通信情報保護學會論文誌
第 11 券 第 2 號, 2001. 4

암호기술 구현물 검증도구 구현*

이 종 후**, 김 충 길**, 이 재 일***, 이 석 래***, 류 재 철**

Implementation of Validation Tool for Cryptographic Modules

Jong-hu Lee**, Chung-kil Kim**, Seok-lae Lee***, Jae-il Lee***, Jae-cheol Ryou**

요 약

암호기술 검증에 대한 연구는 암호 알고리즘의 안전성 및 신뢰성을 검증하는데 집중되어 있는 경우가 대부분이다. 그러나 정보보호 시스템의 수준 향상과 안전성 및 신뢰성 확보를 위해서는 암호기술 자체에 대한 검증뿐만 아니라, 암호기술을 구현한 구현물에 대한 검증이 필요하다. 특히, 암호기술에 대해서 국내외적으로 폭 넓은 표준화가 진행되고 있는 가운데, 이들 기술표준을 정확하게 구현하는 것은 정보보호 시스템의 안전성 및 신뢰성 향상을 가져올 뿐만 아니라, 정보보호 시스템 간의 상호연동성 확보 및 사용자 편익 증대라는 면에서도 매우 중요하다.

본 논문에서는 RSA, KCDSA, SHA-1, HAS-160 등 국내 공인인증체계에서 기술표준으로 적용되고 있는 암호기술의 구현물이 기술표준을 정확하게 준용하여 구현되었는지를 테스트할 수 있는 검증도구를 설계 및 구현하였다. 각각의 암호기술에 대한 검증은 여러 개의 세부항목으로 구성되어 있고, 충분한 테스트 항목을 통해 검증의 정확성을 높였으며, 검증도구와 검증 대상이 원격에 위치한 상태에서 검증을 수행할 수 있도록 하였다. 본 논문에서 설계 및 구현한 검증도구는 RSA, KCDSA, SHA-1, HAS-160 등을 구현한 모든 보안 제품에 적용할 수 있으며, 각종 암호제품의 평가 및 인증에 활용할 수 있을 것으로 기대된다.

ABSTRACT

There are relatively many research results of the validation of the cryptography. But few researches on the validation of cryptography implementations were accomplished. However, developer's misunderstanding the crypt-algorithm or a mistake in implementation of the crypto-algorithm leads to lose reliability of security products. Therefore, as validation of the crypto-algorithm itself also validation of the implementation is important.

The major objective of this paper is to propose Security Products Validation Tool. Our tool validates implementation of the public key algorithm (RSA, KCDSA) and hash algorithm (SHA-1, HAS-160). The validation process is composed of several items and our tool performs validation tests for conformance to related standard.

keyword : *Validation of cryptogaphy implementations, Conformance test to cryptography standards*

Ⅰ. 서 론

사회 전반에 걸쳐 디지털화가 가속화되면서 전자상거래뿐만 아니라 금융, 의료, 전자정부 등 경제·사회 각 부문에서 지속적인 디지털화가 나타날 것으로 기대된다. 이에 따라 국가안보, 경제 질서, 개인 사생활 등에 대한 위협 요인을 제거하고, 안전하고 신뢰할 수 있는 통신 환경을 조성하기 위해 인터넷

* 본 논문은 과학기술부/한국과학재단 지정 충남대학교 부설 소프트웨어연구센터의 지원으로 수행된 과제의 결과입니다.
* 충남대학교 정보통신공학부
** 한국정보보호센터

보안기술에 대한 필요성 및 요구는 급격히 증대하고 있다.

인터넷 보안기술의 중요성이 강조되면서, 세계 각국은 보안분야의 기술력 확보를 위한 연구를 지속적으로 수행해오고 있다. 특히 보안기술은 안전하고 신뢰할 수 있는 통신환경 구축에 있어서 반드시 필요한 기반기술이라는 점에서 국가적인 차원에서 연구가 진행되고 있는 것이 특징이다. 우리나라를 비롯한 세계 각국에서 진행되고 있는 국가 공개키기반구조 구축 작업은 이와 같은 국가적 차원의 연구의 대표적인 예라고 할 수 있다.

공개키기반구조를 비롯한 여러 가지 인터넷 보안기술을 개발 및 운용하는데 있어서 반드시 고려해야 하는 요소 가운데 하나는 보안 시스템이 기본적으로 안전성 및 신뢰성 등의 요구조건을 만족하며, 확장성 및 상호연동성을 갖고있는지 확인하는 것이다. 만약 통신환경에 안전성 및 신뢰성 제공을 위해 구축된 보안 시스템이 이와 같은 사항을 만족하지 못한다면, 사용자는 이러한 보안 시스템의 안전성 및 신뢰성을 의심할 수밖에 없으며, 결과적으로 사용자에게 불이익을 가져다주게 된다. 특히, 인터넷 보안기술은 여러 가지 암호기술에 기반해서 제공되고 있는데, 암호기술은 그 특성상 구현시 발생하는 사소한 취약점이 전체 시스템에 치명적인 영향을 줄 수 있다. 따라서 암호기술의 구현에는 많은 주의가 필요하다.

이와 같은 암호기술은 현재 관용 암호 알고리즘으로는 DES, 공개키 알고리즘은 RSA, 해쉬 알고리즘으로는 SHA-1 등이 전세계적으로 사실상 표준으로 인정받아 널리 사용되고 있다. 따라서 이들 기술표준을 정확하게 준용하여 구현하는 것은 안전성 및 신뢰성 확보라는 측면뿐만 아니라, 보안 시스템들 간의 상호연동성 확보라는 측면에서도 매우 중요하다. 국내에서는 전자서명법의 시행과 함께 공인인증기관 지정제도가 시행되고 있는데, 국내 공인인증체계에서는 RSA, SHA-1 외에도, KCDSA, SEED, HAS-160 등 국내 표준 암호기술을 사용하도록 하고 있다. 국내 공인인증체계의 안전하고 원활한 운용을 위해서는 이와 같은 국내 기술표준을 정확하게 준용하여 구현하여야 한다.

이와 같이 암호기술의 정확한 구현은 각종 보안기술의 안전하고 원활한 운용을 위해서 반드시 필요한 사항이다. 따라서 암호기술의 구현물이 기술표준을 정확하게 준용하여 구현되었는지 여부를 검증하는 절차가 반드시 필요하다.

이에 본 논문에서는 국제적으로 사실상 표준으로 받아들여지고 있는 RSA, SHA-1 및 KCDSA, HAS-160 등 국내 표준 암호기술을 구현한 구현물이 기술표준을 정확하게 준용하여 구현되었는지 여부를 테스트하는 검증도구를 설계 및 구현하였다. 본 논문의 2장에서는 미국의 NIST에서 수행하고 있는 암호기술 구현물에 대한 검증 프로그램에 대해서 살펴본다. 3장에서는 본 논문에서 제안하는 RSA, KCDSA, SHA-1, HAS-160 등의 기술표준 구현물에 대한 검증을 수행할 수 있는 검증도구의 설계 및 구현 내용에 대해서 살펴본다. 그리고 마지막으로 4장에서 결론을 맺는다.

II. 관련 연구

미국의 NIST(National Institute of Standard and Technology)에서는 미연방 정부의 관용 알고리즘 표준인 DES(Data Encryption Standard, FIPS46-3), 전자서명 알고리즘 표준인 DSS (Digital Signature Standard, FIPS 186), 해쉬 알고리즘 표준인 SHS(Secure Hash Standard), 그리고 Skipjack, 3-DES 등의 암호 알고리즘의 구현물이 표준을 준수하였는지 여부를 테스트하고 있다. 이 테스트는 NIST의 암호모듈 검증 프로그램(CMVP: Cryptographic Module Validation Program)의 일환으로 진행되고 있으며, CMT (Cryptographic Modules Testing) 연구소에서 주관하고 있다.

