情報保護學會論文誌
第14券 第5號, 2004. 10

KCDSA 메커니즘을 제공하는 PKCS #11 설계 및 분석*

김 명 희†‡, 김 은 환, 전 문 석
숭실대학교

A Design and Analysis of PKCS #11 supporting the KCDSA mechanism

Myung-Hee Kim†‡, Eun-Hwan Kim, Moon-Seog Jun
Soongsil University

요 약

전자상거래의 활성화에 따라 이동매체의 안전에 대한 요구사항이 늘어나고 있다. 이동매체간의 호환성을 제공하기 위해서는 보안 API(Application Programming Interface)가 쉽고 안전하게 설계되어져야한다. 현재 많은 제품과 개발에 호환성과 확장성 표준을 제시하고 있는 RSA사의 PKCS(Public Key Cryptographic Standard) #11 인터페이스를 선택하여, 국내 전자서명 표준인 KCDSA(Korean Certificate-based Digital Signature Algorithm) 메커니즘을 제공하고자 한다. 그리고, PKCS #11 보안 API에 새로운 키 관리 함수를 정의하므로써 보다 더 안전한 키 관리 기능도 지원한다. KCDSA 개인키와 공개키의 객체 속성과 템플릿을 정의하고, KCDSA 메커니즘으로 전자서명을 생성하고 검증할 수 있도록 한다. KCDSA 메커니즘을 제공하는 PKCS #11을 설계 및 구현하여 성능평가하고, 보안성 및 호환성에 대하여 비교분석한다.

ABSTRACT

According to the improvemnent of electronic commerce, the requirements of security devices are becoming increasingly pervasive. The security API must design easily and securely to support a compatibility feature between security devices. It is chosen the PKCS #11 interface by RSA Labs that shows the compatibility and extensibility standards of many application product and implementation, and supported KCDSA mechanism which is a korean digital signature standard. And the PKCS #11 security API defines new key management function which provides more secure key management ability. We suggest the object attributes and templates of KCDSA private and public key object, generate and verify digital signature using KCDSA mechanism. The PKCS #11 supporting KCDSA mechanism is designed, implemented using C-Language, tested a performance, and analyzed the security and compatibiltiy feature.

Keywords : *PKI, Security API, PKCS #11, KCDSA Mechanism, Digital Signature Algorithm, Private and Public Key*

Ⅰ. 서 론

오늘날 인터넷이 발전함에 따라 전자상거래가 활성화되면서, 비밀키, 인증서 등을 스마트카드와 같은 이동매체에서 안전하게 다루고자 하는 요구사항이 늘어나고 있다. 사용자의 이러한 요구사항을 만족시키기 위하여 많은 전자상거래 응용프로그램이 개발되고 있다. 그러나, 각각 다른 환경에서 각자가 프로그램을 개발하므로 각 제품간의 호환성이 없어 전자상거

접수일 : 2004년 6월 15일 ; 채택일 : 2004년 10월 7일
* 본 연구는 숭실대학교 교내연구비 지원으로 이루어졌음.
† 주저자, ‡ 교신저자 : cryptohee@hanmail.net

래 사용에 제한이 많다. 기존의 보안 API가 변경되는 경우 사용자가 변경된 보안 API와 데이터의 호환성을 확인하기 어렵고, 변경된 보안 API의 적용을 위하여 기존 응용 시스템의 수정이 어렵다. 기능의 수정이나 추가가 발생할 경우 유연하게 적용될 수 있도록 보안 API를 쉽고 안전하게 설계하고, 보안 API 기반의 응용프로그램을 개발할 수 있어야 한다[1-5]. 응용프로그램 개발자가 보안 API를 사용하여 개발하는 경우에, 잘못된 사용으로 인하여 보안상의 문제가 야기될 수도 있다. 이런 문제가 발생하지 않도록 안전한 보안 API를 개발하도록 주의를 기울여야한다.

CCA, Thales-Zaxus-Racal, Compaq-Atalla, PKCS #11 등 여러 보안 API 중에서, 현재 많은 제품과 개발에 호환성과 확장성 표준을 제시하고 있는 RSA사의 PKCS #11 인터페이스를 선택한다. 현재 PKCS #11은 국내 전자서명 표준인 KCDSA 알고리즘에 대한 메커니즘을 제공하지 않고 있다. KCDSA 메커니즘을 개발하여 PKCS #11 보안 API에 추가하므로써, 국내 전자상거래의 라이브러리에 대한 보안 API와의 호환문제를 해결하여 국내에서의 활용도를 높이고자 한다.

그리고, PKCS #11에 KCDSA 메커니즘을 기능적으로 추가하는 것만 아니라, KCDSA 메커니즘을 위해 생성한 KCDSA 공개키와 개인키를 안전하게 보관하여 안전성을 지원하고자 한다. 암호화할 키를 키 관리 함수의 입력 파라미터에 바로 입력하면 변조된 키를 전달할 수 있으므로, 새로운 키 관리 함수를 정의하여 암호화할키 대신 사용자의 PIN(Personal Identification Number : 패스워드)를 입력하여 개인키 및 공개키를 더 안전하게 보관한다.

본 논문의 구성은 다음과 같다. 2장에서 관련 연구로 공개키 암호 방식, PKCS #11, 그리고 KCDSA 전자서명 방식에 대해서 기술한다. 3장에서 KCDSA 메커니즘을 제공하는 PKCS #11을 설계하고, 4장에서 KCDSA 메커니즘을 제공하는 PKCS #11을 분석한다. 그리고, 마지막으로 5장에서 결론을 맺는다.

