바이오정보 연계 등 스마트폰 환경에서 공인인증서 안전 이용 구현 가이드라인

Implementation Guideline for Safe Usage of Accredited Certificate using bio information in Smart phone

V1.00

2016년 5월

KISA 한국인터넷진흥원
KOREA INTERNET & SECURITY AGENCY

< 목 차 >

바이오정보 연계 등 스마트폰 환경에서 공인인증서 안전 이용 구현 가이드라인
Implementation Guideline for Safe Usage of Accredited Certificate using bio information in Smart Phone

1. 개요

본 가이드라인은 스마트폰에서 별도 프로그램 설치 없이 바이오정보 등 다양한 인증기술 연계를 통해 공인인증서를 간편하고 안전하게 이용할 수 있는 방법에 대해 기술한다.

2. 구성 및 범위

공인인증서를 이용해 서버로 로그인하는 과정은 사용자가 가입자 소프트웨어를 통해 공인인증서를 선택한 후 전자서명생성정보 비밀번호를 입력해 암호화된 전자서명생성정보가 복호화에 성공하는지 확인하는 로컬인증(Local Authentication)과 복호화된 전자서명생성정보로 서버가 보내준 난수값을 서명해서 서명 값을 생성한 후 공인인증서와 함께 서버로 보내 서버가 이를 검증하는 원격인증(Remote Authentication)으로 구성된다.

본 가이드라인은 스마트폰 환경에서 공인인증서를 안전하게 저장하고 사용자가 전자서명생성정보 비밀번호를 입력하는 과정에서 비밀번호를 입력하는 대신 다양한 인증수단을 이용하는 방법을 기술한다.

부록에서는 스마트폰 환경에서 바이오정보, NFC IC카드 등 다양한 인증수단을 통해 공인인증서를 간편하고 안전하게 이용할 수 있는 기술을 제공한다.

3. 관련 표준 및 규격

3.1 국외 표준 및 규격

[PKCS5] RSA Laboratories, PKCS#5 v1.5 & v2.0, Password-Based Cryptography Standard, 1993

[PKCS7] RSA Laboratories, PKCS#7 v1.5, Cryptographic Message Syntax, 1993

[PKCS8] RSA Laboratories, PKCS#8 v1.2, Private Key Information Syntax Standard, 1993

[PKCS9] RSA Laboratories PKCS#9 v2.0, Selected Object Classes and Attribute Typeds, 2000

[PKCS12] RSA Laboratories, PKCS#12 v1.0, Personal Information Exchange Syntax Standard, 1999

3.2 국내 표준 및 규격

해당사항 없음

4. 정의

4.1 전자서명법 용어의 정의

본 규격에서 사용된 다음의 용어들은 전자서명법 및 동법 시행령, 공인인증기관의 시설 및 장비 등에 관한 규정(미래창조과학부 고시)에 정의되어 있다.

가) 공인전자서명
나) 전자서명생성정보
다) 전자서명검증정보
라) 인증
마) 공인인증서
바) 가입자
사) 가입자 소프트웨어

4.2 용어의 정의

가) 로컬인증 : 전자서명생성정보를 접근하기 위해 올바른 사용자 여부를 확인하기 위한 방법

가) 트러스트존 : 스마트폰 내 전자서명생성정보 및 전자서명검증정보를 안전하게 저장할 수 있는 저장매체

4.3 용어의 효력

본 규격에서 사용된 다음의 용어들은 공인인증기관 및 가입자 소프트웨어의 구현 정도를 의미하는 것으로 [RFC2119]를 준용하며 다음과 같은 의미를 지닌다.

가) 해야한다, 필수이다, 강제한다 (기호 : M)
반드시 준수해야 한다.

나) 권고한다 (기호 : R)
보안성 및 상호연동을 고려하여 준수할 것을 권장한다.

다) 할 수 있다, 쓸 수 있다 (기호 : O)
주어진 상황을 고려하여 필요한 경우에 한해 선택적으로 사용할 수 있다.

라) 권고하지 않는다 (기호 : NR)
보안성 및 상호연동을 고려하여 사용하지 말 것을 권장한다.

마) 금지한다, 허용하지 않는다 (기호 : X)
반드시 사용하지 않아야 한다.

바) 언급하지 않는다, 정의하지 않는다 (기호 : -)
준수 여부에 대해 기술하지 않는다.

5. 약어

본 가이드라인에서는 다음의 약어가 이용된다.

가) ASN.1 : Abstract Syntax Notation One, 추상적 구문 표기
나) TEE : Trusted Execution Environment, 신뢰된 실행 환경
다) REE : Rich Execution Environment, 일반 실행 환경
라) TA : Trusted Application, 신뢰된 어플리케이션

6. 보안 요구사항

본 가이드라인에서는 다음의 보안 요구사항을 정의한다.

가) 루팅(Rooting)·탈옥(Jail-Break) 등 스마트폰이 불법 변경된 환경에서는 모든 저장소에 접근이 가능하기 때문에, 물리적으로 독립된 안전한 하드웨어 저장소를 활용하는 것을 권고한다.
나) 비밀번호, 생체정보 등 전자서명생성정보 접근을 위한 로컬인증 실패시 횟수를 제한하여야 한다.
다) 스마트폰 내 지문인식 장치의 FAR(오인식률)은 1/50,000을 지원하여야 하며, FRR(오거부율)은 2~3% 이하를 지원하여야 한다.
라) 위조지문 등 스마트폰 내 지문인식 장치의 취약점이 발견되는 즉시 보안조치가 이루어져야 한다.
마) TEE 클라이언트와 TA는 신뢰된 상호인증을 수행하는 것을 권고한다.

7. 로컬 인증

7.1 로컬인증을 위한 인터페이스

공인인증서와 전자서명생성정보가 외부의 접근이 엄격하게 통제되고 있는 안전한 저장소(예 : SE, TEE 등)에 저장되지 않고 일반 저장소(예 : 앱내 등)에 저장될 경우, 전자서명생성정보는 반드시 암호화해서 저장해야 한다.

기존 공인인증서의 쌍이 되는 전자서명생성정보의 경우에도 대칭키 방식으로 암호화되어있으므로 로컬인증(소유 및 생체인증) 후에는 전자서명생성정보 암/복호화를 할 수 있는 키를 생성(또는 유도)하거나, 암/복호화/서명 등을 수행할 수 있는 객체를 제공해야 한다.

다음은 로컬인증을 위한 Java와 Objective C의 인터페이스 명세이다.

```
public interface LocalAuth {
        public void init( Object param ) throws Exception;
        public void startReg( byte[] privKey, RegResult callbackObj );
        public void startAuth( String signAlgName, byte[] encryptedPrivateKey,
                               AuthResult callbackObj );
}

public interface RegResult {
        public void success(byte[] encryptedPrivateKey);
        public void error(Exception e);
}

public interface AuthResult {
        public void success(java.security.Signature sig);
        public void error(Exception e);
}
```

<Java 인터페이스 명세(Android에서 활용 가능)>

```
@interface Signature : NSObject
{
    NSString* signAlgName;                  /* 서명알고리즘 */
    NSData* encryptedPrivateKey;            /* 확장 전자서명생성정보 */
}

/**
* @brief : sign                           [서명 처리(키체인에서 키를 취득하여 서명처리)]
* @param : [IN] NSData* originalData      [서명원문]
* @param : [OUT]NSError* error            [오류코드(정상인경우: nil)]
* @return: [OUT]NSData**                  [서명데이터]
*/
- (NSData*)sign:(NSData*)originalData error:(NSError**)error;
@end

/*인스턴스 메소드 타입*/
@interface LocalAuth : NSObject
{
    NSDictionary* param;
}
```

```
/**
* @brief : initWithPrarm                          [확장 전자서명생성정보 인터페이스 객체 생성]
* @param : [IN] NSDictionary* paramDict           [파라메터]
* @return :[OUT]id                                [생성된 확장 전자서명생성정보 인터페이스 객체]
*/
-(id)initWithPrarm:(NSDictionary*)paramDict;

/**
* @brief : startReg                               [확장 전자서명생성정보 생성 처리]
* @param : [IN] NSData* privateKey                [전자서명생성정보 데이터]
* @param : [OUT]NSError* error                    [오류코드(정상인경우: nil)]
* @return :[OUT]NSData                            [확장전자서명생성정보 데이터(실패:nil)]
*/
- (NSData*)startReg:(NSData*) privateKey error:(NSError**) error;

/**
* @brief : startAuth                [확장 전자서명생성정보 생성 처리(성공시 키체인에 확장전자서명생성정보 저장)]
* @param : [IN] NSData* encryptedPrivateKey       [확장 전자서명생성정보]
* @param : [IN] NSString* signAlgName             [서명알고리즘]
* @param : [OUT]NSError* error                    [오류코드(정상인경우: nil)]
* @return :[OUT]Signature*                        [Signature객체(실패:nil)]
*/
- (Signature*)startAuth:(NSData*) encryptedPrivateKey signAlgName:(NSString*)signAlgName error:(NSError**)error;

@end
```

<Objective C 인터페이스 명세(iOS에서 활용 가능)>

스마트폰 환경에서 공인인증서의 쌍이 되는 전자서명생성정보를 암/복호화하기 위해 상기 인터페이스를 상속받아 완전한 클래스 또는 모듈을 구현한다.

구현된 클래스 또는 모듈을 공인인증서 가입자 S/W 개발 벤더로 제공하면 전자서명생성정보 비밀번호 입력 루틴을 소유 또는 생체기반 인증으로 대체할 수 있다.

가입자 S/W 개발 벤더는 소유 또는 생체기반 인증을 통해 생성된 암호키를 전자서명생성정보 암호로 사용하며 암호화 방법은 기존에 사용하던 PKCS #5, #8 표준을 그대로 따른다. 다만 복수개의 인증수단을 사용하기

위해 7.2절에 확장 기술을 명시하고 있다.

7.2 전자서명생성정보 암호화 확장 기술

전자서명생성정보 암호화 확장 기술은 복수개의 인증수단(예 : NFC 소유 인증, 바이오인증 등)을 통해 전자서명생성정보를 암호화하기 위한 구조를 명시하고 있다. 이는 기존 공인인증서 비밀번호뿐만 아니라 다양한 인증수단을 동시에 사용할 수 있음을 의미한다.

예를 들어, 사용자는 공인인증서 로그인을 위해 기존 전자서명생성정보 비밀번호를 입력해도 되고, 교통카드가 지원되는 복수의 신용카드 중 하나를 선택해도 되고, 지문 등 바이오인식을 사용해도 된다.

7.2.1 복수의 암호화된 전자서명생성정보

복수의 암호화된 전자서명생성정보는 공인인증서의 쌍이 되는 하나의 전자서명생성정보를 여러 개의 인증수단을 통해 생성된 각각의 비밀번호를 통해 전자서명생성정보를 암호화한 집합(EncryptedPrivateKeyInfos)이다.

```
EncryptedPrivateKeyInfos ::= SEQUENCE OF EncryptedPrivateKeyInfo
        -- PKCS #8 정의된 EncryptedPrivateKeyInfo 의 집합

EncryptedPrivateKeyInfo ::= SEQUENCE {
         encryptionAlgorithm EncryptionAlgorithmIdentifier,
         encryptedData EncryptedData
}

EncryptionAlgorithmIdentifier ::= AlgorithmIdentifier
EncryptedData ::= OCTET STRING

AlgorithmIdentifier ::= SEQUENCE {
         algorithm OBJECT IDENTIFIER,
         parameters ANY DEFINED BY algorithm OPTIONAL
}

algorithm ::= CHOICE {
         nfcObjectBasedEncryption(1.2.410.200004.1.101.1),
          -- NFC 객체 인증을 통한 소유 기반 암호화
         fingerprintBasedEncryption(1.2.410.200004.1.101.2),
```

```
          -- 지문 인증을 통한 생체 기반 암호화
         faceprintBasedEncryption(1.2.410.200004.1.101.3),
          -- 얼굴 인증을 통한 생체 기반 암호화
         voiceprintBasedEncryption(1.2.410.200004.1.101.4),
          -- 화자 인증을 통한 생체 기반 암호화
         eyeprintBasedEncryption(1.2.410.200004.1.101.5),
          -- 망막 인증을 통한 생체 기반 암호화
         irisprintBasedEncryption(1.2.410.200004.1.101.6),
          -- 홍채 인증을 통한 생체 기반 암호화
         handprintBasedEncryption(1.2.410.200004.1.101.7),
          -- 손바닥 인증을 통한 생체 기반 암호화
         actofsigningBasedEncryption(1.2.410.200004.1.101.8)
          -- 서명 행위 인증을 통한 생체 기반 암호화
}

parameters := SEQUENCE {
         pbeParam PBEParam,
         objectName UTF8String,
          -- 사용자가 명시한 객체의 이름(예 : 하나카드, 오른쪽 엄지 지문)
        keyFactorIDs KeyFactorIDs OPTIONAL
}

PBEParam ::= CHOICE {
         PBEParameter,
         [1] EXPLICIT PBES2-params
}

KeyFactorIDs ::= SEQUENCE OF KeyFactorID
          -- 암호키 생성 시 사용된 인자ID 집합

KeyFactorID ::= OBJECT IDENTIFIER
          -- 암호키 생성 시 사용된 인자ID

PBEParameter ::= SEQUENCE {
         salt OCTET STRING,
         iterationCount INTEGER
}

PBES2-params ::= SEQUENCE {
         keyDerivationFunc AlgorithmIdentifier {{PBES2-KDFs}},
         encryptionScheme AlgorithmIdentifier {{PBES2-Encs}}
}
```

7.2.2 인증서 내보내기/가져오기 방법

본 절에서는 PKCS #12를 확장해 복수의 암호화된 전자서명생성정보를 PFX에 삽입하는 방법을 기술한다.