이 가운데, DES와 Skipjack에 대한 검증은[1]에 기반해서 이루어지고 있으며, 3-DES에 대한 검증은[2]에 기반해서 이루어지고 있다. 또한 전자서명 및 해쉬 알고리즘에 대한 검증은 역시 CMT 연구소에서 개발한 전자서명 표준 검증 시스템(DSSVS: Digital Signature Standard Validation System)[3]을 통해 이루어지고 있다. 이와 같은 검증 프로그램에 의해서 현재 DES 구현물은 101개, 3-DES 구현물은 32개, Skipjack 구현물은 4개가 NIST의 인증을 받았으며, DSA 및 SHA 구현물은 RSA Data Security의 B-SAFE Crypto-C 등 39개 업체, SHA 구현물은 Entrust Technologies Ltd.의 Entrust Security Kernel 등 40개 업체가 NIST의 인증을 받았다.[4] 그러나 아직까지 RSA 및 ECDSA에 대한 검증은 수행하고 있지 않다.

본 장에서는 이와 같은 암호 모듈 검증 프로그램

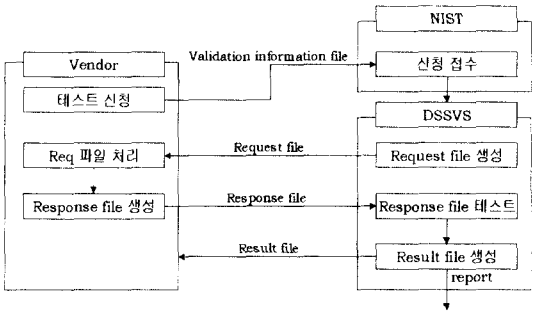
가운데 검증도구를 개발하여 전자서명 알고리즘과 해쉬 알고리즘에 대해서 검증을 수행하고 있는 DSSVS에 대해서 살펴보고자 한다. DSSVS는 크게 전자서명 알고리즘인 DSS를 검증하는 부분과 해쉬 알고리즘인 SHA-1을 검증하는 부분으로 구분할 수 있다.

2.1 DSSVS 설계 원칙 및 검증 절차

DSSVS는 DSS와 SHA-1을 구현한 하드웨어 및 소프트웨어에 대한 원격 테스트를 수행할 수 있도록 설계되었으며, 제품의 보안 강도를 측정하는 것이 아닌 표준의 준수 여부를 검증하는 프로그램이다. 즉, 이 프로그램은 DSS와 SHA-1를 구현하는 과정에서 발생할 수 있는 오류를 발견하도록 도와주는 역할을 한다고 할 수 있다. 따라서 NIST의 인증은 제품의 보안 강도에 대한 평가로 해석할 수 없다. 또한 DSSVS의 테스트는 구현물에서 생성한 모든 결과물에 대해서 테스트를 수행하지 않고, 결과물의 일부만을 선택해서 테스트하는 통계적인 방법으로 이루어진다. DSSVS에 의한 테스트 절차는 [그림 1]과 같다.

NIST로부터 인증을 받고자하는 업체는 신청서를 제출한다. 이 때, 테스트를 위해 필요한 업체의 정보 등을 저장한 파일(.inf)을 함께 제출한다. 신청을 접수한 NIST에서는 이를 처리하고, 테스트 파일을 DSSVS에 입력한다. 이 때, 업체에서 제출해야 하는 정보는 다음과 같다.

- 구현물 이름 및 버전
- 업체명 및 담당자
- 테스트 대상 제품이 소프트웨어인 경우, 운영체제, 사용 프로세서 등 운용 환경
- 제품에 대한 간략한 설명 등



[그림 1] DSSVS에 의한 테스트 절차

DSSVS는 입력받은 테스트 파일을 기반으로 해서 요청 파일(.req)을 생성하여 업체에 전달한다. 이 파일에는 테스트를 위해 업체의 구현물이 처리해야 하는 정보가 저장되어 있다. 요청 파일을 받은 업체에서는 이를 처리하여, 그 결과를 응답 파일(.rsp)에 저장하여 DSSVS로 전달한다. DSSVS는 이를 검토하여 결과 파일(.out)을 생성한다.

2.2 SHS 테스트

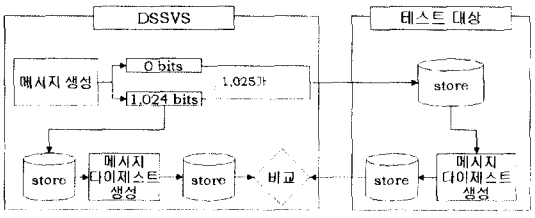
SHS에 대한 테스트는 크게 다음과 같은 3가지 종류로 구분할 수 있다.

- 다양한 길이의 메시지(messages of varying length)에 대한 테스트
- 선택된 긴 메시지(selected long messages)에 대한 테스트
- 임의로 생성된 메시지(pseudorandomly generated messages)에 대한 테스트

이들 각각에 대해서 살펴보면 다음과 같다.

2.2.1 다양한 길이의 메시지에 대한 테스트

SHS의 구현물은 임의의 길이를 가진 메시지에 대해 정확하게 메시지 다이제스트를 생성할 수 있어야 한다. 이를 확인하기 위해서 DSSVS는 0비트부터 1,024비트의 길이를 가지는 1,025개의 메시지를 임의로 생성하여 테스트 대상에게 전달하고, 각각의 메시지에 대한 메시지 다이제스트를 생성하여 저장한다. 메시지를 전달받은 테스트 대상은 이 메시지 각각에 대해서 메시지 다이제스트를 생성하여 DSSVS에 전달한다. 테스트 대상으로부터 메시지 다이제스트를 전달받은 DSSVS는 자신이 생성하여 저장하고 있는 메시지 다이제스트와 비교하여 모든 메시지 다이제스트가 동일하면 테스트를 통과한 것으로 한다. [그림 2]는 다양한 길이를 가지는 메시지에 대한 SHS 테스트 절차를 보여준다.



[그림 2] SHS 테스트 - 다양한 길이의 메시지

2.2.2 선택된 긴 메시지

SHS 구현물은 1,024비트 이상의 긴 메시지에 대해서 정확한 메시지 다이제스트를 생성할 수 있어야 한다. 테스트 절차는 다양한 길이의 메시지 테스트와 동일하며, 테스트에 사용하는 메시지는 1,024비트 이상의 다양한 길이를 갖는 100개의 메시지이다.

2.2.3 임의로 생성된 메시지

SHS 구현물은 임의로 생성된 메시지에 대해서 정확하게 메시지 다이제스트를 생성할 수 있어야 한다. 테스트 절차는 다양한 길이의 메시지 테스트와 동일하며, 테스트에 사용하는 메시지는 420비트의 길이를 갖는 시드(seed) 값으로부터 생성된 100개의 메시지이다.

2.3 DSS 테스트

DSSVS에 의한 DSS 테스트는 다음과 같은 6가지 항목으로 구성된다.