II. 관련 연구

2.1 공개키 암호 방식

SEED, DES, AES 등과 같은 대칭키 암호 방식은 빠른 속도와 오랫동안 검증된 안전성 등의 장점이 있으나 키 관리 문제에서 약점을 가지고 있다. 전자상거래시 두 사람이 안전하게 암호화된 메시지를 주고 받기 위해서는 서로 같은 키를 가지고 있어야 하는데 키를 안전하게 전달하고 보관하는 것이 매우 어렵다. 특히 다자간에 메시지 교환을 하기 위해서는 각각의 전송시에 서로 다른 키를 요구하기 때문에 키의 분배 문제 또한 매우 중요하다. 그래서 이러한 키의 분배 및 관리 문제를 해결하기 위해 공개키 암호 방식이 개발되었다. 암호화와 복호화에서 서로 다른 키를 사용하는 RSA, DSA, KCDSA 등과 같은 공개키 암호 방식을 적용하여 사용자인증, 정보보호, 무결성보장 및 부인봉쇄 등의 서비스를 제공함으로써 전자상거래시 네트워크를 통해 유통되는 개인정보에 대한 안전성 및 신뢰성을 확보할 수 있다.

전자상거래 시스템은 키가 안전하게 보존된다는 가정 하에서 시스템의 안전성이 보장된다. 그러나 키의 안전한 관리는 매우 어려운 일이므로 시스템의 안전성 문제가 여전히 남아있게 된다. 이를 해결하기 위해 공개키 암호 방식을 가장 유용하게 적용할 수 있는 기반은 스마트 카드이다. 스마트 카드는 사용자와 공급자 모두에게 보안성, 편리성, 다기능성, 비용효과성과 같은 사용 이점이 많기 때문이다. 현재 스마트 카드가 보안성이 뛰어난 것으로 인정되나 해킹될 경우 카드에 저장된 개인 정보와 비밀키까지도 추출이 가능해 IC칩에 저장된 정보 등에 대한 위·변조는 물론 개인 정보의 악용까지도 초래할 수 있다[6-8].

이러한 문제점을 미리 방지하거나 해결하기 위하여, 보안성과 이동성이 뛰어난 스마트 카드와 같은 보안 장치의 요구사항을 만족하는 보안 API 중 현재 많은 전자상거래 제품과 개발에 호환성과 확장성 표준을 제시하고 있는 RSA사의 PKCS #11 인터페이스를 본 논문에서는 선택하여 연구하고자 한다.

2.2 PKCS #11

PKCS는 전자상거래 인증기관과 제품간의 상호 운용이 가능하도록 공개키기반구조(Public Key Infrastructure, PKI)의 데이터 형식, 통신 프로토콜, API등에 대한 기본을 규정하고 있다. PKCS에서는 #1에서 #15까지 규정되어 있으며, RSA Security 사의 사이트에 게재되어 있다. 그중에서 자주 이용하는 것 중의 하나인 PKCS #11은 암호토큰(Cryptographic Token)과 함께 동작하는 요구를 지원해

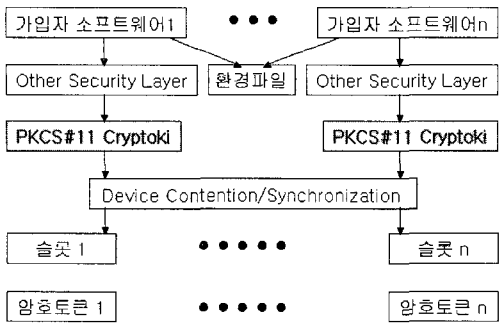


그림 1. PKCS #11 기본 모델 구조

주기 위한 응응프로그램으로부터 불러오는 프로그래밍 인터페이스이며, Cryptoki라고 불려지고 있다. PKCS #11 인터페이스는 Netscape사의 브라우저에서 기본적으로 키와 인증서의 저장 포맷에 사용되며, 그뿐만 아니라 메일 프로그램, 다른 회사의 인증기관 소프트웨어에서의 키 관리에도 사용되고 있다. PKCS #15은 이러한 PKCS #11에서 정의한 인터페이스를 통해 암호토큰의 정보가 저장되는 형태를 결정한다.

2.2.1 PKCS #11 기본 구조

PKCS #11의 기본 모델 구조는 그림 1과 같으며, PKCS #11 기반의 가입자 소프트웨어와 암호토큰에 대한 인터페이스 제공 모델을 보여준다[9,10]. 암호토큰은 암호관련 정보를 저장하거나 암호기능을 수행할 수 있는 논리적 관점의 디바이스이며, 사용자 인증서와 개인키가 여기에 저장된다. 슬롯(Slot)은 암호토큰과의 입출력을 수행할 수 있는 논리적 리더기이며, 다중 쓰레드(Multi Thread)의 동기화는 뮤텍스(Mutexes) 기법으로 지원한다. 메커니즘(Mechanism)은 암호기능을 구현하는 절차이며, PKCS #11은 AES, RSA, DSA 메커니즘 등을 지원하고 있다. 그리고, 저장 객체(Storage Object)로써 데이터 객체, 인증서 객체, 키 객체를 생성하거나 처리할 수 있어야 하며, 각각의 객체는 객체 생성, 수정, 검색 등을 위해 사용하는 객체 속성 집합인 템플릿(Template)을 정의한다. 본 논문에서는 PKCS #11 보안 API에 KCDSA 개인키 및 공개키 객체의 속성과 템플릿을 정의하여 KCDSA 메커니즘을 제공할 수 있도록 하며, 현재 국내 표준인 [11]에서 KCDSA 메커니즘을 제공할 수 있도록 지원하고자 한다.

2.2.2 키 관리 함수

PKCS #11에서 정의하고 있는 키 관리 함수에는 C_GenerateKey, C_GenerateKeyPair, C_WrapKey, C_UnwrapKey, C_DeriveKey 등이 있다.

