rsa_selftest.c 파일에 정의되어 있다.

RSA-PSS 전자서명 알고리즘은 서명을 생성할 때 랜덤할 salt 값을 이용하여 메시지를 인코딩하나 KAT 시험 수행을 위하여 인코딩 과정에서 사용되는 salt 값을 0으로 설정하여 사용한다. 〔Fig. 14.〕는 fips_rsa_selftest.c 파일에 정의된 RSA 암호에 대한 KAT 코드이다. 사전에 정의된 테스트벡터를 fips_pkey_signature_test() 함수의 인자로 하여 호출하고 있다. fips_pkey_signature_test() 함수에서는 먼저 소스코드에 고정된 메시지에 대해서 사전에 정의된 개인키를 이용하여 서명을 생성한 후 그 결과가 사전에 계산되어 저장된 서명값과 일치하는지 확인한다 〔Fig. 15.〕. 이를 통과하면 다시 생성된 서명에 대해 검증과정을 수행하여 통과하면 RSA 암호에 대한 KAT가 성공으로 끝난다.

```
int FIPS_selftest_rsa()
    {
    int ret = 0;
    RSA *key = NULL;
    EVP_PKEY pk;
    key=FIPS_rsa_new();
    setrsakey(key);
    pk.type = EVP_PKEY_RSA;
    pk.pkey.rsa = key;

    if (!fips_pkey_signature_test(FIPS_TEST_SIGNATURE,
                &pk, kat_tbs, sizeof(kat_tbs) - 1,
                kat_RSA_PSS_SHA256, sizeof(kat_RSA_PSS_SHA256),
                EVP_sha256(), RSA_PKCS1_PSS_PADDING,
                "RSA SHA256 PSS"))
        goto err;
```

Fig. 14. RSA KAT procedure

```
int fips_pkey_signature_test(int id, EVP_PKEY *pkey,
            const unsigned char *tbs, size_t tbslen,
            const unsigned char *kat, size_t katlen,
            const EVP_MD *digest, int pad_mode,
            const char *fail_str)
    {
    int subid;
    int ret = 0;
    unsigned char *sig = NULL;
    unsigned int siglen;
    __fips_constseg
    static const unsigned char str1[]="12345678901234567890";
                                ...
    if (pkey == NULL)
        {
        if (!FIPS_digestfinal(&mctx, sig, &siglen))
            goto error;
        }
    else if (pkey->type == EVP_PKEY_RSA)
        {
        if (!FIPS_rsa_sign_ctx(pkey->pkey.rsa, &mctx,
                    pad_mode, 0, NULL, sig, &siglen))
            goto error;
        }
                                ...
    if (kat && ((siglen != katlen) || memcmp(kat, sig, katlen)))
        goto error;
                                                Sign KAT
    }
```

