웹 표준 기반 공인인증서비스 도입 및 구현 기술 안내서

2014. 9.

버전 1.0

KISA 한국인터넷진흥원
Korea Internet & Security Agency

목차

소개

공인인증서는 인터넷 등 온라인 환경에서 인증 및 전자서명 용도로 다양한 분야에 사용되고 있다. 하지만 공인인증서 발급과 이용을 위해 필요한 소프트웨어가 액티브X와 같은 웹 브라우저 플러그인으로 제공됨으로 인해 많은 이슈를 야기하게 되었다.

웹 브라우저 플러그인은 웹 브라우저의 기능을 확장시켜 주는 장점을 제공하였지만, 호환성 및 보안성을 저하시키는 주범으로 지목되었고 웹 브라우저 업체 또한 플러그인을 웹 표준으로 대체하려는 노력을 하고 있다.

이에 웹 브라우저 플러그인으로 인해 발생했던 공인인증서비스 호환성, 편의성 등 문제점을 개선하고자 웹 브라우저 플러그인을 사용하지 않고 웹 표준 구현기술만을 활용하는 공인인증서 발급 및 이용 기술 개발을 추진하게 되었다.

웹 표준 기반 공인인증서비스

웹 표준 기반 공인인증서비스란 웹 브라우저 자체의 확장기능을 배제하고 W3C가 제정한 웹 관련 표준 구현기술만을 활용하여 공인인증서를 이용할 수 있도록 구성한 서비스를 의미한다.

웹 표준으로 구현된 공인인증서 발급 및 이용 서비스에 대한 일반적인 모델은 [그림 1]과 같이 표현될 수 있다.

[그림 12] 웹 표준 기반 공인인증서비스 모델 예시

기존에 웹 브라우저 플러그인이 처리하던 공인인증서 관련 기능은 HTML5, 자바스크립트 등 웹 표준 구현기술로 구현되어 웹 브라우저가 담당하게 됨으로써 사용자는 웹 브라우저 접속만으로 공인인증서를 이용할 수 있도록 이용환경이 구성된다. 웹 표준으로 구현된 공인인증서비스는 각종 공인인증서 저장매체와의 연동을 통해 공인인증서의 안전한 저장 및 이용 편의성을 제공한다.

웹 표준 기반 공인인증서비스를 위해서는 공인인증기관은 공인인증서 발급을 별도 통신채널과 프로토콜을 사용하지 않고 웹 브라우저 만으로 발급할 수 있도록 시스템을 갖추고, 전자거래, 전자민원 등 이용기관은 발급·저장된 공인인증서를 이용하여 전자서명문을 생성할 수 있도록 웹 페이지 내에 기능을 구현하여야 한다. 또한 공인인증기관 및 보안업체는 안전한 공인인증서 저장을 위해 다양한 저장매체를 도입하고 웹 표준과 연동하여야 한다.

공인인증서 발급

기존에 웹브라우저 플러그인을 통해 발급하던 공인인증서를 웹 표준 구현기술 만으로 발급하기 위해서는 공인인증기관의 발급시스템과 프로토콜의 변경, 클라이언트인 웹브라우저에서의 발급기능 구현이 필요하다. 공인인증기관은 HTTP를 통해 발급할 수 있도록 발급방식을 변경하고, 웹브라우저에서는 전자서명키 쌍과 인증서 요청양식의 생성, 응답을 처리하여 저장하는 기능 등을 구현하여야 한다.

1. 서버 측면

공인인증기관은 웹 브라우저만으로 공인인증서를 발급하기 위해서는 발급시스템에 대한 변경이 필요하다. 공인인증서의 발급과 관리는 발급시스템과 클라이언트 간의 요청과 응답을 위한 메시지를 통해 이루어지며, 이 때 메시지는 특정 프로토콜이 아닌 HTTP(또는 HTTPS)를 통해 전송되어야 한다.

HTTP Transfer for the CMP

기존 공인인증서 발급시스템을 활용하고 웹 표준을 적용할 수 있는 방법은 'HTTP Transfer for the Certificate Management Protocol'를 준용하는 것이다[RFC6712]. 공인인증서 발급·관리와 관련한 모든 요청과 응답은 HTTP를 통해 PKI 메시지로 처리 가능하며 기존 발급시스템의 활용 또한 가능하다.

[RFC6712]를 준용하기 위해서, CMP(Certificate Management Protocol) 메시지는 반드시 HTTP를 사용하여야 하며 HTTP/1.0, HTTP/1.1을 모두 지원해야 한다. 또한 GET이 아닌 POST방식만을 사용하여 메시지를 전송하여야 한다. PKIMessage를 전송하는 동안 HTTP 헤더의 콘텐츠 타입으로 application/pkixcmp으로 설정하여야 한다.

DER 인코딩[ITU.X690.1994]된 PKIMessage[RFC4210]는 HTTP POST 요청의 본문에 담아 보내며 HTTP 요청이 성공하면 서버는 HTTP 응답에 CMP 응답메시지를 담아 보낸다. 이 경우 HTTP 응답코드는 200이어야 하며 요청 성공과 관련한 다른 응답코드(2XX)는 이 경우 사용해선 안된다.

이와 관련한 더 상세한 표준내용은 [RFC6712], [RFC4210] 등의 표준문서를 참고하면 된다.

CSR 등 발급방법

CMP를 HTTP를 통해 전송하는 방법 이외에도 SSL 인증서 발급 시 이용되는 CSR(Certificate Signing Request), HTML5 keygen 태그를 활용한 방법 등이 있다[PKCS10][Keygen].

2. 클라이언트 측면

웹브라우저에서 공인인증서 발급 기능을 구현하기 위해서는 전자서명 키쌍 생성, 전자서명 생성·검증, 개인키 암호화, 전자서명 대상 원문의 축약 등을 위해 암호화 알고리즘 구현과 인증서 요청 및 응답 메시지 처리 등을 위한 기능 구현이 필요하다.

자바스크립트 암호 알고리즘 구현

웹 표준 구현기술만으로 공인인증서 발급 및 이용이 가능하기 위해서는 자바스크립트를 사용하여 암호 라이브러리를 개발하거나 W3C에서 표준화하고 있는 웹 암호 API(Web Cryptography API)를 사용하여 개발하는 방법이 있다.

공인인증서 발급 및 이용에 필요한 암호 라이브러리 및 ASN.1 등을 자바스크립트만으로 구현한 다수의 오픈소스가 존재하며, 필요 시 다음과 같은 오픈소스를 참조할 수 있다.

- SJCL (Stanford Univ.): http://bitwiseshiftleft.github.io/sjcl/
- Crypto-js (Google): http://code.google.com/p/crypto-js/
- MSR JavaScript Cryptography Library (Microsoft) : http://research.microsoft.com/en-us/downloads/29f9385d-da4c-479a-b2ea-2a7bb335d727/
- Forge (Digital Bazaar): http://digitalbazaar.com/forge/
- PidCrypy (Versaneo): https://www.pidder.de/pidcrypt/

CMP 요청 및 응답 메시지 처리

공인인증서 발급을 위해서는 암호라이브러리 외에도 기존 TCP 방식으로 구현되어 있는 CMP를 HTTP 방식으로 전송하도록 구현해야한다. 아래는 웹 표준을 사용해서 구현한 CMP의 일부이다. CMP 요청을 위한 PKIMessage는 Base64 인코딩되어 POST 데이터로 서버에 전송된다. CMP 응답 또한 Base64 디코딩하여 메시지를 처리한다.

```
self.ir(refNum, authCode, function(asn) {
    if (asn instanceof cmp.error) {
        resultCallback(asn);
        return;
    }
    data = asn1.toDer(asm).getBytes();

    HTTPTransfer.send( self.url.irip, HTTPTransfer.pkiReq, data,
                       function(version, flags, messageType, result) {
        if (version instanceof cmp.error) {
            resultCallback(version);
            return;
        }
```

```
            try {
                self.decodeMessage(result);
            } catch (ex) {
                resultCallback(new cmp.error(ex));
                return;
            }

            if (!cmp.testIssue)
                doConf(resultCallback);
            else {
                resultCallback(self.certAndKey);                    }
    }); // end of send irip
}); // end of ir
```

웹 암호 API 활용

웹 암호 API의 경우 현재 표준규격 제정이 진행 중이며, 일부 최신 웹브라우저에는 관련 기능이 탑재되어 있어 사용이 가능하며 아래의 웹사이트에서 웹 암호 API에 대한 정보를 얻을 수 있다.