C_GenerateKey 함수는 비밀키(Secret Key)를 생성하고, C_GenerateKeyPair 함수는 개인키(Private Key)와 공개키(Public Key)를 생성한다. 그리고, C_WrapKey 함수는 비밀키 또는 개인키를 암호화하고, C_UnwrapKey 함수는 암호화된 키를 복호화한다. C_DeriveKey 함수는 기본키(Base Key)로부터 새로운 키를 생성한다. 이 중에서 노출되면 가장 위험한 비밀키와 인증서 등의 개인 데이터를 암호화하여 안전하게 보관하기 위한 C_WrapKey 함수를 살펴보고자 한다.

```
CK_DEFINE_FUNCTION(CK_RV, C_WrapKey)
(
  CK_SESSION_HANDLE      hSession,
  CK_MECHANISM_PTR       pMechanism,
  CK_OBJECT_HANDLE       hWrappingKey,
  CK_OBJECT_HANDLE       hKey,
  CK_BYTE_PTR            pWrappedKey,
  CK_ULONG_PTR           pulWrappedKeyLen
)
```

여기서, hSession은 세션 핸들이고, pMechanism은 암호 메커니즘에 대한 포인터이다. hWrappingKey는 암호화키의 핸들이고, hKey는 키를 암호화하기 위한 키의 핸들이다. hKey 입력 파라미터로는 DES, RC5, AES 메커니즘 등으로 생성된 비밀키 객체 또는 RSA, DSA 메커니즘 등으로 생성된 공개키 객체 중에서 시스템에 가장 안전한 암호를 지원할 수 있는 키를 선택하여 입력한다. 본 논문에서는 안전할 뿐만 아니라 국내 전자상거래 제품과의 호환성을 지원하는 KCDSA 메커니즘으로 생성된 키 객체를 선택하여 입력할 수 있도록 한다. 그리고, pWrappedKey는 암호화된 키를 수신하여 보관하는 장소에 대한 포인터이고, pulWrappedKeyLen은 암호화된 키 길이를 수신하여 보관하는 장소에 대한 포인터이다.

C_WrapKey 함수는 다음과 같은 경우 중에서 사용할 수 있다.

- 비밀키를 hWrappingKey 파라미터에 입력하고, 이것을 암호화하거나 복호화하는 KCDSA 공개키를 hKey 파라미터에 입력한다.

- 비밀키를 hWrappingKey 파라미터에 입력하고, 이것을 암호화하거나 복호화하는 다른 비밀키를 hKey 파라미터에 입력한다.
- KCDSA 개인키를 hWrappingKey 파라미터에 입력하고, 이것을 암호화하거나 복호화하는 비밀키를 hKey 파라미터에 입력한다.

2.3 KCDSA 전자서명 방식

전자서명 알고리즘 KCDSA는 국내표준으로 채택되어 국내 인증기관간의 상호연동을 지원하는 전자서명 인증서버 등에서 활용 중이다. 유선환경 뿐만 아니라 무선환경에서도 전자서명이 원활히 이루어질 수 있도록 무선 PKI 기반의 인증서 규격, 인증서 관리 프로토콜 등에 관한 작업들도 활발하다.

KCDSA 알고리즘은 DSA와 GOST 알고리즘과 달리 곱에 대한 역원(Multiplicative Inverse) 계산을 하지 않기 때문에 속도가 빠르므로 스마트 카드와 같은 제한적인 환경에서 더 효율적이다. 랜덤 값을 생성하는 오라클을 대신하는 해쉬함수가 안전함이 증명되어 KCDSA 알고리즘은 보안측면에서도 매우 안전하다.

이러한 KCDSA 전자서명 방식을 기반으로 한 전자 현금 시스템, 추적 가능한 전자화폐 시스템, 거스름 재사용 가능한 전자 수표지불 시스템 등에 KCDSA 알고리즘이 많이 활용되고 있다[12-14].

본 논문에서는 이런 장점을 가지고 있는 국내 표준인 인증서 기반 부가형 전자서명의 생성과 검증을 위한 알고리즘 KCDSA 전자서명 방식을 이용하여, 국내외 전자상거래 시스템과 보안 API에 호환성을 제공하고자 한다[15,16].

2.3.1 KCDSA 파라미터

시스템 파라미터 p, q, g와 사용자 파라미터 x, y는 다음과 같다.

- p : $2^{|p|-1} < p < 2^{|p|}$, $|p| = 512 + 256i$ $(0 \le i \le 6)$의 크기를 가지며, $(p-1)/2q$ 역시 소수이거나 최소한 q보다 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.
- x : $0 < x < q$를 만족하는 정수로서 랜덤하게 선택된 서명자의 비공개 서명키이다.
- y : $y = g^{x^{-1}} \bmod q$로 계산되는 서명자의 공개 검증키이다. 즉, x^{-1}는 $xx^{-1} = 1 \bmod q$와 $0 < x^{-1} < q$를 만족하는 수이다.

2.3.2 KCDSA 서명 생성 과정

서명자는 다음과 같은 과정을 거쳐서 서명된 메시지 $\{h(m) \| \Sigma\}$를 출력하며, 서명 Σ는 $\{r \| s\}$이다.

① 난수값 k를 $\{1, \ldots\ldots, q-1\}$에서 랜덤하게 선택한다.
② 서명의 첫 부분 $r = g^k \bmod p$를 계산한다.
③ 서명의 두 번째 부분 $s = x(k-h(m)) \bmod q$를 계산한다.
④ 서명 $\Sigma = \{r \| s\}$를 만들어 서명된 메시지 $\{h(m) \| \Sigma\}$를 출력한다.

2.3.3 KCDSA 서명 검증 과정

검증자는 다음과 같은 과정을 거쳐서 수신된 메시지 $h(m)$의 서명이 Σ인지를 확인할 수 있다.

① 서명된 메시지 $\{h(m) \| \Sigma\}$로부터 검증할 메시지 $h(m)$, 서명의 첫 부분 r, 서명의 두 번째 부분 s를 추출한다. 이 때 $0 < r < 2^{|q|}$이고 $0 < s < q$임을 확인한다.
② 서명자의 공개 검증키 y를 이용하여 $r = y^s g^{h(m)} \bmod p$가 성립하는지 확인한다.
③ ①과 ②의 확인 과정이 모두 통과되면 서명 Σ는 받은 메시지 $h(m)$에 대하여 공개 검증키 y에 대응하는 비공개 서명키 x로 서명하였음이 확인된 것이다.