Fig. 15. RSA Sign KAT

4.1.4 ECDH KAT 시험

ECDH 키 설정 알고리즘에 대해서도 KAT를 수행하며 키 합의 과정을 수행하여 도출된 키가 사전에 계산된 키와 일치하는지 확인한다. RSA 암호에 대한 KAT와 마찬가지로 FIPS-OpenSSL은 P-224/256/384/512에 대한 키 길이를 제공하나 가장 짧은 키 길이인 P-224에 대해서만 ECDH를 이용하여 도출된 키에 대한 KAT를 수행한다. 〔Fig. 16.〕은 fips_ecdh_selftest.c 파일에 정의된 ECDH에 대한 KAT 코드이다. 사전에 정의된 키와 타원곡선 점을 이용하여 공유키를 계산한 후 사전에 계산된 키와 비교하여 ECDH의 올바른 동작을 확인한다.

```
int FIPS_selftest_ecdh(void)
    {
    EC_KEY *ec1 = NULL, *ec2 = NULL;
    const EC_POINT *ecp = NULL;
    BIGNUM *x = NULL, *y = NULL, *d = NULL;
    unsigned char *ztmp = NULL;
    int rv = 1;
    size_t i;

    for (i = 0; i < sizeof(test_ecdh_data)/sizeof(ECDH_SELFTEST_DATA); i++)
        {
        ECDH_SELFTEST_DATA *ecd = test_ecdh_data + i;
        if (!fips_post_started(FIPS_TEST_ECDH, ecd->curve, 0))
            continue;
        ztmp = OPENSSL_malloc(ecd->zlen);

        x = BN_bin2bn(ecd->x1, ecd->x1len, x);
        y = BN_bin2bn(ecd->y1, ecd->y1len, y);
        d = BN_bin2bn(ecd->d1, ecd->d1len, d);
                                ...
        ecp = EC_KEY_get0_public_key(ec2);
        if (!ecp)
            {
            rv = -1;
            goto err;
            }

        if (!ECDH_compute_key(ztmp, ecd->zlen, ecp, ec1, 0))
            {
            rv = -1;
            goto err;                   Computing shared key
            }

        if (!fips_post_corrupt(FIPS_TEST_ECDH, ecd->curve, NULL))
            ztmp[0] ^= 0x1;                             KAT
        if (memcmp(ztmp, ecd->z, ecd->zlen))
            {
            fips_post_failed(FIPS_TEST_ECDH, ecd->curve, 0);
            rv = 0;
            }
```

Fig. 16. ECDH KAT

4.2 조건부 자가시험

암호모듈이 탑재하고 있는 알고리즘의 정상동작을 확인하기 위한 KAT 시험은 암호모듈이 초기화될 때 알고리즘 별로 1회 수행되지만, 조건부 자가시험은 특정 조건이 발생할 때마다 수행되어야 한다. 대표적인 조건부 자가시험은 공개키 키 쌍 생성 시 수행되는 키 쌍 일치 시험(Pairwise key test)과 난수발생기의 출력이 발생할 때마다 수행되는 연속적인 난수발생기 시험 (Continuous test)이 있다. FIPS-OpenSSL에서는 〔Table 4.〕와 같이 탑재하고 있는 난수발생기에 대한 연속적인 난수발생기 시험과 RSA/DSA/ECDSA 대한 키 쌍 일치 시험을 수행한다.

Table 4. Conditional test in FIPS-OpenSSL

Algorithm	Test
DRBG	Continuous test on Hash_DRBG, HMAC_DRBG, CTR_DRBG
RSA	Pairwise consistency test on each generation of a RSA key pair
DSA	Pairwise consistency test on each generation of a DSA key pair
ECDSA	Pairwise consistency test on each generation of a ECDSA key pair

4.2.1 키 쌍 일치 시험

RSA, DSA, ECDSA와 같은 공개키 기반 암호에서 키 쌍을 생성한 경우 키 쌍 일치 시험이 수행된다. 〔Fig. 17.〕은 RSA 키 쌍을 생성하는 함수이며 fips_check_rsa() 함수(〔Fig. 18.〕)를 이용하여 생성된 키 쌍의 유효성을 확인한다. 〔Fig. 18.〕은 RSA 암호를 위해 생성된 키 쌍의 유효성을 시험하는 함수이며, 생성된 키 쌍에 대해 서명생성과 검증, 그리고 암호화와 복호화 과정을 통하여 유효성을 검증한다.

