ISSN 1598-0170 (Print)
ISSN 2287-1136 (Online)
http://www.jksii.or.kr

웹 로컬스토리지 데이터 보안을 위한 연구

A Study on Data Security of Web Local Storage

김 지 수[1] 문 종 섭[1*]
Ji-soo Kim Jong-sub Moon

요 약

HTML5의 로컬스토리지는 HTML5에서 지원하는 웹 스토리지로, 디바이스에 파일 형태로 저장되어 온・오프라인 모두에서 호환 가능하고 영구 보관이 가능하다는 특징을 가진다. 그러나 로컬스토리지는 데이터를 평문상태로 저장하기 때문에, 파일에 대한 접근 및 수정이 가능하다. 또한 각 도메인에 대한 로컬스토리지를 파일명을 통해 분류하기 때문에, 파일명이 변조되거나 다른 디바이스로 유출되면 로컬스토리지 파일의 재사용이 가능하다는 문제점이 존재한다.

본 논문에서는 로컬스토리지 파일이 생성된 도메인 및 디바이스에 대한 무결성, 기밀성 보장을 위한 방법을 제안한다. 로컬스토리지에 저장되는 데이터를 암·복호화하여 보관하는 방법으로, 암호키는 서버에 보관되며 암호키를 요청하는 단계에서 로컬스토리지를 생성한 디바이스 및 도메인에 대한 인증이 이루어진다. 이를 통해 로컬스토리지의 도메인 및 디바이스간의 기밀성과 무결성을 보장한다. 최종적으로, 제안 방법에 따른 실험을 진행하여 본 논문에서 설명하는 로컬스토리지에 대한 비정상적인 접근에 대해 탐지하는 것을 보였다.

☞ 주제어 : 로컬스토리지, 무결성, 기밀성, 암호화, 해시

ABSTRACT

A local storage of HTML5 is a Web Storage, which is stored permanently on a local computer in the form of files. The contents of the storage can be easily accessed and modified because it is stored as plaintext. Moreover, because the internet browser classifies the local storages of each domain using file names, the malicious attacker can abuse victim's local storage files by changing file names.

In the paper, we propose a scheme to maintain the integrity and the confidentiality of the local storage's source domain and source device. The key idea is that the client encrypts the data stored in the local storage with cipher key, which is managed by the web server. On the step of requesting the cipher key, the web server authenticates whether the client is legal source of local storage or not. Finally, we showed that our method can detect an abnormal access to the local storage through experiments according to the proposed method

☞ keyword : Local Storage, Integrity, Confidentiality, Encrypt ,Hash

1. 서 론

인터넷 서비스 기술이 발달함에 따라 웹 서비스는 더 많은 사용자가 이용하게 되었고, 그에 따라 다양한 브라우저가 등장하게 되었다. 여러 브라우저 간의 호환을 위해 새로운 웹 표준인 HTML5[1] 가 등장했다. 기존의 HTML 언어에서 ActiveX를 통해서만 구현 가능했던 기능들의 상당수가 HTML5 API로 구현 되었고, 자바스크립트와의 연동 또한 용이하여 외부 플러그인 없이도 고수준의 웹 어플리케이션의 구현이 가능해졌다.

웹 스토리지[2] 는 HTML5와 함께 등장하였다. 웹 스토리지는 기존의 쿠키[3]가 가진 적은 용량의 한계를 극복하고, 데이터의 직접적인 저장 및 참조를 위해 고안된 기법으로, 로컬스토리지와 세션 스토리지 두 종류가 존재한다. 로컬스토리지는 데이터가 사용자의 로컬 디스크에 저장되어 유효기간 없이 보관되고, 세션스토리지는 세션이 종료되면서 사라지는 차이점을 제외하면 동일한 성질을 가진다. 로컬스토리지는 데이터의 직접적인 저장이 가능하기 때문에 도메인에서 사용자 식별에 사용되는 UUID(Universally unique identifier) 값 등을 저장하는데 주로 사용되고 있다. 최근 국내 은행에서 공인인증서를 저장하는데 로컬스토리지가 사용되면서, 추후 로컬스토리지의 이용 범위가 더욱 넓어질 것으로 기대된다.