- 인터넷 익스플로러: http://msdn.microsoft.com/ko-kr/library/ie/dn265046(v=vs.85).aspx
- 크롬: https://goo.gl/ovgTRQ
  https://code.google.com/p/chromium/issues/detail?id=245025
- 파이어폭스: https://bugzilla.mozilla.org/show_bug.cgi?id=865789,
  https://nightly.mozilla.org/
- 사파리(웹킷): https://bugs.webkit.org/show_bug.cgi?id=122679,
  http://nightly.webkit.org/

아래와 같이 웹 암호 API를 사용하여 전자서명 키쌍을 생성할 수 있다.

```
var subtle = window.crypto.subtle;

var publicKey;
var privateKey;
var extractable = false;

var algorithmKeyGen = {
    name: "RSASSA-PKCS1-v1_5",
    modulusLength: 2048,
    publicExponent: new Uint8Array([0x01, 0x00, 0x01]),  // Equivalent to 65537
    hash: {
        name: "SHA-256",
    }
};
```

```
subtle.generateKey(algorithmKeyGen , extractable, ['sign']).then( function(e) {
      publicKey = new crypto.key(e.publicKey);
      privateKey = new crypto.key(e.privateKey);

      resultCallback ( {
            publicKey: publicKey,
            privateKey: privateKey
    } );
}, function(e) {
         resultCallback(new crypto.error(e));
} );
```

생성된 키쌍을 이용하여 전자서명을 수행하는 것은 다음과 같다.

```
var subtle = window.crypto.subtle;

var algorithmSign = {
   name: "RSASSA-PKCS1-v1_5"
};

plain = crypto.util.stringToArraybuffer(input);
Subtle.sign(algorithmSign, privkey, plain.buffer).then(
      function(e) {
            resultCallback(crypto.util.arrayBufferToString(e));
      }, function(e) {
            resultCallback(new cryoto.error(e));
      });
```

웹 암호 API에 관한 보다 상세한 내용은 W3C 페이지를 참고할 수 있다[WebCrypto].

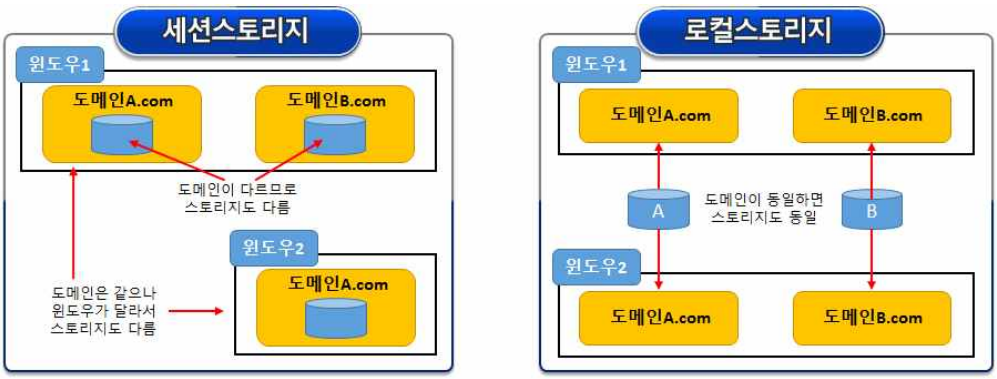
공인인증서 저장

웹 브라우저 플러그인 기반의 기존 환경에서는 주로 하드디스크와 이동식 디스크 내의 /NPKI/ 디렉토리 하에 공인인증서가 저장되었다. 하지만 웹 표준 환경에서는 파일시스템에 대한 접근이 자유롭지 않기 때문에 기존 방식으로 저장하는 것은 어렵다. 웹 표준 방식으로 공인인증서를 발급 받은 후 저장할 수 있는 저장매체는 웹브라우저 내의 웹 저장소, 보안토큰, 소프트토큰, 스마트인증 등이 있다.

[그림 13] 공인인증서 저장매체

1. 웹 스토리지(Web Storage)

HTML5의 웹 스토리지(Web Storage)는 HTML5 표준으로 웹사이트의 데이터를 웹브라우저에 저장할 수 있는 새로운 자료구조로 기존 쿠키(Cookie)의 단점을 개선하기 위해 개발되었다[WebStorage]. 현재 웹 스토리지는 모든 웹브라우저에서 지원 가능하며 저장된 데이터는 서버로 전송되지 않는다.



[그림 14] 웹 스토리지

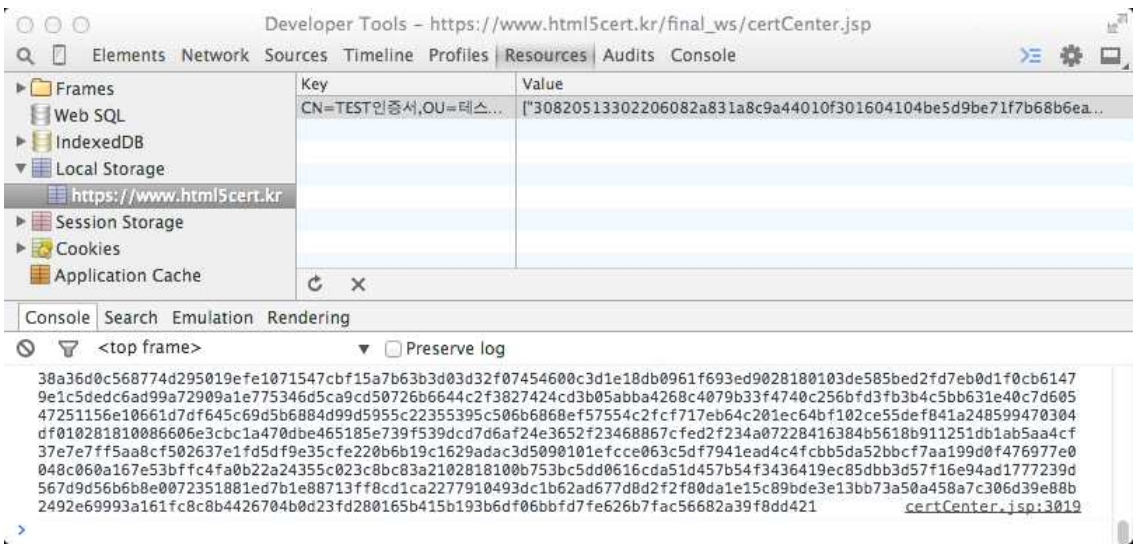
HTML5에서는 웹 스토리지로 두 개의 객체를 제공한다. 하나는 세션 단위로 데이터를 저장하는 세션 스토리지(Session Storage)와 만료 기간이 없는 로컬 스토리지(Local Storage)이다. 세션 스토리지는 데이터가 영구적으로 저장되지 않아 웹브라우저 종료 시 데이터가 삭제

되며 저장된 데이터에 대해서는 탭 브라우징, 새 창에서도 서로 별개로 취급한다. 로컬 스토리지는 명시적으로 지우지 않는 한 영구적 보관이 가능하다.