PKCS #12에서 PFX는 다음과 같은 구조로 정의되어 있다.

```
PFX ::= SEQUENCE {
        version INTEGER {v3(3)}(v3,...),
        authSafe ContentInfo,
        macData MacData OPTIONAL
}

MacData ::= SEQUENCE {
        mac DigestInfo,
        macSalt OCTET STRING,
        iterations INTEGER
}
```

이 중 authSafe에 정의된 value는 다음과 같이 ContentInfo의 배열로 정의되어 있다.

```
AuthenticatedSafe ::= SEQUENCE OF ContentInfo
        -- Data if unencrypted
        -- EncryptedData if password-encrypted
        -- EnvelopedData if public key-encrypted
```

AuthenticatedSafe에는 암호화된 인증서와 전자서명생성정보가 포함되는데 일반적인 암호화 방법은 아래와 같다.

① 인증서와 전자서명생성정보는 SafeBag 이라는 구조체로 변환한다.
② SafeBag의 집합체인 SafeContents을 구성한다.
③ SafeContents를 DER 인코딩 한다.
④ DER 인코딩된 SafeContents를 PKCS #12에 정의된 PBE 방식으로 암호화 한다.

```
SafeContents ::= SEQUENCE OF SafeBag

SafeBag ::= SEQUENCE {
        bagId BAG-TYPE.&id ({PKCS12BagSet})
        bagValue [0] EXPLICIT BAG-TYPE.&Type({PKCS12BagSet}{@bagId}),
        bagAttributes SET OF PKCS12Attribute OPTIONAL
}

PKCS12Attribute ::= SEQUENCE {
        attrId ATTRIBUTE.&id ({PKCS12AttrSet}),
        attrValues SET OF ATTRIBUTE.&Type ({PKCS12AttrSet}{@attrId})
}

PKCS12AttrSet ATTRIBUTE ::= {
        friendlyName ¦ -- from PKCS #9
        localKeyId, -- from PKCS #9
        ... -- Other attributes are allowed
}
```

PKCS #12에 정의된 SafeBag의 종류는 6가지로 다음과 같다.

```
PKCS12BagSet BAG-TYPE ::= {
        keyBag ¦
        pkcs8ShroudedKeyBag ¦
        certBag ¦
        crlBag ¦
        secretBag ¦
        safeContentsBag,
        ... -- For future extensions
}
```

본 가이드라인에서는 PFX에 “전자서명생성정보 암호화 확장”을 포함하기 위해 SecretBag을 사용하고, SecretBag의 Type으로는 encryptedPrivateKeyInfos에 해당하는 형식을 정의해서 사용한다.

```
SecretBag ::= SEQUENCE {
        secretTypeId BAG-TYPE.&id ({SecretTypes}),
        secretValue [0] EXPLICIT BAG-TYPE.&Type
}

SecretTypes BAG-TYPE ::= {
        encryptedPrivateKeyInfos ¦
        ... -- For future extensions
}

encryptedPrivateKeyInfos BAG-TYPE = {1.2.410.200004.7.1.2}
        -- secretValue 는 EncryptedPrivateKeyInfos를
           DER 인코딩 한 값을 EXPLICIT 로 표현한 값이다.
```

전자서명생성정보 암호화 확장의 경우, 소유기반 인증과 생체기반 인증을 통해 생성된 암호키로 암호화된 복수 전자서명생성정보의 집합으로 구성되어 있으나, 생체 인증정보의 경우, 같은 매체라 하더라도 등록 시 패턴 정보의 변경이 있기 때문에 모든 기기에 대해서 동일한 암호키를 유도하는 것은 어려울 수 있다. 그렇기 때문에 인증서/전자서명생성정보를 내보내기 할 때는 모든 장치에서 동일하게 암호키 유도가 가능한 암호화된 전자서명생성정보만 포함되도록 해야 한다.

또한 인증서, 전자서명생성정보 쌍을 찾는데 도움을 주는 속성인 localKeyID(PKCS #9)를 전자서명생성정보 암호화 확장이 들어가는 SafeBag의 bagAttributes에 동일하게 적용한다.

8. 스마트폰 환경에서 안전한 공인인증서 저장방법

본 가이드라인에서는 스마트폰 환경에서 공인인증서와 전자서명생성정보를 안전하게 저장하는 방법을 기술한다.

8.1 안드로이드 운영체제

안드로이드 운영체제에서는 전자서명생성정보를 저장하기 위해 TEE 또는 키스토어(Keystore)를 활용하는 것을 권고한다.

공인인증서의 쌍이 되는 전자서명생성정보를 키스토어(또는 TEE)에 직접 저장하거나, 인증서/전자서명생성정보 파일을 키스토어(또는 TEE)에 저장된 비밀키로 암호화해서 앱 내 저장소에 저장하는 것이 바람직하다.

본 가이드라인에서는 인증서 내보내기 기능 구현을 고려해 인증서/전자서명생성정보를 암호화해 앱 내 저장소에 저장하는 방법을 기술한다.

8.1.1 안드로이드 키스토어

안드로이드 키스토어는 단말로부터 비밀키를 추출하는 것을 어렵게 하기 위해 컨테이너에 비밀키를 저장한다. 컨테이너에 저장된 비밀키는 인가되지 않은 사용으로부터 키 데이터의 추출을 방지하며, TEE를 지원하는 OS 버전인 경우, 하드웨어적으로 안전한 저장소에 키를 저장한다.

안드로이드 키스토어를 사용하는 앱의 경우, 앱 삭제 시 키스토어 내 공인인증서 관련 정보도 삭제되는 것을 안내해야 한다.

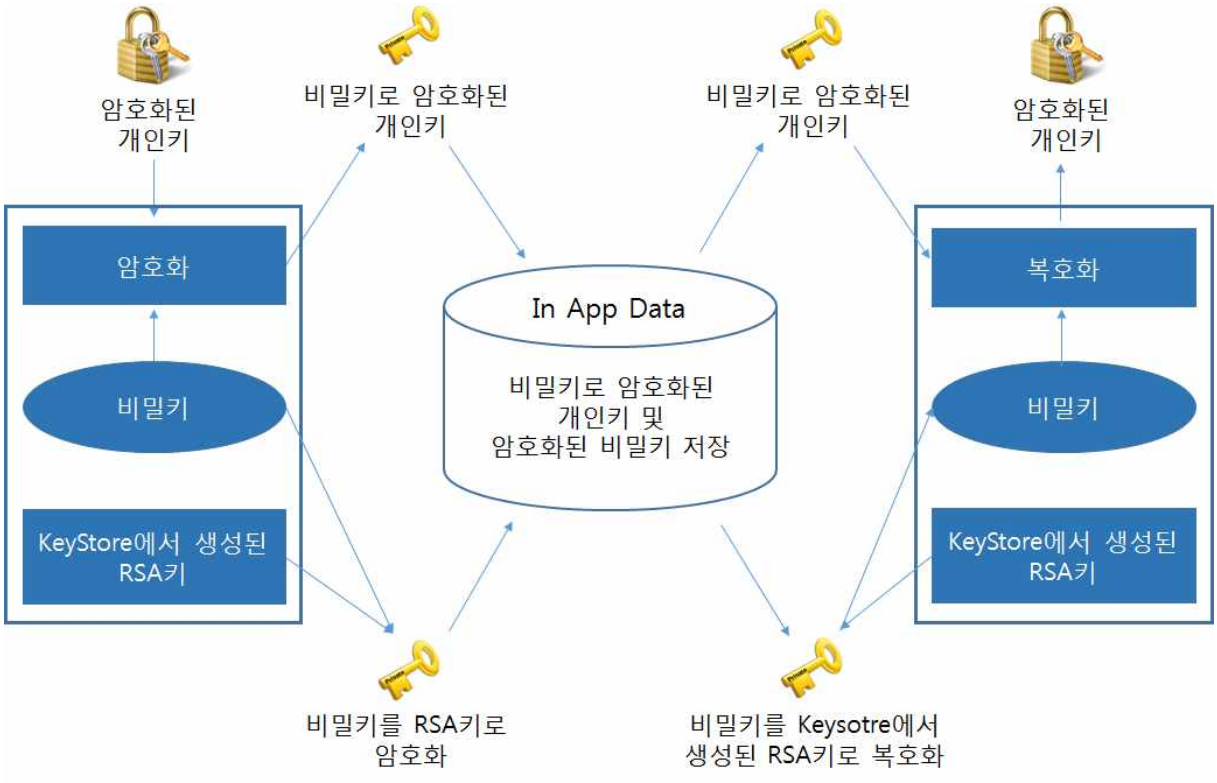
8.1.2 키스토어 지원 키

키스토어에서 지원되는 키의 종류는 안드로이드 운영체제 버전 별로 다음과 같다.

① 안드로이드 4.3(API 18) 이상 5.x 이하 : 인증서, 전자서명검증정보(RSA), 전자서명생성정보(RSA)
② 안드로이드 6.0(API 23) : 인증서, 전자서명검증정보(RSA), 전자서명생성정보(RSA), 비밀키(SecretKey), MAC키

8.1.3 키스토어의 활용

① 안드로이드 4.3 이상 5.x 이하 환경



<안드로이드 4.3 이상 5.x 이하(RSA 키 이용)>

[전자서명생성정보의 안전한 저장방법]

1. RSA 키 쌍(전자서명검증정보, 전자서명생성정보)을 생성하고 키스토어에 저장한다.
2. 임의의 비밀키(SecretKey)를 생성한다.
3. 비밀키를 과정 1에서 생성한 RSA 전자서명검증정보로 암호화 후 저장한다.
4. 암호화된 전자서명생성정보(PKCS#8-EncryptedPrivateKeyInfo)를 과정 2에서 생성한 비밀키로 암호화 후 저장한다.(이중 암호화)
5. 비밀키를 해제한다.

[전자서명생성정보 이용 방법]

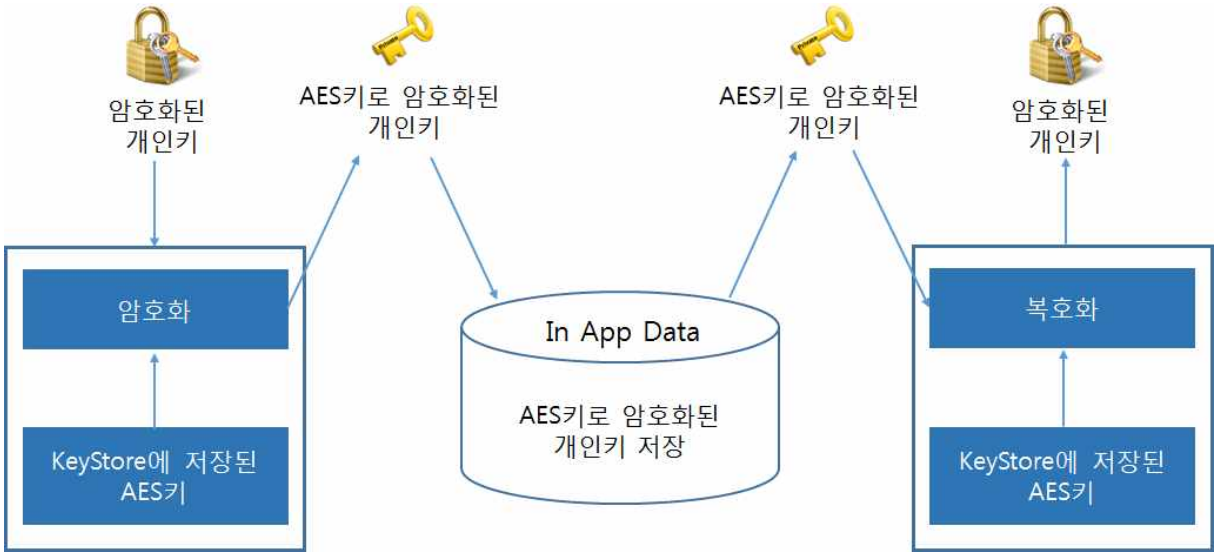
1. 키스토어에 저장된 RSA 전자서명생성정보 객체(예 : java.security.PrivateKey)를 로드한다.