[1] CIST (Center for Information Security Technologies), Korea Univ. Anam Campus, Anam-dong 5-ga, Seongbuk-gu , Seoul 136-713, Korea
* Corresponding author (jsmoon@korea.ac.kr)
[Received 7 March 2016, Reviewed 24 March 2016, Accepted 11 April 2016]

(표 1) 로컬스토리지 기본 설정
(Table 1) Default Setting of Local Storage

Browser	IE	Chrome	Firefox	Opera	Safari
FIle Format	XML	SQLite	SQLite	SQLite	SQLite
Classification method of Domain	File name	File name	Database Column	File name	File name
Save Using Plain Text	○	○	○	○	○
Default Path	%userprofile%\App Data\Local\Microsoft\Internet Explorer\DOMStore	%userprofile%\App Data\Local\Google\Chrome\User Data\Default\Local Storage	%userprofile%\App Data\Roaming\Mozilla\Firefox\Profiles\xxxxx.default\webappsstore.sqlite	%userprofile%\App Data\Roaming\Opera Software\Opera Stable\Local Storage	%userprofile%\App Data\Local\Apple Computer\Safari\LocalStorage

로컬스토리지는 저장되는 단계에서 입력된 데이터를 평문 그대로 로컬 디스크에 저장하며, 파일의 형태로 디스크에 저장되기 때문에, Cross-site Scripting(XSS)[4] 공격부터 물리적인 공격까지 다양한 악의적인 공격을 통해 파일의 내용을 열람하거나, 파일 자체를 네트워크를 통해 외부로 전송하는 것이 가능하다. 획득한 로컬스토리지 파일은 쉽게 수정이 가능하고, 값을 획득하였을 경우, 획득한 값을 이용하여 동일한 로컬스토리지를 생성하는 것 또한 가능하다. 또한 획득한 로컬스토리지 파일을 생성한 도메인이 아닌 다른 도메인에서 재사용이 가능하다. 따라서 로컬스토리지 설계 시 저장되는 데이터 보안에 대한 고려가 필요하다.

본 논문에서는 로컬스토리지 파일이 외부 공격자에 의해 탈취되었을 때, 외부 도메인에서 획득한 로컬스토리지 값의 확인 및 재사용이 불가능하게 하기 위한 방법을 제시한다. 값에 대한 확인 및 재사용이 불가능하게 함으로써 로컬스토리지의 생성원에 대한 무결성 및 데이터의 기밀성에 대해 보장한다.

본 논문의 구성은 2장에서 로컬스토리지의 개요와 관련된 연구를 설명하고, 3장에서 로컬스토리지의 보안 취약점 분석 및 발생 가능한 공격을 설명한다. 4장에서 로컬스토리지 생성원의 무결성, 기밀성을 보장하기 위한 방안을 제시하고, 5장에서 제안 방법을 검증한다. 6장에서 결론으로 본 논문을 서술한다.

2. 배경지식

2.1 로컬스토리지

HTML5 표준에 신규 추가된 웹 스토리지 기능은 웹 어플리케이션 개발 및 사용 시에 저장되어야할 데이터들의 보관을 위해 고안되었다. HTML5 등장 이전에는 쿠키 혹은 외부 플러그인을 이용하여 데이터를 컴퓨터 내부에 저장하였는데, 쿠키 의 경우 적은 용량의 한계를 가지고 있고, 외부 플러그인 또한 브라우저간의 호환이 자유롭지 않다는 단점이 존재했다. 웹 스토리지는 이러한 단점들을 개선하기 위해 등장했다.

로컬스토리지는 쿠키와 같이 사용자의 로컬 디스크에 저장되지만, 최대크기가 4kb 인 쿠키와 달리 10MB 까지 저장 가능하다. 또 로컬스토리지 는 단순 문자열 뿐만 아니라 스크립트 객체 정보를 저장할 수 있다는 이점이 있다. 구조화된 스크립트 객체에 대한 저장이 가능하다는 특성을 이용하여 웹 페이지 구성에 사용될 스크립트 구문을 로컬스토리지에 저장해 두어, 서버 측 트래픽 부담을 감소시키는 방식의 구현도 가능하다.