DSA와 ECDSA 전자서명 알고리즘에서도 키 쌍을 생성한 후 RSA 암호의 경우와 동일하게 각각 fips_check_dsa(), fips_check_ecdsa()가 호출되어 생성된 키 쌍의 유효성을 검증한다. fips_check_dsa()와 fips_check_ecdsa()에서는 fips_pkey_signature_test()를 호출하여 서명 생성과 검증을 통하여 생성된 키 쌍의 유효성을 검증한다. 만약, 유효성 검증이 실패로 끝나면 자가시험 실패 오류 코드를 반환한다.

```
static int rsa_builtin_keygen(RSA *rsa, int bits, BIGNUM *e_value, BN_GENCB *cb)
    {
    BIGNUM *r0=NULL,*r1=NULL,*r2=NULL,*r3=NULL,*tmp;
    BIGNUM local_r0,local_d,local_p;
    BIGNUM *pr0,*d,*p;
    int bitsp,bitsq,ok= -1,n=0;
    BN_CTX *ctx=NULL;
                ...
    /* calculate d mod (p-1) */
    if (!BN_mod(rsa->dmp1,d,r1,ctx)) goto err;

    /* calculate d mod (q-1) */
    if (!BN_mod(rsa->dmq1,d,r2,ctx)) goto err;

    /* calculate inverse of q mod p */
    if (!(rsa->flags & RSA_FLAG_NO_CONSTTIME))
        {
        p = &local_p;
        BN_with_flags(p, rsa->p, BN_FLG_CONSTTIME);
        }
    else
        p = rsa->p;
    if (!BN_mod_inverse(rsa->iqmp,rsa->q,p,ctx)) goto err;

#ifdef OPENSSL_FIPS
    if(!fips_check_rsa(rsa))
        goto err;
#endif
```

Fig. 17. Calling RSA Pairwise consistency test function

```
int fips_check_rsa(RSA *rsa)
    {
    const unsigned char tbs[] = "RSA Pairwise Check Data";
    unsigned char *ctbuf = NULL, *ptbuf = NULL;
    int len, ret = 0;
    EVP_PKEY pk;
        pk.type = EVP_PKEY_RSA;
        pk.pkey.rsa = rsa;

    /* Perform pairwise consistency signature test */
    if (!fips_pkey_signature_test(FIPS_TEST_PAIRWISE, &pk, tbs, 0,
            NULL, 0, NULL, RSA_PKCS1_PADDING, NULL)
        || !fips_pkey_signature_test(FIPS_TEST_PAIRWISE, &pk, tbs, 0,
            NULL, 0, NULL, RSA_X931_PADDING, NULL)
        || !fips_pkey_signature_test(FIPS_TEST_PAIRWISE, &pk, tbs, 0,
            NULL, 0, NULL, RSA_PKCS1_PSS_PADDING, NULL))
        goto err;
    /* Now perform pairwise consistency encrypt/decrypt test */
    ctbuf = OPENSSL_malloc(RSA_size(rsa));
    if (!ctbuf)
        goto err;

    len = RSA_public_encrypt(sizeof(tbs) - 1, tbs, ctbuf, rsa, RSA_PKCS1_PADDING);
    if (len <= 0)
        goto err;
    /* Check ciphertext doesn't match plaintext */
    if ((len == (sizeof(tbs) - 1)) && !memcmp(tbs, ctbuf, len))
        goto err;
    ptbuf = OPENSSL_malloc(RSA_size(rsa));

    if (!ptbuf)
        goto err;
    len = RSA_private_decrypt(len, ctbuf, ptbuf, rsa, RSA_PKCS1_PADDING);
    if (len != (sizeof(tbs) - 1))
        goto err;
    if (memcmp(ptbuf, tbs, len))
        goto err;
```

Sign/verify

Encrypt/decrypt

Fig. 18. Calling RSA Pairwise consistency test function

4.2.2 연속적인 난수발생기 시험

난수발생기에서 난수열이 생성되어 출력될 때마다 연속적인 난수발생기 시험이 수행된다. 난수발생기는 일반적으로 인스턴스 생성 과정, 난수열 생성 과정, 그리고 리씨딩 과정으로 구성된다. 연속적인 난수발생기 시험은 사용자가 요청한 길이의 난수를 난수열 생성 함수에서 생성할 때 수행되는 조건부 자가시험이다. 〔Fig. 19.〕는 fips_drbg_hash.c 파일에 정의되어 있는 HASH_DRBG의 난수열 생성 함수 hash_gen() 함수의 소스코드이다. 사용자가 요청한 길이의 난수를 생성하기 위해, 해시함수의 출력 길이를 기본 블록 길이로 하여 for 루프를 돌면서 난수를