   ※ 전자서명생성정보 객체 : 전자서명생성정보를 이용해 암호연산을 할 수는 있으나, 실제 전자서명생성정보 값(Raw Key)을 추출할 수

는 없는 형태의 객체

2. 암호화된 비밀키를 로드하고, 과정 1에서 로드된 전자서명생성정보 객체로 복호화한다.
3. 이중 암호화된 전자서명생성정보를 로드해, 과정 2에서 얻은 비밀키로 복호화 후 암호화된 전자서명생성정보(PKCS#8-EncryptedPritvateKeyInfo)를 얻는다.
4. 비밀키를 해제한다.

② 안드로이드 6.0 이상 환경



<안드로이드 6.0 이상(AES Key 이용)>

[전자서명생성정보의 안전한 저장 방법]

1. 비밀키(예 : AES Key)를 생성하고 키스토어에 저장한다.
2. 암호화된 전자서명생성정보를 과정 1에서 생성한 비밀키로 암호화(예 : AES/CBC) 후 저장한다.(이중 암호화)

[전자서명생성정보 이용 방법]

1. 키스토어에 저장된 비밀키 객체(예 : javax.crypto.SecretKey)를 로드한다.
2. 이중 암호화된 전자서명생성정보를 로드해 과정 1에서 얻는 비밀키 객체로 복호화 후 암호화된 전자서명생성정보

(PKCS#8-EncryptedPrivateKeyInfo)를 얻는다.

8.1.4 키스토어 지원 및 제약사항

안드로이드 운영체제 버전에 따라 키스토어 지원 가능 키 및 사용에 대한 제약사항이 있다. 표와 같이 운영체제별 제약사항에 따라 해당 동작 시 사용자에게 키 삭제 등과 같은 제약 관련 정보를 안내해야 한다.

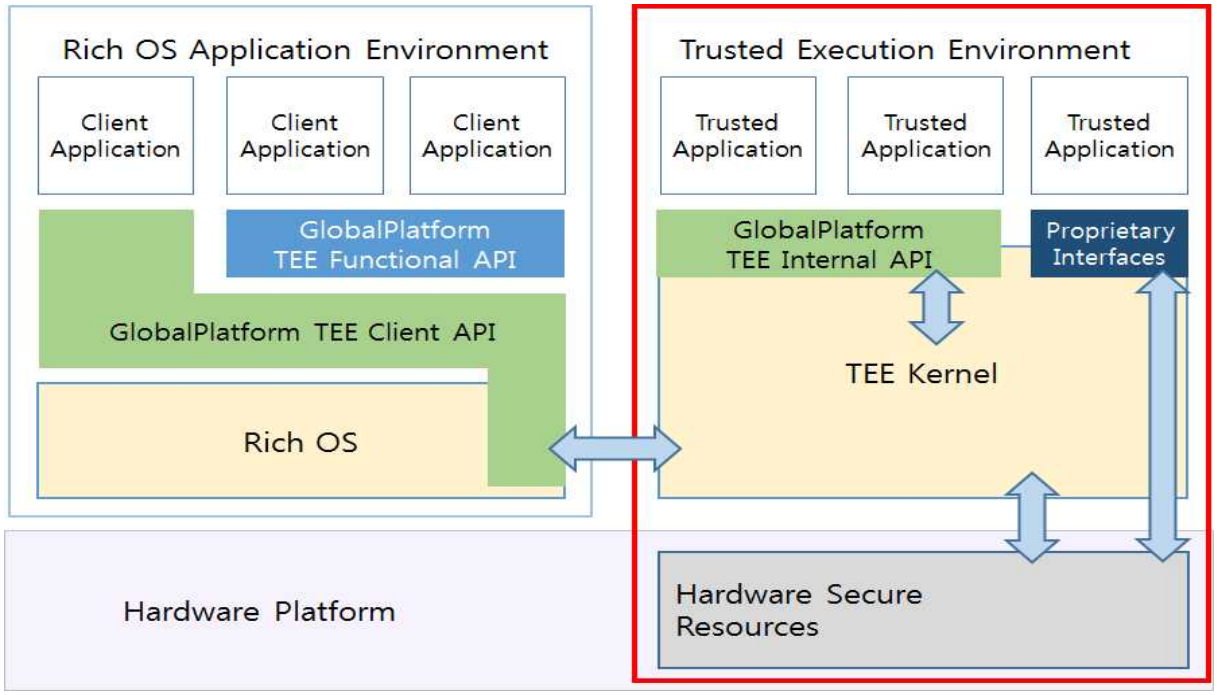
<table>
<tr><td colspan="2" rowspan="2">구분</td><td colspan="3">Android OS Version</td></tr>
<tr><td>Android 4.3
Android 4.4</td><td>Android 5.0</td><td>Android 6.0</td></tr>
<tr><td colspan="2">지원 키 종류</td><td colspan="2">인증서, 전자서명검증정보, 전자서명생성정보
(RSA 방식만 지원)</td><td>인증서,
전자서명검증정보,
전자서명생성정보,
대칭키(비밀키 :
SecretKey), MAC키</td></tr>
<tr><td rowspan="8">디바이스
잠금
방식
변경</td><td>Pin -> 비밀번호</td><td>키가 모두 삭제 됨</td><td>영향 없음</td><td>영향 없음</td></tr>
<tr><td>Pin -> 잠금 없음</td><td>키가 모두 삭제 됨</td><td>잠금 없음 비활성화 됨
(키를 삭제하는 경우
비활성화 풀림)</td><td>영향 없음</td></tr>
<tr><td>pin -> 드래그</td><td>키가 모두 삭제 됨</td><td>드래그 비활성화 됨
(키를 삭제하는 경우
비활성화 풀림)</td><td>영향 없음</td></tr>
<tr><td>Pin -> Pin</td><td>키가 모두 삭제 됨</td><td>키가 모두 삭제 됨</td><td>영향 없음</td></tr>
<tr><td>드래그 -> Pin</td><td>키가 모두 삭제 됨</td><td>키가 모두 삭제 됨</td><td>영향 없음</td></tr>
<tr><td>드래그 -> 잠금 없음</td><td>키가 모두 삭제 됨</td><td>키가 모두 삭제 됨</td><td>영향 없음</td></tr>
<tr><td>잠금 없음 -> 드래그</td><td>키가 모두 삭제 됨</td><td>키가 모두 삭제 됨</td><td>영향 없음</td></tr>
<tr><td>잠금 없음 -> Pin</td><td>키가 모두 삭제 됨</td><td>영향을 받지 않음</td><td>영향 없음</td></tr>
<tr><td rowspan="2">키 추출</td><td>RSA Key</td><td colspan="3">PublicKey, PrivateKey 객체를 가져올 수 있음
(PrivateKey의 Raw Key는 가져올 수 없음)</td></tr>
<tr><td>AES Key</td><td colspan="2">N/A</td><td>SecretKey 형태의
객체를 가져올 수
있음
(AES Key의 Raw
Key는 가져올 수
없음)</td></tr>
</table>

<안드로이드 운영체제별 키스토어 제약사항>

8.1.5 트러스트존(TrustZone)

안드로이드 운영체제를 사용하는 스마트폰 단말의 경우, ARM사의 트러스

트존을 제공하는 경우가 있으며, 트러스트존 기술이 제공하는 TEE(Trusted Execution Environment)를 통해 안전한 키 저장소를 구현할 수 있다. TEE는 안드로이드 운영체제가 구동되는 REE(Rich Execution Environment) 영역과 물리적으로 격리되어 있으며, REE에서 TEE 내 자원을 접근하기 위해서는 TEE Client(이하 TC)와 TEE Trusted Application(이하 TA) 간의 통신 인터페이스를 통해 제한적인 자원에 대해서만 접근할 수 있다.



<트러스트존 구성 환경>

[전자서명생성정보의 안전한 저장 방법]

1. TC에서 TA로 비밀키(예 : AES Key) 생성을 요청한다.
2. TA는 비밀키와 이를 접근할 수 있는 비밀키 핸들을 생성하고, TEE 내부에 저장 후 비밀키 핸들 값을 리턴한다.
3. TC는 비밀키 핸들을 이용해 암호화된 전자서명생성정보(EncryptedPrivateKeyInfo)의 암호화(예 : AES/CBC)를 요청한다.(이중 암호화)
4. TA는 비밀키 핸들에 쌍이 되는 비밀키를 로드해 전달된 전자서명생성정보를 암호화 한 후 리턴한다.
5. TC는 비밀키 핸들과 이중 암호화된 전자서명생성정보를 앱 내 저장소에 저장한다.

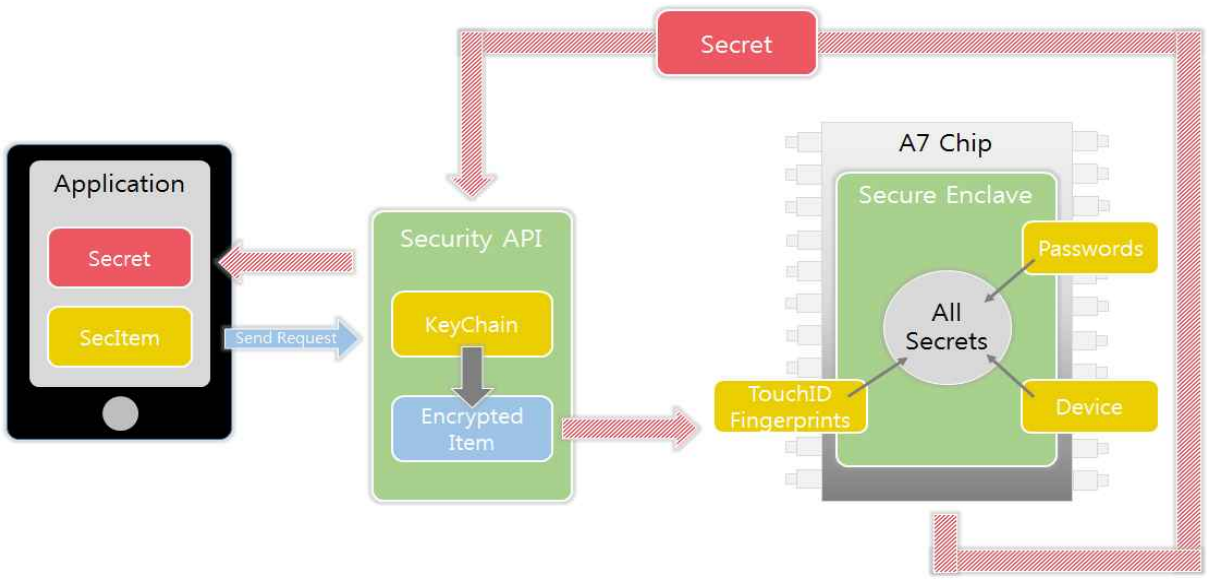
[전자서명생성정보 이용 방법]

1. TC는 비밀키 핸들과 이중 암호화된 전자서명생성정보를 로드한다.
2. TC는 TA로 비밀키 핸들과 이중 암호화된 전자서명생성정보를 보내 복호화를 요청한다.
3. TA는 비밀키 핸들을 통해 TEE 내부에 저장된 비밀키를 찾고, 이중 암호화된 전자서명생성정보를 복호화 후 암호화된 전자서명생성정보(EncryptedPrivateKeyInfo)를 리턴한다.

TEE 클라이언트가 TA로 접근할 때 TA uuid를 통해 접근을 하는데 호출되는 TA 입장에서는 호출된 TEE 클라이언트가 신뢰된 TEE 클라이언트가 아닐 수 있으므로 TEE 클라이언트와 TA 간에 상호인증을 수행하는 것을 권고한다.

8.2 애플(iOS) 운영체제

애플 운영체제의 경우, 인증서와 전자서명생성정보를 저장함에 있어 단순히 앱내 저장소에 저장하는 것 보다 키체인(KeyChain)에 직접 저장하거나, 키체인에 비밀키를 저장하고, 해당 비밀키로 인증서와 전자서명생성정보를 암호화하여 앱내 저장소에 저장하는 것이 바람직하다.



<애플(iOS) 키체인 구성 및 동작과정>

키체인을 사용하면 데이터 암호화 및 타 어플리케이션의 키체인 항목 접근 제한이 가능하기 때문에 일반적으로 데이터를 저장하는 방법(NSUserDefaults, CoreData 등) 보다 전자서명생성정보를 안전하게 저장할 수 있다.