웹 스토리지는 동일출처정책(Same-Origin Policy, SOP)에 따라 해당 도메인(Origin)만이 접근 가능한 저장소이다. 이는 프로토콜, 호스트, 포트 중 하나라도 다르면 별도의 공간으로 인식한다는 것을 의미한다. 저장 가능한 데이터의 용량은 보통 Origin당 약 5MB까지 가능하며 5MB를 넘어설 경우, QUOTA_EXCEEDED_ERR 오류가 발생하고 추가 공간을 요구할 수 없다. 문자열(UTF-16)만 저장 가능하며 Object 타입을 저장하는 경우 toString()을 호출한 형태로 저장된다.

로컬 스토리지는 window 객체의 localStorage 컬렉션을 통해 저장 및 조회가 가능하다. 데이터는 key/value 쌍으로 구성되며 아래와 같이 setItem()과 getItem()으로 값을 저장하거나 가져올 수 있다. key와 value 모두 string으로 저장되며 setItem()은 기존 아이템이 있을 경우 덮어쓰고, getItem()으로 값을 찾지 못하는 경우 에러를 발생하지 않고 null을 리턴한다.

웹 스토리지는 클라이언트 디바이스에 직접 저장되고 네트워크로 전송되지 않기 때문에 네트워크 레벨에서는 안전할 수 있다. 웹 스토리지는 물리적으로는 파일로 저장되며, 사용자 계정 하위 디렉토리에 저장된다. 페이지가 로드되기 전에 웹브라우저는 물리적으로 저장되어 있는 데이터를 읽어 메모리에 올려둔다. 웹브라우저 개발자 도구 등을 통해 웹 스토리지에 저장된 데이터 접근이 가능하며, 수정 및 삭제가 가능하다.



[그림 15] 로컬 스토리지에 공인인증서 저장 예

공인인증서와 개인키를 웹 스토리지에 저장하고자 하는 경우, 세션 스토리지는 데이터가 영구적으로 저장되지 않아 사용이 어렵다. 로컬 스토리지의 경우에도 사용자가 웹브라우저 메뉴 등을 통해 삭제하거나 웹브라우저 개발도구 등을 통해 임의 수정이 가능하다. 따라서 웹 스토리지에 공인인증서를 저장하는 경우 추가적인 암호화, 사용자인증 등 보안수단을 강구하여야 한다. 또한 물리적으로는 사용자 계정 하위 디렉토리에 파일로 저장되며 웹 암호

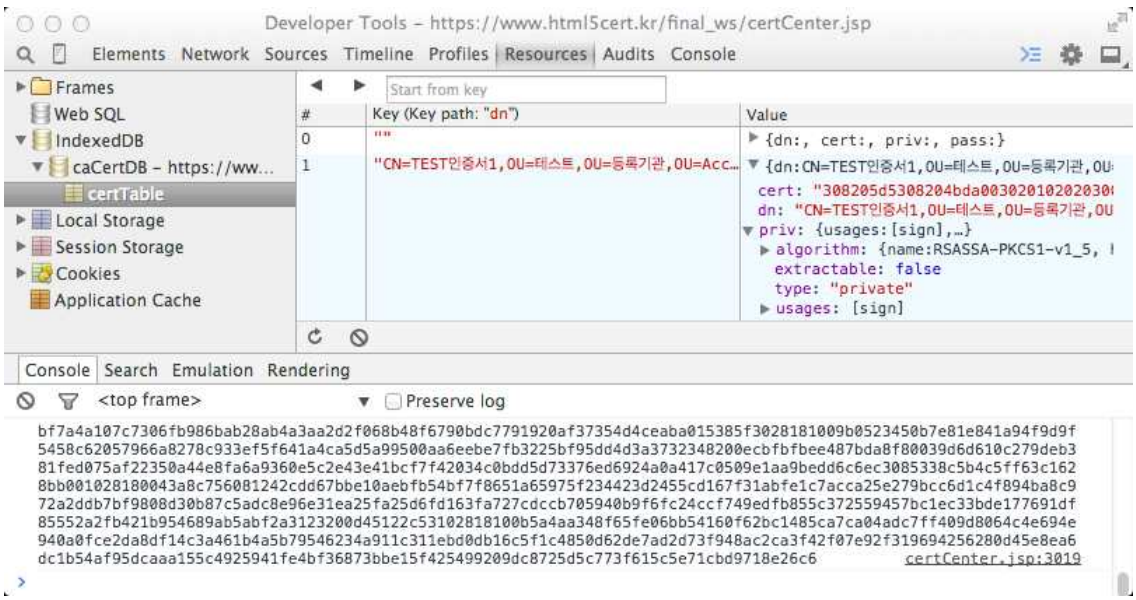
API로 생성된 Key 객체의 경우 저장되지 않아 웹 암호 API를 이용하는 경우 데이터 변환 등 주의가 필요하다.

2. 인덱싱된 데이터베이스(IndexedDB)

웹 스토리지가 용량이 작고 구조화된 데이터를 다루기엔 부족하기 때문에 웹브라우저에서 많은 양의 구조화된 데이터를 영구적으로 저장하여 검색 및 이용할 수 있도록 만든 HTML5 API가 인덱싱된 데이터베이스(IndexedDB)이다. 현재 IE 8, 9 버전 및 하위 버전은 지원되지 않는다.

IndexedDB는 기본적으로 자바스크립트 객체(Object)를 저장하며 기존 RDBMS(관계형 데이터베이스 관리 시스템)와는 다르게 Object Store를 만들고 자바스크립트를 통해 저장한다. SQL과는 개념이 비슷하나 쿼리 대신 Index를 사용하는 것이 차이점이다.

웹 스토리지와 마찬가지로 동일출처정책에 따라 해당 도메인만 접근 가능한 저장소이다. 개개의 데이터 항목의 크기나 데이터베이스 자체의 크기 제한을 두지 않지만, 브라우저는 저장용량 제한을 둘 수도 있다.



[그림 16] IndexedDB에 공인인증서 저장 예

웹 스토리지와 마찬가지로 클라이언트 디바이스에 직접 저장되고 네트워크로는 전송되지 않는다. IndexedDB 또한 웹브라우저 개발도구에서 저장된 데이터에 대한 열람 및 삭제가 가능하다. 따라서 웹 스토리지와 마찬가지로 안전한 저장을 위해서는 추가적인 암호화, 사용자인증 등 다양한 보안수단을 강구하여야 한다.

웹 암호 API를 이용하는 경우, 키 객체 저장이 가능한 IndexedDB를 저장소로 이용할 수 있다. IndexedDB에 웹 암호 API를 이용해 생성된 키 객체 등을 저장하는 경우 웹 스토리

지와는 달리 별도의 데이터 변환 등이 필요치 않다.

웹 브라우저 내에 공인인증서를 저장하는 경우 IndexedDB가 지원되는 경우에는 웹 암호 API 적용 등을 고려하여 웹 스토리지 보다는 IndexedDB를 우선적으로 사용할 것을 권고한다.

3. W3C 파일 API

공인인증서의 발급과 저장에 W3C 웹 표준의 파일 API를 사용할 수 있다[FileAPI]. 발급받은 공인인증서와 개인키를 [PKCS12]로 변환한 후 mime-type을 application/x-pkcs12로 설정하여 사용자 단말기로 다운로드 할 수 있으며, 사용자는 다운로드된 [PKCS12] 파일 .p12 또는 .pfx 파일을 웹브라우저에서 이용할 수 있다.

mime-type 설정을 이용한 다운로드 방식 이외에도 W3C File Writer API를 이용할 수 있으나, 현재 표준화가 진행 중에 있어 일부 웹브라우저에서만 한정적으로 지원되고 있다[FileWriter]. File Writer API를 이용하는 경우 사용자가 파일 다이얼로그 박스로 파일을 선택하는 것보다 웹브라우저를 이용하여 비교적 자유롭게 .p12, pfx 파일을 처리할 수 있다.

