블록 암호 알고리즘 정확성 테스트 모듈 구현

정성민*, 박성근*, 김석우*, 서창호**, 김일준***

*한세대학교 정보통신학과, **공주대학교 응용수학과, ***국가보안기술연구소

The Implementation of Block Cipher Algorithm Correctness Test Module

Jeong Seong Min*, Park Seong Gun*, Kim Seok Woo*, Su Chang Ho**, Kim Il Jun***

*Dep. of Information & Communication Hansei Univ.

**Dep. of Applied Mathematics Kongju Univ.

***National Security Research Institute

요 약

정보보호 평가는 크게 시스템 평가인 CC평가와 암호모듈 평가인 CMVP평가로 나눌 수 있다. 본 논문은 암호모듈 평가 방법으로 북미 CMVP의 3가지 블록 알고리즘시험방법인 KAT(Known Answer Test), MCT(Monte Calro Test), MMT(Multi-block Message Test)를 JAVA환경에 적용하여 시범 구현하였다. 대상 알고리즘은 CMVP의 4가지 블록 알고리즘(DES, TDES, AES, Skipjack)을, 테스트 방법으로 MOVS, TMOVS, AESAVS를 선정하여 FIPS표준을 적용하였다. 구현 환경으로는 JCA기반의 Cryptix를 채택하여 CMVP의 블록 암호 알고리즘 테스트 시스템 중 일부를 구현하였다.

I. 서론

정보보호에 대한 인식이 확산되면서 정보보호 제품 수요가 증가되고 있으며, 이에 따른 국내 CC평가 인증제도가 2002년 하반기부터 시행되고 있다[9]. 반면에, 상용 암호 서비스에 대한 평가제도는 한국 정보보호진흥원의 K1~7E 평가에서 시작되어 최종적으로 암호알고리즘에 대한 평가를 병행하고 있다[9]. 본 논문은 북미에서 현재 활발히 진행 중인 CMVP 내의 암호 알고리즘 시험 기술을 국내에 보다 효율적으로 통합하기 위한 작업의 일부로써, 블록 암호알고리즘 시험 모듈을 Java 환경에서 시범 구현하였다. 본 논문에서는 새로운 시험방법이나 알고리즘 강도 시험이 아닌 Java환경하의 여러 프로그램 이식 및 NIST 시험 방법의 구현을 위한 테스트 베드 구축에 의미를 부여한다.

본 논문의 제 II 장에서 암호 알고리즘 테스트 방법의 모듈 구현 환경인 운영모드, 시험방법, 시험환경에 대하여 기술하고, 제 III 장에서는 자바 기반 암호 API인 JCA · JCE에 대해 설명하였고, 마지막으로 결론 및 향후 연구방향에 대해 기술하였다.

II. NIST 표준 암호 알고리즘 테스트 모듈 구현 환경

1. 블록 암호알고리즘 운영 모드

NIST에서 기술한 블록 암호 알고리즘의 구동 방식에는 ECB, CBC, OFB, CFB등 4가지 방식의 운영모드가 있고, 미국 CMVP에서 DES[1] 및 암호 알고리즘에 대한 암호 알고리즘 구현 정확성 검증에 대한 표준 문서에는 4가지 운영 모드에 대한 KAT(The Known Answer Tests), MCT(The Monte Carlo Test), MMT(The Multi-block Message Test)의 3가지 검증 방식을 채택하고 있다.[2-4]

1)ECB(Electronic CodeBook) 모드

ECB 모드는 DES 암호 방식의 사용 방식 중 가장 간단한 방식으로 평문을 64비트씩 나누어 암호화하는 방식이다. 평문을 64비트씩 나눌 때 마지막 블록이 64비트가 되지 않을 때는 임의의 약속된 비트 모양을 패딩(padding)하게 된다.