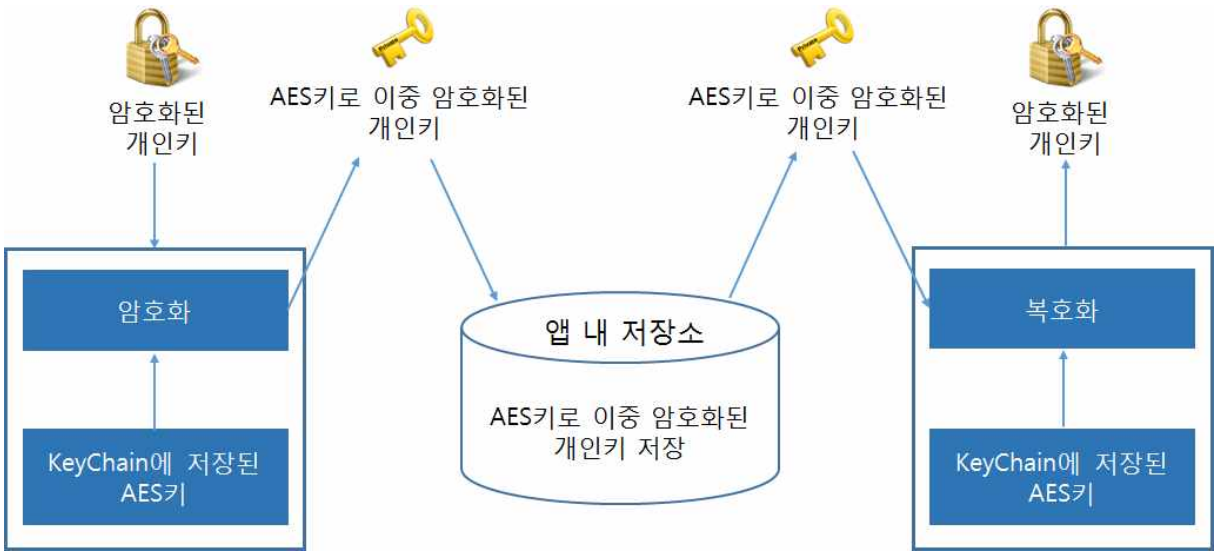
어플리케이션이 키체인에 항목을 추가하거나 검색, 변경, 삭제와 같은 작업을 수행하기 위해서는 키체인 서비스에서 제공하는 API를 사용해야 한다.

키체인의 경우, 애플 운영체제의 휴대폰 잠근 기능(예 : Passcode/지문 등)과 결합 또는 별도 패스워드 설정이 가능하며, 잠금 기능 또는 패스워드와 결합시 보안성이 더욱 높아진다.

키체인에 비밀키를 삽입할 때 운영체제의 잠금 기능과 추가적인 패스워드를 연동하도록 속성을 설정할 수 있으며, 해당 속성이 설정된 경우, 비밀키를 사용하기 위해 접근을 시도하면 운영체제 잠금 기능 및 패스워드 입력창을 띄워 인증을 통과해야만 비밀키 접근 권한을 부여받을 수 있다.

8.2.1 키체인의 활용

① iOS 2.0 이상 환경



<키체인에 저장된 AES 키를 이용한 암호화된 전자서명생성정보 저장>

[전자서명생성정보의 안전한 저장 방법]

1. 비밀키(예 : AES Key)를 생성하고 키체인에 저장한다.
2. 암호화된 전자서명생성정보를 과정 1에서 생성한 비밀키로 암호화 후 저장한다.(이중 암호화)

[전자서명생성정보 이용 방법]

1. 키체인에 저장된 비밀키 객체를 로드한다.
2. 이중 암호화된 전자서명생성정보를 로드해 과정 1에서 얻는 비밀키 객체로 복호화 후 암호화된 전자서명생성정보(PKCS#8-EncryptedPrivateKeyInfo)를 얻는다.

8.2.2 키체인 지원 및 제약사항

애플 운영체제 버전에 따라 키체인 기능에 대한 지원 범위와 제약사항이 있다. 아래 표와 같이 키체인을 잠금 장치 연동할 경우, 운영체제별 제약사항을 확인하여 사용자에게 충분히 안내 후 기능을 수행해야 한다.

구분	iOS Version		
	2.0 이상	8.0 이상	9.0 이상
지원 키 종류	인증서, 전자서명검증정보, 전자서명생성정보, 대칭키(비밀키 : SecretKey), MAC키 등 (키 형태의 제한 없음)		
잠금 장치 연동 지원	(지원하지 않음)	Passcode	Passcode, 지문(실패 시 Passcode 사용)
추가 패스워드 연동 지원	(지원하지 않음)	(지원하지 않음)	지문 + Password

<애플(iOS) 운영체제 버전별 키체인 제약사항>

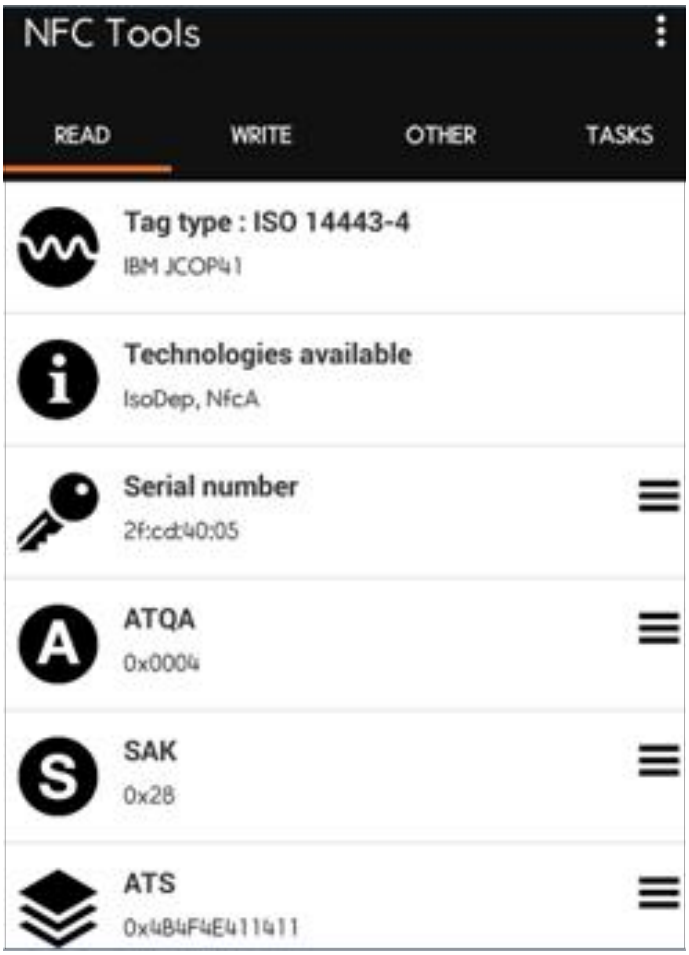
부록 1. 소유 및 생체기반 간편 공인인증기술

1. NFC

1.1 소유기반(NFC 객체) 전자서명생성정보 암호화 방법

NFC의 경우, 거의 모든 스마트폰에서 제공하고 있는 인터페이스로 소유기반 인증을 통해 전자서명생성정보 암호에 필요한 암호키 생성에 적합한 구조를 제공한다. 또한 이용자들이 사용하는 교통카드나 교통카드 기능을 가진 신용·체크카드들(이하 NFC 객체)도 NFC 통신이 가능하기 때문에 스마트폰을 통해 해당 카드들의 정보를 읽을 수 있어, 소유 기반 인증을 통한 암호키 생성이 용이하다.

다음은 스마트폰에서 NFC 객체를 접촉했을 때 NFC통신을 통해 얻을 수 있는 정보들이다.



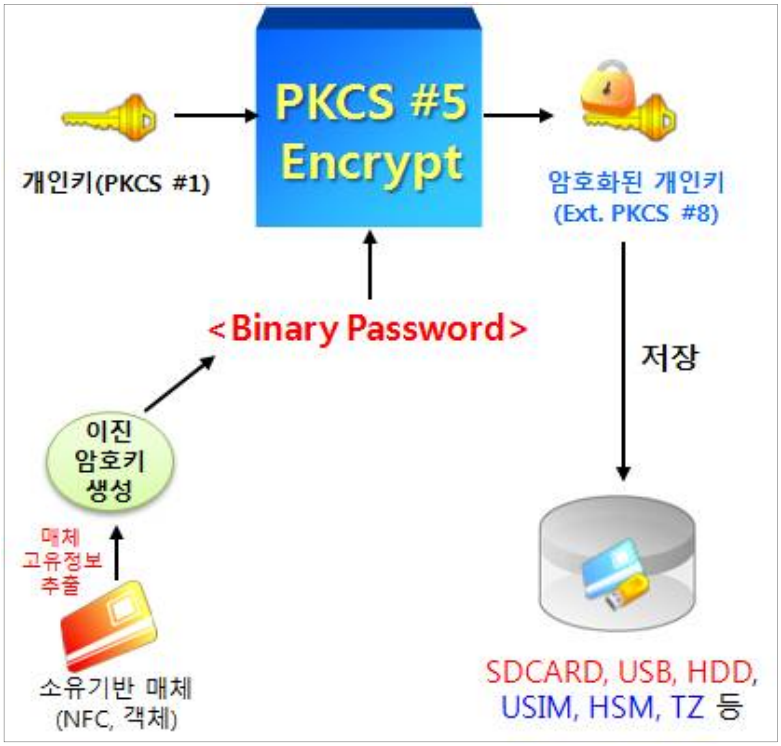
<NFC 객체로부터 얻을 수 있는 정보의 예>

NFC객체로부터 Tag Type, Technologies Available, Serial Number, ATQA, SAK, ATS값 등을 얻을 수 있으며 태그할 때마다 태그된 NFC객체의 고유정보를

리턴하기 때문에 전자서명생성정보를 암호화에 사용되는 패스워드를 생성하는데 사용할 수 있다.

여기에서는 NFC 객체로부터 얻을 수 있는 인자 값을 KeyFactor 라고 하고, 각 KeyFactor별 고유 ID를 KeyFactorID라고 한다.

모든 NFC객체가 위에 기술된 정보를 제공하는 것은 아니므로, 수집할 수 있는 인자들을 나열한 후 해당 인자들로부터 값을 얻어 이진암호(Binary Password)를 생성하고, 생성된 이진암호를 전자서명생성정보 암호로 사용해 공인인증서의 쌍이 되는 전자서명생성정보를 PKCS#5, #8 스펙에 따라 암호화 한다.



<NFC 객체를 이용한 전자서명생성정보 암호화 과정>

KeyFactorIDs ::= SEQUENCE OF KeyFactorID

-- NFC 객체가 제공하는 인자ID의 집합

KeyFactorID ::= OBJECT IDENTIFIER

-- NFC 객체의 경우, 수집할 수 있는 정보가 다양하기 때문에,
Binary Password 조합 과정에서 참여시킨 Key Factor 종류를
명시해야한다.

종류	OBJECT IDENTIFIER
Tag Type	1.2.410.200004.7.1.3.1
Technologies Available	1.2.410.200004.7.1.3.2
Serial Number	1.2.410.200004.7.1.3.3
ATQA	1.2.410.200004.7.1.3.4
SAK	1.2.410.200004.7.1.3.5
ATS	1.2.410.200004.7.1.3.6

<NFC 객체가 제공하는 정보에 대한 KeyFactorID 목록>

KeyFactors := SEQUENCE OF KeyFactor

-- NFC 객체로부터 수집 된 인자 값을 집합

-- 전자서명생성정보 암호화 시 사용되는 Binary Password

KeyFactor := OCTET STRING

-- NFC 객체로부터 수집 된 인자 값(예 : ATS 값)

스마트폰 앱에서 NFC 객체를 이용한 간편 공인인증 과정의 예시이다.