로컬스토리지에 저장된 데이터는 직접적인 삭제가 없다면 영구히 보존된다. 로컬 디스크에 저장되기 때문에, 서버와 독립적으로 관리되며, 오프라인 상태에서도 로컬스토리지에 저장된 정보를 열람 및 편집할 수 있다. 쿠키와 다르게 서버 측에 값을 전달하지 않기 때문에, 네트워크 트래픽 비용을 줄여 준다는 장점이 있다. 로컬스토리지는 브라우저 종류에 따라 설정되어있는 위치에 각자 저장되며, 저장되는 위치는 표 1과 같다.

로컬스토리지는 key/value 쌍으로 저장되어지며, 자바스크립트를 통해 값의 추가, 삭제 및 참조가 이루어진다. 로컬스토리지를 사용하기 위한 HTML5 API는 표 2와 같다. API를 통해 간단하게 로컬스토리지에 데이터 입·출력이 가능하고, 브라우저의 종류와 관계없이 HTML5가 지원되는 브라우저라면 동일한 API를 사용하기 때문에 호환성이 보장된다.

(표 2) 로컬스토리지 API
(Table 2) Local Storage API

Funtion	Purpose
- localStorage.SetItem("key","value") - localStorage[key] = value	key/value pair add to local storage list.
- localStorage.key(n)	Return the name of the nth key in the local storage list.
- localStorage.getItem(key) - value = localStorage[key]	Return the current value associated with the given key
- localStorage.removeItem(key) - delete localStorage[key]	Remove value associated with the given key
- localStorage.clear()	Remove all of local storage's items

2.2 관련연구

로컬스토리지에 대한 많은 활용이 이뤄지지 않았기 때문에, 로컬스토리지의 보안 이슈는 끊임없이 제기 되었음에도 불구하고, 보안에 대한 연구는 시작 단계에 머물러 있는 상태이다. 반면 웹 스토리지 보안에 대한 연구는 상당부분 진행되어 있고, 대부분이 쿠키 보안에 집중되어 있다. 로컬스토리지와 쿠키는 사용자의 로컬 환경에 저장되어 보관되는 Client-side Storage 라는 동일한 특성을 가진다. 하지만 로컬스토리지의 경우 쿠키와 달리 네트워크 패킷을 통해 전송되지 않는다는 차이점으로 인해 쿠키에 대한 보안 방안을 로컬스토리지에 적용시키기에 다소 제한적이다. 쿠키의 기밀성 무결성 보장을 위해 여러 연구가 발표되었고[5,6,7], 그 중 로컬스토리지에 응용 가능한 연구도 역시 존재한다.

Heng wu 등[8]은 쿠키의 기밀성, 무결성 보장을 위한 방법으로 서버 측에 쿠키 암·복호화에 사용되는 키를 보관하고, 클라이언트가 Mac 주소를 식별자로 하여 쿠키 정보를 요청하면 서버 측에서 보관하고 있는 키를 이용하여 쿠키를 암·복호화 하여 전송하는 방법을 제안하였다. 통신 초기에 서버와 클라이언트가 공개키를 교환한다. 이후 서버에서 시간 정보를 이용해 쿠키를 생성하고, 타임스탬프와 서명 정보를 클라이언트의 공개키로 암호화하여 전송한다. 클라이언트는 이를 복호화하고 서명과 타임스탬프를 검증한다. 검증이 완료되면 Media Access Control Address(Mac주소)를 식별자로 사용하는 키 링에 쿠키정보를 저장한다.

이 논문에서 제안하는 방식에서 Mac주소와 같은 디바이스의 고유 값을 식별자로 사용하여 보관정보를 암호화하여 보관한다는 점은 쿠키와 동일하게 로컬 디스크에 보관되고, 특정 서버와 연결되어야 사용 가능한 로컬스토리지 특성에 적용하기에 적당하다.

로컬스토리지 보안에 대해 다음과 같은 연구가 진행되었다.

Jemel 등[9]은 로컬스토리지의 기밀성 보장을 위한 방법으로 Strong Master Password(SMP)[10]를 이용하여 저장되는 로컬스토리지의 내용과 로컬스토리지 파일의 메타데이터를 암호화하여 데이터의 기밀성 및 무결성을 보장하고, 또한 각 유저의 로컬디스크에 보안 영역인 Secured Local data Storage(SecLDS) 할당하여 로컬스토리지를 저장하는 방식을 제안하고 있다. 사용자는 로컬스토리지 사용전 SMP를 입력하여 모든 값들을 복호화 한 뒤 사용하게 된다. 이 모든 작업은 브라우저에서 지원되며 로컬스토리지의 메타데이터를 암호화하여 보관하기 때문에 파일 이름 변경을 통한 외부 도메인에서의 재사용을 막을 수 있다. 이는 곧 본 논문에서 설정한 공격자 모델에 안전하다 할 수 있다.