Ⅲ. 제안하는 KCDSA 메커니즘을 제공하는 PKCS #11 설계

3.1 새로운 키 객체 정의

3.1.1 새로운 KCDSA 개인키 객체 정의

KCDSA 전자서명을 생성하는 경우에는 2.3.1에서 살펴본 파라미터 중에서 시스템 파라미터 p, q, g와 사용자 파라미터 x가 필요하다. 그래서, p, q, g, x를 각각 객체속성으로 CKA_PRIME, CKA_SUBPRIME, CKA_BASE, CKA_VALUE로 정의한다. 개인키를 저장하는 KCDSA 개인키 객체의

표 1. 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	Private value x

표 2. 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

객체 클래스는 CKO_PRIVATE_KEY로 정의하고, 키 타입은 CKK_KCDSA로 정의한다. KCDSA 개인키 객체 속성의 내용을 요약하면 표 1과 같다.[17]

정의한 개인키 객체 속성, 객체 클래스, 키 타입으로 구성되는 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)}
};
```

3.1.2 새로운 KCDSA 공개키 객체 정의

KCDSA 전자서명을 검증하는 경우에는 2.3.1에서 살펴본 파라미터 중에서 시스템 파라미터 p, q, g와 사용자 파라미터 y가 필요하다. 그래서, p, q, g, y를 각각 객체속성으로 CKA_PRIME, CKA_SUBPRIME, CKA BASE, CKA VALUE로 정의한다. 공개키를 저장하는 KCDSA 공개키 객체의 객체 클래스는 CKO_PUBLIC_KEY로 정의하고, 키 타입은 CKK_KCDSA로 정의한다. KCDSA 공개키 객체 속성의 내용을 요약하면 표 2와 같다[17].

정의한 공개키 객체 속성, 객체 클래스, 키 타입으로 구성되는 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 새로운 키 관리 함수 정의

KCDSA 개인키를 생성하여 네트워크를 통해 암호화하지 않은 개인키 파일을 전송하거나, 플로피 디스크로 전달하면 타인이 개인키 파일 일부분 또는 전체를 수정할 수 있다[18]. KCDSA 개인키를 안전하게 보관하기 위하여 암호화할 경우에는 C_WrapKey 함수의 hKey 파라미터에 AES(Advanced Encryption Standard) 메커니즘의 대칭키를 입력하여 KCDSA 개인키를 암호화한다. AES 메커니즘은 키와 암호화할 데이터 블럭 사이즈가 같은 경우

에 암복호화하고자 하는 것이 가능하므로, 키를 더 안전하게 암호화하여 보관할 수 있다. 그뿐만 아니라, AES 메커니즘의 여러 모드 중에서 마지막 블럭의 암호화된 데이터에 의해 키파일 변조 공격이 더 어려운 CBC(Cipher Block Chaining) 모드를 선택하여 키를 암호화하는 것이 좋다.

그리고, PKCS #11 보안 API는 공개키의 공개 파라미터를 제대로 보관하지 않거나 공개키에 대한 인증 체크를 하지 않는다. 공개 파라미터 p, q, g, y를 변경하여 공개키 생성에 악의의 영향을 주거나, C_WrapKey 함수에 공개키가 그대로 입력되므로 개인키에 대한 정보를 가지고 있는 악의의 사용자는 언제든지 공개키 파일을 임의로 변조할 수 있다. 그러므로 공개키도 서명 검증을 하기전에 공개키에 대한 인증이 반드시 필요하다.

C_WrapKey 함수의 hKey 파리미터에 키가 그대로 입력되면 개인키 및 공개키에 악의의 영향을 줄 수 있으므로, 본 논문에서는 C_SeucreWrapKey 새로운 키 함수를 정의하여 키를 더 안전하게 보관하고자 한다. hKey 파라미터에 키를 입력하는 C_WrapKey 함수와 달리 사용자가 PIN을 바로 입력하여 생성된 키로 개인키 및 공개키를 암호화하는 것이다. 그러면 암호화하는 키를 다른 사람은 알지 못하므로 키를 안전하게 보관할 수 있다.