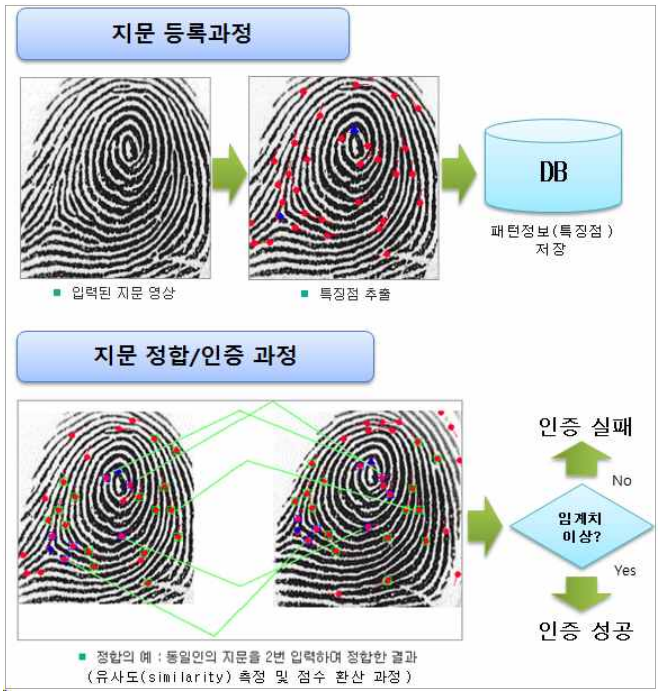
<NFC 객체를 전자서명생성정보 암호로 등록하는 과정>

<등록된 NFC 객체를 태그하여 전자서명생성정보 패스워드 입력 없이 공인인증 로그인 과정>

2 생체

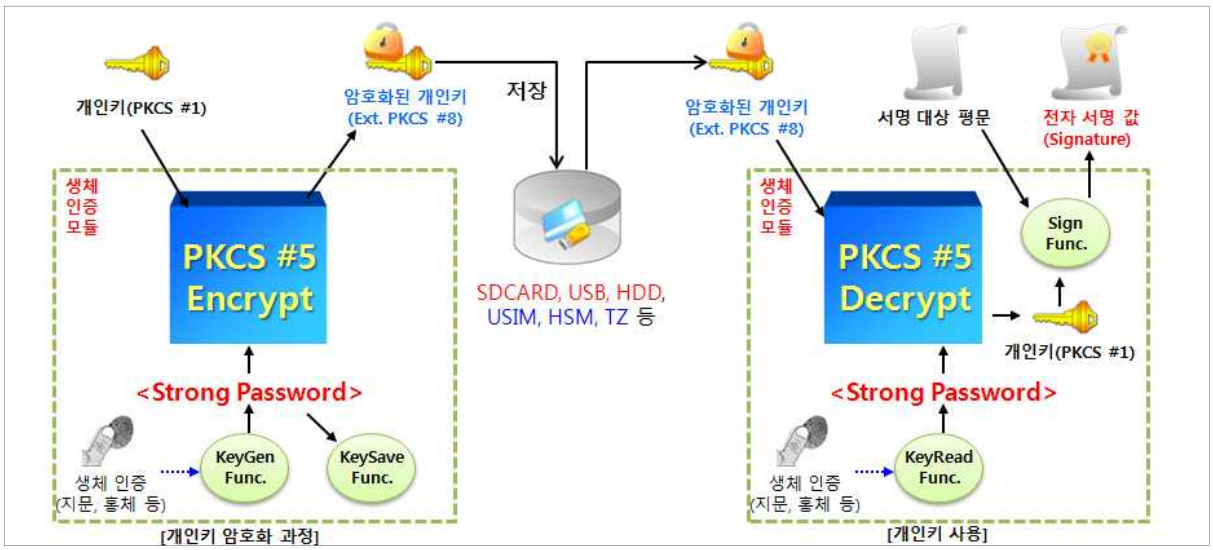
2.1 생체기반 전자서명생성정보 암호화 방법

생체인증의 경우, 지식기반 인증이나 소유기반 인증과는 달리 인증 시 입력되는 데이터가 매번 같지 않아 입력된 생체 데이터와 등록된 생체 데이터의 유사도(similarity)를 점수로 계산하여 설정된 임계치 이상인 경우 인증 성공으로, 그렇지 않은 경우 인증 실패로 판단한다. 그렇기 때문에 입력 시 매번 달라지는 생체 데이터로는 전자서명생성정보 암호화에 필요한 유일한 비밀키를 생성할 수가 없다. 그러므로 생체 인증 모듈은 내부적으로 생체 인증 정보(예 : 특징점, 패턴 등)를 등록하는 과정 중 안전한 난수 생성기에 의해 비밀키를 생성하여, 생체 인증 정보와 쌍으로 저장해야한다. 또한 비밀키 생성 과정 중 수집된 생체 패턴 정보를 난수 생성기의 씨드(Seed) 로 참여시킬 수 있다.



<일반적인 지문 등록 및 지문 인증 과정>

인증과정에서 등록된 생체 패턴 정보에 부합하는 생체입력정보가 들어왔을 경우, 쌍으로 보관되어 있던 비밀키를 리턴하거나 해당 비밀키를 이용해 전자서명생성정보를 암/복호화 할 수 있는 API를 제공해야 한다.



<생체 인증 후 전자서명생성정보 암호 생성, 이용 과정>

위 그림에서 KeyGenFunc API는 전자서명생성정보 암호화에 사용되는 강력한 비밀키(String Password)를 생성하는 역할을 수행한다. 비밀키 생성 과정에서 저장된 생체 패턴을 보안 난수 생성기(예 : DRBG)의 Seed 로 참여시킬 수 있다.

여기서 생성된 비밀키(Strong Password)는 전자서명생성정보 암/복호화에 사용되며, 암호화 방법은 PKCS #5, #8 스펙에 명시된 방법을 사용한다.

KeySaveFunc 는 생체 인증 정보 등록과정에서 생성된 비밀키를 저장하는 역할을 수행한다.

※ 암호키와 생체 패턴 정보의 저장소

암호키와 생체 패턴 정보는 접근이 강력하게 통제된 저장 공간에 저장할 것을 권고한다.(예 : SE, TEE 등)

KeyReadFunc 는 생체 인증 성공 후 생체 인증의 쌍이 되는 비밀키를 리턴하는 역할을 수행한다.

SignFunc 는 복호화된 전자서명생성정보를 이용해 전자서명(예 : RSA-PSS)을 수행하고, 전자서명 값을 리턴한다.

3. ASN.1 인코딩 예제

실제 암호화 완료 후 완성된 ANS.1 DER 인코딩 결과의 예시는 다음과 같다.

o NFC 객체를 이용한 전자서명생성정보 암호화(EncryptedPrivateKeyInfo)

```
SEQUENCE {
    SEQUENCE {
      SEQUENCE {
        OBJECT IDENTIFIER nfcObjectBasedEncryption(1.2.410.200004.1.30),
        SEQUENCE {
          SEQUENCE {
            OCTET STRING C6 4E 7C F8 F7 55 13 CC
              INTEGER 2048
            },
            UTF8String    "MyBusCard",
            SEQUENCE {
                OBJECT IDENTIFIER 1.2.410.200004.7.1,
                OBJECT IDENTIFIER 1.2.410.200004.7.3,
                OBJECT IDENTIFIER 1.2.410.200004.7.6
            }
        }
    }
      OCTET STRING
        F1 A2 16 3B AF 27 26 CB 52 B6 BF 05 CB 6F 62 A8
        F0 FE 8B BD 2F 8D C3 39 FA 33 05 87 2C 05 87 F2
        D5 D5 89 8F F0 46 3A 69 5E 0E 3B F7 BF F0 F3 CE
        84 A9 F6 D0 C6 A8 6C 6F 31 C4 95 60 B8 A1 A1 56
        E6 51 65 FA 70 22 D0 84 A0 19 57 34 5B 88 BD 2C
```

```
        23 C3 10 CF FF B2 F2 C1 3A 5F DB AB 95 4E DC 13
        01 99 F7 04 EF ED DE B7 3A 14 03 46 35 73 03 6A
        F1 86 A8 BB 17 92 71 D8 EF DE FE 5C 03 60 F4 5A
                [ Another 1136 bytes skipped ]
  }
}
```

o 지문 인증을 통해 생성된 암호키로 전자서명생성정보 암호화

```
SEQUENCE {
    SEQUENCE {
      SEQUENCE {
        OBJECT IDENTIFIER fingerprintObjectBasedEncryption(1.2.410.200004.1.40),
        SEQUENCE {
          SEQUENCE {
              OCTET STRING 54 7E 2C 4E 74 71 DB A5,
              INTEGER 2048
            },
            UTF8String    "오른쪽 엄지 지문",
        }
    }
      OCTET STRING
      74 A3 26 12 81 4D 66 BF 32 02 CD B9 8C 1B 64 59
      4E 2C 1C A2 BD 6F DE 55 8D E6 67 22 80 D7 19 E5
      4C 44 D4 A2 E5 68 6B 9B 59 3E A3 4E 72 C2 C2 26
      39 5B 4B C8 E6 CD 85 9B 29 69 25 E3 ED E1 D8 11
      B0 C8 A3 11 87 27 F1 C2 75 49 2B E8 30 B0 1F 57
              [ Another 1136 bytes skipped ]
  }
}
```

부록 2. FIDO 인증기술과 공인인증서 연계 기술

1. 개 요

본 가이드라인은 FIDO 인증기술을 통해 등록된 바이오 정보(지문 정보 등)를 이용하여 비밀번호를 입력하지 않고 공인인증서를 이용할 수 있는 방법에 대해 기술한다.

본 가이드라인에서는 트러스트존에 공인인증서를 안전하게 저장 및 이용하는 서비스 모델과 앱 내 저장소에 공인인증서를 안전하게 저장 및 이용하는 서비스 모델에 대하여 기술한다.

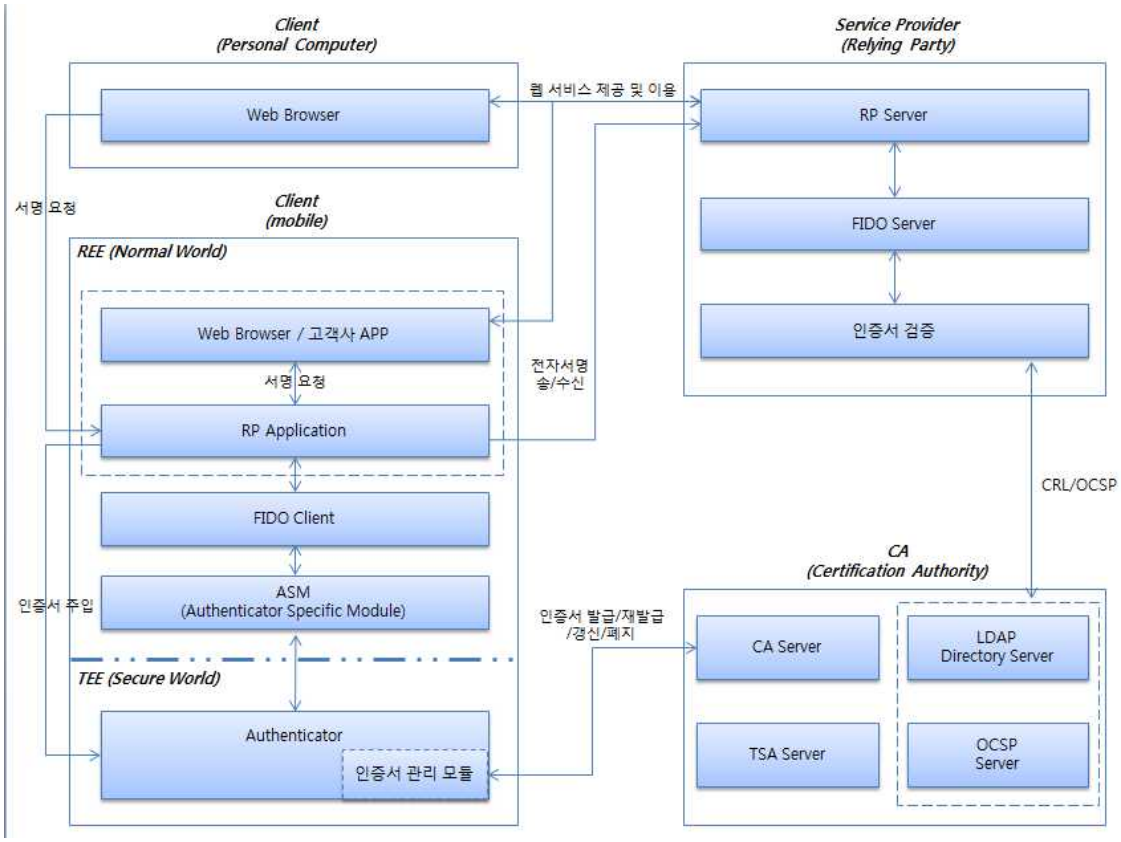
본 가이드라인은 트러스트존을 이용한 서비스 모델을 권장하며, 트러스트존을 이용할 수 없는 경우에는 앱 내 저장소를 이용한 서비스 모델을 구현 및 서비스 할 수 있다.

본 가이드라인에서 사용되는 FIDO 인증기술은 FIDO 1.0 UAF 규격을 이용하며, FIDO 인증기술과 공인인증서 발급 및 이용 기술을 연계하는 방법에 대해 언급하고, FIDO 인증기술에 대해서는 본 가이드라인에서 언급하지 않는다.