하지만 이 방식은 암호키인 SMP의 입력이 사용자에 의해 이루어지기 때문에 키로거(Key logger) 공격에 취약하다. 또한 이 논문에서 제안하는 보안 영역인 SecLDS은 브라우저가 가질 수 있는 시스템 제어 권한이 브라우저에 따라 상이하기 때문에 HTML5의 장점인 호환성을 보장하기 힘들다.

Myeon et. al 등[11]은 로컬스토리지의 기밀성 보장을 위한 방법으로 로컬스토리지 저장시 암·복호화 기능을 추가하여 데이터를 저장하는 방법을 제안한다. 데이터 암호화에 사용되는 암호화 키는 사용자에게 직접 입력을 받는다. 로컬스토리지 상에는 어떠한 암호학적 비밀정보도 저장하지 않는다. 암호학적으로 안전한 암호화 방식을

사용하고 어떠한 비밀정도도 저장하지 않기 때문에 로컬스토리지 파일이 탈취 되었을 때, 보관되는 데이터의 기밀성이 보장된다.

하지만 로컬스토리지가 생성된 도메인에 대한 검증 수단이 존재하지 않고 암호의 입력이 사용자에 의해 이루어지기 때문에 키로거 공격에 취약 하고, 또한 본 논문에서 설계한 리다이렉션을 이용한 공격자 모델에 취약하다.

본 논문에서는 암·복호화 및 키 입력 등 모든 과정을 자동화하여 여러 공격자 모델에 대해 안전한 로컬스토리지 설계 기법을 제안한다.

3. 문제 정의

3.1 로컬스토리지의 보안 취약점

로컬스토리지는 자바스크립트를 통해 추가되고 값에 대한 참조가 이루어지기 때문에 Web Application Secure 정책인 동일 출처 정책(SOP, Same Origin Policy)을 보장받는다. 동일 출처 정책이란 특정 도메인에서 로드 된 문서 혹은 스크립트가 다른 도메인을 엑세스 하지 못하도록 막는 정책이다[12, 13, 14]. 동일 출처 정책으로 세션 및 도메인간의 참조가 불가능하기 때문에 안전하다 생각할 수 있으나 다음과 같은 취약요소를 내포하고 있다.

(1) 로컬스토리지 파일이 특정 경로에 평문의 형태로 저장되어 있는 문제.

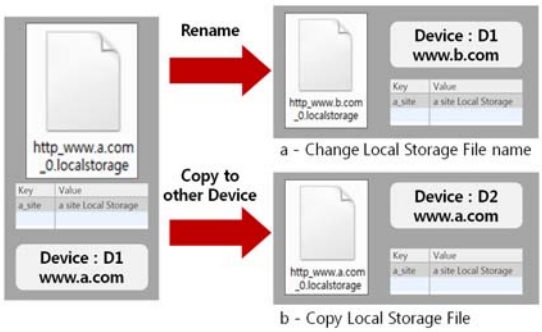
- 표 1에 나타낸 것과 같이 로컬스토리지는 사용자의 로컬 디스크의 특정위치에 평문형태로 저장된다. 이러한 특성으로 인해 로컬스토리지는 다양한 보안 취약점에 노출되고, 공격자는 어렵지 않게 파일에 접근을 할 수 있다. 또한 평문 상태로 저장되기 때문에 값의 확인, 수정 및 삭제가 가능하다.

(2) 로컬스토리지 파일이 생성원이 아닌 다른 디바이스에서 사용 가능한 문제.

- 표 1에서 보는 바와 같이, 브라우저에서 현재 접속한 도메인에 해당하는 로컬스토리지를 탐색할 때 오직 파일명 혹은 컬럼명을 사용하여 현재 접속한 도메인에 대한 로컬스토리지를 탐색한다. 로컬스토리지 생성 디바이스에 대한 검증루틴이 존재하지 않기 때문에 공격자가 사용자의 로컬스토리지 파일을 획득하게 되면, 그림 1-b 와 같이 공격자의 디바이스에서 로컬스토리지를 사용 하는 것이 가능하다.