- 소수 테스트(primality test)
- 파라미터 p, q 생성 테스트
- 다른 구현물에서 생성한 p, q, g의 정확성 검사 테스트
- 공개키쌍 (x,y) 생성 테스트
- 서명 생성 테스트
- 서명 확인 테스트

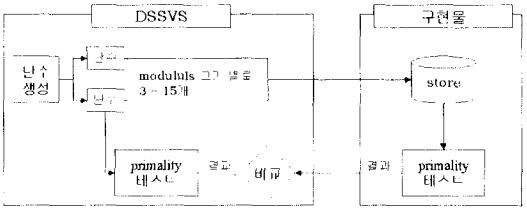
2.3.1 소수 테스트(primality test)

DSS의 구현물에는 소수 테스트를 위해 다음과 함수가 사용된다.

```
prime(w)    if w is prime return true;
            else return false;
```

이 함수는 $2 < w < 2^{1,MAX}$인 정수 w를 입력 받아 w의 소수 여부를 판단하는 함수이다. 따라서 소수 테스트는 테스트 대상의 prime(w) 함수가 소수 여부를 정확하게 판단하는지 확인하는 것으로 이루어진다.

DSSVS는 [그림 3]과 같이 모듈러 크기에 따라 [표 1]을 통해 지정된 개수의 임의의 수를 생성하여 테스트 대상에게 전달하고, 각각의 수에 대한 소수 여부를 판단하여 저장한다. 테스트 대상은 전달받은 수에 대해 소수 테스트를 수행한 후 그 결과를 DSSVS



[그림 3] DSS 테스트 - 소수 테스트

[표 1] 소수 테스트와 p, q, g 생성 테스트에 필요한 난수 및 p, q, g쌍의 개수

모듈러 크기 (비트)	소수 테스트와 p, q, g 생성 테스트에 필요한 난수 및 p, q, g 쌍의 개수			
	모든 사이즈에 대해서 테스트 하는 경우	512, 768, 1024에 대해서 테스트하는 경우	960, 1024에 대해서 테스트하는 경우	1024에 대해서 테스트하는 경우
512	15	15	-	-
576	15	-	-	-
640	13	-	-	-
704	9	-	-	-
768	7	15	-	-
832	6	-	-	-
896	4	-	-	-
960	3	-	7	-
1024	3	7	6	10

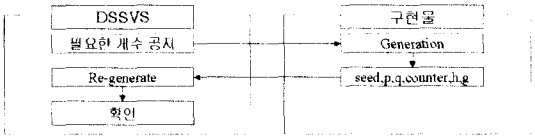
에 전달한다. DSSVS는 테스트 대상으로부터 전달받은 결과와 자신이 생성한 결과를 비교한다.

2.3.2 p, q, g 생성(Generation of p, q and g)

DSS 구현물은 FIPS에서 승인된 방법을 통해 소수 p, q를 생성해야 하며, (p, q) 쌍에 대한 파라미터 g 또한 정확하게 생성해야 한다. g의 정확성은 g를 생성하는데 사용되는 파라미터 h를 검사함으로써 알 수 있다. 즉 h와 g는 다음과 같은 조건을 만족해야 한다.

$$1 < h < p - 1$$
$$h^{(p-1)/q} \bmod p > 1$$

DSSVS는 [그림 4]와 같이 테스트에 필요한 p,q,g 쌍의 개수를 테스트 대상에 알린다. 이 때 몇 개의 p, q, g 쌍을 테스트 할 것인가는 [표 1]에 기반해서 결정한다. 테스트 대상은 모듈러 크기별로 (seed, q,

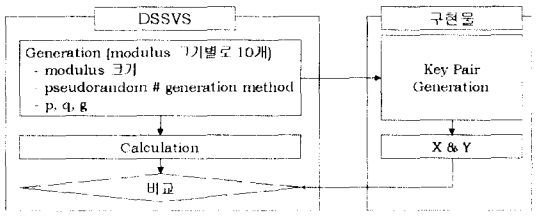


〔그림 4〕 DSS 테스트 - p, q, g 테스트

p, counter, h, g)를 생성하여 DSSVS에 전달한다. DSSVS는 전달 받은 값을 다시 계산함으로써 테스트를 수행한다.

2.3.3 공개키쌍 생성 (Key Generation for Private and Public Key Pairs)

〔그림 5〕와 같이 DSSVS는 공개키쌍을 생성하는데 필요한 정보인 모듈러 크기, 난수 생성 방법, 파라미터 p, q, g를 검증 대상에 전달하고, 이 정보를 이용하여 공개키쌍을 생성해서 저장한다. 이 때 각 모듈러 크기별로 10개의 공개키쌍을 생성하도록 한다. DSSVS로부터 필요한 정보를 받은 테스트 대상은 이를 이용하여 공개키쌍을 생성한 뒤, 이를 DSSVS에 전달한다. DSSVS는 자신이 생성한 공개키쌍과 테스트 대상으로부터 받은 공개키쌍을 비교하며, 모두 동일하면 테스트를 통과한 것으로 한다.

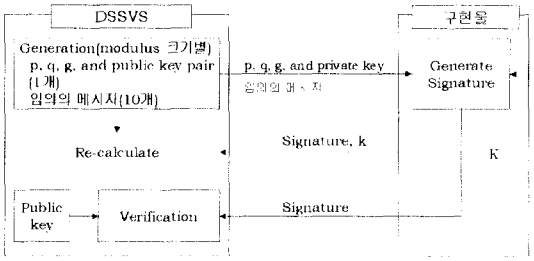


〔그림 5〕 DSS 테스트 - 공개키쌍 생성

2.3.4 전자서명 생성(Signature Generation)

〔그림 6〕과 같이 DSSVS는 파라미터 p, q, g를 생성하고 이를 통해 하나의 공개키쌍을 생성하고, 10개의 임의의 메시지를 생성하여 저장한 뒤, 이를 테스트 대상에게 전달한다. 테스트 대상은 DSSVS로부터 받은 정보를 이용하여 메시지에 대해 전자서명을 수행한 결과를 DSSVS에게 전달한다. DSSVS은 서명된 메시지에 대해 전자서명을 확인하는 테스트를 수행한다.

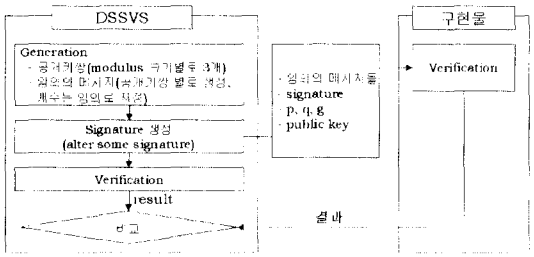
이 때 테스트 대상에서 전자서명을 생성하는데 내부 파라미터 k가 사용된다. 따라서 서명된 메시지와 함께 k를 DSSVS에 전달하여야 한다. 서명된 메시지와 k를 전달받은 DSSVS는 k를 이용하여 메시지에 전자서명하고 이를 확인한다.



〔그림 6〕 DSS 테스트 - 전자서명 생성

2.3.5 전자서명 확인(Signature Verification)

전자서명 확인에서는 테스트 대상이 전자서명의 유효성 확인을 올바르게 수행하는지 테스트한다. 즉, 전자서명 메시지가 변조된 경우, 이를 확인할 수 있는지 테스트한다. 〔그림 7〕과 같이 DSSVS는 모듈러 크기별로 3개의 공개키쌍과 임의의 메시지를 생성하여 저장한 뒤, 이를 이용해 메시지에 전자서명한다. 그후, 메시지, 전자서명된 메시지, 파라미터 p, q, g, 공개키를 테스트 대상에게 전달한다. 이 때 DSSVS는 테스트를 위해 몇 개의 전자서명된 메시지를 변경한다. 테스트 대상은 전달받은 정보를 통해 전자서명의 유효성 여부를 확인하여 그 결과를 DSSVS에게 전달한다. DSSVS는 테스트 대상으로부터 전달 받은 결과를 자신의 확인 결과와 비교한다.