또한 웹브라우저의 드래그앤드롭 기능과 결합하여 사용자 단말기의 파일시스템에 [PKCS12]로 저장한 인증서를 웹브라우저에서 편리하게 이용할 수 있다[Drag&Drop]. 드래그앤드롭은 대부분의 웹브라우저에서 지원 가능하며, 이용페이지에서 전자서명이 요구되는 경우 [PKCS12] 파일을 사용자가 드래그앤드롭으로 이용페이지에 넣고 비밀번호를 입력함으로써 전자서명 처리가 가능해진다.

파일형태로 공인인증서와 개인키를 저장하여 이용하는 경우, 파일에 대한 관리를 전적으로 사용자가 함으로 인한 불편함 등이 발생할 수 있으며, 복사·유출·분실 등에 대한 위험이 여전히 존재한다.

4. 보안토큰(HSM)

보안토큰은 하드웨어 장치 내에서 공인인증서를 생성 및 저장하고 전자서명을 수행하는 USB 인터페이스를 갖춘 개인용 휴대 저장매체를 말한다.

보안토큰은 하드웨어 칩 기반으로 공인인증서 및 개인키를 보관하고 전자서명 등 암호 연상 기능을 수행함에 따라 전자서명 생성 요청에 대해 보안토큰은 내부의 개인키를 이용하여 전자서명을 생성하고 결과값만을 보안토큰 외부로 전달한다. 결과적으로 공인인증서와 관련한 모든 연산을 보안토큰 칩 내부에서 수행함으로써 개인키에 대한 유출위험이 없다.

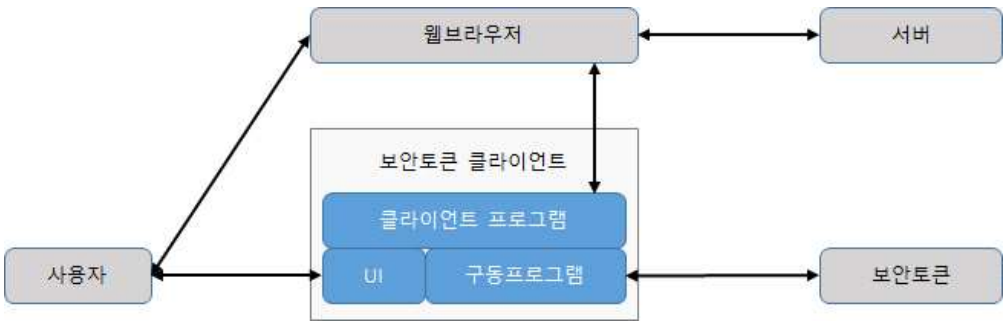
보안토큰을 이용하기 위해서는 표준 API를 사용하여야 한다[PKCS11]. 따라서 하드웨어 장치에 대한 드라이버 프로그램과 이용을 위한 표준 API의 설치가 필요하다. 또한 웹페이지 내 데이터에 대해 전자서명하기 위해서는 웹브라우저와의 연동이 필요하다.

현재 별도 소프트웨어 없이 웹브라우저 자체만으로 보안토큰 하드웨어 매체와 웹브라우저 간에 연동하는 것은 불가능하다. 이와 관련한 표준 제정 논의가 진행되고 있으나 현재로서는 별도의 소프트웨어 설치가 필요하며, 웹브라우저 플러그인을 설치하고 이용하거나 로컬서버로 작동하는 별도 소프트웨어를 설치하는 방법, 연동을 위한 별도의 외부서버를 활용하는 방법 등이 있다. 본 가이드라인에서는 플러그인, 외부서버 이용은 배제하고 로컬서버를 활용하는 방법에 대해 기술한다.

로컬서버 방식은 웹브라우저가 로컬PC에 설치된 소프트웨어에 접속하여 전자서명을 요청하고 응답을 처리하는 방식이다. 유럽에서 많이 사용되고 있는 e-ID에서도 이와 같은 방식으로 e-ID 카드리더기와 웹브라우저 간 연동을 처리한다[e-ID].

사용자 PC에 설치되어 웹브라우저와 연동되는 보안토큰 클라이언트 소프트웨어는 단일 소프트웨어로 구현될 수 있으며, 다음과 같이 편의상 여러 개의 모듈로 나뉠 수 있다.

- 보안토큰 클라이언트 : 단일 소프트웨어 또는 모듈로 구현된 보안토큰을 이용하기 위해 필요한 온전한 형태의 클라이언트 프로그램을 칭함
- 클라이언트 프로그램 : 보안토큰 클라이언트의 부분 또는 모듈로써 웹브라우저와의 통신을 담당하는 부분을 칭함
- 클라이언트 구동프로그램 : 보안토큰 하드웨어 매체와의 연결을 담당하는 부분을 칭함
- 사용자 인터페이스(UI) : PIN 입력 등 사용자에게 필요한 인터페이스 제공



[그림 17] 보안토큰과 웹브라우저 간 연동 모델

웹브라우저는 보안토큰 클라이언트에 HTTP(또는 HTTPS) 접속 후 JSON 타입의 명령어를 전달함으로써 보안토큰을 이용한 공인인증서비스를 제공하게 된다. 로컬서버와 통신하기 위한 포트는 8443포트를 사용하며, XMLHTTPRequest(이하 XHR) 등을 활용한다.

```
target.contentWindow.postMessage(JSON-Message, https://127.0.0.1:8443);
```

웹브라우저와 보안토큰 클라이언트 간의 통신 메시지 형식은 부록1에 기술되어 있다.

5. 소프트토큰(Software HSM)

공인인증서를 안전하게 저장·이용하는 보안토큰과 같은 물리적 하드웨어 매체가 작동하는 방식을 소프트웨어로 구현한 것이 소프트웨어 방식의 보안토큰 즉, 소프트토큰이다.

소프트토큰을 웹브라우저와 연동하기 위해서는 보안토큰과 동일한 연동방식을 이용할 수 있으며, 별도의 연동방식을 이용할 수도 있다.

6. 스마트인증(Mobile HSM)

스마트인증은 스마트폰과 같은 모바일 기기의 USIM, eSE 등 보안모듈을 모바일 통신을 통해 PC와 같은 타 기기에 연결하여 전자서명생성키 등 비밀정보를 안전하게 저장 및 보관하고 키 생성 및 전자서명 생성 등을 처리하는 하드웨어와 소프트웨어를 총칭하는 것이다.

기존 스마트인증 서비스는 보안토큰 인터페이스와 동일하게 만들고 웹브라우저 플러그인을 통해 서비스가 되었으나, 플러그인 없이 동작하기 위해서는 별도의 연동방법이 필요하다. HTML5 WebRTC를 활용한 방법, 외부의 서버를 통한 방법 등 다양한 연동방법이 가능하다 [WebRTC].

스마트인증의 일반적인 절차는 다음과 같다.

1. 사용자는 웹페이지에서 스마트인증을 통해 전자서명을 요청한다.
2. 서버 또는 웹페이지에서 스마트앱으로 연결을 요청하고 서명하고자 하는 데이터를 전송한다.
3. 사용자는 스마트앱을 통해 전자서명을 생성하여 제출한다.

[그림 18] 스마트인증 이용 절차 예시

공인인증서 이용

웹브라우저를 통해 발급된 인증서를 이용하기 위해서는 각 저장매체에 저장된 인증서 목록을 조회하고 해당 인증서를 이용하여 전자서명문을 생성하는 기능의 구현이 필요하다. 하지만 웹의 보안정책, 동일출처정책으로 인해 브라우저 내의 웹스토리지에 저장된 인증서를 이용하는 경우 제약이 발생한다[RFC6454]. 이에 본 가이드라인에서는 웹스토리지에 공인인증서를 발급·저장한 경우 동일출처(Same-Origin)과 교차출처(Cross-Origin) 환경에서 공인인증서를 이용하는 방법을 기술한다.

