PKCS#11 에 기반한 KCDSA 메커니즘 설계

김명희*, 전문석**
*안철수연구소
**숭실대학교 컴퓨터학과 교수
e-mail : cryptohee@hanmail.net

Design of KCDSA Mechanism based on PKCS#11

Myung-Hee Kim*, Moon-Seog Jun**
*AhnLab, Inc.
**Dept. of Computer, Soong-Sil University

요 약

다양한 개발 환경과 분산 응용 서비스에 적용할 수 있는 보안 API 기술의 필요성이 대두되고 있다. 암호토큰 호환성을 지원하는 RSA 사의 Cryptoki 모델을 선택하여, PKCS#11 에 기반한 국내 전자서명 표준인 KCDSA 알고리즘과 HAS-160 에 대한 메커니즘을 설계한다. 설계한 KCDSA 메커니즘을 PKCS#11 에 추가하여 기능을 향상시키므로써, 국내전자상거래에 호환성과 효율성을 제공하는 효과를 기대할 수 있다.

1. 서론

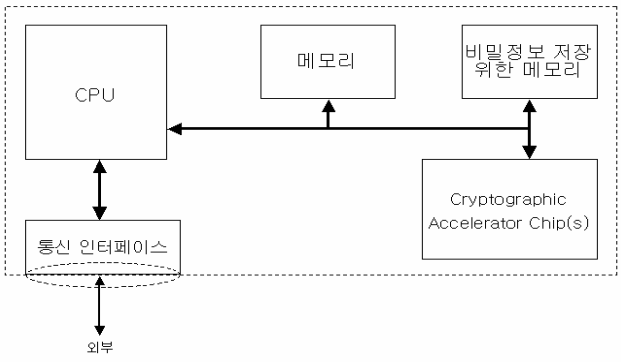
오늘날 컴퓨터 네트워크를 이용한 분산 응용 서비스가 기하급수적으로 증가하고 있다. 그와 함께 서비스의 보안 위협 요소도 증가하면서, 보안 기술은 절대적으로 필요하게 되었다. 그러나, 분산 응용 서비스와 암호 모듈을 각자가 개발하므로 호환성이 없고 개발 노력들이 중복적으로 투자되는 것이 현실이다. 이러한 문제점을 해결하기 위해서는, 다양한 개발환경과 분산 응용 서비스에 적용할 수 있는 공통적 사용 가능한 보안 API 기술이 필요하다.[10] 많은 보안 API 중에서 개인키 저장매체 USB, 스마트카드 등의 암호토큰에 대한 요구가 증가됨에 따라, 본 논문에서는 암호토큰의 호환성을 제공하는 표준인 RSA 사의 PKCS#11 인터페이스를 선택하여 보안 API 의 호환문제를 해결하고 국내에서의 활용도를 높이고자 한다.([1], [2], [9])

본 논문의 구성은 다음과 같다. 2 장에서 관련연구에 대해서 기술하고, 3 장에서 제안하는 메커니즘에 대하여 설명한다. 4 장에서 제안한 메커니즘을 분석하고, 5 장에서 결론을 맺는다.

2. 관련연구

2.1 스마트 카드의 보안 기능

인터넷이 발전함에 따라 전자상거래가 활성화 되면서, CPU, 운영체제, 보안모듈, 메모리 등을 갖추고 있는 스마트 카드의 사용이 증가되고 있다. 스마트 카드 내에 저장되는 비밀키, 인증서 등을 안전하게 다루고자 하는 요구사항이 늘어나고 있다. 그런데, 각각 다른 환경에서 개발하므로써 호환성이 없어 전자상거래 사용에 제한이 있으므로 공통적으로 사용가능한 보안 API 를 사용하여야 한다.