생성하여 덧붙인다. 이때, FIPS_digestfinal() 함수를 이용하여 생성된 난수 블록에 대해 연속적인 난수발생기 시험 함수인 fips_drbg_cprng_test()를 호출한다. 연속적인 난수발생기 시험은 연속적으로 생성되는 난수블록들이 동일한 값을 갖지 않는지 확인하는 것으로서 이전에 생성된 난수 블록과 현재 생성된 난수 블록을 비교하여 연속된 두 블록이 동일할 경우 실패로 간주한다. 따라서, 이전에 생성된 난수 블록을 저장할 필요가 있으며 이는 DRBG의 상태를 관리하는 DRBG_CTX 자료형의 lb 멤버 원소에 저장된다 (lb는 last block을 의미하며 해시 함수 출력길이에 해당하는 버퍼임). 즉, [Fig. 20.]와 같이 fips_drbg_cprng_test() 함수에서는 DRBG_CTX 자료형의 lb 멤버에 저장된 직전에 생성된 난수블록과 현재 생성된 난수 블록을 비교하고 동일한 경우 오류를 반환한다. 만약, 두 블록이 서로 다를 경우 성공을 반환하며 다음번 비교를 위하여 현재 생성된 난수 블록을 DRBG_CTX의 lb 멤버에 저장한다. 즉, SHA-256 기반의 Hash_DRBG를 이용하여 1024 비트의 난수열을 생성하는 경우 총 4번의 연속적인 난수발생기 시험이 수행되는 것이다.

```
static int hash_gen(DRBG_CTX *dctx, unsigned char *out, size_t outlen)
    {
    DRBG_HASH_CTX *hctx = &dctx->d.hash;
    if (outlen == 0)
        return 1;
    memcpy(hctx->vtmp, hctx->V, dctx->seedlen);
    for(;;)
        {
        FIPS_digestinit(&hctx->mctx, hctx->md);
        FIPS_digestupdate(&hctx->mctx, hctx->vtmp, dctx->seedlen);
        if (!(dctx->xflags & DRBG_FLAG_TEST) && !dctx->lb_valid)
            {
            FIPS_digestfinal(&hctx->mctx, dctx->lb, NULL);
            dctx->lb_valid = 1;
            }
        else if (outlen < dctx->blocklength)
            {
            FIPS_digestfinal(&hctx->mctx, hctx->vtmp, NULL);
            if (!fips_drbg_cprng_test(dctx, hctx->vtmp))
                return 0;
            memcpy(out, hctx->vtmp, outlen);
            return 1;
            }
        else
            {
            FIPS_digestfinal(&hctx->mctx, out, NULL);
            if (!fips_drbg_cprng_test(dctx, out))
                return 0;
            outlen -= dctx->blocklength;
            if (outlen == 0)
                return 1;
            out += dctx->blocklength;
            }
        ctx_add_buf(dctx, hctx->vtmp, NULL, 0);
        }
    }
```

Fig. 19. Continuous test in DRBG

```
/* Continuous DRBG utility function */
int fips_drbg_cprng_test(DRBG_CTX *dctx, const unsigned char *out)
    {
    /* No CPRNG in test mode */
    if (dctx->xflags & DRBG_FLAG_TEST)
        return 1;
    /* Check block is valid: should never happen */
    if (dctx->lb_valid == 0)
        {
        FIPSerr(FIPS_F_FIPS_DRBG_CPRNG_TEST, FIPS_R_INTERNAL_ERROR);
        fips_set_selftest_fail();
        return 0;
        }
    if (drbg_stick)
        memcpy(dctx->lb, out, dctx->blocklength);
    /* Check against last block: fail if match */
    if (!memcmp(dctx->lb, out, dctx->blocklength))
        {
        FIPSerr(FIPS_F_FIPS_DRBG_CPRNG_TEST, FIPS_R_DRBG_STUCK);
        fips_set_selftest_fail();
        return 0;
        }
    /* Save last block for next comparison */
    memcpy(dctx->lb, out, dctx->blocklength);
    return 1;
    }
```