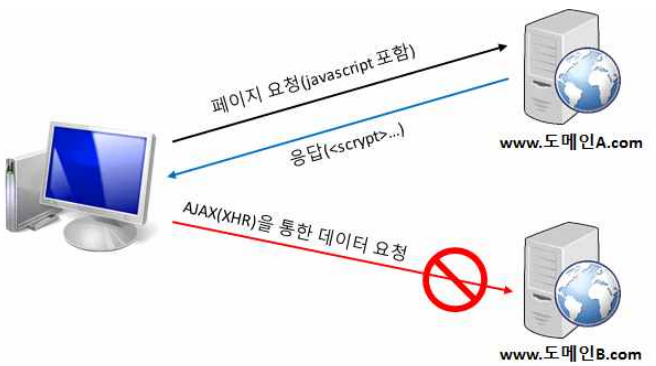
1. 동일출처 환경에서의 공인인증서 이용

동일출처정책은 한 출처(Origin)에서 로드된 문서나 스크립트가 다른 출처 자원과 상호작용을 하지 못하도록 제한하는 보안정책이다. 두 웹페이지의 프로토콜, 호스트, 포트가 동일하면 동일출처(Same-Origin)이며, 하나만 달라져도 다른 출처로 처리된다.

공인인증서 저장매체로 웹 스토리지를 이용하는 경우는 동일출처정책 영향을 받게 되며, 이로 인해 공인인증서를 발급한 웹사이트(이하 발급사이트)와 공인인증서를 이용하는 웹사이트(이하 이용사이트)가 상이한 경우 발급된 공인인증서를 조회·이용하지 못하는 제약사항이 발생한다.

따라서 동일출처 환경으로만 공인인증서 발급과 이용을 제한하거나, 동일출처 환경에서 발급된 공인인증서를 타 웹사이트에서 이용할 수 있는 기능을 제공할 수 있다. 이 경우 웹스토리지에 발급된 인증서를 [PKCS12] 형식의 파일로 내보내고 타 웹사이트에서 해당 파일을 불러와 사용하는 등 다양한 방안이 이용될 수 있다.

공인인증서의 발급과 저장이 [PKCS12]로 이루어지고, 이용사이트에서 [Drag&Drop]으로 공인인증서가 이용되는 경우, 동일출처 및 교차출처 여부와 관계없이 이용이 가능하나 공인인증서의 저장 및 관리에 대해 사용자의 주의가 필요하다.



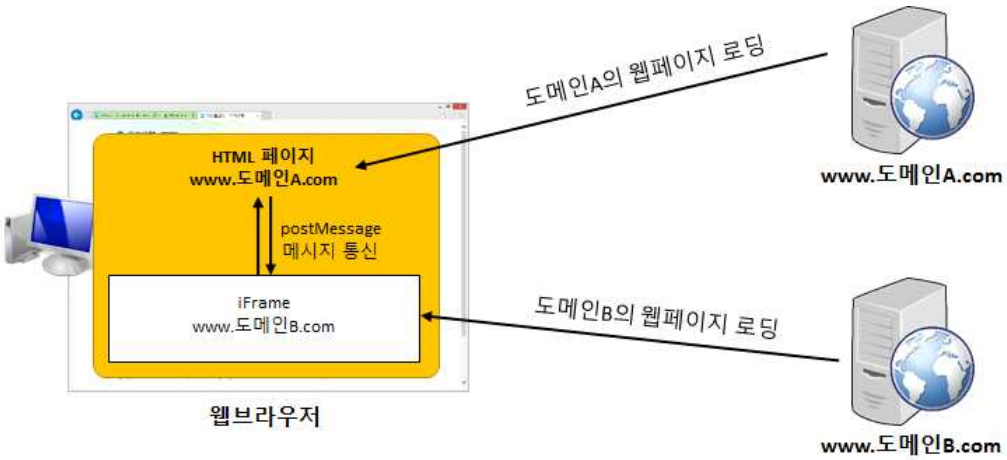
[그림 19] Same-Origin 환경에서의 공인인증서 이용

2. 교차출처 환경에서의 공인인증서 이용

웹 메시징(Web Messaging)

HTML5의 웹 메시징은 기존 HTML에 없던 메시지 통신방식으로 HTML5에서 새로 추가되었다[WebMessage]. 이는 일반적인 Windows API에서 사용하던 메시지 이벤트를 이용하는 방식과 거의 동일한 형태이다. 웹 메시징을 이용하는 일반적인 방식은 다음과 같다.

1. postMessage()를 이용하여 메시지를 목표의 메시지큐에 넣는다.
2. dispatch를 통하여 메시지큐에서 메시지를 뽑아서 목표에 전달한다.
3. onmessage 이벤트 핸들러에서 메시지를 받아서 처리한다.



[그림 20] Web Messaging 메시지 통신 기능

웹 메시징은 cross-document messaging과 channel-messaging 두가지 방식이 있으며, postMessage 메소드의 정의는 다음과 같다.

```
windows.postMessage(message, targetOrigin, [ports]);
• message :     송신 메시지
• targetOrigin: 목적지 도메인, 메시지 수신 도메인 지정, 특정 도메인이 아닌 경우 '*' 지정
• ports:        메시지 포트(생략 가능)
```

웹 메시징을 활용한 공인인증서 이용

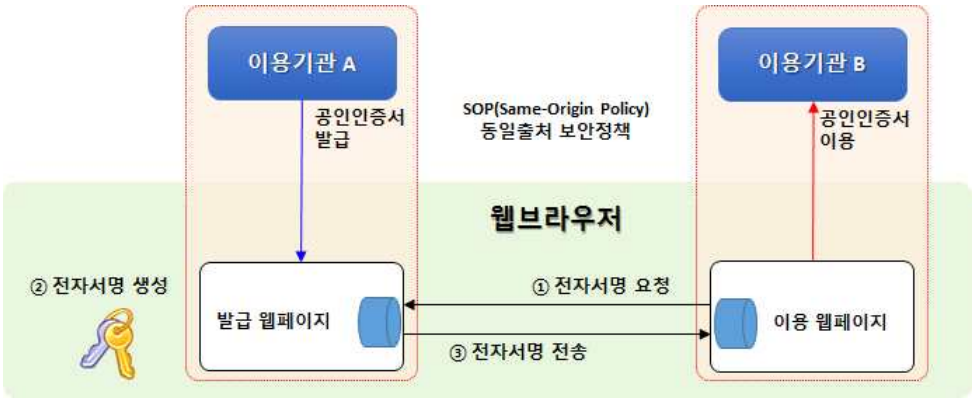
발급사이트와 이용사이트가 다른 교차출처 환경 하의 이용사이트에서 발급사이트의 웹 스토리지에 발급된 공인인증서를 이용하기 위해서 Web Messaging 표준이 활용될 수 있다. 발급사이트와 이용사이트 간에 postMessage 통신을 통해 발급사이트에서 생성된 전자서명문을 이용사이트에 전송함으로써 공인인증서를 이용할 수 있다.

[그림 21]과 같이 이용기관A는 발급 페이지로부터 공인인증서를 발급받아 저장하고 있고, 이용기관B는 이용페이지만 있는 경우, 웹 메시징을 활용한 공인인증서 이용방법은 다음과

같다.

1. 이용기관B는 이용기관A에 인증서 목록을 요청한다.
2. 이용기관A는 정당한 요청 여부를 검증한 후 이용기관B에 인증서 목록을 전송한다.
3. 사용자는 이용기관B에서 사용할 인증서를 선택한 후 비밀번호를 입력한다.
4. 이용기관B는 이용기관A에 선택한 인증서, 비밀번호와 서명원문을 전송한다.
5. 이용기관B는 전자서명을 수행하고 전자서명문을 이용기관B로 전송한다.
6. 이용기관B는 서버로 전자서명문을 전송한다.