```
CK_DEFINE_FUNCTION(CK_RV, C_SecureWrapKey)
(
  CK_SESSION_HANDLE hSession,
  CK_MECHANISM_PTR pKeyEncryptionMechanism,
  CK_MECHANISM_PTR pKeyDerivationMechanism,
  CK_OBJECT_HANDLE  hWrappingKey,
  CK_UTF8CHAR_PTR  pPin,
  CK_ULONG    ulPinLen,
  CK_BYTE_PTR   pWrappedKey,
  CK_ULONG_PTR  pulWrappedKeyLen
)
```

여기서, hSession은 세션 핸들이고, pKeyEncryptionMechanism은 키 암호 메커니즘에 대한 포인터이다. pKeyDerivationMechanism은 사용자가 입력한 PIN으로부터 키를 생성하는 메커니즘에 대한 포인터이고, hWrappingKey는 암호화할 키의 핸들이다. pPin은 사용자 PIN에 대한 포인터이고, ulPinLen은 사용자 PIN의 길이이다. pWrappedKey는 암호화된 키를 수신하여 보관하는 장소에 대한 포인터이고, pulWrappedKeyLen은 암호화된 키 길이를 수신하여 보관하는 장소에 대한 포인터이다.

3.3 제안하는 KCDSA 메커니즘을 제공하는 PKCS #11 서명 생성 과정

본 논문에서는 메커니즘의 포인터 CK_MECHANISM_PTR hMechanism에 국내 전자서명 표준 KCDSA 메커니즘 기능을 추가적으로 제공하고자 한다. SHA-1 해쉬알고리즘으로 압축한 데이터를 KCDSA 전자서명 방식으로 서명을 생성하고 검증하는 것이 일반데이터를 서명하는 것보다 더 빠르고 안전하다. 그래서, SHA-1 메커니즘과 KCDSA 메커니즘을 제공하는 CK_MECHANISM_TYPE을 CKM_KCDSA_SHA1로 정의하여 사용한다. CKM_KCDSA_SHA1 메커니즘의 서명 생성에 대한 전체 알고리즘은 그림 2와 같다.

① 본 논문에서 제안하는 KCDSA 메커니즘을 제공하는 PKCS #11 보안 API가 지원하는 함수 포인터들을 CK_FUNCTION_LIST 구조체에 가져온다. (C_GetFunctionList)
② 지정된 위치에 config 파일이 있는지 검색하여 config 파일이 있으면 환경변수를 먼저 가져온다. 그리고, 토큰 라이브러리를 초기화하고, 토큰에 대한 슬롯을 등록(Register)한다. 표 1에서 정의한 KCDSA 개인키 객체 속성을 템플릿에 추가하여 KCDSA 메커니즘을 PKCS #11 보안 API에서 제공할 수 있도록 한다. 그리고나서 각각의 객체들을 초기화한다. (C_Initialize)
③ 사용중인 시스템의 모든 슬롯 리스트를 획득한다. (C_GetSlotList)
④ 각각의 슬롯에 대한 메커니즘 리스트를 획득한다. (C_GetMechanismList)
⑤ 변수들을 초기화한 후에, Legacy 체크를 위해 CKF_SERIAL_SESSION 플래그가 TRUE로 설정되어 있는지 확인한다. CKF_SERIAL_SESSION 플래그가 TRUE로 설정되어야 Parallel 지원이 가능하다. 동기화를 지원하기 위해 뮤텍스를 생성하고, 슬롯을 통해 토큰을 가져온다. KCDSA 메커니즘을 제공하는지 확인한 후에 새로 생성한 세션의 핸들에 세션의

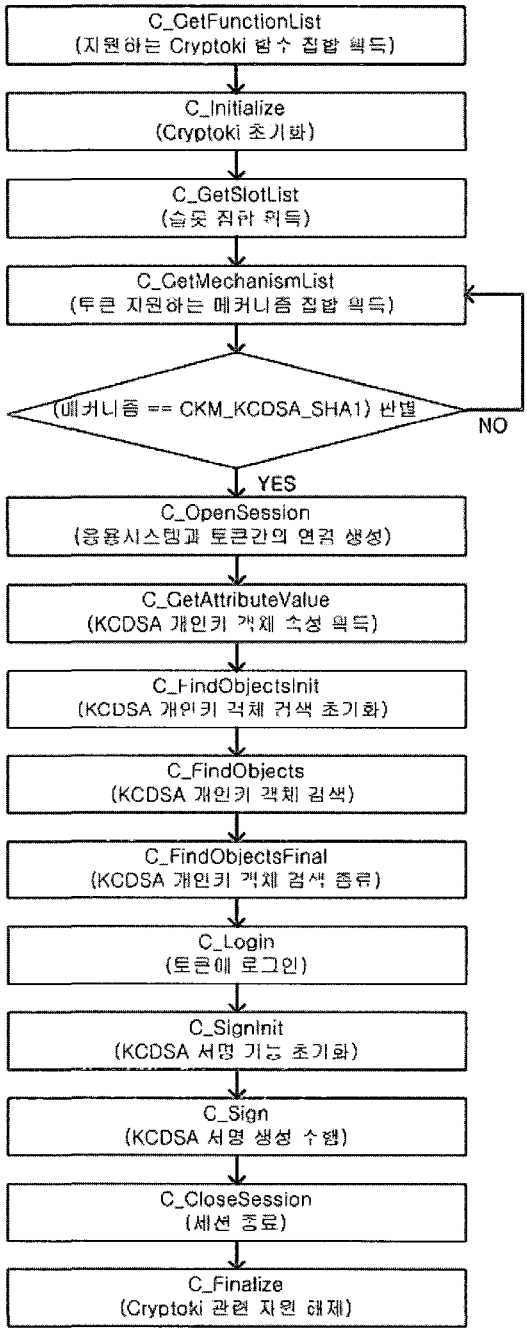


그림 2. KCDSA 메커니즘을 제공하는 PKCS #11 서명 생성 과정

정보와 데이터를 설정한다. (C_OpenSession)

⑥ 세션이 생성된 후에, 현재의 세션이 있는지 검색하여 세션의 정보를 가져온다. 그래서 세션의 해쉬테이블에서 키 객체를 찾는다. (C_GetAttributeValue)

⑦ 3.1.1에서 정의하여 안전하게 저장된 KCDSA 개인키 객체 템플릿의 토큰과 세션을 검색하기 위해 초기화한다. (C_FindObjectsInit)

⑧ KCDSA 개인키 객체 템플릿을 검색하여 복사해온다. (C_FindObjects)

⑨ KCDSA 개인키 객체 템플릿 검색을 종료한다. (C_FindObjectsFinal)

⑩ 사용자 PIN을 입력하여 암호토큰에 로그인한다. PIN을 몇 번 잘못 입력하면 암호토큰을 이용할 수 없게 된다. (C_Login)

⑪ 3.1.1에서 정의한 KCDSA 개인키 객체 템플릿, 3.2에서 정의한 C_SecureWrapKey 함수에 의해 안전하게 보관된 키를 복호화하기 위한 사용자의 PIN, 서명하고자 하는 데이터를 입력하여 KCDSA 메커니즘으로 서명하기 위해 초기화한다. (C_SignInit)

⑫ KCDSA 메커니즘으로 서명 생성을 수행한다. (C_Sign)

⑬ 뮤텍스를 생성하여 세션의 엔트리를 제거하므로써 세션을 종료한다. (C_CloseSession)

⑭ 모든 세션을 해제하면서 PKCS #11 라이브러리를 종료한다. (C_Finalize)

3.4 제안하는 KCDSA 메커니즘을 제공하는 PKCS #11 서명 검증 과정