(3) 파일명 변경을 통해, 생성된 도메인이 아닌 다른 도메인에서 사용 가능한 문제.

- 로컬스토리지 사용 시 도메인에 대한 검증루틴이 존재 하지 않기 때문에, 그림 1-a와 같이 파일명 변경을 통해 외부 도메인에서 생성된 로컬스토리지의 사용이 가능하다.



(그림 1) 로컬스토리지 파일 복사 및 파일명 변경
(Figure 1) Copy&Rename Local Storage File

3.2 발생 가능 위협

위에서 살펴본 것처럼 로컬스토리지는 파일의 형태로 저장되고 참조되는 과정에서 어떠한 검증 과정도 존재하지 않기 때문에 아래와 같은 위협이 발생할 수 있다.

(1) 로컬스토리지 탈취

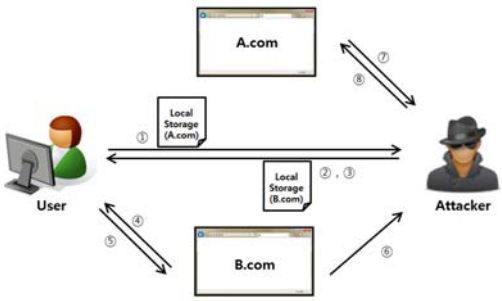
- 특정 위치에 파일의 형태로 저장되기 때문에 XSS, 악성코드 및 물리적인 공격 등 다양한 방법을 통해 사용자의 로컬스토리지 파일이 공격자에게 탈취당할 수 있다. 이를 이용하여 공격자는 악성 스크립트를 삽입, 해당 페이지에 접근하는 사용자들의 로컬스토리지 내용 열람 및 파일 유출이 가능하다.

(2) 탈취한 로컬스토리지를 이용한 2차 공격.

- 본 논문에서 설명한 보안 취약점 중 '다른 디바이스에서 재사용 가능한 문제' 와 '생성된 도메인이 아닌 다른 도메인에서 사용 가능한 문제' 를 이용하여 로컬스토리지의 직접적인 재사용이 가능하고, 또한 리다이렉션(Redirection attack)[15]을 이용한 피싱 공격 등 2차적인 공격이 가능하다.

3.3 공격자 모델

본 논문에서 공격자 모델을 그림 2와 같이 설정한다. 본 공격 환경 및 시나리오는 위 기술한 내용 중 Redirection attack을 이용한 피싱 공격에 근거하여 설계되었다.



(그림 2) 로컬스토리지, 리다이렉션 공격 절차
(Figure 2) Local Storage, redirection attack process

3.3.1 객체 설명

- A.com : 정상적인 도메인. 로컬스토리지에 인증서 및 사용자 식별에 사용되는 값을 저장해두고, 사용자 접속 시 인증서 값을 사용자에게 확인시켜준다. 사용자가 인증서 내용을 확인 한 후 계정 정보를 입력하여 인증서의 내용 확인 혹은 로그인에 필요한 이차 인증을 진행한다.
- B.com : A.com를 위조한 도메인. 로컬스토리지의 값을 읽어와 출력해주고, 사용자에 의해 값이 입력되면 입력된 값이 공격자에게 전송된다.
- 공격자 : 공격자는 여러 가지 공격을 통하여 사용자의 로컬스토리지 파일에 접근이 가능하다.
- 사용자 : 사용자는 홈페이지에서 출력되는 인증서 정보를 확인한 후, 정상적인 값이 출력되면 계정 정보를 입력하여 로그인을 시도.
- 로컬스토리지(A.com) : A.com 도메인에서 정상적으로 생성된 로컬스토리지 파일. 사용자 식별에 사용되는 값이 저장되어 있다.
- 로컬스토리지(B.com) : 공격자에 의해 변조된 로컬스토리지 파일. 도메인 'A.com'에서 생성된 로컬스토리지(A.com) 파일의 파일명 혹은 데이터베이스 컬럼명을 변경하여 'B.com'에서 사용이 가능하도록 변경된 파일이다.