위의 이용절차에서 이용기관A와 이용기관B간의 통신은 모두 웹브라우저 내에서 웹 메시징을 통해서 이루어지며, 외부 서버 등으로 전송되지 않는다.



[그림 21] 공인인증서 상호연동

웹 서비스의 보호

웹 표준으로 구현되는 공인인증서 발급 및 이용 기능은 웹 취약점에 대한 보호, 자바스크립트 소스에 대한 보호 등 콘텐츠 보호기능을 필요로 한다. 또한 공인인증서를 웹브라우저 내에 저장하는 경우 피싱, 파밍 등에 대한 대비가 필요하다. 이와 같은 보안위협에 대처하기 위해서 아래와 같은 보호수단을 적용할 수 있다.

1. 자바스크립트 샌드박스

샌드박스는 외부로부터 들어온 프로그램이 보호된 영역에서 동작해 시스템이 부정하게 조작되는 것을 막는 보안 형태이며 웹브라우저에서 자바스크립트를 실행할 때도 역시 샌드박스 환경에서 실행된다.

최근의 웹 사이트들은 자신의 웹페이지에 트위터, 페이스북 등이 제공하는 콘텐츠를 iframe을 이용하여 통합 제공하고 있다. 이러한 사용자경험은 웹사이트 보안에 문제점을 야기할 수 있다.

iframe 태그의 sandbox 속성은 iframe 내에 삽입된 페이지에 다음과 같은 추가적인 제한을 걸 수 있다.

- 실제 존재하는 않는 도메인(Origin)에 속한 것으로 간주
- 플러그인 실행 금지
- 자바스크립트 사용 금지
- 폼 요소에 의한 페이지 호출 금지

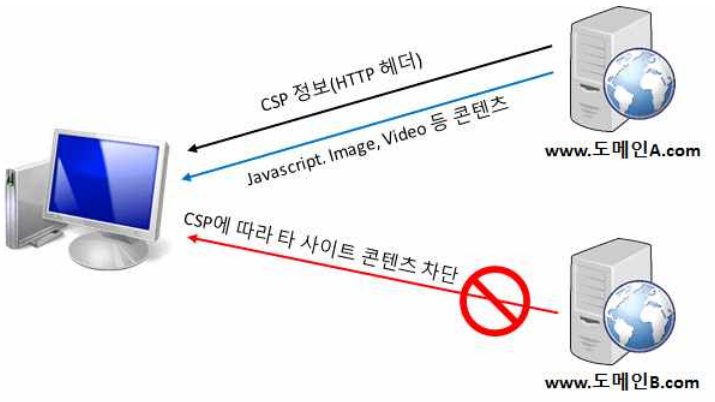
이러한 샌드박스 제한은 너무 엄격한 것이기 때문에 allow-script(자바스크립트 허용), allow-same-origin(동일 도메인에 소속된 리소스 이용) 등의 속성값 설정을 통해 샌드박스 제한을 완화할 수 있다.

2. 웹 콘텐츠 보호(W3C CSP)

웹 콘텐츠의 변경을 통한 악의적인 공격행위에 대응하기 위해서는 기본 SSL을 통한 채널 암호화 이외에도 다음 항목에 대한 대응이 필요하다.

- 악의적인 HTML 태그 삽입(iframe, Object, Embed 등)
- 악의적인 Layer 삽입
- Script에 대한 Cross Domain Include를 악용한 공격
- 하이퍼링크에 대한 위변조
- 사용자 입력으로 변경되지 않는 Hidden 속성의 Input value 에 대한 변경

HTML5 표준의 CSP(Content Security Policy)는 XSS(Cross Site Scripting)와 데이터 끼워넣기(Data Injection) 같은 공격을 감지하거나 경감시키기 위해서 보안층(Security Layer)을 추가한 것이다. 이런 류의 공격은 정보를 갈취하거나 악성코드를 유포시키려는 등의 모든 목적을 위해 이용된다.



[그림 22] 콘텐츠 보안 정책(CSP)

CSP는 하위호환성에 중점을 둬서 디자인되어 이것을 지원하는 서버에 지원하지 않는 브라우저가 잘 작동되고 그 반대도 가능합니다. CSP를 지원하지 않는 브라우저는 CSP를 무시하고, 단순히 웹 콘텐츠에 대한 표준 동일출처정책을 기본값으로 해서 평소대로 작동합니다. 웹 사이트가 CSP 헤더를 제공하지 않으면 마찬가지로 브라우저들은 동일출처정책 표준을 사용합니다.

CSP는 웹서버에 Content-Security-Policy HTTP를 설정하는 것만큼 쉽게 설정할 수 있다. Content Security Policy을 설정하는 것은 어떤 정책을 실행하게 할 것인지를 결정하는 것을 포함하고 이런 정책 실행 부분을 Content-Security-Policy 헤더를 사용해서 정책을 실행하게 설정하는 것이다.

3. 자바스크립트 코딩

자바스크립트가 웹브라우저 캐쉬에 저장되지 않도록 HTTP 헤더에 캐쉬를 방지할 수 있도록 설정하여야 한다. 이를 위해 HTTP 1.1에서 지원되는 Cache-Control 헤더, HTTP 1.0의 Pragma: No-Cache 헤더를 설정하면 된다.

```
<HTML><HEAD>
<META HTTP-EQUIV="Pragma" CONTENT="no-cache">
<META HTTP-EQUIV="Expires" CONTENT="-1">
</HEAD><BODY>
</BODY>
</HTML>
```

자바스크립트는 접근 수정자(private, public, protected)를 제공하지 않기 때문에, 기본적으로 모든 객체의 속성(변수, 메서드)은 public이다. 변수나 메소드를 외부에서 접근할 수 없는 private으로 만들기 위해서 클로져(Closure)가 사용하여 유사하게 구현할 수 있다.

자바스크립트 코드를 최적화하기 위해 다음과 같은 도구들을 사용할 수 있다.

- Google Closure : https://developers.google.com/closure/
- YUI Compressor : http://yui.github.io/yuicompressor/
- UglifyJS : https://github.com/mishoo/UglifyJS
- JSLine : http://jslint.com/
- Packer : http://dean.edwards.name/packer/

또한 중요 데이터를 담은 변수를 더 이상 사용하지 않는 경우, 즉시 값을 초기화하여 악용되지 않도록 하여야 한다. 자바스크립트는 변수에 대한 명시적인 타입이 없고, 변수에 값을 할당함으로써 초기화 한다. 함수 내의 지역변수는 함수가 끝날 때 변수에 null값 할당, return 값이 있는 함수에서는 finally에서 해당 변수에 null값 할당 등을 통해 변수에 대한 메모리 해제가 가능하다.

4. 자바스크립트 동적 로딩

근래의 웹페이지들은 Ajax(Asynchronous Javascript and XML)이 사용자에게 편리한 UI를 제공함에 따라 더욱 많은 기능들을 Ajax으로 적용하고 있는 추세이다. 이에 따라 하나의 페이지를 구성하는데 필요한 자바스크립트 모듈의 개수도 증가하게 되었고, 자바스크립트 모듈 개수와 크기의 증가는 웹브라우저가 웹페이지를 로딩 하는데 걸리는 시간이 지연되는 문제를 발생시킨다. 또한 해당 페이지에서는 전혀 사용되지 않는데도 불구하고 모든 자바스크립트를 로딩하는 문제도 있다.

이러한 문제를 해결하기 위한 방법이 자바스크립트 파일을 동적으로 로딩하여 필요한 경우에만 사용하도록 할 수 있다.