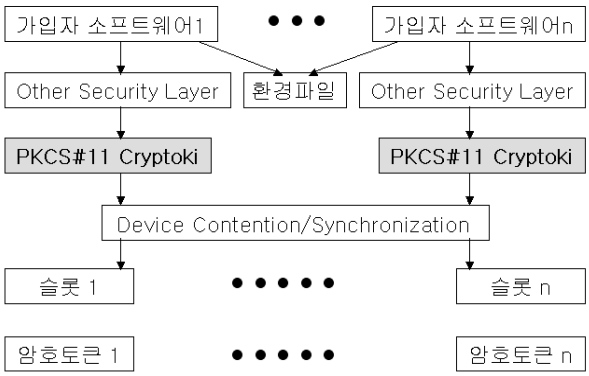


(그림 1) 안전한 Coprocessor 의 간단한 구조

(그림 1)에서 보면, 외부와 접해있는 통신 인터페이스를 공통적으로 사용하기 위해 일반적이며 안전한 보안 API 가 반드시 필요하다.

2.2 PKCS#11 기본 모델 구조

RSA 의 Cryptoki 는 공개키 암호표준인 PKCS#11 로써 어플리케이션이 암호토큰과 함께 동작하는 요구를 지원해 주기 위한 인터페이스로 개발되어 현재 사용중이다. (그림 2)는 PKCS#11 기반의 가입자 소프트웨어와 암호토큰에 대한 인터페이스 제공 모델을 보여준다([1],[9]). 암호토큰(Cryptographic Token)은 암호관련 정보를 저장하거나 암호기능을 수행할 수 있는 논리적 관점의 디바이스이며, 슬롯(Slot)은 암호토큰과의 입출력을 수행할 수 있는 논리적 리더기이다. 각각의 객체는 객체 생성, 수정, 검색 등을 위해 사용하는 객체 속성 집합인 템플릿(Template)을 정의하고, 다중 쓰레드의 동기화를 위해서는 뮤텍스(Mutexes) 기법을 사용한다.



(그림 2) RSA 의 Cryptoki 모델

2.3 KCDSA 방식

이산대수문제에 근간한 전자서명방식으로 국내 전자서명 방식의 표준으로 제정된 KCDSA(Korean Certificate-based Digital Signature Algorithm)의 시스템 변수와 사용자 변수를 설명한다. 본 논문에서 설명되지 않은 부분은 표준문서에 정의된 내용을 따른다.([3], [8])

● 시스템 변수

p : $2^{|p|-1} < p < 2^{|p|}$, $|p| = 512 + 256i(0 \le i \le 6)$의 크기를 가지며, $(p-1)/2q$ 역시 소수이거나 최소한 q 보다 큰 소수들의 곱으로 구성되는 소수이다.

q : p-1 를 나누는 소수이며, $2|q|-1 < q < 2|q|$, $|q| = 128 + 32j(0 \le j \le 4)$의 크기를 가진다.

g : $a^{(p-1)/q} \bmod p$, $1 < a < p-1$ 이고, $a^{(p-1)/q} \bmod p > 1$ 을 만족한다.

● 사용자 변수

x : $0 < x < q$ 를 만족하는 정수로서 랜덤하게 선택된 서명자의 비공개 서명키이다.

y : $y = g^{x_1} \bmod q$ 로 계산되는 서명자의 공개 검증키이다. 여기서 x^{-1} 는 $xx^{-1} = 1 \bmod q$ 와 $0 < x^{-1} < q$ 를 만족하는 수이다.

3. 제안하는 메커니즘

현재 PKCS#11 의 Cryptoki 는 암호기능을 구현하는 절차인 메커니즘(Mechanism)으로 국내 전자서명 표준인 KCDSA 알고리즘에 대한 메커니즘은 지원하지 않고 있다. 본 논문에서 KCDSA 메커니즘과 그 메커니즘 사용을 지원하는 함수로써 개발에 대한 설계부분을 제안할 것이다. KCDSA 메커니즘의 CK_MECHANISM_TYPE 은 CKM_KCDSA 로 정의한다.

3.1 KCDSA 공개키 객체

객체 클래스 CKO_PUBLIC_KEY 와 키 타입 CKK_KCDSA 를 가지는 KCDSA 공개키 객체는 공개키를 저장한다. (표 1) 은 KCDSA 공개키 객체 속성을 나타낸다.