2)CBC(Cipher Block Chaining) 모드

DES 암호 방식의 CBC 모드는 출력 암호문이 다음 평문 블록에 영향을 미치게 하여 각 암호문 블록이 전단의 암호문의 영향을 받도록 만든 방식으로 ECB에서 발생하는 동일한 평문에 의한 동일한 암호문이 발생하지 않도록 구성한

동작 모드이다. CBC 모드 동작은 처음 입력된 평문 블록 M_1은 초기 벡터 IV_0(initial vector)와 EX-OR되어 DES 암호기에 입력된다. 암호기 출력 암호문 C_1은 다음 단 평문 블록 M_2와 EX-OR되어 DES 암호기에 입력된다.

3)CFB(Cipher FeedBack) 모드

CFB 모드도 CBC 모드와 마찬가지로 평문 블럭이 동일한 경우 동일한 암호문이 나타나지 않도록 전단의 암호문이 다음 단의 평문에 영향을 미치도록 구성하는 방식이다. CFB 모드 동작은 CBC를 연상시키게 구성되어 있으나, 다른 점은 암호문이 수신자의 암호 입력으로 사용되는 것이다.

4)OFB(Output FeedBack) 모드

CBC 모드와 CFB 모드 동작은 암호화할 평문 블럭에 오류가 발생하면 모든 암호문 블럭에 영향을 미치며 또한 전송중인 암호문 블럭에 오류가 발생하면 이후 복호화 된 모든 평문 블럭에 영향을 미치게 된다. OFB 모드 동작은 평문 블럭이 동일하면 암호문이 같아지는 ECB 모드의 단점과 오류 전파가 발생하는 CBC 모드와 CFB 모드를 개선한 동작 모드이다.

2.KAT(The Known Answer Test)

KAT 테스트 방식에는 GFSbox, KeySbox, 정형화된 키, 정형화된 평문에 대한 암호화 과정 및 복호화 과정에서 출력되는 결과 데이터를 참조값으로 하여 시험대상물의 결과 데이터를 검증한다.

KAT는 세가지 키(128, 192, 256)에 관해 규칙 및 비 규칙적인 값의 변화에 따라 제공되는 알려진 데이터 값을 참조한다. 정형화된 키에 대한 테스트 방식은 키 값을 고정시키고 평문과 IV값을 다양하게 변화시켜 테스트 한다. 정형화된 평문 테스트는 반대로 평문을 고정시키고 키 값과 IV 값을 다양하게 변화시킨 후 알고리즘 결과 값을 출력시킨다.

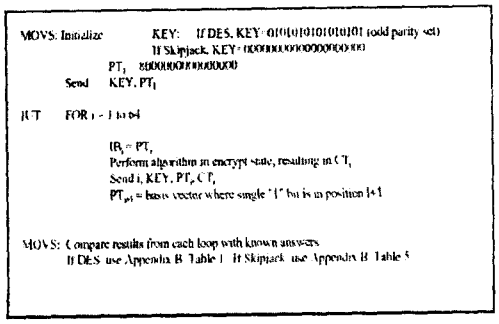

그림 1: 정형화된 평문에 대한 ECB of KAT

3.MCT(Monte Carlo Test)

MCT는 DES, SkipJack, AES등 블록 암호 알고리즘에 따라 ECB, CBC, CFB, OFB등 4가지 모드별로 테스팅을 수행한다. MCT는 알고리즘의 수행에 필요한 초기값(IV, 평문, 키)의 정보를 갖고 있는 Request_file과 초기값에 따라 MCT 테스팅 알고리즘의 수행에 따른 결과로서 생성된 Response_file(이터레이션값, 키 값, 평문, 암호문의 셋트)로 구성된다. 다음 그림2 는 AES 알고리즘(ECB 동작모드) 상에서 MCT 테스팅 알고리즘의 한 예를 보인다.

```
Key[0] = Key
PT[0] = PT
For i = 0 to 99
  Output Key[i]
  Output PT[0]
    For j = 0 to 999
      CT[j] = AES(Key[i], PT[j])
      PT[j+1] = CT[j]
  Output CT[j]
    If (keylen = 128)
      Key[i+1] = Key[i] xor CT[j]
    If (keylen = 192)
      Key[i+1] = Key[i] xor (last 64 bits of
                             CT[j-1] || CT[j])
    If (keylen = 256)
      Key[i+1] = Key[i] xor (CT[j-1] || CT[j])
      PT[0] = CT[j]
```