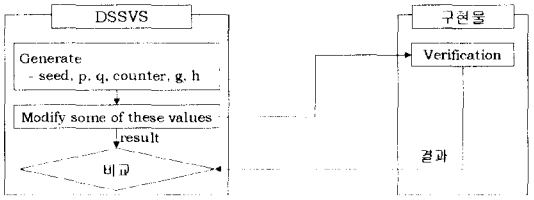


〔그림 7〕 DSS 테스트 - 전자서명 확인

2.3.6 p, q, g의 정확성 확인

DSS의 구현물은 파라미터 p, q, g를 직접 생성할 수도 있지만, 외부로부터 가져와서 사용할 수도 있다. 이러한 경우에는 외부로부터 가져온 파라미터들의 정확성을 확인할 수 있는 수단이 갖춰져야 한다. 이 테스트에서는 이 수단이 올바르게 동작하는지 확인한다.

DSSVS는 〔그림 8〕과 같이 (seed, q, p, counter, g, h)를 생성하고, 이 들 가운데 몇 개의 값을 변경한 뒤, 테스트 대상에서 전달한다. 이 때 테스트를 위해 생성되는 (seed, q, p, counter, g, h)의 개



〔그림 8〕 DSS 테스트 - p, q, g의 정확성 확인

수는 〔표 1〕을 참조한다. 테스트 대상은 전달 받은 정보들에 대해 유효성 검사를 수행하여 그 결과를 DSSVS에게 전달한다. DSSVS는 전달받은 결과를 검사한다.

Ⅲ. 검증도구 설계 및 구현

본장에서는 본 논문에서 설계 및 구현한 검증도구를 소개한다. 검증도구는 크게 전자서명 알고리즘에 대해서 테스트하는 부분과 해쉬 알고리즘에 대해서 테스트하는 부분으로 구분할 수 있다. 검증 대상이 되는 전자서명 알고리즘은 RSA와 KCDSA이며, 해쉬 알고리즘은 SHA-1과 HAS-160이다. 이 들 알고리즘은 국내 공인인증체계에서 사용되고 있는 암호기술로, 검증기술이 가장 먼저 적용되어야 하는 기술들이다.

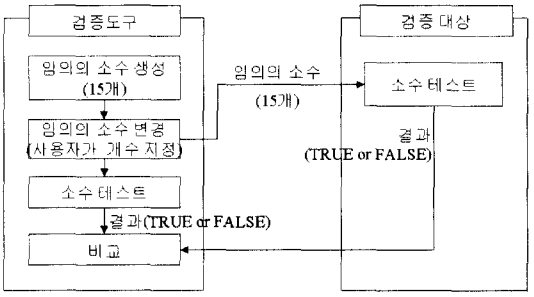
3.1 RSA 검증

RSA 구현물에 대한 검증 부분에서는 검증 대상이 RSA 공개키 알고리즘을 적합하게 구현하였는지 테스트한다. 이를 위해서 크게 다음과 같은 5가지 기능을 테스트하며, 5가지의 테스트를 모두 통과하여야만 적합하게 구현한 것으로 간주한다. 이와 같은 테스트 항목의 선정은 NIST의 연구에 기반해서 이루어졌다.

- 소수(素數) 테스트
- p, q, n 생성의 적합성 테스트
- 공개키쌍 생성 테스트(e, d 생성 테스트)
- 전자서명 생성 테스트
- 전자서명 유효성 검사 테스트

3.1.1 소수(素數) 테스트

소수 테스트에서는 검증 대상이 공개키쌍 생성을 위해 필요한 파라미터인 p, q의 소수 여부를 적절하게 판단할 수 있는지 테스트한다. 테스트 절차는 〔그림 9〕와 같다.



〔그림 9〕 소수 테스트

① 검증도구는 임의의 소수 15개를 생성한다. 이 때, 다음과 같은 소수 생성 방법을 사용한다.
(1) 임의의 홀수 n 선택
(2) a 〈 n인 임의의 정수 a 선택
(3) Miller-Rabin 수행, 만약 n이 소수가 아니라면 (1)을 수행
(4) n이 소수 테스트를 통과하면 n을 소수로 선택, 그렇지 않으면 (2)를 수행

이 때, 소수 여부를 확인하기 위해서 사용한 Miller-Rabin 방법은 소수 여부를 확인하기 위해서 가장 많이 사용되는 방법으로 자세한 내용은 〔그림 10〕과 같다.

```
WITNESS (a,n)

let bkbk-1 . . . B0 be the binary representation of (n-1)
d = 1
for i=k downto 0
   do x = d
   d = (d * d) mod n
   if d == 1 and x != 1 and X != n-1
   then return TRUE
   if bi == 1
   then d = (d * a) mod n
if d != 1
then return TRUE
return FALSE
```

〔그림 10〕 Miller-Rabin 알고리즘

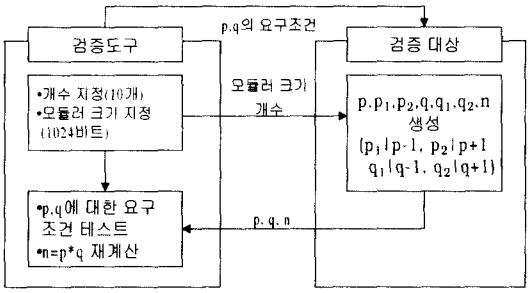
② 생성한 소수 가운데 일부를 임의로 변경한다. 변경된 수를 포함한 15개의 수를 파일에 저장하고, 이 파일을 검증 대상에게 전달한다.
③ 15개의 수에 대해서 소수 여부를 테스트하여 결과를 저장한다. 소수 여부를 테스트 하는 방법은 ①과 동일하다.
④ 검증 대상은 전달받은 값에 대해서 소수 여부를 판단하여 그 결과를 파일에 저장하고, 이를 검증도구에게 전달한다.
⑤ 검증도구는 자신의 소수 테스트 수행 결과와 검증 대상의 수행 결과를 비교하여 결과를 파일에

저장한다. 15개의 수에 대해서 모두 결과가 일치하면 테스트를 통과한 것으로 간주한다.

이 때, 실제 검증 수행시 검증 수행자가 변경되는 소수의 개수와 Miller-Rabin 알고리즘의 수행 횟수를 지정할 수 있도록 하였다.

3.1.2 p, q, n 생성 테스트

이 테스트에서는 검증 대상이 요구조건에 따라 강한소수(strong prime)인 p, q을 생성하고 이로부터 n을 생성할 수 있는 능력을 갖추었는지 테스트한다. 이에 대한 테스트 절차는 [그림 11]과 같다.



[그림 11] p, q, n 생성의 테스트

① 검증도구는 테스트에 필요한 (p, q, n)의 개수와 모듈러 크기를 지정하여 자신의 로그 파일에 기록하고 검증 대상에게 알린다. 이 때 테스트에 필요한 (p, q, n)쌍의 개수는 10개이며, 모듈러 크기는 1024비트이다. 또한 p, q에 대한 요구조건이 함께 전달되는데, 이는 강한 소수에 대한 요구조건으로, 그 내용은 다음과 같다.[7]

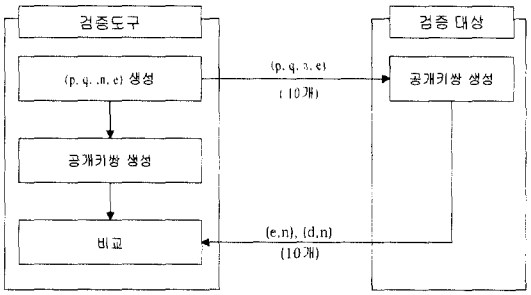
- p-1과 q-1의 최대공약수는 작아야 한다.
- p-q와 q-1은 각각 p와 q에 대해서 큰 소인수를 가져야 한다.
- p-1과 q-1은 큰 소인수를 가져야 한다.
- p+1과 q+1은 큰 소인수를 가져야 한다.
- (p-1)/2과 (q-1)/2은 소수이어야 한다.