Compare current block with the last block

Store the current block for the next comparison

Fig. 20. Comparison method in continuous test

V. FIPS 140-2 자가시험과 KS X ISO/IEC 19790 자가시험 비교 및 고찰

국내 암호모듈 검증제도의 기준인 KS X ISO/IEC 19790:2015는 FIPS 140-2와 많은 유사점을 가지고 있으나 용어와 기준 특면에서 다소 차이가 존재한다. 본 절에서는 자가시험 보안요구사항 영역에서의 차이점을 소개한다.

5.1 FIPS 140-2와 KS X ISO/IEC 19790 비교

FIPS 140-2에서는 암호모듈이 초기화될 때 전원인가 자가시험 (Power-on selftest)를 수행하여 암호모듈에 대한 무결성 시험과 탑재한 승인 알고리즘에 대한 KAT 기반의 알고리즘 시험을 수행한다. 반면, KS X ISO/IEC 19790:2015에서는 암호모듈이 초기화될 때 동작 전 자가시험을 수행할 것을 요구하고 있으며, 동작 전 자가시험에서 필수로 수행되어야하는 시험은 암호모듈에 대한 무결성 시험만이다. FIPS 140-2와는 달리 KAT 기반의 알고리즘 시험은 조건부 시험으로 수행 가능하며 해당 알고리즘이 최초 호출될 때 1회에 한해서 수행될 수 있다. 따라서, 국내 암호모듈 검증제도에서는 KAT 기반의 알고리즘 시험을 동작 전 자가시험에서 암호모듈에 대한 무결성 시험과 함께 수행되도록 하거나, 또는 동작 전 자가시험이 완료된 후 해당 알고리즘이 최초로 호출되었을 때 수행되도록 할 수 있어 유연한 운영이 가능하다.

키 쌍 일치시험과 연속적 난수발생기 시험은 국내 암호모듈 검증제도와 FIPS 140-2에서 동일한 기준을 적용하고 있다. 하지만, DH, ECDH에 대한 키 쌍 일치 시험을 요구하지 않는 FIPS 140-2에 반하여 국내 암호모듈 검증제도에서는 DH, ECDH 알고리즘에서 키 쌍을 생성할 경우 키 쌍 일치 시험을 수행할 것을 요구한다.

5.2 고찰

4장에서 분석한 것과 같이 FIPS 140-2에 의거

하여 개발된 FIPS-OpenSSL에서는 탑재된 승인 알고리즘에 대한 자가시험 수행 시, 최소 크기의 키, 그리고 최소 운영모드에 대해서만 자가시험을 수행한다. 예를 들어, AES 암호에 대해서는 128 비트 ECB 모드에 대해서만 KAT 시험을 수행하고 RSA 암호에 대해서는 2048 비트 키에 대해서만 시험을 수행한다. 특히, NIST 권고 타원곡선 파라미터를 사용하는 타원곡선 기반 암호에서까지 최소 키 길이에 대한 자가시험을 수행한다.

대칭키 암호, 해시함수, MAC, 난수발생기의 경우에는 키 길이가 늘어남에 따라 단순히 내부 루프 횟수가 늘어나는 것이기 때문에 최소 키 비트에 대해서 KAT 시험을 수행해도 무방하다. 또한, RSA 암호, DSA 서명과 같이 암호 운영을 위해 키 쌍 또는 도메인 파라미터를 생성해야하는 알고리즘의 경우에도 최소 키에 대해서 KAT 시험을 수행해도 무방하다 (유한체 연산과 지수승 알고리즘에서 키 사이즈와 상관없이 동일한 알고리즘 사용). 하지만, 대칭키 암호 운영모드의 경우에는 각 운영모드마다 동작 절차가 상이하기 때문에 탑재된 운영모드에 맞는 KAT 시험이 필요하다. 또한, 타원곡선 기반 암호 시스템(ECDH, ECDSA)의 경우 NIST 권고 도메인 파라미터를 사용하기 때문에 탑재된 각 커브에 대한 KAT 시험 수행이 필요하다. 즉, 각 커브 파라미터마다 서로 다른 감산 소수를 사용하기 때문에 이에 맞는 감산 방식이 필요하며 이를 시험하기 위한 KAT 시험이 필요하다. 향후 FIPS 140-2의 자가시험이 이러한 부분을 보완할 수 있도록 개정되길 기대해본다.