그림 2: MCT algorithm for AES-ECB mode

그림2 에서 위의 테스팅 알고리즘의 수행에 필요한 초기값은 PT[0]인 평문과 key[0]인 키값이다. ECB모드상에서 MCT는 초기값으로서 IV는 필요로하지 않는다. 위 테스팅 수행 결과로서 Output Key[i], PT[0], CT[j]는 response file에 해당하는 내용이다.아래 그림3은 Cryptix JCE에 의해 128비트 키사이즈의 AES를 ECB 동작모드 상에서 MCT 수행을 한 예이다.

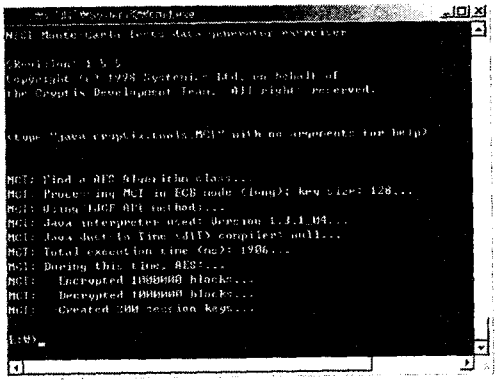

그림 3: Cryptix에 기반한 AES-ECB MCT 수행

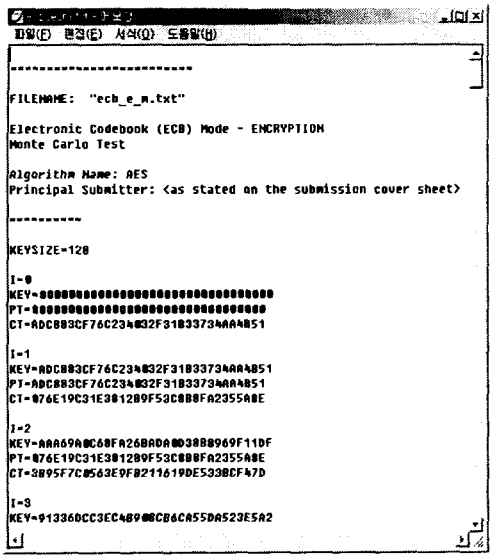

그림 4: 그림3 수행 결과 생성된 response 파일

그림4는 128비트 AES-ECB모드 MCT 수행 결과 생성된 response 파일이다. 그림에서 이터레이션값 I=0일때 PT(평문)와 Key값이 초기값이고,해당하는 CT(암호문)값이 AES 알고리즘을 천번 수행 한 후의 암호문이다. 즉 I값이 하나 증가할 때마다 천번의 알고리즘을 수행하므로 I값이 999가 되면 AES-ECB모드 MCT는 암복호화 기능을 100,000번 정도 수행한다고 할 수 있다. 현재 이터레이션에 필요한 키 값은 전 이터레이션에 해당하는 키와 암호문의 xor한 결과 값이고, 전 이터레이션의 암호문은 다음 이터레이션의 평문으로 사용된다. MCT는 Request file에 따른 Response 결과로서 알고리즘 수행의 정확성을 평가한다.

4.MMT(Multi-block Message Test)

한 블럭에서 다음 블록까지 정보의 상태를 연결하는 멀티 블럭 메시지 진행을 구성하는 것에 대한 테스트이다. 테스트 되는 모드는 ECB, CBC, OFB와 블록 길이가 64bit이면 CFB64, 1비트이면 CFB1, 8bit이면 CFB8이 있다. 4가지 모드에서 블럭의 길이 만큼 테스트를 수행하며 블록의 길이가 $1 \le blocklengh \le 10$이면 모드에 따라 10개의 메시지를 지원한다.

MMT테스트에서 요구되는 파일은 키와 초기벡터(ECB 모드 제외), 그리고 암호화될 평문으로 구성되어 진다. 암호문의 추가된 테스트 파일은 처음 암호문이 포함되지 않은 테스트 파일과 일치해야한다.