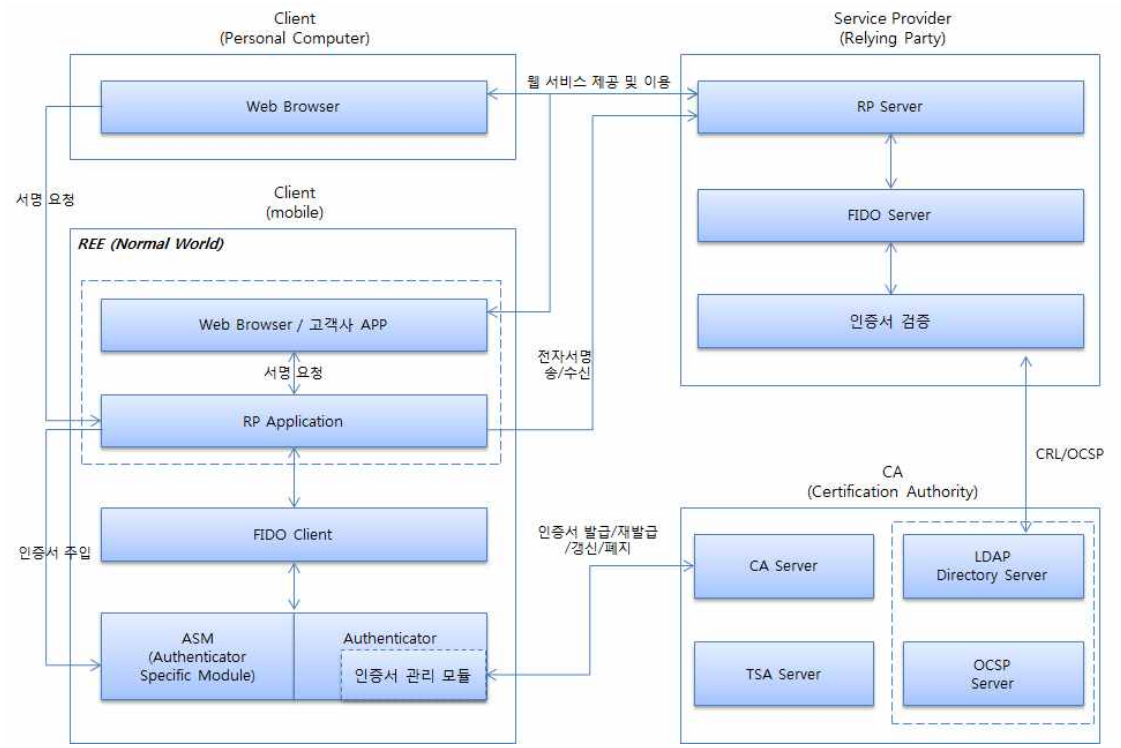
2. FIDO 인증 기술을 이용한 공인인증서 이용 서비스 모델

2.1 시스템 구성 및 운영

FIDO 인증 기술을 이용하여 공인인증서를 이용하기 위해서는, FIDO 1.0 UAF 기반의 시스템이 구성되어야 한다.



<트러스트존(TEE) 기반 전체 시스템 구성도>



<일반적인 실행환경(REE)에서 앱 내 저장소 이용 기반 전체 시스템 구성도>

FIDO 구성 요소 중, Client 내에서 동작하는 FIDO ASM/Authenticator 는 Client 내 트러스트존에서 동작하게 되며, 트러스트존 환경을 이용할 수 없는 경우

에는 일반적인 실행환경(REE)을 통해 클라이언트 구성요소들이 실행될 수 있다.

FIDO Authenticator 는 공인인증서를 발급 및 관리할 수 있는 기능이 포함되어야 하며, 참조번호, 인가코드 등 사용자가 RPC에서 입력한 정보를 FIDO Authenticator 에 안전하게 전달 가능 하여야 한다.

FIDO Server는 Client 로부터 전달받은 공인인증서를 검증하는 기능이 포함되어야 하며, 공인인증서의 상태 확인을 위해 CA(Certification Authority)의 인증서 생성/관리 시스템이나 디렉토리 서버(LDAP) 혹은 OCSP 서버와 연결되어야 한다.

2.2 공인인증서 전자서명생성정보 저장 방안

공인인증서와 전자서명생성정보는 FIDO Authenticator 만 접근 가능한 안전한 저장 공간에 저장되어야 하며, 어떠한 경우에도 허가된 응용 프로그램(RPC) 이외의 다른 RPC가 Authenticator APP 내 저장소에 접근하지 못하도록 해야 한다.

트러스트존(TEE) 환경에서 동작하는 FIDO Authenticator 의 경우 TEE 내 FIDO Authenticator 전용 저장 공간에 공인인증서와 전자서명생성정보가 저장된다.
TEE 환경을 이용할 수 없는 경우에는 REE 환경에서 동작하는 FIDO Authenticator 만 접근 가능한 안전한 저장 공간(USIM, 금융IC카드, MicroSD, APP내 저장소)에 공인인증서와 전자서명생성정보를 저장할 수 있다.

이 때, 전자서명생성정보는 FIDO Authenticator에서 바이오 인증을 통해 획득할 수 있는 바이오 정보 Hash 값 혹은 바이오 정보를 대표할 수 있는 유일값을 Password 로 하여 PKCS #5, PKCS#8 로 암호화 되어야 한다.

만약 바이오 인증을 통해 획득할 수 있는 바이오 정보 Hash 값 혹은 바이오 정보를 대표할 수 있는 유일값을 획득할 수 없는 단말기의 경우, 사용자 소유의 단말기 내 타인의 바이오 정보가 저장되어서는 안된다는 사항에 대하여 고지하여야 한다.

전자서명생성정보를 암호화 하는 방법은 다음과 같다.

(1) PKCS#5에서 정의한 PBES2 암호화 기법을 이용한다. 이 때, PBKDF2 키 생성 함수와 블록 암호화 알고리즘을 사용한다.

(2) PBKDF2 키 생성 함수를 사용 시 파라미터는 다음과 같이 정의된다.

파라미터	설명
Password	바이오 정보 Hash 정보, 혹은 바이오 정보를 대표할 수 있는 유일값,
Salt	8바이트 Random
Iteration Count	2048

(3) PBKDF2 키 생성 함수를 이용하여 20바이트의 추출키를 유도하고, 처음 16바이트를 암호화 키로 정의하며, 나머지 4바이트를 SHA-1 으로 해쉬하여 생성된 20바이트 중 처음 16바이트를 초기 벡터(IV) 로 정의하고 PBES2 암호화 기법을 이용해 전자서명생성정보를 암호화 한다.

또한, Authenticator APP 내에서 바이오정보 Hash 값 혹은 바이오 정보를 대표할 수 있는 유일값은 바이오 정보가 인식된 결과로 획득한 시점부터 전자서명생성정보 비밀번호로 사용이 완료된 후 반드시 메모리 내에서 삭제해야 하며, 그 어떤 경우에도 안전하지 않은 저장 장치 혹은 응용 프로그램(RPC)에 저장되거나 외부에 노출 되어서는 안 된다.

2.3 FIDO 전자서명검증정보/전자서명생성정보 이용 방안

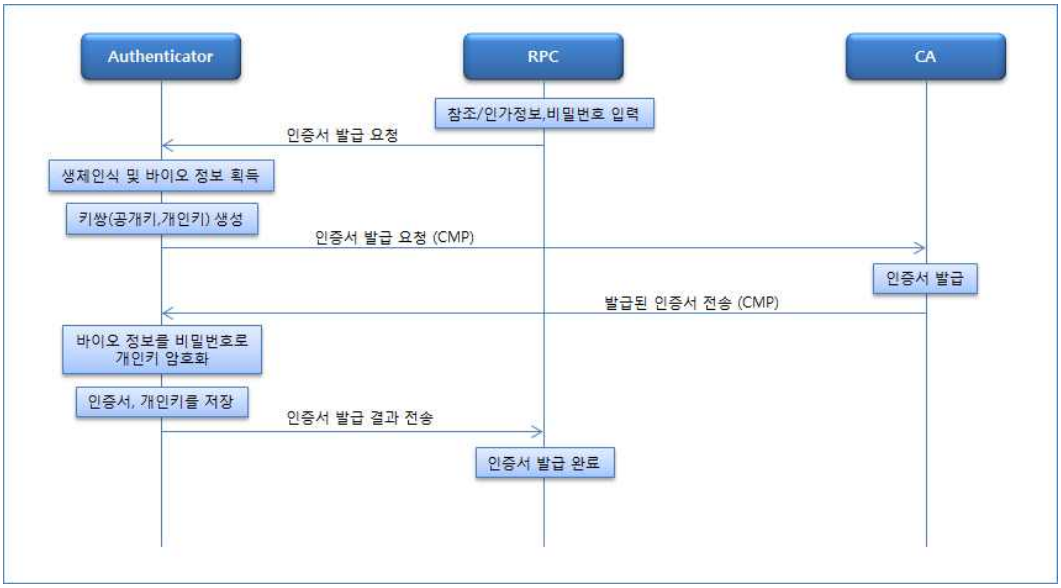
FIDO는 FIDO Authenticator 에서 바이오 인증을 수행한 후, FIDO 전자서명검증정보(UAuth.pub) 와 FIDO 전자서명생성정보(UAuth.priv)를 이용한 PKI 서명 기술을 이용하여 FIDO Server 와의 통신을 통해 인증을 수행한다.

본 가이드라인에서는 저장소에 저장되어 있는 공인인증서의 전자서명검증정보와 전자서명생성정보를 획득하여 각각 FIDO 전자서명검증정보와 FIDO 전자서명생성정보로 사용한다. 이를 통해 FIDO 전자서명검증정보/전자서명생성정보 기반의 PKI 서명을 통한 FIDO 인증 및 공인인증서를 이용한 전자서명으로 이용할 수 있다.

FIDO는 클라이언트에 등록된 바이오 정보들을 기반으로 인증하는 기술이므로, 클라이언트에 여러 사람의 바이오 정보가 등록되어 있을 경우, 공인인증서의 소유자가 아닌 다른 사용자의 바이오 정보를 이용하여 인증될 수 있다. 이를 위하여, FIDO 전자서명생성정보(i.e 공인인증서 전자서명생성정보) 저장 시 사용자가 선택한 바이오 정보의 Hash 값 혹은 바이오 정보를 대표하는 유일값을 비밀번호로 하여 암호화 하여야 한다.

2.3 이용 절차

2.3.1 공인인증서 발급



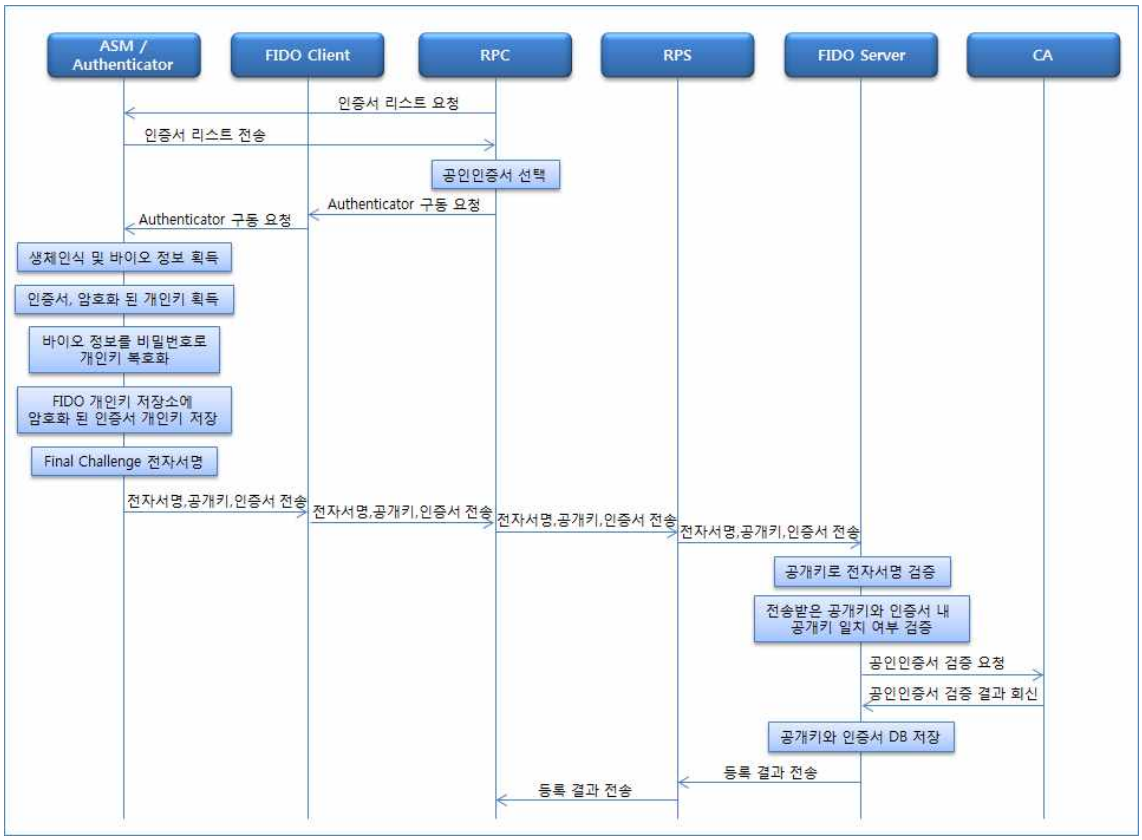
<인증서 발급 흐름도>

RPC 에서는 등록기관(RA) 으로부터 대면확인 후 부여 받은 참조번호 및 인가코드와 인증서 전자서명생성정보에서 사용할 비밀번호를 입력한 뒤 Authenticator 로 인증서 발급을 요청한다.