(표 1) KCDSA 공개키 객체 속성

속성	데이터 타입	내용
CKA_PRIME	Big integer	Prime p
CKA_SUBPRIME	Big integer	Subprime q
CKA_BASE	Big integer	Base g
CKA_VALUE	Big integer	Public value y

KCDSA 공개키 객체 템플릿은 다음과 같다.

```
CK_OBJECT_CLASS class = CKO_PUBLIC_KEY;
CK_KEY_TYPE keyType = CKK_KCDSA;
CK_UTF8CHAR label[] = "A KCDSA public key object";
CK_BYTE prime[] = {…};
CK_BYTE subprime[] = {…};
CK_BYTE base[] = {…};
CK_BYTE value[] = {…};
CK_BBOOL true = TRUE;
CK_ATTRIBUTE template[] = {
    {CKA_CLASS, &class, sizeof(class)},
    {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
    {CKA_TOKEN, &true, sizeof(true)},
    {CKA_LABEL, label, sizeof(label)-1},
    {CKA_PRIME, prime, sizeof(prime)},
    {CKA_SUBPRIME, subprime, sizeof(subprime)},
    {CKA_BASE, base, sizeof(base)},
    {CKA_VALUE, value, sizeof(value)}
};
```

3.2 KCDSA 개인키 객체

객체 클래스 CKO_PRIVATE_KEY 와 키 타입 CKK_KCDSA 를 가지는 KCDSA 개인키 객체는 개인키를 저장한다. (표 2)는 KCDSA 개인키 객체 속성을 나타낸다.

(표 2) KCDSA 개인키 객체 속성

속성	데이터 타입	내용
CKA_PRIME	Big integer	Prime p
CKA_SUBPRIME	Big integer	Subprime q
CKA_BASE	Big integer	Base g
CKA_VALUE	Big integer	Public value x

KCDSA 개인키 객체 템플릿은 다음과 같다.

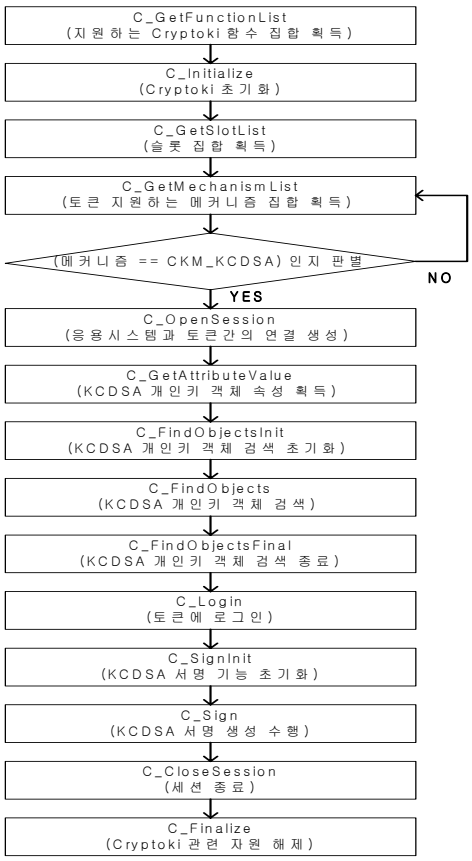
```
CK_OBJECT_CLASS class = CKO_PRIVATE_KEY;
CK_KEY_TYPE keyType = CKK_KCDSA;
CK_UTF8CHAR label[]= “A KCDSA private key object”;
CK_BYTE subject[] = {…};
CK_BYTE id[] = {123};
CK_BYTE prime[] = {…};
CK_BYTE subprime[] = {…};
CK_BYTE base[] = {…};
CK_BYTE value[] = {…};
CK_BBOOL true = TRUE;
CK_ATTRIBUTE template[] = {
   {CKA_CLASS, &class, sizeof(class)},
   {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
   {CKA_TOKEN, &true, sizeof(true)},
   {CKA_LABEL, label, sizeof(label)-1},
   {CKA_SUBJECT, subject, sizeof(subject)},
   {CKA_ID, id, sizeof(id)},
   {CKA_SENSITIVE, &true, sizeof(true)},
   {CKA_SIGN, &true, sizeof(true)},
   {CKA_PRIME, prime, sizeof(prime)},
   {CKA_SUBPRIME, subprime, sizeof(subprime)},
   {CKA_BASE, base, sizeof(base)},
   {CKA_VALUE, value, sizeof(value)}
};
```