5. 통합 테스트 환경 구축

위에서 설명한 각 테스트들을 하나의 프로그램으로 합쳐서 Java로 구현된 Cryptix 기반으로 통합 테스트 환경을 구축하였다.[10] Java로 구현한 큰 이유는 이식성이 뛰어나므로, 후에 국내 표준 암호 알고리즘 및 추가 알고리즘을 손쉽게 이식할 수 있기 때문이다. 사용자는 먼저 테스트 방법을 결정하며 사용자의 정보와 테스트 받고자 하는 알고리즘에 대해 테스트 방법을 설명을 하여 Request File로 저장하여 온라인 상에서 전송할 수 있다. 이때, 사용자가 정해진 테스트 방법 중 일부를 임의대로 선택하고 평문이나 키, 벡터 값을 임의대로 넣어 테스트를 할 수 있다. 사용자가 전송한 Request File 파일을 입력으로 하여 테스트를 하며, 결과 데이터는 Response File로 저장되어 사용자의 알고리즘 정확성 시험에 대한 판단 기준으로 사용된다.[2-4]

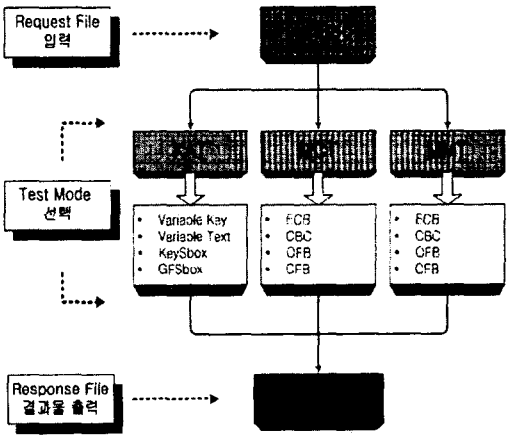

그림 5: 테스트 과정

Ⅲ.자바 JCA, JCE

1.자바 암호 라이브러리

1)JCA(JAVA Cryptographic Architecture)

JCA를 기반으로 암호 라이브러리를 구현할 때의 장점은 분산처리, 아키텍처, 독립, 보안성, 그리고 이식성 등 기존 Java의 이점들을 그대로 얻을 수 있다는 것이다.[11]

JCA는 java.security 패키지와 그 서브 패키지 안에 있는 많은 클래스들로 구성되어 있다. 이 클래스들은 전자 서명과 메시지 다이제스트 같은 기능에 관한 일반적인 API를 제공한다.

표1 : JCA 암호 관련 주요 클래스

클래스	기 능
Message Digest	MD5 또는 SHA 등의 메세지 다이제스트 알고리즘의 기능을 제공한다
Signature	어플리케이션에 대해서 디지털 서명 알고리즘의 기능을 제공하기 위해서 사용된다.
KeyPair Generator	공개열쇠와 비공개열쇠의 페어를 생성하기 위해서 사용한다. 열쇠 페이 제네레이터는, getInstance 팩토리 메소드 (지정된 클래스의 인스턴스를 돌려주는 static 메소드)를 사용해 구축된다.
Key Factory	Key 형태의 불투명한 암호열쇠, 기본 열쇠 데이터의 투명한 열쇠 사양과의 사이의 변환을 하기 위해서(때문에) 사용한다.
KeyStore	열쇠와 증명서의 메모리내 콜렉션을 나타내, 2 종류의 엔트리(키, 신뢰할수 있는 증명서)를 관리한다.
Algorithm Parameters	AlgorithmParameters 오브젝트를 취득하기위해, getInstance 팩토리 메소드 (지정된 클래스의 인스턴스를 돌려주는 static 메소드가 어떤 것인지를 호출한다.
Algorithm Parameters Generator	특정의 알고리즘으로 사용되는 파라미터세트를 생성하기 위해서 사용된다. 파라미터 제네레이터는, getInstance 팩토리 메소드 (지정한 클래스의 인스턴스를 돌려주는 static 메소드)를 사용해 구축된다.
Secure Random	암호용으로 강화한 의사 random number 제네레이터 (PRNG)를 제공한다.