CKM_KCDSA_SHA1 메커니즘의 서명 검증에 대한 전체 알고리즘은 그림 3과 같다.

① 본 논문에 제안하는 KCDSA 메커니즘을 제공하는 PKCS #11 보안 API가 지원하는 함수 포인터들을 CK_FUNCTION_LIST 구조체에 가져온다. (C_GetFunctionList)

② 지정된 위치에 config 파일이 있는지 검색하여 config 파일이 있으면 환경변수를 먼저 가져온다. 그리고, 토큰 라이브러리를 초기화하고, 토큰에 대한 슬롯을 등록(Register)한다. 표 2에서 정의한 KCDSA 공개키 객체 속성을 템플릿에 추가하여 KCDSA 메커니즘을 PKCS #11 보안 API에서 제공할 수 있도록 한다. 그리고나서 각각의 객체들을 초기화한다. (C_Initialize)

③ 사용중인 시스템의 모든 슬롯 리스트를 획득

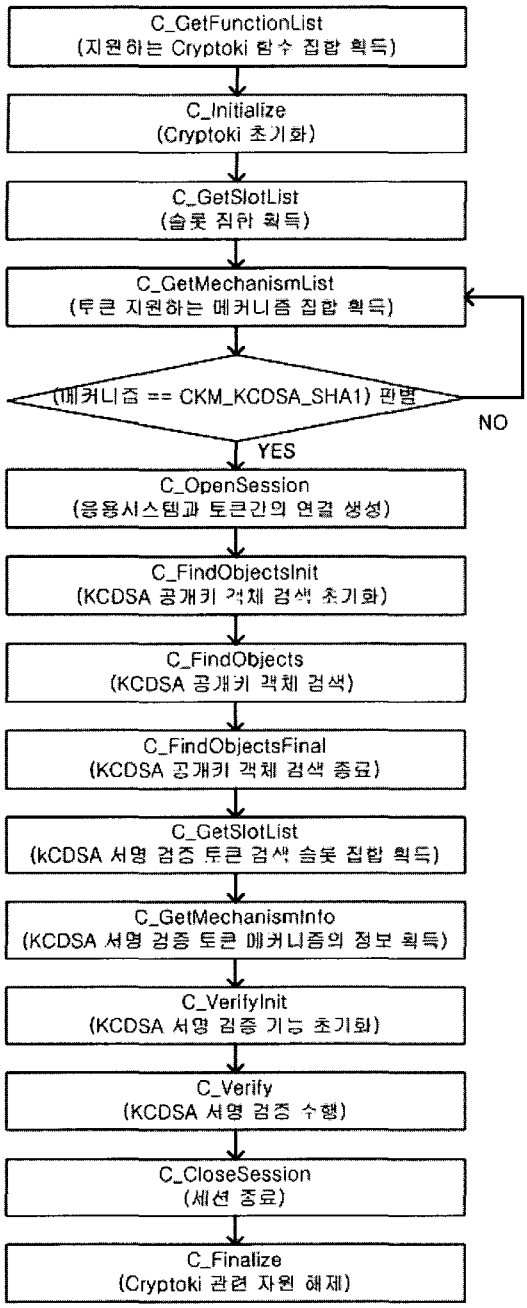


그림 3. KCDSA 메커니즘을 제공하는 PKCS #11 서명 검증 과정

한다. (C_GetSlotList)

④ 각각의 슬롯에 대한 메커니즘 리스트를 획득한다. (C_GetMechanismList)

⑤ 변수들을 초기화한 후에, Legacy 체크를 위해 CKF_SERIAL_SESSION 플래그가 TRUE로 설정되어 있는지 확인한다. CKF_SERIAL_SESSION 플래그가 TRUE로 설정되어야 Parallel 지원이 가능하다. 동기화를 지원하기 위해 뮤텍스를 생성하고, 슬롯을 통해 토큰을 가져온다. KCDSA 메커니즘을 제공하는지 확인한 후에 새로 생성한 세션의 핸들에 세션의 정보와 데이터를 설정한다. (C_OpenSession)

⑥ 3.1.2에서 정의하여 안전하게 저장된 KCDSA 공개키 객체 템플릿의 토큰과 세션을 검색하기 위해 초기화한다. (C_FindObjectsInit)

⑦ KCDSA 공개키 객체 템플릿을 검색하여 복사해온다. (C_FindObjects)

⑧ KCDSA 공개키 객체 템플릿 검색을 종료한다. (C_FindObjectsFinal)

⑨ 서명 검증할 토큰을 찾기위해 사용중인 시스템의 모든 슬롯 리스트를 획득한다. (C_GetSlotList)

⑩ 해당하는 슬롯의 메커니즘 정보를 검색한다. (C_GetMechanismInfo)

IV. 제안한 KCDSA 메커니즘을 제공하는 PKCS #11 분석

PKCS #11 보안 API에서 기본적으로 제공하는 전자서명 RSA, DSA 메커니즘 뿐만 아니라, 본 논문에서 제안한 KCDSA 메커니즘과 새로 정의한 키 관리 함수를 C언어로 구현하여 Pentium III 866 CPU와 256M RAM 환경에서 테스트한 결과가 표 3과 같다. 키 생성 테스트는 키 생성 과정을 20번 실행 테스트했을 경우 나타나는 시간상의 수치이고, 서명 생성과 검증 테스트는 서명 생성과 검증 과정을 10000번 실행 테스트를 실행했을 경우 나타나는 시간상의 수치이다.

전자서명 메커니즘의 성능평가를 비교한 표 3의 결과를 살펴보면, DSA와 KCDSA 메커니즘의 성능평가 결과는 거의 비슷한 것을 알 수 있다. DSA보다 본 논문에서 제안하는 안전한 KCDSA 메커니즘의 서명 생성 시간이 약간 느려진 이유는, 새로 정의한 C_SecureWrapKey 함수로 키를 안전하게 저장하기위해 사용자 PIN으로부터 키를 생성하여 암호화하는 과정에서 나타난 결과이다. 반면에, DSA와 KCDSA 메커니즘이 RSA 메커니즘보다 키쌍과 전자서명 생성하는 데 걸리는 시간은 빠르고, 전자서명을 검증하는데 걸리는 시간은 느리다. 표 3의 전자서명 메커니즘 성능평가 비교결과와 앞의 2.3에서

설명한 KCDSA 전자서명 방식의 장점을 종합하여 살펴보면, 스마트카드와 같은 이동성매체에서는 KCDSA 메커니즘을 사용하는 것이 더욱 좋다는 것을 알 수 있다.

KCDSA 메커니즘을 제공하는 PKCS #11 보안 API는 일반적인 PKCS #11 보안 API에서 정의된 RSA, DSA 메커니즘과 함께 새로 정의한 키 관리 함수 C_SecureWrapKey 함수의 기능을 안전하게 실행한다. 그리고, PKCS #11에서 지원하는 여러 메커니즘 중에서 KCDSA 메커니즘을 선택하여 전자서명을 생성하고 검증하는 과정도 안전하게 실행된다.

본 논문에서 제안한 KCDSA 메커니즘을 제공하는 PKCS #11 보안 API와 일반적인 PKCS #11 보안 API에 대한 비교 분석은 표 4에서 설명한다. 논문 [19]와 [20]의 보안 API에 대한 평가 기준을 참고하여 KCDSA 메커니즘을 제공하는 PKCS #11 보안 API를 일반적인 PKCS #11과 어떻게 비교분석할지 적절한 요소들을 아래와 같이 비교기준을 선별하여 표 4와 같이 서로를 비교분석한다.

- 다양한 응용프로그램 지원: 현재 작성되고 있는 다양한 응용프로그램 뿐만 아니라 향후 제시될 응용프로그램에서도 다양한 메커니즘을 제공할 수 있어야 한다.
- 국내 라이브러리와의 호환성: PKCS #11 보안 API는 국내 전자서명 표준 KCDSA 메커니즘을 지원하지 않고있다. 국내 전자서명 표준 KCDSA 메커니즘을 제공하여 국내 전자상거래 인증기관과 제품간의 상호 운용이 가능하도록 한다.