Authenticator 는 바이오 정보를 입력 받는 UI를 호출한 뒤 사용자가 입력한 바이오 정보를 인증하며, 인증이 성공할 경우 바이오 정보 Hash 값 혹은 바이오 정보 대표값을 획득하게 된다.

Authenticator 는 CA 에 접속하여 인증서 발급 요청 및 인증서를 발급 받게 되며, 바이오 정보 Hash 값 혹은 바이오 정보 대표값을 비밀번호로 전자서명생성정보를 암호화 한 뒤 Authenticator 만 접근 가능한 저장 공간에 인증서와 바이오 정보로 암호화 된 전자서명생성정보를 저장하고, 발급 결과를 RPC 로 전달한다. 이때, Authenticator 와 CA 는 RFC 4210(2510) 에 명시된 CMP 프로토콜을 준용하여 인증서 발급 프로세스를 수행해야 한다.

2.3.2 공인인증서 등록



<공인인증서 등록 흐름도>

RFC는 공인인증서 등록을 위하여 Authenticator에게 Authenticator 만 접근 가능한 저장 공간에 저장된 인증서에 대한 리스트를 요청하고, 획득한 인증서를 화면에 보여준다.

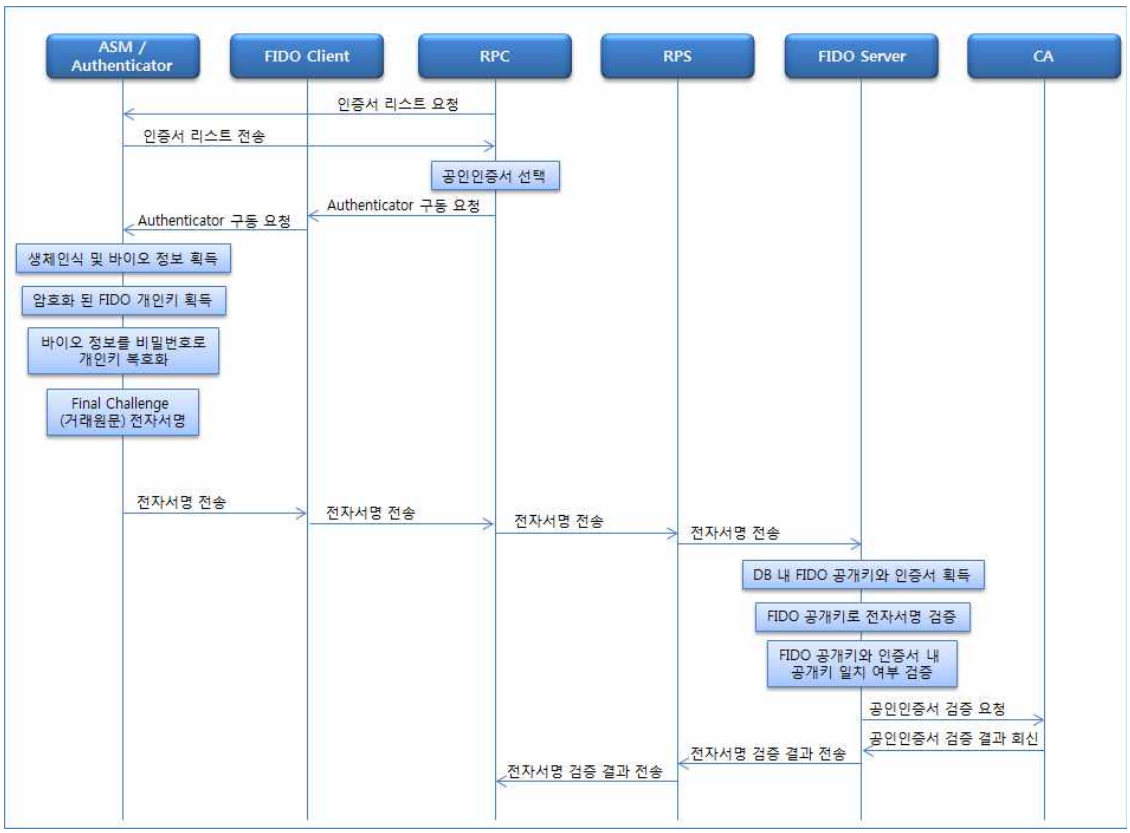
복수개의 인증서들이 저장 공간에 저장되어 있는 경우, 편의성을 고려하여 저장 공간에 저장된 인증서 리스트의 index를 같이 Authenticator 로 전달할 수 있다. 만약, 저장 공간에 저장된 인증서가 1개만 존재할 경우, 인증서 리스트를 화면에 보여주는 단계를 생략할 수 있다.

RPC 에서는 사용자가 인증서를 선택하면 FIDO Registration 프로토콜을 이용하여 사용자 등록을 진행한다. FIDO Registration 과정 중, FIDO Client에서 ASM/Authenticator를 실행하게 되고, Authenticator 에서는 바이오 정보를 입력받는 UI를 호출한 뒤 사용자가 입력한 바이오 정보를 인증하며, 인증이 성공할 경우, 바이오 정보 Hash 값 혹은 바이오 정보 대표값을 획득하게 되고, 사용자가 선택한 인증서의 바이오 정보로 암호화 된 전자서명생성정보를 복호화 한다. 전자서명생성정보 복호화가 성공하면, 공인인증서 내 전자서명검증정보를 획득하여 FIDO 전자서명검증정보(UAuth.pub)로 설정하고, 바이오 정보로 암호화 된 전자서명생성정보는 Authenticator 만 접근 가능한 FIDO 전자서명생성정보 저장소

(UAuth.priv)에 저장한다. 이후, FIDO 전자서명생성정보를 이용하여 FIDO Final Challenge 값 등을 전자서명(Signature) 한 뒤 FIDO 전자서명검증정보와 공인인증서를 함께 RPS 로 전달한다. 이 때 공인인증서는 FIDO Registration 메시지 내 확장 태그를 이용하여 공인인증서를 추가한다.

RPS 는 전달받은 전자서명값과 공인인증서를 FIDO Server 로 전달하고, FIDO Server 에서는 FIDO 전자서명검증정보를 획득하여 전자서명을 검증한다. 전자서명검증이 성공할 경우, FIDO 전자서명검증정보와 공인인증서 내 전자서명검증정보가 동일한지를 확인하고, 동일한 경우 FIDO Server DB 내 FIDO 전자서명검증정보와 공인인증서를 저장한 뒤 등록 결과를 RPS 에 전달하게 되며, RPS 는 RPC 에 등록 결과를 전달한다.

2.3.3 전자서명



<공인인증서 기반 전자서명 흐름도>

FIDO는 사용자 인증을 위한 Authentication 프로세스와 거래 내역 등 Transaction 에 대한 확인을 위한 Transaction Confirmation 프로토콜을 제공하며, 프로세스 내부적으로 PKI 기반의 전자서명을 수행한다.

RPC는 전자서명을 위하여 Authenticator에게 Authenticator 만 접근 가능한 저장 공간에 저장된 인증서에 대한 리스트를 요청하고, 획득한 인증서를 화면에 보여준다.

복수개의 인증서들이 저장 공간에 저장되어 있는 경우, 편의성을 고려하여 저장 공간에 저장된 인증서 리스트의 index를 같이 Authenticator 로 전달할 수 있으며, 만약 저장 공간에 저장된 인증서가 1개만 존재할 경우, 인증서 리스트를 화면에 보여주는 단계를 생략할 수 있다.

RPC 에서는 사용자가 인증서를 선택하면 FIDO Authentication 프로토콜(사용자 인증) 혹은 Transaction Confirmation 프로토콜(거래내역 등 각종 트랜잭션 정보 인증)을 이용하여 바이오 정보 인증 후 PKI 기반의 전자서명 생성 및 검증을 진행한다.

FIDO Authentication 혹은 Transaction Confirmation 과정 중, FIDO Client 에서 ASM/Authenticator를 실행하게 되고, Authenticator 에서는 바이오 정보를 입력 받는 UI를 호출한 뒤 사용자가 입력한 바이오 정보를 인증하며, 인증이 성공할 경우, 바이오 정보 Hash 값 혹은 바이오 정보 대표값을 획득하게 된다. 이후, Authenticator 내 저장되어 있는 바이오 정보 Hash 값 혹은 바이오 정보 대표값으로 암호화 되어 있는 FIDO 전자서명생성정보(UAuth.priv)를 획득 및 복호화 한 뒤 FIDO Server 로 전달할 정보를 FIDO 전자서명생성정보로 암호화 하여 전자서명(Signature) 한다.
이 때, FIDO Server 로 전달할 정보는 FIDO Authentication 의 경우 Final Challenge 정보가 포함된 정보이며, FIDO Transaction Confirmation 의 경우, Final Challenge 및 거래 원문이 포함된 정보이다. 생성된 전자서명 정보는 RPC와 RPS를 거쳐 FIDO Server 로 전송된다.

FIDO Server 에서는, DB 에 저장된 FIDO 전자서명검증정보(UAuth.pub)와 공인인증서를 획득한 뒤, 전달받은 전자서명 정보를 복호화 하여 전자서명 검증한다. 전자서명 검증이 성공한 경우, FIDO 전자서명검증정보와 공인인증서 내 전자서명검증정보 정보가 동일한지를 확인하며, 동일한 경우, CA에 접속하여 공인인증서를 검증한다. 이후, FIDO Server 전자서명 검증 결과를 RPS에 전송하며, RPS는 RPC에 검증 결과를 전송한다.

3. 메시지 규격

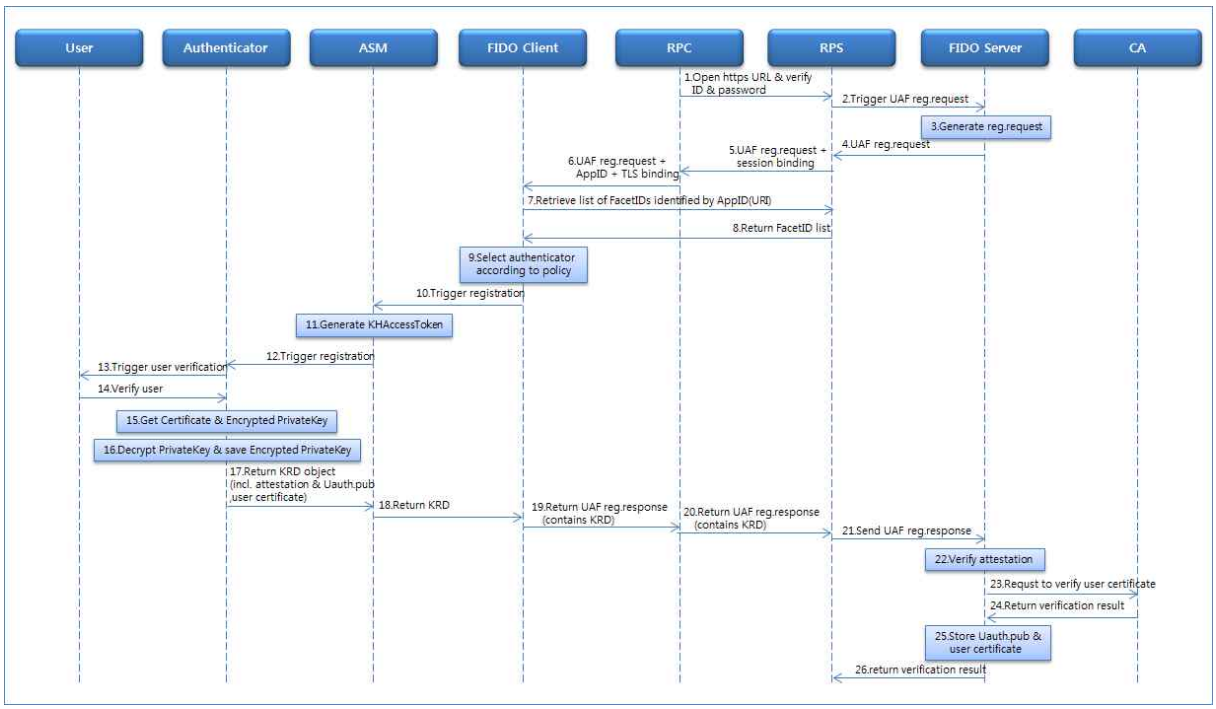
3.1 구성 및 범위

FIDO 인증기술과 공인인증서 이용 기술에 대한 연계 기술 서비스에서 사용되는 메시지들을 도출하고 각 메시지에 대한 표준을 정의한다. 서비스에서 사용되는 메시지들은 FIDO 1.0 UAF 규격 내 명시된 FIDO Registration, FIDO Authentication (FIDO Transaction Confirmation) 단계에서 각각의 생성 주체에 의해 생성되고 전달된다.