② 검증 대상은 지정된 개수만큼 (p, q, n)을 생성, 디스켓에 저장하여 검증도구에게 전달한다. 이 때, (p, q, n)을 생성하는데 사용되는 파라미터인 p1, p2, q1, q2를 함께 전달한다.

③ (p, q, n)을 전달 받은 검증도구는 p와 q가 요구조건을 모두 만족하는지 확인하고, 만족하는 경우에는 p와 q를 이용하여 n을 계산한 뒤, 검증 대상이 생성하여 전달한 n과 비교한다. 모든 결과가 동일할 때만 테스트를 통과한 것으로 간주하며, p, q가 요구조건을 만족하지 못할 경우에는 테스트는 실패이다.

3.1.3 공개키쌍 생성 테스트

공개키쌍 생성 테스트에서는 검증 대상이 검증도구로부터 제공 받은 파라미터를 이용하여 공개키쌍을 적합하게 생성할 수 있는지 테스트한다. 이에 대한 테스트 절차는 [그림 12]와 같다.



[그림 12] 공개키쌍 생성 테스트

① 검증도구는 공개키쌍 생성에 필요한 (p, q, d, e)를 10개 생성한다.

② 검증도구는 생성한 10개의 (p, q, d, e)을 검증 대상에게 전달한다.

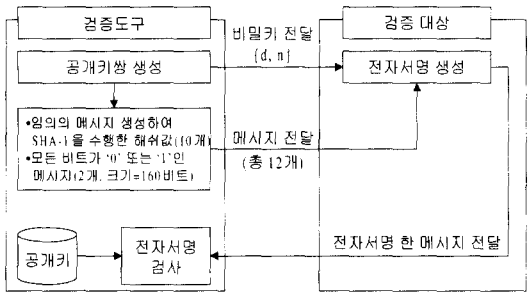
③ 검증도구는 생성한 파라미터들을 이용하여 10개의 공개키쌍 {e, n}, {d, n}을 생성하여 저장한다.

④ 검증 대상은 전달받은 파라미터들을 통해 10개의 공개키쌍을 생성하여 검증도구에게 전달한다.

⑤ 검증도구는 자신이 생성한 공개키쌍과 검증 대상으로부터 전달받은 공개키쌍을 비교하여 일치 여부를 확인한다. 10개의 공개키쌍이 모두 일치할 경우에만 테스트를 통과한 것으로 간주한다.

3.1.4 전자서명 생성 테스트

이 테스트에서는 검증 대상이 비밀키를 이용하여 전자서명을 적합하게 생성할 수 있는지 테스트한다. 여기서는 단순히 비밀키를 이용한 전자서명만을 테스트하며, 해쉬함수를 이용하여 메시지 다이제스트를 생성하는 것에 대한 테스트는 제외한다. 테스트 절차는 [그림 13]과 같다.



〔그림 13〕 전자서명 생성 테스트

① 전자서명에 사용되는 메시지의 크기는 160비트이다. 검증도구는 하나의 공개키쌍을 생성한다. 그리고 10개의 임의의 메시지를 생성하여 이에 대해서 SHA-1을 수행한다. 이 때 추가적으로 모든 비트가 '0' 또는 '1'인 메시지를 생성한다.

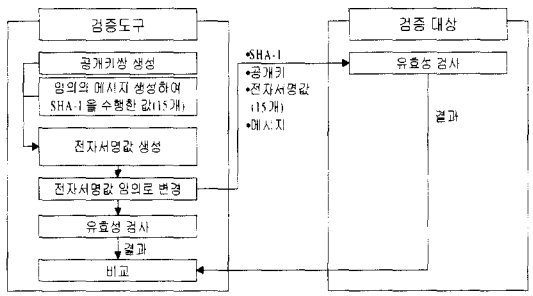
② 검증도구는 생성한 공개키쌍과 메시지를 저장하고, 비밀키와 12개의 메시지를 검증 대상에게 전달한다.

③ 검증도구로부터 비밀키와 메시지를 전달받은 검증 대상은 각각의 메시지에 전자서명을 첨부한다. 그리고 이를 검증도구에게 전달한다.

④ 검증도구는 12개의 전자서명 한 메시지에 대해서 자신이 생성한 공개키를 이용하여 유효성 검사를 수행한다. 12개의 메시지가 모두 성공적으로 유효성 검사를 통과하면 테스트를 통과한 것으로 간주한다.

3.1.5 전자서명 유효성 검사 테스트

이 테스트에서는 검사 대상이 전자서명한 메시지에 대해서 유효성 검사를 수행하여 이상이 없는 전자서명인지 확인할 수 있는 능력을 갖추었는지 테스트한다. 테스트 절차는 〔그림 14〕와 같다.



〔그림 14〕 전자서명 유효성 검사

① 검증도구는 공개키쌍과 15개의 임의의 메시지를 생성하고, 이 들 메시지를 입력으로 하여 SHA-1을 수행한다.

② 검증도구는 임의의 메시지에 각각 전자서명을 하고, 이 가운데 몇 개의 메시지를 임의로 변경하여 저장한다. 이 때 변경되는 메시지의 개수는 사용자가 지정할 수 있다.

③ 검증도구는 임의로 변경된 메시지를 포함하여 전자서명한 메시지와 전자서명 유효성 검사에 필요한 공개키를 검증 대상에게 전달한다.

④ 검증 대상은 전달받은 메시지에 대해서 유효성 검사를 수행하여 그 결과를 검증도구에게 전달한다.

⑤ 검증도구는 검증도구 내에서 수행한 유효성 검사의 결과와 검증 대상으로부터 전달받은 결과를 비교한다. 모든 결과가 동일할 경우, 테스트를 통과한 것으로 간주한다.

3.2 KCDSA 검증

KCDSA 검증에서는 검증 대상이 KCDSA 전자서명 알고리즘을 적합하게 구현하였는지 테스트한다. KCDSA 검증은 NIST의 DSS 검증과 유사하며, 크게 다음과 같은 6가지 기능을 테스트한다. 이 때, 6가지의 테스트를 모두 통과하여야만 적합하게 구현한 것으로 간주한다.

- 소수(素數) 테스트
- p, q, g 생성 테스트
- 다른 구현물에서 생성한 p,q,g의 정확성 검사 테스트
- 공개키쌍 생성 테스트
- 전자서명 생성 테스트
- 전자서명 확인 테스트

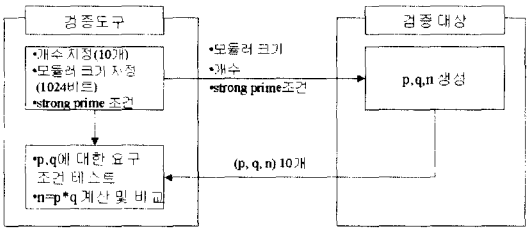
검증은 RSA 검증과 마찬가지로 1,024비트의 KCDSA 구현물에 대해서만 이루어진다.

3.2.1 소수(素數) 테스트

소수 테스트에서는 검증 대상이 공개키쌍 생성을 위해 필요한 파라미터의 소수 여부를 적절하게 판단할 수 있는지 테스트한다. 테스트 절차는 RSA 검증에서의 소수 테스트와 동일하다.