- SSL(Secure Socket Layer) 및 Crypto API와의 호환성: 웹서버와 브라우저간의 통신 암호화 목적으로 이용되는 SSL은 현재 많은 웹서버와 브라우저에서 지원하고 있다. 본 논문에서 구현한 KCDSA 메커니즘을 제공하는 PKCS #11 라이브러리는 Microsoft의 암호 라이브러리를 사용하여 구현하였으므로 SSL과 호환되지 않고 Microsoft Crypto API와의 호환성을 지원한다.
- 사용자 인증: 암호토큰에 접근하기 위해서는 C_Login 함수를 통과해야 하므로 사용자 인증 기능을 충분히 제공한다. 그리고, 노출되기 쉬운 개인키 및 공개키를 암호화하기 위해서는 사용자 PIN을 입력하므로써 사용자 인증의 기능을 강화한다.
- 키 보안성: 일반적인 PKCS #11 보안 API는 암호화하기위해 함수의 파라미터에 키를 직접 입력하므로 안전한 키가 입력되는지에 대한 확인이 필요하다. 본 논문에서 제안한 새로운 키 관리 함수에서는 키 대신 사용자의 PIN을 직접 입력하여 키에 대한 보안성을 강화한다. 그러므로, 사용자의 KCDSA 공개키와 개인키를 안전하게 보관하여 사용할 수 있다.
- API 이용한 프로그래밍 안전성: 전자상거래 응용프로그램 개발자가 PKCS #11 보안 API를 사용하여 개발하는 경우에, 잘못된 사용으로 인하여 보안상의 문제가 야기될 수도 있다. 그러므로, 응용프로그램 개발자에 의해 초래되는 보안 사고를 방지하기 위해 취해지는 프로그래밍 안전성 지원이 매우 중요하다. 일관성있는 이름부여, 부주의로 인한 정보 유출을 방지하기 위한 정보 은닉, 보안 API의 일관성있는 순서 등을 제공하여 보안 API를 이용하는 응용프로그램 개발자가 실수할 확률을 줄여주도록 한다.

표 4. 일반적인 PKCS #11과 KCDSA 메커니즘을 제공하는 PKCS #11의 비교

	일반적인 PKCS #11	KCDSA 메커니즘을 제공하는 PKCS #11
다양한 응용프로그램 지원	지원함	지원함
국내 라이브러리와의 호환성	지원하지 않음	지원함
SSL과의 호환성	지원함	지원하지 않음
Crypto API와의 호환성	지원하지 않음	지원함
사용자 인증	지원함	지원함
키 보안성	지원하지 않음	지원함
API 이용한 프로그래밍 안전성	지원함 (일부분 불안정함)	지원함

V. 결 론

본 논문에서는 PKCS #11 보안 API에 국내 전자상거래의 호환성과 효율성을 제공하기 위한 KCDSA 메커니즘을 제공할 수 있도록 기능을 추가하여 확장시켰다. 새로운 KCDSA 개인키 객체와 공개키 객체를 정의하고, 새로운 키 관리 함수 C_SecureWrapKey 함수를 정의하였다. 기존의 C_WrapKey에 키를 바로 입력하면 변조된 키를 전달할 수 있는 위험성이 있기 때문에, 새로 정의한 C_SecureWrapKey 함수에 키 대신 사용자 PIN을 입력하므로써 키 보관에 대한 안전성이 매우 향상되었다. 그리고, RSA, DSA, KCDSA 메커니즘을 C 언어로 구현하여 성능평가를 비교하고, 본 논문에서 제안한 안전한 KCDSA 메커니즘의 장단점을 알 수 있도록 기존 PKCS #11과 서로 비교 분석하였다.

향후 연구과제로는, PKCS #11 보안 API에 안전성과 효율성을 제공하는 본 논문의 결과가 현재 국내 표준인 [11]의 참고가 되도록 지속적인 연구가 계속해서 이루어질 것이다. 그리고, EC-KCDSA (Elliptic Curve KCDSA) 메커니즘을 제공하는 PKCS #11을 설계 및 분석하여 구현하고, PKCS #11 보안 API에 대한 제3자의 공격을 분석하여 보안과 대책 방법 등에 대해서도 계속 연구하고자 한다. 보안성을 갖춘 안전한 보안 API 뿐만 아니라, 성능면과 사용자 편이성 문제점도 고려한 보안 API를 설계 및 개발하고자 한다.

참 고 문 헌

[1] Jolyon Clulow, "The design and analysis of cryptographic application programming interfaces for devices," Master's thesis, University of Natal, Durban, 2003.

[2] NSA Cross Organization CAPI Team, "Security Service API: Cryptographic API Recommendation Updated and Abridged Edition," The National Security Agency, July 1997.

[3] Mike Bond, Ross Anderson, "API-Level Attacks on Embedded Systems," IEEE Computer Magazine Oct 2001, pp. 67-75, 2nd May 2001.

[4] Mike Bond, "Attacks on Cryptoprocessor Transaction Sets," Cryptographic Hardware and Embedded System-CHES 2001 Third International Workshop, Springer LNCS, V2162, pp. 220-234, 2001.