3.3 PKCS#11 기반의 KCDSA 메커니즘 설계

PKCS#11 에 기반한 Cryptoki 라이브러리는 KCDSA 알고리즘을 이용하여 전자서명 생성 및 암호키 복호화 등의 기능을 지원하기 위해서는 CKM_KCDSA_KEY_PAIR_GEN 메커니즘 과 CKM_KCDSA_HAS160 메커니즘을 지원해야 한다. HAS160 해쉬알고리즘에 기반한 KCDSA 전자서명 메커니즘은 CKM_KCDSA_HAS160 로 정의한다.[7]

KCDSA 알고리즘에 대한 키쌍 생성 메커니즘은 KM_KCDSA_KEY_PAIR_GEN 으로 정의하며, CKA_PRIME, CKA_SUBPRIME, CKA_BASE 속성을 가지는 공개키과 개인키를 생성한다. KCDSA 개인키는 C_Sign 함수로 서명을 생성하고, KCDSA 공개키는 C_Verify 함수로써 서명을 검증한다.

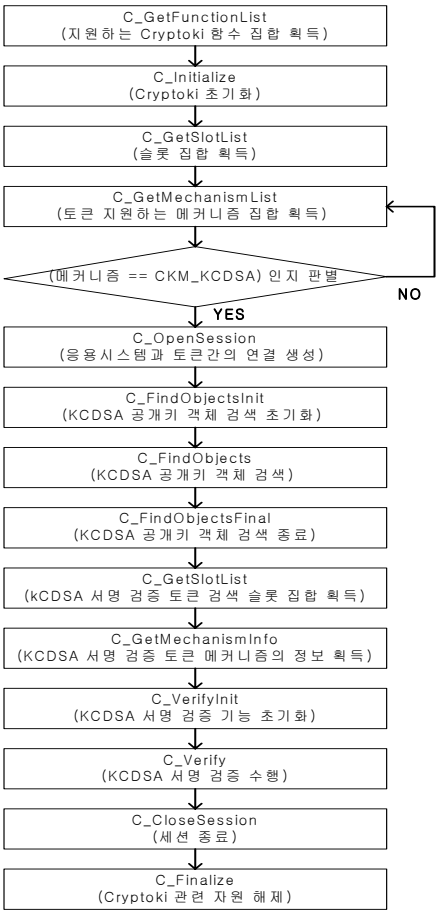
(그림 3)는 CKM_KCDSA_HAS160 서명 생성 과정을 순서도로 설계한 것이다. Cryptoki 가 지원하는 함수 포인트 집합을 획득(C_GetFunctionList)한 후에, Cryptoki 를 초기화(C_Initialize)한다. 슬롯 집합(C_GetSlotList)과 토큰이 지원하는 메커니즘 집합(C_GetMechanismList)을 획득한다. 만약 메커니즘이 CKM_KCDSA 인지 판별한 후에, CKM_KCDSA 가 아니면 토큰이 지원하는 메커니즘 집합을 다시 얻어온다. 메커니즘이 CKM_KCDSA 이면 응용 시스템과 토큰간의 연결을 생성(C_OpenSession)한다. 연결이 생성되면, KCDSA 전자서명 알고리즘의 메커니즘에 대한 개인키 객체 속성을 획득(C_GetAttributeValue)하여, 개인키 객체 검색을 초기화(C_FindObjectsInit)한다. KCDSA 개인키 객체를 검색(C_FindObjects)하여 작업한 후에, KCDSA 개인키 검색 작업을 종료(C_FindObjectsFinal)한다. 그 다음 암호토큰에 로그인한다. KCDSA 메커니즘으로 서명을 초기화(C_SignInit)한 후, KCDSA 메커니즘으로 서명 생성을 수행(C_Sign)한다. 서명한 후에 세션을 종료(C_CloseSession)하고, Cryptoki 와 관련된 자원을 해제(C_Finalize)한다.