본 가이드라인에서는, FIDO 1.0 UAF 메시지 중 FIDO 인증기술과 공인인증서 이용 기술에 대한 연계와 관련된 메시지만을 기술하며, 그 외 FIDO 1.0 UAF 메시지와 동일한 내용은 언급하지 않는다.

3.2 FIDO Registration

FIDO 인증기술과 공인인증서 이용 기술에 대한 연계 기술 서비스에서는 공인인증서 등록 시 FIDO Registration 프로세스를 이용한다.



<공인인증서 등록 기능이 포함된 FIDO Registration 프로토콜>

RPC에서 공인인증서를 선택한 뒤 선택한 공인인증서의 인덱스 정보를 FIDO OperationHeader 의 Extension 필드에 추가하여 ASM 으로 전달할 수 있으며, Authenticator 에 저장되어 있는 공인인증서가 1개일 경우, 공인인증서의 인덱스 정보를 생략할 수 있다.

[FIDO OperationHeader]

```
dictionary OperationHeader {
    required Version    upv;
    required Operation op;
    DOMString           appID;
    DOMString           serverData;
    Extension[]         exts;
        -- exts[1] : 공인인증서 인덱스 정보 (Optional)
};
```

ASM에서는 전달 받은 OperationHeader 내 공인인증서 인덱스 정보를 Authenticator로 전송 시 메시지 규격인 Register Command 내 확장 태그를 이용하여 공인인증서 인덱스 정보를 추가할 수 있으며, 전달 받은 OperationHeader 내 공인인증서의 인덱스 정보가 없을 경우, Register Command 내 공인인증서 인덱스 정보를 생략할 수 있다.

FIDO Msg. No	TLV Structure	설명	비고
1	UINT16 TAG	TAG_UAFV1_REGISTER_CMD	
(중략)			
1.9	UINT16 TAG	TAG_EXTENSION (0x3E11)	메시지 추가
1.9.1	UINT16 TAG	TAG_EXTENSION_ID (0x3E13)	
1.9.1.1	UINT16 Length	BIOHSM_CERT_IDX 길이	
1.9.1.2	UINT8[] DATA	BIOHSM_CERT_IDX	
1.9.2	UINT16 TAG	TAG_EXTENSION_DATA (0x3E14)	
1.9.2.1	UINT16 Length	공인인증서 인덱스 정보 길이	
1.9.2.2	UINT8 User Certificate Index	공인인증서 인덱스 정보	

Authenticator에서 UAF reg.response 에 포함되는 Assertion 메시지에 전자서명검증정보 (UAuth.pub) 정보가 포함되고 공인인증서 정보가 추가되며, 구조는 아래와 같다.

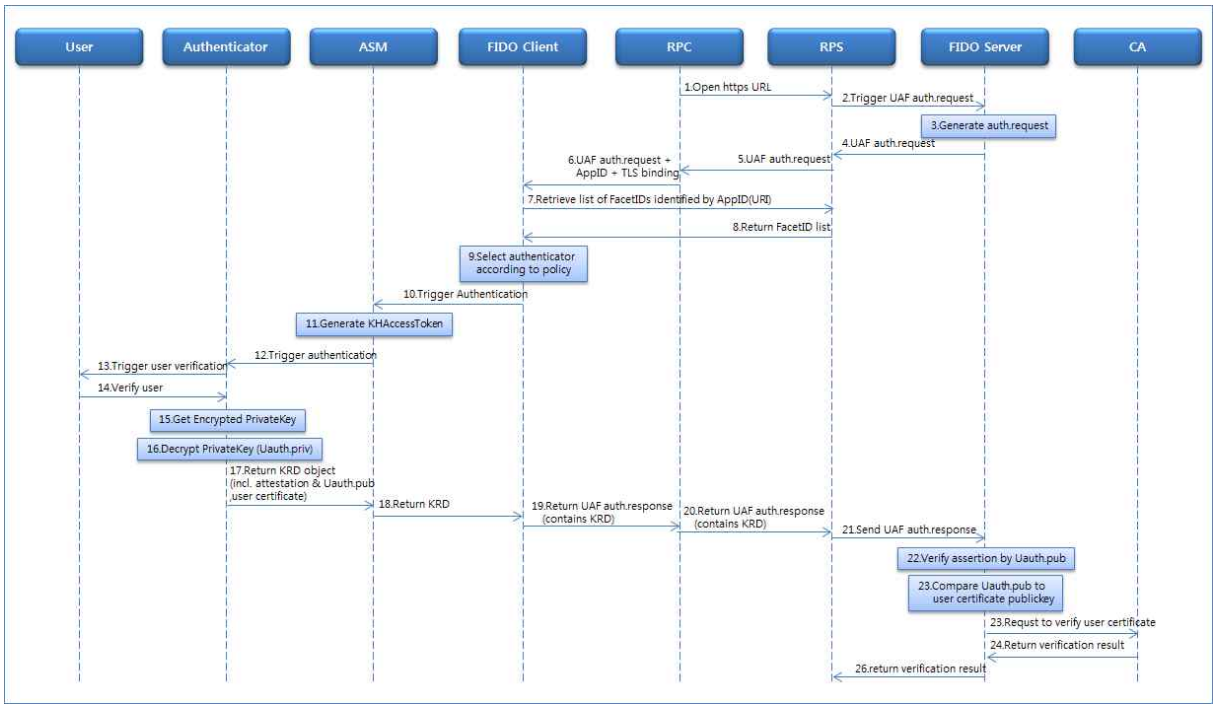
FIDO Msg. No	TLV Structure	설명	비고
1	UINT16 TAG	TAG_UAFV1_REG_ASSERTION	
(중략)			
1.2.7	UINT16 TAG	TAG_PUB_KEY	
1.2.7.1	UINT16 Length	FIDO 전자서명검증정보 (UAuth.pub) 길이	공인인증서 전자서명검증정보 정보
1.2.7.2	UINT8 PublicKey	FIDO 전자서명검증정보 (UAuth.pub) 정보	
1.2.8	UINT16 TAG	TAG_EXTENSION (0x3E11)	메시지 추가
1.2.8.1	UINT16 TAG	TAG_EXTENSION_ID (0x3E13)	
1.2.8.1.1	UINT16 Length	BIOHSM_CERTIFICATE 길이	
1.2.8.1.2	UINT8[] DATA	BIOHSM_CERTIFICATE	
1.2.8.2	UINT16 TAG	TAG_EXTENSION_DATA (0x3E14)	
1.2.8.2.1	UINT16 Length	공인인증서 길이	
1.2.8.2.2	UINT8 User Certificate	공인인증서 정보	
(이하 생략)			

상기 기술한 내용 이외의 나머지 메시지들은 FIDO 1.0 UAF의 Registration 프로세스에서 정의한 메시지와 동일하다.

3.3 FIDO Authentication (FIDO Transaction Confirmation)

FIDO 인증기술과 공인인증서 이용 기술에 대한 연계 기술 서비스에서는 공인인증서 기반의 전자서명 시 FIDO Authentication (FIDO Transaction Confirmation) 프로세스를 이용한다.

RPC에서 공인인증서를 선택한 뒤 선택한 공인인증서의 인덱스 정보를 FIDO OperationHeader 의 Extension 필드에 추가하여 ASM 으로 전달할 수 있으며, Authenticator 에 저장되어 있는 공인인증서가 1개일 경우, 공인인증서의 인덱스 정보를 생략할 수 있다.



<공인인증서 기반 전자서명 기능이 포함된 FIDO Authentication 프로토콜>

[FIDO OperationHeader]

```
dictionary OperationHeader {
    required Version    upv;
    required Operation  op;
    DOMString           appID;
    DOMString           serverData;
    Extension[]         exts;
        -- exts[1] : 공인인증서 인덱스 정보 (Optional)
};
```

ASM에서는 전달 받은 OperationHeader 내 공인인증서 인덱스 정보를 Authenticator로 전송 시 메시지 규격인 Sign Command 내 확장 태그를 이용하여 공인인증서 인덱스 정보를 추가할 수 있으며, 전달 받은 OperationHeader 내 공인인증서의 인덱스 정보가 없을 경우, Sign Command 내 공인인증서 인덱스 정보를 생략할 수 있다.

FIDO Msg. No	TLV Structure	설명	비고
1	UINT16 TAG	TAG_UAFV1_SIGN_CMD	
(중략)			
1.9	UINT16 TAG	TAG_EXTENSION (0x3E11)	메시지 추가
1.9.1	UINT16 TAG	TAG_EXTENSION_ID (0x3E13)	
1.9.1.1	UINT16 Length	BIOHSM_CERT_IDX 길이	
1.9.1.2	UINT8[] DATA	BIOHSM_CERT_IDX	
1.9.2	UINT16 TAG	TAG_EXTENSION_DATA (0x3E14)	
1.9.2.1	UINT16 Length	공인인증서 인덱스 정보 길이	
1.9.2.2	UINT8 User Certificate Index	공인인증서 인덱스 정보	

상기 기술한 내용 이외의 나머지 메시지들은 모두 FIDO 1.0 UAF의 FIDO Authentication (FIDO Transaction Confirmation) 프로세스에서 정의한 메시지와 동일하다.

4. 프로토콜 규격

4.1 개요

FIDO 인증기술과 공인인증서 이용 기술에 대한 연계 기술 서비스의 활성화를 위해서는 서로 다른 서비스 주체들의 시스템 간 통신이 가능하여야 하며, 이를 위해서는 표준화된 프로토콜 규격이 필수적이다. 본 가이드라인에서는 이러한 프로토콜 규격에 대해서 기술한다.

4.2 구성 및 범위

본 가이드라인에서는 시스템 간의 메시지 전송 시 프로토콜에 대해 기술한다. 본 가이드라인에서는, 공인인증서 이용기술과 FIDO UAF 1.0 기술을 연계하는 부분에 대해서만 기술하며, 그 밖의 FIDO 관련 프로토콜에 대해서는 규정하지 않는다.

4.3 전송 프로토콜

FIDO 인증기술과 공인인증서 이용 기술에 대한 연계 기술 서비스에서 사용되는 전송 프로토콜은 아래와 같은 기본적인 요구사항을 만족해야 한다.

- 국제 표준 프로토콜을 사용하며, 글로벌 인프라와의 연동이 가능해야 한다.
- 인터넷 기반에서 범용적으로 사용되는 프로토콜을 수용해야 한다.
- 기능에 있어서 확장이 가능한 형태이어야 한다.

본 프로토콜은 상기의 모든 요구사항을 수용할 수 있는 인터넷 관련 표준 프로토콜 규격을 준수하도록 한다.

4.4 보안 프로토콜

인터넷에서의 대표적인 보안 프로토콜은 아래와 같이 정의할 수 있다.

- 네트워크(IP) 레벨 : IPSec
- 트랜스포트(TCP) 레벨 : SSL/TLS
- 애플리케이션 레벨 : SSH, Kerberos 등

이러한 구분을 바탕으로 보안 프로토콜을 적용함에 있어서, FIDO 인증기술과 공인인증서 이용 기술에 대한 연계 기술 프로토콜은 애플리케이션 레벨의 보안, 즉 메시지 보안을 위해 다음과 같은 표준 규격을 사용하도록 한다.

- RPC 와 RPS 구간 : FIDO 1.0 UAF 프로토콜
- RPS 와 FIDO Server 구간 : FIDO 1.0 UAF 프로토콜
- FIDO Server 와 CA 구간 : CRL(RFC 3280), OCSP(RFC 2560)
- Authenticator 와 CA 구간 : CMP(RFC 4210(2510))

통신채널의 보안을 위해서 RPC 와 RPS 구간 간에 TLS v1.2 이상을 사용하여야 하며, RPS 와 FIDO Server 가 물리적으로 분리되어 있는 경우 RPS 와 FIDO Server 구간도 SSL/TLS를 사용하여야 한다.