[5] Peter Gutmann, "The Design of A Cryptographic Security Architecture," Proceedings of the 8th USENIX Security Symposium, Washington, D.C., USA, August 23-26, 1999.

[6] Paul Kocher, Joshua Jaffe, Benjamin Jun, "Introduction to Differential Power Analysis and Related Attacks," Technical report, Cryptography Research Inc., 1998.

[7] Paul Kocher, Joshua Jaffe, Benjamin Jun, "Differential Power Analysis," Advances in Cryptology-CRYPTO '99, Springer LNCS, v1666, pp. 388-397, 1999.

[8] 이성은, 장홍종, 박인재, 한선영, "다중 암호화 기법을 활용한 하이브리드 스마트카드 구현," 정보보호학회논문지, 13(2), pp. 81-89, 2003.

[9] Jolyon Clulow, "On the Security of PKCS #11,"CHES 2003, LNCS 2779, 2003.

[10] RSA Laboratories, "PKCS #11 v2.20 : Cryptographic Token Interface Standard-Draft 5," 2004.

[11] 인터넷보안기술포럼, "암호토큰을 위한 PKCS #11 프로파일 표준," 2003.

[12] 류영규, 윤호선, 염홍열, "분배된 비밀 공유 기법을 이용한 KCDSA 매직 잉크 서명 방식," 정보보호학회논문지, 9(2), pp. 13-23, 1999.

[13] 오형근, 이임영, 김지연, 박성준, "공정한 은닉 KCDSA 서명에 기반한 추적 가능한 전자화폐 시스템," 정보보호학회논문지, 9(4), pp. 85-97, 1999.

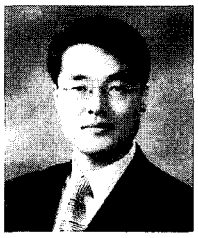
[14] 이상곤, 윤태은, "EC-KCDSA 부분 은닉서명을 이용한 거스름 재사용 가능한 전자수표지불 시스템," 정보보호학회논문지, 13(1), pp. 3-10, 2003.

[15] Chae Hoon Lim, "The Revised Version of KCDSA," 2000.

[16] KCDSA Task Force Team, "The Korean Certificate-based Digital Signature Algorithm," 1998.
[17] 김명희, 전문석, "PKCS#11에 기반한 KCDSA 메커니즘 설계," 한국정보처리학회 춘계학술발표논문집, 11(1), pp. 1015-1018, 2004.
[18] Vlastimil Klima, Tomas Rosa, "Attack on Private Signature Keys of the OpenPGP format, PGPTM programs and other applications compatible with OpenPGP," March 2001.
[19] R. L. Rivest, M. E. Hellman, J. C. Anderson, "Response to NIST's proposal," Comm. ACM, 35(7), pp. 41-52, 1992.
[20] 주학수, 이언경, 김승주, "암호라이브러리 및 암호API 개발현황," 정보보호학회지, 12(4), pp. 94-103, 2002.

〈著者紹介〉

김 명 희 (Myung-Hee Kim) 정회원
1995년 2월 : 경남대학교 전산통계학과 졸업
1998년 2월 : 창원대학교 전자계산학과 석사
1999년 9월~현재 : 숭실대학교 컴퓨터학과 박사과정
2001년 12월~현재 : (주)안철수연구소 주임연구원
〈관심분야〉 정보보호, 암호학, 공개키기반구조(PKI)



김 은 환 (Eun-Hwan Kim) 정회원
1990년 2월 : 숭실대학교 전자계산학과 졸업
1997년 2월 : 숭실대학교 컴퓨터학과 석사
2003년 2월 : 숭실대학교 컴퓨터학과 박사
1990년 3월~1995년 8월 : 국방과학연구소 연구원
1997년 9월~현재 : 숭실대학교 전산원 전임교수
〈관심분야〉 정보보호, 암호학, 네트워크 및 인터넷 보안

전 문 석 (Moon-Seog Jun) 정회원
1980년 2월 : 숭실대학교 전자계산학과 졸업
1986년 2월 : University of Maryland 전산과 석사
1989년 2월 : University of Maryland 전산과 박사
1989년 3월~1991년 2월 : Morgan State University부설 Physical Science Lab 책임연구원
1991년 3월~현재 : 숭실대학교 정보과학대학 정교수
〈관심분야〉 정보보호, 암호학, 네트워크 및 인터넷 보안, 공개키기반구조(PKI)