3.2.2 p, q, g 생성 테스트

KCDSA에서 공개키쌍 생성을 위해 사용되는 파라



〔그림 15〕 p, q, g 생성 테스트

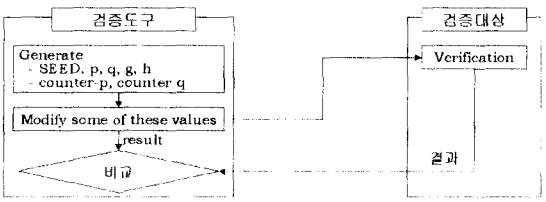
미터는 다음과 같은 연산을 통해서 구할 수 있다.

$$1 < h < p-1$$
$$g = h^{(p-1)/q} \bmod p > 1$$
$$p = 2qq' + 1 (q > q')$$

이 테스트에서는 검증 대상이 p, q, g를 적합한 방법으로 생성할 수 있는지 테스트하며, 테스트 절차는 〔그림 15〕와 같다. 테스트 절차는 RSA 검증에서의 p, q 생성 테스트와 유사하며, 다만 검증대상에서 검증도구에게 전달하는 파라미터만이 달라진다.

3.2.3 p, q, g의 정확성 확인 테스트

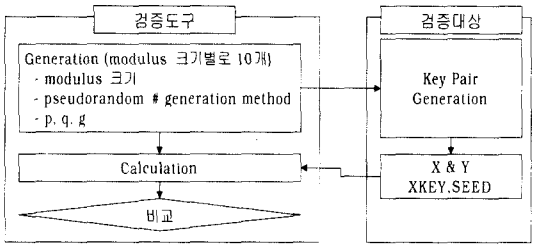
KCDSA의 검증에서는 RSA 검증에서는 포함되지 않은 p, q, g의 정확성 확인 테스트를 수행한다. 이는 검증 대상이 키를 생성하는데 있어서 내부에서 생성하지 않고 외부에서 파라미터를 받아서 사용할 경우에 파라미터의 정확성 여부를 확인할 수 있는 능력을 갖추었는지 여부를 테스트하는 것이다. 테스트 절차는 〔그림 16〕과 같다. 이는 NIST의 DSA 검증과 일부 파라미터를 제외하고는 동일하다.



〔그림 16〕 p, q, g의 정확성 확인 테스트

3.2.4 공개키쌍 생성 테스트

공개키쌍 생성 테스트에서는 RSA 검증과 마찬가지로 검증대상이 검증도구로부터 제공받은 파라미터



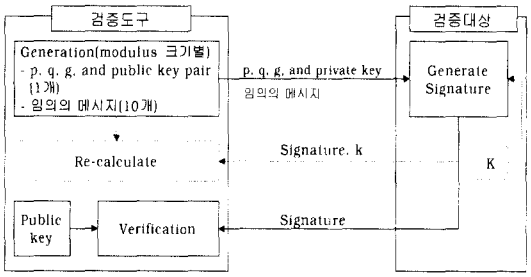
〔그림 17〕 공개키쌍 생성 테스트

를 이용하여 공개키쌍을 적합하게 생성할 수 있는지 테스트한다. 테스트 절차는 〔그림 17〕과 같다.

이 테스트에서는 검증도구에게 의사난수 생성 방법을 전달하고, 검증대상이 이에 따라 난수를 생성하도록 하고, 공개키쌍 외에 XKEY와 seed까지 전달받아 의사난수 생성 테스트를 함께 수행하도록 하였다.

3.2.5 전자서명 생성 테스트

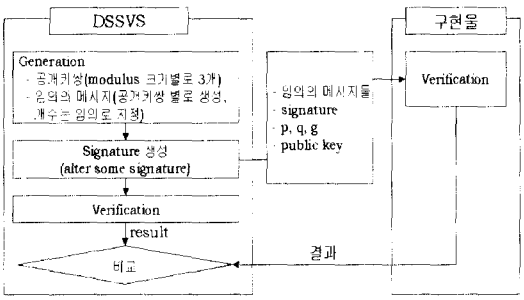
이 테스트에서는 검증 대상이 비밀키를 이용하여 전자서명을 적합하게 생성할 수 있는지 테스트하며, 테스트 절차는 〔그림 18〕과 같이 RSA 검증의 전자서명 생성 테스트와 동일하다. 다만 KCDSA의 경우 전자서명 생성시 비밀키 외에 파라미터 k가 사용되므로, 검증대상이 k를 사용하여 전자서명을 생성한 경우에는 검증도구는 검증대상으로부터 k를 전달받아야 한다.



〔그림 18〕 전자서명 생성 테스트

3.2.6 전자서명 유효성 검사 테스트

이 테스트에서는 검증대상이 전자서명한 메시지에 대해서 유효성 검사를 수행하여 이상이 없는 전자서명인지 여부를 확인할 수 있는 능력을 갖추었는지 테스트한다. 테스트 절차는 〔그림 19〕와 같으며, RSA 검증시 전자서명 유효성 검사 테스트와 동일하다.



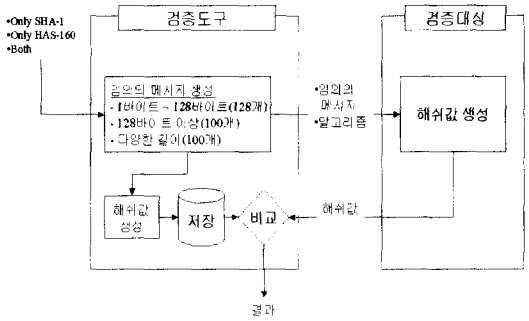
〔그림 19〕 전자서명 유효성 검사 테스트

3.3 해쉬 알고리즘 검증

해쉬 알고리즘 검증에서는 검증 대상이 해쉬 알고리즘을 적합하게 구현하였는지 테스트한다. 이 때 검증 대상이 되는 해쉬 알고리즘은 SHA-1과 HAS-160이며, 검증 절차는 알고리즘의 종류에 관계 없이 동일하다. 이 테스트는 검증도구에서 제공하는 입력을 검증 대상이 처리하여 출력하는 결과와 검증도구에서 출력하는 결과가 일치하는지 여부를 확인하는 것으로 이루어진다. 검증도구는 크게 다음과 같은 3가지 종류의 메시지를 출력하고 이를 검증 대상에게 전달한다. 검증도구로부터 메시지를 전달받은 검증 대상은 각각의 메시지를 해쉬함수에 입력으로 하여 나온 결과를 다시 검증도구에게 전달한다.

- 1바이트부터 128바이트까지의 크기를 가지는 임의의 메시지(총 128개)
- 128바이트 이상의 크기를 가지는 임의의 메시지 100개
- 임의의 메시지 100개

테스트 절차는 〔그림 20〕과 같다.



〔그림 20〕 해쉬 알고리즘 검증

① 검증도구는 3가지 종류의 임의의 메시지 328개를 생성한다. 이 때 검증 수행자는 SHA-1에 대해서만 검증을 수행할 것인지, 혹은 HAS-160에 대해서만 검증을 수행할 것인지, 아니면 두 알고리즘 모두에 대해서 검증을 수행할 것인지 지정할 수 있으며, 두 알고리즘 모두에 대해서 검증을 수행할 경우에는 동일한 메시지가 사용된다.

② 검증도구는 생성한 메시지를 자신의 저장장치에 저장하고, 메시지를 검증 대상에게 전달한다.

③ 검증도구는 자신이 생성한 각각의 메시지를 해쉬함수에 입력하여 그 결과를 저장한다.

④ 검증 대상은 검증도구로 전달받은 각각의 메시지의 해쉬값을 계산하여 검증도구에게 전달한다.

⑤ 검증도구는 자신의 계산한 결과와 검증 대상으로부터 전달받은 결과와의 일치 여부를 확인한다. 이 때 모든 결과와 일치해야만 테스트를 통과한 것으로 한다.