동적으로 자바스크립트를 로딩하는 방법 중 하나는 script 태그를 자바스크립트 코드 내에서 직접 생성하는 것으로 script 태그를 생성하고 src에 로딩할 주소를 넣음으로서 로딩하게 된다.

script 태그를 사용하지 않고 자바스크립트를 로딩하는 방법으로 XHR을 사용하는 방법이 있다. XHR을 이용하여 서버에 파일을 요청한 후 요청한 파일을 응답으로 받은 다음 이를 eval() 함수를 사용하여 로딩하는 것이다.

이외에도 iframe이나 document.write()를 이용하는 등 자바스크립트를 동적으로 로딩하기 위한 다양한 방법들이 존재한다.

5. 자바스크립트 난독화(Obfuscation)

자바스크립트는 일반적으로 클라이언트 측 언어로 사용되기 때문에 소스코드가 공개되는 단점을 가지고 있다. 개발된 소스코드를 도용당할 문제 및 숨기고자 하는 로직이나 알고리즘이 그대로 드러나는 문제 등이 있다.

난독화(Obfuscation)는 프로그래밍 언어로 작성된 코드에 대해 읽기 어렵게 만드는 작업이다. 프로그램에 사용된 아이디어나 알고리즘 등을 숨기고자 하는 경우 사용되며, 프로그램의 일부 또는 전체를 변경한다. 난독화는 코드의 가독성을 낮추어 역공학(Reverse Engineering)에 대비하는 것으로 무단복제 등을 방지할 수 있다. 또한 난독화는 보안장비를 우회하려는 악성코드에서도 많이 사용되고 있는 추세이다.

난독화 기법으로는 변수명 등 문자열을 다른 문자열로 대체하여 사람이 해석하기 어렵게 하는 방법, 소스코드를 잘게 잘라 변수에 저장했다가 재조합하는 방법, eval() 함수, XOR 8bit ASCII 인코딩을 이용하는 방법 등 다양한 방법이 있으며, 나름의 장단점을 가진다. 일부 난독화 기법은 소스를 해독하기 어렵게 하는 기능 이외에도 웹브라우저 개발도구에서 디버깅이 어렵게 하거나, 코드의 유효기간 설정, Injection 공격 등에 대한 자체 대응기능 등 다양한 기능을 포함하고 있으므로 이러한 기능을 적절히 활용함으로써 자바스크립트 소스를 보호할 수 있다.

자바스크립트 코드 난독화는 소스코드 보호를 위한 최소한의 요구사항이며 소스코드 보호를 위한 다양한 보호수단과 함께 사용되어야 한다.

6. 자바스크립트 암호화(Javascript Encryption)

자바스크립트 이외에 자바스크립트 소스를 보호하기 위한 방법으로 소스코드 암호화를 고려할 수 있다. 자바스크립트 코드를 암호화하기 위해서는 웹브라우저에서 암호화를 위한 키를 생성하고 서버와 키를 안전하게 공유함으로써 자바스크립트 코드를 암호화할 수 있다.

이와 관련한 일반적인 절차는 아래와 같다.

- 웹브라우저에서 자바스크립트를 암호화하기 위한 암호화 키 생성
- 서버의 공개키로 암호화 키를 암호화하여 전송
- 서버는 해당 자바스크립트 코드를 암호화하여 웹브라우저에 전송
- 웹브라우저에서는 암호화 키를 이용하여 자바스크립트 복호화 후 이용

하지만 자바스크립트 특성상 웹브라우저에서 정상적인 서비스 이용을 위해서는 반드시 복호화되어야 하고 복호화된 자바스크립트 코드, 암호화 키 등의 값은 웹브라우저 개발자 도구

등을 통해 노출될 수 있다. 따라서 자바스크립트 암호화는 일반적인 데이터 암호화와는 달리 웹서버-웹브라우저 간의 통신구간에서만 제한적으로 보안을 제공한다고 볼 수 있다.

암호 알고리즘은 AES, SEED 등 블록암호 알고리즘을 이용할 수 있다.

7. 자바스크립트 무결성 확인

자바스크립트는 웹브라우저에서 해석되어 실행되는 인터프리터 언어의 특징으로 인해 웹브라우저 개발도구 등에서 모든 소스코드가 노출되는 것이 외 DOM 객체에 대한 위변조, 자바스크립트의 변조 등이 가능하다. 이러한 자바스크립트 소스코드에 대한 위변조는 보안 문제로 이어지며, 이를 방지하기 위해서는 자바스크립트 실행 시 소스코드가 변조되지 않았는지 무결성을 검사해야 한다.

자바스크립트 무결성 검사 방식은 서버 측면 검사(Server-side Integrity Test)와 클라이언트 측면 검사(Client-side Integrity Test)로 나누어진다.

- **서버 측면 검사:** 서버에서 콘텐츠 배포 시 해당 콘텐츠에 대한 무결성 정보(hash 등)를 포함하여 배포하고, 클라이언트에서 다음 페이지 요청 시 또는 주기적으로 서버에 검증 요청을 하는 방식이다. 무결성 검증을 위한 정보가 서버에 있고, 서버가 최종 검사의 주체라는 점에서 상대적으로 보안이 우수하지만, 검증 요청에 대한 부가적인 통신이 필요하고, 동적인 페이지인 경우 콘텐츠 무결성 정보의 생성이 복잡해진다는 단점이 있다.
- **클라이언트 측면 검사:** 콘텐츠 배포 시 클라이언트에서 무결성 정보를 생성하고, 이를 실시간 또는 주기적으로 검사하는 방식이다. 무결성에 문제가 없는 경우 별도의 통신이 필요하지 않지만, 무결성을 검사하는 모듈이 클라이언트에 있기 때문에 해당 모듈에 대한 추가적인 보안 수단이 필요하다.

이외에 Web Crypto API 이용이 가능한 경우, localStorage 등 웹브라우저 내 저장소에 자바스크립트 소스 코드를 저장한 후 해쉬를 통해 무결성을 검증하는 방법이 W3C 표준화 과정에서 사용례로써 논의되고 있다.

맺음말

웹 표준 기반 공인인증서비스는 운영체제, 웹브라우저, 단말기에 상관없이 웹을 통해 공인인증서를 발급받아 이용하고자 하는 모든 서비스에 적용가능하다.

공인인증서 이용방식을 웹 표준으로 구현함으로써 사용자가 별도의 공인인증서 소프트웨어를 설치하지 않아도 되므로 이와 관련한 이용불편, 접근성 및 보안 문제 등이 해소될 것이다. 더 나아가서는 데스크톱PC, 스마트폰, 스마트TV 등 웹브라우징이 가능한 모든 환경에서 공인인증서를 이용할 수 있는 환경이 만들어지게 될 것으로 기대된다.

다만 웹을 통해 서비스되고 웹표준 구현기술인 자바스크립트를 이용함으로 인해 발생할 수 있는 웹취약점 관련한 보안문제에 대해서는 적절한 대비가 필요하다.

부록 1. 보안토큰 연동 메시지 규격

웹브라우저와 보안토큰 클라이언트간의 메시지 형식은 다음과 같다.