VI. 결 론

본 논문에서는 미국에서 암호모듈검증을 획득한 대표적인 소프트웨어 라이브러리 형태의 암호모듈인 FIPS-OpenSSL의 소스코드를 분석하여 암호모듈 검증 기준의 중요 보안요구사항인 자가시험 부분이 소프트웨어적으로 어떻게 구현되어있는지 분석하였다. FIPS-OpenSSL은 구조가 복잡하고 함수 포인터를 이용하였기 때문에 디버깅을 통한 구조 분석하였으며 크게 전원인가 자가시험과 조건부 자가시험으로 나누어 분석을 수행하였다. 전원인가 자가시험은 대부분 KAT 형태로 시험이 수행되며 〔Fig. 9.〕와 같이 정형화된 형태를 가지고 있다. 또한, 공개키 기반 암호에서 키 쌍이 생성될 때 수행되는 키 쌍 일치 시험과 난수발생기에서 난수를 생성할 때마다 수행되는 연속적 난수발생기 시험의 구조를 분석하였다. 뿐만 아니라, FIPS 140-2과 국내 암호모듈 검증기준 사이의 차이점과 FIPS 140-2의 단점에 대해서도 논하였다. 본 논문의 자가시험 기능에 대한 소스코드 분석을 통해 향후 국내 암호모듈 개발업체들이 해당 기능을 안전하고 정확하게 구현하여 시험기간을 단축시킬 것을 기대해본다.

References

〔1〕 Cryptographic Module Validation Program, https://csrc.nist.gov/projects/cryptographic-module-validation-program, (10. Oct. 2019)

〔2〕 FIPS PUB 140-2, "Security Requirements for Cryptographic Modules", May. 2001.

〔3〕 Implementation Guidance for FIPS 140-2 and the Cryptographic Module Validation Program, https://csrc.nist.gov/csrc/media/projects/cryptographic-module-validation-program/documents/fips140-2/fips1402ig.pdf (10. Oct. 2019)

〔4〕 ISO/IEC 19790:2012/Cor.1:2015, "Security Requirements for Cryptographic Modules", July. 2015.

〔5〕 ISO/IEC 24759:2014/Cor.1:2015, "Test Requirements for Cryptographic Modules", July. 2015.

〔6〕 Korea Cryptographic Module Validation Program (KCMVP), "https://www.nis.go.kr:4016/AF/1_7_3_1.do" (10. Oct. 2019)

〔7〕 FIPS-OpenSSL, "https://www.openssl.org/docs/fips.html", "https://wiki.openssl.org/index.php/FIPS_modules", (10. Oct. 2019)

〔8〕 nasm assembler, "https://www.nasm.us", (10. Oct. 2019)

〔9〕 perl scripter, "https://www.perl.org/get.html", (10. Oct. 2019)

〈저자소개〉

서 석 충 (Seog Chung Seo) 정회원
2011년 8월: 고려대학교 정보보호학과 박사
2013년 11월: 삼성전자 종합기술원 전문연구원
2014년 4월: 삼성전자 DMC 연구소 책임연구원
2019년 2월: 국가보안기술연구소 선임연구원
2019년 3월~현재: 국민대학교 정보보안암호수학과 조교수
〈관심분야〉 암호최적화, 공개키 암호, 암호모듈검증, 네트워크보안