3.3.2 공격 절차

① 공격자는 XSS, 악성코드 등 공격을 통해 사용자 로컬스토리지(A.com) 파일을 획득.
② 공격자는 로컬스토리지(A.com)의 파일 명 혹은 데이터 베이스 컬럼 명을 'B.com'로 변경하고 사용자의 로컬스토리지 저장 영역에 저장.
③ 피싱 기법을 통해 도메인 'B.com' 로 접속 유도.
④ 사용자가 도메인 'B.com'에 접속하면. 도메인 'B.com'은 로컬스토리지(B.com) 파일의 값을 읽어 출력. 이때 출력되는 값은 로컬스토리지(A.com) 의 내용과 동일.
⑤ 사용자는 출력 값을 확인 후, 계정 정보 입력.
⑥ 도메인 'B.com' 는 공격자에게 사용자에 의해 입력된 계정 정보값 전송
⑦ 공격자는 사용자의 로컬스토리지(A.com) 파일을 사용하여 도메인 'A.com'에 접속. 도메인 'A.com' 는 로컬스토리지(A.com) 값을 읽어 출력.
⑧ 공격자는 도메인 'B.com'에서 전달 받은 계정 정보 값을 사용하여 정상 인증.

4. 제안 방법

본 논문에서는 로컬스토리지의 보안상 취약점을 보완하기 위해 데이터를 암호화하여 로컬스토리지에 보관하는 방식을 제안한다. 이 때 암·복호화 과정은 모두 브라우저 내부에서 이루어지며, 키의 입력 또한 사용자에 의한 입력이 아닌 브라우저 내부에서 서버에 요구하여 키를 받아온다. 암호키는 서버에서 생성하며, 키 생성단계에서 클라이언트는 서버에 디바이스 고유 식별 정보를 등록하고 키 요청 시 이를 인증한 후 키를 전달 받는다.

4.1 전제 조건

본 논문에서 제안하는 기법의 안전성을 보장하기 위해서는 아래와 같은 고려사항 및 보안 요구사항을 만족해야 한다.

(1) 항상 같은 클라이언트에 의해 암·복호화 되어야 하며, 키 관리의 용이함을 위해 대칭키 암호화 방식

이 사용된다.

(2) 디바이스 고유 식별 정보의 길이와 암호화에 사용되는 키의 길이는 대칭키 암호 알고리즘의 보안 강도 권고사항인 128비트 혹은 그 이상의 값을 사용한다.

(3) 암호 키는 서버에서 생성 및 관리된다.

(4) 인증 프로토콜에서 디바이스 고유 식별 정보에 대한 어떠한 정보 유출도 발생되어서는 안 된다.

(5) 인증 프로토콜에서 사용되는 값은 재사용이 불가능하도록 신선성(freshness)이 보장되어야 한다.

(6) 만약 로컬스토리지 파일이 탈취되었다 하더라도, 공격자는 이를 이용하여 인증 프로토콜을 통과할 수 없어야한다.

4.2 전체 개요

본 논문에서 제안하는 기법을 수행하기 위해서는 키의 보관이 우선적으로 고려된다. 암호키가 디바이스 내부에 저장된다면, 암호키 역시 여러 공격에 노출되어 안전하지 않으므로 암호화에 사용되는 키는 서버에 보관한다.

서버에 키를 보관하게 되면 클라이언트는 로컬스토리지 사용에 앞서 서버에게 키를 요청해야 한다. 이때 서버는 키를 요청하는 디바이스가 로컬스토리지를 생성한 디바이스와 동일한지 식별할 수 있어야 한다. 이를 해결하기 위해 로컬스토리지가 생성될 때 디바이스의 고유 식별 정보를 서버에 전달하고, 키를 요청할 때마다 디바이스 고유 식별 정보에 대한 인증이 이루어져야 한다. 이 때 디바이스 고유 식별 정보는 로컬스토리지에 저장하지 않는다.