1. 토큰 목록 요청

- 요청 메시지

```
{
        messageNumber:0,
        sessionID:"string",
        operation:"GetTokenList"
}
```

예시) messageNumber:0,sessionID:"a1234",operation:"GetTokenList"

- 응답 메시지

```
{
        messageNumber:0,
        sessionID:"string",
        operation:"GetTokenList",
        resultCode:1234,
        resultMessage:"ok",
        list:[ {
                tokenIdentifier:number,
                tokenName:"string"
        }]
}
```

예시) messageNumber:0, sessionID:"a1234",operation:"GetTokenList", resultCode:0, resultMessage:"ok", { tokenIdentifier:"1234", tokenName:"kisatoken" }

2. 인증서 목록 요청

- 요청 메시지

```
{
        messageNumber:0,
        sessionID:"string",
        operation:"GetCertificateList"
}
```

예시) messageNumber:0,sessionID:"a1234",operation:"GetCertificateList"

- 응답 메시지

```
{
        messageNumber:0,
        sessionID:"string",
        operation:"GetCertificateList",
        resultCode:1234,
        resultMessage:"ok",
        list:[ {
                subject:"utf8 dn",
                issuer:"utf8 dn",
                serial:"hexa string",
                validFrom: "long time",
                validTo: "long time",
                certIdentifier: "number",
                keyIdentifier: "number"
        }]
}
```

예시) messageNumber:0, sessionID:"a1234",operation:"GetCertificateList", resultCode:0, resultMessage:"ok", { subject:"홍길동", issuer:"한국인증",serial:"12345", validFrom:"1234567", validTo:"9876543",certIdentifier:"1234",keyIdentifier: "3456" }

3. 인증서 획득

- 요청 메시지

```
{
        messageNumber:0,
        sessionID:"string",
        operation:"GetCertificate",
        certIdentifier: "number"
}
```

예시) messageNumber:0,sessionID:"a1234"operation:"GetCertificate",certIdentifier: "12345"

- 응답 메시지

```
{
        messageNumber:0,
        sessionID:"string",
        operation:"GetCertificate",
        resultCode:1234,
        resultMessage:"ok",
        certificate:"base64 string"
}
```

예시) messageNumber:0,sessionID:"a1234",operation:"GetCertificatet", resultCode:0, resultMessage:"ok", certificate: "MzA4MDEyMzRh ..... YmNkMTIzNA=="

4. 전자서명 생성

- 요청 메시지

```
{
        messageNumber:0,
        sessionID:"string",
        certIdentifier:"number",
        keyIdentifier:"base64 string",
        type:"pkcs#1_hash" or "pkcs#1_content" or "cms_hash" or "cms_content"
        toBeSigned:"base64 string"
}
```

예시) messageNumber:0,sessionID:"a1234", operation:"GenerateSignature, certIdentifier:"b9876", keyIdentifier: "7ZmN6ri464+Z", type:"cms_hash", toBeSigned:"7KCE7J6Q7ISc66qFLg=="

- 응답 메시지

```
{
        messageNumber:0,
        sessionID:"string",
        operation:"GenerateSignature",
        resultCode:1234,
        resultMessage:"ok",
        signature:"base64 string"
}
```

예시) messageNumber:0, sessionID:"a1234",operation:"GenerateSignature", resultCode:0, resultMessage:"ok", certificate: "c2lnZW5kIGRhdGE="

5. 키 생성

- 요청 메시지

```
{
        messageNumber:0,
        sessionID:"string",
        operation:"GenerarteKeyPair",
        algorithm:"RSA",
        modularLength:2048,
        purpose:"cmp"
}
```

예시) messageNumber:0, sessionID:"a1234", operation:"GenerarteKeyPair, algorithm:"RSA", modularLength:2048, purpose:"cmp"

- 응답 메시지

```
{
        messageNumber:0,
        sessionID:"string",
        operation:"GenerateKeyPair",
        resultCode:0,
        resultMessage:"ok",
        publicKey:"base64 string",
        keyIdentifier:"base64 string"
}
```

예시) messageNumber:0,sessionID:"a1234", operation:"GenerarteKeyPair, resultCode:0, resultMessage:"ok", publicKey:"MzA4Mjk4NzY1NDMyMQ==", keyIdentifier:" 7ZmN6ri464+Z"

6. 인증서 삽입 및 키 매칭

- 요청 메시지

```
{
        messageNumber:0,
        sessionID:"string",
        operation:"PushCertificate",
        keyIdentifier:"base64 string",
        certificate:"base64 string",
}
```

예시) messageNumber:0, sessionID:"a1234", operation:"PushCertificate" keyIdentifier:" 7ZmN6ri464+Z", certificate:"MzA4Mn.....h5ejEyMzQ="

- 응답 메시지

```
{
        messageNumber:0,
        sessionID:"string",
        operation:"PushCertificate",
        resultCode:0,
        resultMessage:"ok",
        certIdentifier:"number"
}
```

예시) messageNumber:0, sessionID:"a1234", operation:"PushCertificate" resultCode:0, resultMessage:"ok", certIdentifier:"b2345"

참고자료

[CryptoJS] Google, JavaScript implementations of standard and secure cryptographic algorithms, https://code.google.com/p/crypto-js/, CryptoJS 3.1

[CSP] W3C, Content Security Policy 1.0, http://www.w3.org/TR/CSP/, 2012

[CSP2] W3C, Content Security Policy Level 2, http://www.w3.org/TR/CSP2/, 2014

[Drag&Drop] W3C, HTML5, Drag and Drop http://www.w3.org/TR/html5/editing.html#dnd, 2014

[e-ID] BSI TR-03112 Das eCard-API-Framework, https://www.bsi.bund.de/DE/Publikationen/TechnischeRichtlinien/tr03112/index_htm.html, 2014

[FileAPI] W3C, File API, http://www.w3.org/TR/FileAPI/. 2013

[FileWriter] W3C, File API: Writer, http://www.w3.org/TR/file-writer-api/, 2014

[HTML5] W3C, HTML5, http://www.w3.org/TR/html5/, 2014

[IndexedDB] W3C, Indexed Database API, http://www.w3.org/TR/IndexedDB/, 2013

[ITU.X690.1994] ITU-T, Information technology – ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER), 1994

[Keygen] W3C, HTML5 keygen element, http://www.w3.org/TR/html5/forms.html#the-keygen-element, 2014

[MSRCRYPTO] Microsoft Research, MSR JavaScript Cryptography Library, v1.2, http://research.microsoft.com/en-us/downloads/29f9385d-da4c-479a-b2ea-2a7bb335d727/, 2014

[PKCS1] RSA Laboratories PKCS#1, RSA Cryptography Standard v2.1, 2001

[PKCS5] RSA Laboratories PKCS#5, Pasword-Based Cryptography Standard v2.0, 1999

[PKCS8] RSA Laboratories PKCS#8, Private-Key Information Syntax Standard, 1993

[PKCS10] RSA Laboratories PKCS#10, Certification Request Syntax Specification, 2000

[PKCS11] RSA Laboratories PKCS#1, Cryptographic Token Interface Standard v2.1, 2001

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

[RFC2511] IETF, RFC 2511, Internet X.509 Certificate Request Message Format, March 1999

[RFC4210] IETF, Internet X.509 Public Key Infrastructure Certificate Management Protocol (CMP), 2005

[RFC6454] IETF, The Web Origin Concept, 2011

[RFC6712] IETF, Internet X.509 Public Key Infrastructure -- HTTP Transfer for the Certificate Management Protocol (CMP), 2012

[Sandbox] W3C, HTML5 Sandboxing, http://www.w3.org/TR/html5/browsers.html#sandboxing, 2014

[SCJL] Stanford University, Stanford Javascript Crypto Library, http://crypto.stanford.edu/sjcl/, 2009

[WebCrypto] W3C, Web Cryptography API, http://www.w3.org/TR/WebCryptoAPI/, 2014

[WebMessage] W3C, HTML5 Web Messaging, http://www.w3.org/TR/webmessaging/, 2012

[WebRTC] W3C, WebRTC 1.0: Real-time Communication Between Browsers, http://www.w3.org/TR/webrtc/, 2013

[WebStorage] W3C, Web Storage, http://www.w3.org/TR/webstorage/, 2013

[XHR] W3C, XMLHttpRequest Level 1, http://www.w3.org/TR/XMLHttpRequest/, 2014

[XHR2] W3C, XMLHttpRequest Level 2, http://www.w3.org/TR/XMLHttpRequest2/, 2014