2) JCE(Java Cryptography Extension)

JCE는 JCA와 유사한 기능을 하면서도 가장 중요한 차이점은 Sun에서 구현된 암호 라이브러리 외에 다른 암호 서비스 라이브러리들도 사용 가능하다는 확장성을 들 수 있다. 그 중에서도 본 논문에서 기술한 블록암호에 대한 MCT, KAT 테스팅 기능을 기본적으로 가지고 있는 Cryptix 라는것도 자바 JCE의 한 종류이다.
JCE는 javax.crypto 패키지와 그 서브 패키지들로 구성되어 있다. 중요 클래스는 다음과 같다.

표 2 : JCE 암호 관련 주요 클래스

클래스	기 능
Cipher	암·복호화 하는 기능을 수행하는 클래스이다.
KeyAgreement	키 교환 프로토콜의 기능을 제공한다.
KeyGenerator	비밀키 생성 기능을 제공한다.
Mac	메시지 인증 코드 알고리즘의 기능을 제공한다.
SecretKey	비밀키에 대한 인터페이스이다.
SecretKeyFactory	비밀키의 팩토리를 나타낸다.

아래의 싱글 라인 자바 소스는 DES암호 알고리즘을 위한 키를 생성하는 객체를 생성하는 경우를 예시한다.

KeyGeneratorkey Generator =

KeyGenerator.getInstance("DES");

이와같이 자바는 암호 서비스를 사용하기 용이한 구조로 인해 최근 인터넷 전자상거래나 웹 어플리케이션에서 광범위하게 사용되고 있다.

Ⅳ.결론 및 향후 연구 방향

미국NIST의 경우 DES, TDES, AES, Skipjack, SHA-1, DSA등 표준 암호 알고리즘에 대한 검증방식이 이미 개발되어 있고 , CMVP라는 암호 모듈 제품 평가 체계 프로그램 내에서 암호 알고리즘 구현 정확성 및 모듈 적합성 시험을 수행하고 있다. 국내의 경우에도 MOVS등과 같은 암호 알고리즘 및 모듈 적합성 시험 환경이 활발히 구축 중에 있으며, 현재 국내 표준화 알고리즘인 SEED, HAS-160, KCDSA 및 향후 개발될 기타 알고리즘들에 대한 검증 툴이 절실히 필요하게 되리라 예측된다.[7]

본 논문에서는 국내의 암호알고리즘 시험 환경 활성화의 일환으로써, CMVP 시험 및 표준화 연구, 자바 기반 시험환경 하에서 블록 암호 알고리즘 정확성 시험 모듈의 일부를 적용·구현하였다. 향후, 국내 표준화 암호알고리즘 시험에 관한 선행 연구 결과들을 이식·포함하고, CMVP의 시험 방법 및 절차를 국내 환경으로 보완·확장하여 추가의 테스트 베드 구축을 구현코저 한다.

본 연구 결과가 관련 연구 분야 및 향후 연구 목표에 자그마한 초석이 될 수 있기를 기대한다.

참고 문헌

[1] NIST, *FIPS 46-3 Data Encryption Standard,* NIST, Oct. 1999

[2] NIST, *NIST SP 800-17 : MOVS,* NIST, 1998

[3] NIST, *NIST special pub 800-20 : Modes of Operation Validation System for the Triple Data Encryption Algorithm(TMOVS)*, NIST, Apr. 2000

[4] NIST, *The Advanced Encryption Standard Algorithm Validation Suit(AESAVS)*, NIST, July 2002

[5] Bruce Schneier, *Applied Cryptography: Protocols, Algorithms, and Source Code in C, 2nd Edition* , John Wiley & Sons, 1997

[6] Jonathan Knudesen, *자바 보안과 암호화*, O'REILLY, 1998

[7] TTA, "128-bit Symmetric Block Cipher(SEED)", 한국 정보 통신 기술 협회, 1999

[8] Jess Garms, *Professional Java Security*, WROX Press , 2002

[9] http://www.kisa.or.kr

[10] http://www.cryptix.org

[11] http://java.sun.com/security/

[12] http://csrc.nist.gov/cryptval/