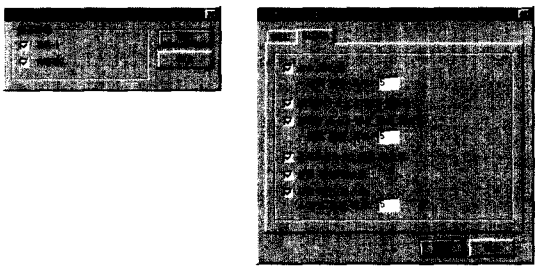
3.4 검증도구 구현

본 논문에서 구현한 암호기술 구현물 검증도구는 전자서명 알고리즘인 KCDSA와 RSA, 해쉬 알고리즘인 SHA-1과 HAS-160의 구현이 올바르게 이루어졌는지 테스트한다. 검증은 검증도구에서 생성한 검증에 필요한 정보를 검증 대상에게 전달하면, 검증 대상은 이 정보를 이용하여 전자서명 또는 해쉬를 수행한 후, 결과를 검증도구에게 전달한다. 검증도구는 검증 대상으로부터 전달받은 정보를 이용하여 검증을 수행한다. 이 때 검증에 필요한 정보는 모두 파일 단위로 전달되어지며, 이를 위해 검증도구와 검증 대상은 여러 가지 파일을 생성하는데, 이를 정리하면 〔표 2〕와 같다.

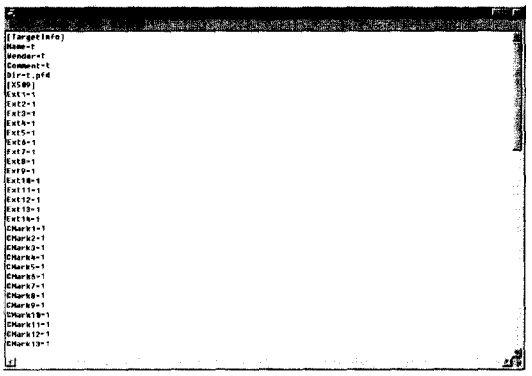
검증을 위해서는 〔그림 21〕과 같이 해쉬 알고리즘과 전자서명 알고리즘에 대한 검증을 위해 필요한 사항을 설정한다. 이 때, 사용자의 필요에 의해서 일부 항목에 대해서만 검증을 수행할 수 있도록 하였다. 검증도구는 이와 같은 설정 작업을 통해 프로파일 정보와 초기정보를 생성한다. 〔그림 22〕와 〔그림 23〕은 프로파일 정보와 초기정보 파일의 내용이다. 이 때 초기정보 파일은 해쉬 알고리즘에 대한 파일과 전자서명 알고리즘에 대한 파일이 각각 별개로 유지되며, 프로파일 정보 파일은 검증 대상에 대해서 하나가 존재한다.

〔표 2〕 검증 수행에 과정에서 생성되는 파일

구분 (파일명)	내용
프로파일 정보 (.pfd)	· 검증에 앞서 검증도구에서 생성하는 정보 · 검증도구에서 검증을 위해 내부적으로 사용
초기정보 (.gep)	· 검증에 앞서 검증도구에서 생성하는 정보 · 검증대상은 이를 이용하여 테스트 정보를 생성
테스트 정보 (.rep)	· 검증도구로부터 전달받은 정보를 통해 검증 대상에서 생성 · 검증도구에 전달되어 검증 수행에 사용
결과정보 (.res)	· 검증도구가 검증 수행후 생성하는 정보 · 검증대상의 검증 통과/실패 여부 기록
로그정보 (.log)	· 검증 수행 과정에서 검증도구가 생성하는 정보 · 결과정보보다 자세한 내용이 기록
검증도구 결과 (.cot)	· 검증도구에서 검증을 위해서 초기정보를 이용하여 생성하는 정보 · 테스트 정보와 검증도구 결과를 통해 검증 수행

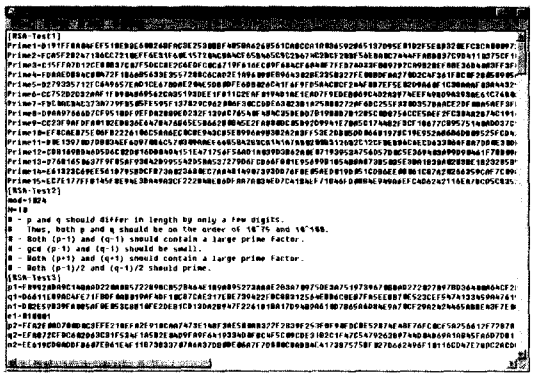


〔그림 21〕 검증도구 초기 설정 화면

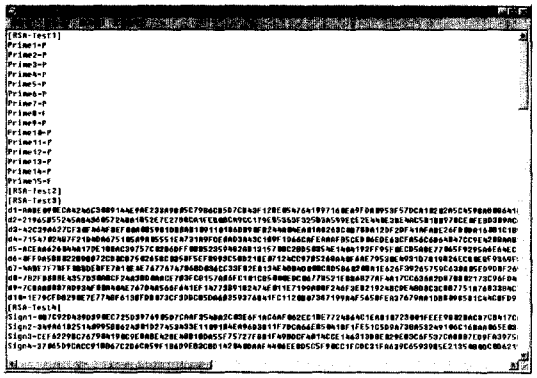


〔그림 22〕 프로파일 파일

검증도구는 생성된 초기정보 파일을 검증 대상에게 전달한다. 검증 대상은 초기정보 파일의 내용을 읽고, 이를 이용해서 검증 대상의 암호모듈을 이용해 초기정보 파일에 기술된 바에 따라 테스트 정보를 생성하고, 이를 테스트 정보 파일에 저장하여 검증도구에게 전달한다. 이 때, 검증 대상은 반드시



〔그림 23〕 초기 정보 파일

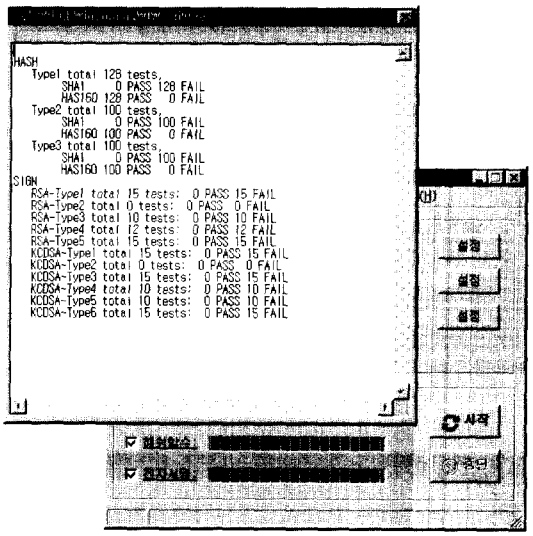


〔그림 24〕 검증도구 결과 파일

검증도구에서 요구한 형식에 따라 테스트 정보 파일을 생성하여야 한다.

검증 대상으로부터 테스트 정보를 전달받은 검증도구는 최초에 생성했던 초기정보를 이용하여 검증에 필요한 정보를 생성하고, 이를 검증도구 결과 파일에 저장한다. 예를 들어 해쉬 알고리즘에 대한 테스트의 경우, 초기정보 파일에는 메시지가 저장되어 있으며, 검증도구는 이를 통해 해쉬값을 계산하여 그 결과를 검증도구 결과 파일에 저장한다. 검증도구 결과 파일의 생성이 완료되면 검증도구는 검증도구 결과와 검증 대상이 생성한 테스트 정보를 이용해 검증을 수행한다. 검증도구는 검증도구 결과 파일과 테스트 정보 파일을 이용해서 테스트를 수행하기 때문에 이 2개의 파일의 형식은 반드시 동일해야 한다. 〔그림 24〕는 검증도구 결과 파일의 내용이다.