(그림 3) CKM_KCDSA_HAS160 서명 생성

(그림 4)는 CKM_KCDSA_HAS160 서명 검증 과정을 순서도로 설계한 것이다. Cryptoki 가 지원하는 함수 포인트 집합을 획득(C_GetFunctionList)한 후에, Cryptoki 를 초기화(C_Initialize)한다. 슬롯 집합(C_GetSlotList)과 토큰이 지원하는 메커니즘 집합(C_GetMechanismList)을 획득한다. 만약 메커니즘이 CKM_KCDSA 인지 판별한 후에, CKM_KCDSA 가 아니면 토큰이 지원하는 메커니즘 집합을 다시 얻어온다. 메커니즘이 CKM_KCDSA 이면 응용 시스템과 토큰간의 연결을 생성(C_OpenSession)한다. 연결이 생성되면, KCDSA 전자 서명 알고리즘의 메커니즘에 대한 공개키 객체 검색을 초기화(C_FindObjectsInit)한다. KCDSA 공개키 객체를 검색(C_FindObjects)하여 작업한 후에, KCDSA 공개키 검색 작업을 종료(C_FindObjectsFinal)한다. KCDSA 메커니즘의 서명 검증 토큰을 검색하여 슬롯 집합을 획득(C_GetSlotList)한 후에, KCDSA 서명 검증을 위한 토큰 메커니즘의 정보를 획득(C_GetMechanismInfo)한다. KCDSA 메커니즘의 서명 검증 기능을 초기화(C_VerifyInit)한 후에, KCDSA 메커니즘의 서명을 검증(C_Verify)한다. 서명

을 검증한 후에 세션을 종료(C_CloseSession)하고, Cryptoki 와 관련된 자원을 해제(C_Finalize)한다.



(그림 4) CKM_KCDSA_HAS160 서명 검증

4. 제안한 메커니즘의 분석

본 논문에서 설계하여 C 언어로 구현한 KCDSA 메커니즘은 PKCS#11 에 정의된 전자서명 알고리즘 RSA, DSA 메커니즘과 호환성을 이룬다. KCDSA 메커니즘으로 전자서명을 생성하고 검증하는 과정도 안전하게 실행된다.

5. 결론

본 논문에서는 암호토큰 호환성을 지원하는 RSA 사의 Cryptoki 모델을 선택하여, PKCS#11 에 기반한 국내 전자서명 표준인 KCDSA 알고리즘과 HAS-160 에 대한 메커니즘을 설계하였다. 설계한 KCDSA 메커니즘을 PKCS#11 에 추가하여 기능을 향상시키므로써, 국내전자상거래에 호환성과 효율성을 제공하는 효과를 크게 기대하고자 한다.

향후 KCDSA 메커니즘을 지원하는 PKCS#11 API 에 대한제 3 자의 공격에 대한 보안 분석과 대책 방법 등에 대해서도 계속 연구가 이루어져야 한다.([4],[5],[6])

참고문헌

[1] RSA Laboratories, “PKCS#11 v2.10: Cryptographic Token Interface Standard”, 1999.
[2] RSA Laboratories, “PKCS#15 v1.1: Cryptographic Token Information Syntax Standard”, 2000.
[3] KCDSA Task Force Team, “The Korean Certificate-based Digital Signature Algorithm”, 1998.
[4] Jolyon Clulow, “On the Security of PKCS #11”, CHES 2003, LNCS 2779, pp. 411-425, 2003.
[5] Jolyon Cluw, “The design and security of public key crypto APIs”, 2001.
[6] Jolyon Clulow, “The design and analysis of cryptographic application programming interfaces for devices”, Master’s thesis, University of Natal, Durban, 2003.
[7] TTAS.KO-12.0011, “해쉬함수표준 - 제 2 부 : 해쉬함수알고리즘표준(HAS-160)”, 1998.
[8] 서문석, 김광조, “KCDSA 및 EC-KCDSA 에 근간한 은닉서명”, KIISC 종합학술발표회논문집(CISC’99), Vol.9, No.1, 1999, pp.141-150
[9] 인터넷보안기술포럼, “암호토큰을 위한 PKCS#11 프로파일 표준”, 2003.
[10] 주학수, 이언경, 김승주, “암호라이브러리 및 암호 API 개발현황”, 정보보호학회지 제 12 권 4 호, 2002, pp.94-103.