검증이 완료되면, 검증도구는 테스트의 성공/실패 여부를 결과 정보에 기록한다. 결과 정보 파일에는 성공/실패 여부와 함께 간략한 설명이 기록된다. 또한 검증 과정에서는 로그 정보 파일이 생성되는데,



〔그림 25〕 검증 결과

이는 결과 정보 파일에 비해서 보다 자세한 내용이 생성된다. 〔그림 25〕는 검증도구를 통해서 볼 수 있는 검증 결과이다.

이와 같은 검증도구의 구현 환경은 〔표 3〕과 같다.

〔표 3〕 개발 및 운용 환경

개발환경				운용환경
기종	운영체제	언어	라이브러리	운영체제
PII 350	Windows98	C++	Openssl 0.9.4[10]	Windows 95/98

또한 검증도구의 신뢰성 확보를 위해 관련 표준에서 제공하는 테스트 벡터와 이미 구현되어 있는 다른 암호모듈과의 연동시험을 수행했다. 이 때 사용된 다른 암호모듈은 오픈소스 암호모듈인 OpenSSL과 Ctyptlib이다. 이와 같은 오픈소스 암호모듈은 오픈소스 정책에 따라 전세계의 다양한 다양한 개발자들이 개발에 참여하고 있고, 메일링 리스트 등을 통한 지속적인 보완이 이루어지고 있어 비공개적으로 개발되고 있으며, 이런 면에서 안전성 및 신뢰성에 대한 상당 수준의 검증을 거쳤다고 볼 수 있다.

Ⅳ. 결 론

인터넷을 통해 제공되는 응용 서비스가 대규모화되고 복잡해짐에 따라 인터넷 상의 정보를 보호하기 위한 보안기술 역시 빠르게 발전하고 있다. 그러나 안전하고 신뢰할 수 있는 통신환경의 구축을 위해서는 정보보호 서비스를 위해 사용되는 보안기술들의 안전성이 보장되어야 한다. 또한 사용자 편익 증진 및 서비스의 원활한 운용을 위해서는 보안기술들 간의 상호연동성이 이루어져야 한다.

보안기술에 있어서 안전성과 신뢰성 및 상호연동성의 확보를 위해서는 암호기술의 정확한 구현이 필수적이며, 이는 공개되어 있는 암호 알고리즘을 정확하게 준용하여 구현함으로써 이루어진다. 계속해서 새로운 보안시스템이 개발되고 있는 가운데, 이러한 보안시스템이 암호기술을 정확하게 구현하였는지 여부를 검증하는 작업은 매우 중요하다.

본 논문에서는 현재 전세계적으로 사실상의 표준 암호기술로 받아들여지고 있는 RSA와 SHA-1이 올바르게 구현되었는지 여부를 검증할 수 있는 검증도구를 설계 및 구현하였다. 검증도구는 RSA와 SHA-1에 대한 검증과 함께 국내 기술표준인 KCDSA와 HAS-160의 구현물에 대한 검증도 수행한다.

각각의 암호기술에 대한 검증은 여러 개의 세부항목으로 구성되어 있으며, 충분한 테스트 자료를 사용하여 검증의 정확성을 높였다. 또한 검증도구와 검증대상이 원격에 위치한 상태에서 검증을 수행할 수 있도록 하였다. 그러나 본 논문에서 설계 및 구현한 검증도구는 암호기술의 정확한 구현 여부만을 테스트할 수 있다. 즉, 암호기술을 잘못 구현하여 발생할 수 있는 보안 헛점에 의한 안전성 문제 및 신뢰성에 대한 검증은 가능하지만, 암호기술 자체의 안전성 검증이나 구현물의 보안강도 평가는 수행하지 않는다.

본 논문에서 설계 및 구현한 검증도구는 RSA, KCDSA, SHA-1, HAS-160 등을 구현한 모든 보안 제품에 적용할 수 있다. 따라서 각종 암호제품의 평가 및 인증에 활용할 수 있을 것으로 기대된다. 예를 들어 공인인증기관 지정제도에 의한 공인인증기관 평가 및 인증시 암호기술의 정확한 구현 여부를 검증하기 위해 본 논문에서 설계 및 구현한 검증도구를 이용할 수 있다.

참 고 문 헌

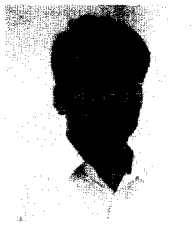
〔1〕 Sharon Keller, Miles Smid, "*NIST Special Publication 800-17 Modes of Operation Validation System(MOVS): Requirements and Procedures*", NIST, 1998. 2.

[2] Sharon Keller, "*NIST Special Publication 800-20 Mode of Operation Validation System for the Triple Data Encryption Algorithm(TMOVS): Requirements and Procedures*", NIST, 2000. 4.
[3] Jim Foti, "*Digital Signature Standard Validation System(DSSVS) User's Guide(For Version 2.3 of the DSSVS Software Tool)*", NIST Information Technology Laboratory, 1999. 5.
[4] NIST CMVP Web Site, "http://csrc.nist.gov/cryptval".
[5] 한국통신기술협회, "*해쉬함수표준 - 제2부 : 해쉬함수알고리즘(HAS-160)(Hash Function Standard - Part 2 : Hash Function Algorithm (HAS-160))*", 한국정보통신기술협회, 1998. 11.
[6] 한국통신기술협회, "*부가형 전자서명 방식 표준 - 제2부 : 확인서 이용 전자서명 알고리즘(Digital Signature Mechanism with Appendix - Part 2 : Certificate-based Digital Signature Algorithm))*", 한국정보통신기술협회, 1998. 11.
[7] Bruce Schneier, "*Applied Cryptography 2nd Edition*", John Wiley & Sons, Inc., 1996.
[8] William Stallings, "*Network and Internetwork Security*", Prentice Hall, 1995.
[9] OpenSSL Web Site, "http://www.openssl.org".

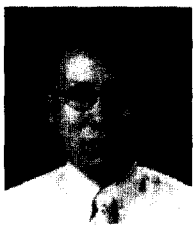
〈著者紹介〉

이 종 후(Jong-hu Lee)
1997년 2월 : 충남대학교 컴퓨터과학과 졸업
1999년 2월 : 충남대학교 컴퓨터과학과 석사
2000년 3월~현재 : 충남대학교 컴퓨터과학과 박사과정
〈관심분야〉 네트워크 보안, PKI

김 충 길(Chung-gil Kim)
1998년 2월 : 충남대학교 물리학과 졸업
2000년 2월 : 충남대학교 컴퓨터과학과 석사
〈관심분야〉 네트워크 보안, 무선인터넷 보안



이 재 일(Jae-Il Lee)
1986년 : 서울대학교 계산통계학과 졸업
1988년 : 서울대학교 계산통계학과 석사
1991년~1996년 6월 : 한국IBM 소프트웨어연구소
1996년 7월~현재 : 한국정보보호센터 인증관리센터팀장
〈관심분야〉 유·무선 PKI, 전자상거래보안



이 석 래(Seok-lae Lee)
1992년 2월 : 한양대학교 전자통신공학과 졸업
1994년 2월 : 한양대학교 전자통신공학과 석사
1994년 2월~1999년 6월: LG 전자
1999년 7월~현재 : 한국정보보호센터 선임연구원
〈관심분야〉 데이터 보안, 통신공학

류 재 철(Jae-Cheol Ryou)
1985.2 한양대학교 산업공학과 졸업
1988.5 Iowa State Univ. 전산학 석사
1990.12 Northwestern Univ. 전산학 박사
1991.2~현재 : 충남대학교 정보통신공학부 부교수
〈관심분야〉 인터넷 보안