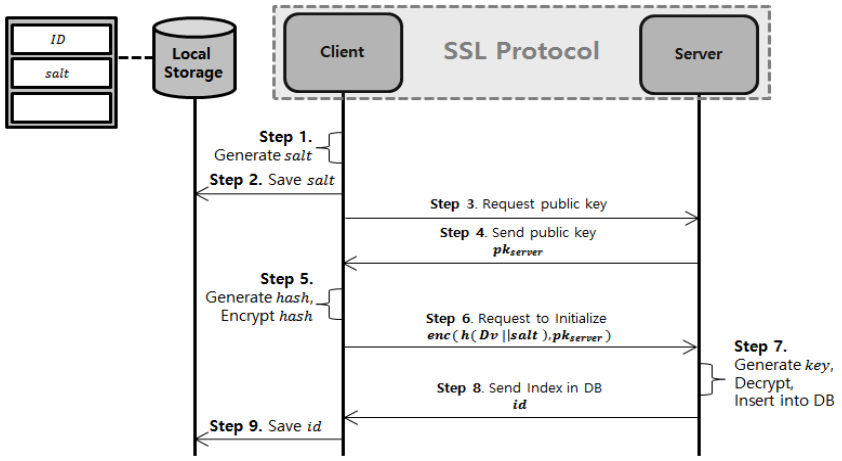
만약 클라이언트가 모든 웹 사이트에 대해 동일한 디바이스 고유 식별 정보를 등록하게 되면, 공격자가 특정 서버를 공격하여 서버에 저장된 암호 키 및 디바이스 고유 식별 정보를 획득했을 때, 공격자는 디바이스 고유 식별 정보를 재사용하여 다른 웹 사이트에 대한 키를 획득할 수 있다. 이를 해결하기 위해 클라이언트는 각 웹 사이트별로 임의의 값 salt 를 생성하고, 디바이스 고유 식별 정보와 생성한 salt를 이용하여 해시 값을 생성한다. 이를 식별 정보로 사용하여 디바이스에 대한 인증을 진행한다. 클라이언트는 각 웹 사이트에 대해 항상 동일한 해시 값을 생성해야 하므로 salt 정보는 로컬스토리지 상에 저장한다.

하지만 이러한 해시 값 역시 유출 된다면, 유출된 웹 사이트에 대해서는 암호키의 유출이 가능해진다. 따라서 해시 값은 초기 로컬스토리지 생성 단계에서만 교환되고, 이후부터는 해시 값에 대한 교환 없이 디바이스에 대한 인증이 이루어져야 한다. 이를 해결하기 위해 로컬스토리지 생성단계에서 서버는 키와 해시 값이 저장된 데이터베이스의 인덱스 정보를 클라이언트에 전송한다. 클라이언트는 이 인덱스 정보를 로컬스토리지에 저장하고, 키 요청시마다 인덱스 정보를 통해 서버에게 인증하고자 하는 해시 값의 위치를 알려준 후, 인증을 진행하여 통신상에서의 해시 값 유출을 방지한다. 또 초기 생성 단계에서 해시 값이 유출될 수 있다. 이를 막기 위해 서버는 공개키·비밀키를 생성하여 클라이언트에게 전송하고, 클라이언트는 서버의 공개키로 해시 값을 암호화한 후 서버에 전송한다. 서버는 이를 비밀키로 복호화하여 데이터베이스에 등록한다. 이러한 과정을 통해 해시 값의 유출을 원천적으로 방지한다.

인증 프로토콜은 난수 기반의 인증 프로토콜을 사용한다. 매번 다른 난수를 생성함으로써, 인증 프로토콜의 신선성을 보장한다. 서버가 먼저 난수를 생성하여 클라이언트에 전송하면, 클라이언트는 로컬스토리지에 저장되어 있는 salt와 디바이스 고유 값을 이용하여 기존의 등록한 해시 값을 생성한 후, 생성된 해시 값과 전달받은 난수를 이용한 해시 값을 생성한다. 이 후 클라이언트는 난수를 생성하여 새로 생성된 해시 값과 난수를 서버에게 전달한다. 서버는 앞서 생성한 난수와 데이터베이스에 저장되어있는 해시 값을 이용한 해시 값을 생성한 후, 클라이언트로부터 전달받은 해시 값과 비교한다. 만일 동일하다면 정상적인 디바이스로 판단하고, 클라이언트로부터 전달받은 난수와 데이터베이스 상의 해시 값을 사용한 해시 값을 생성한다. 이 해시 값과 키 정보를 클라이언트에게 전달한다. 앞서 과정과 동일하게 클라이언트는 전달받은 해시 값을 검증한다. 값이 동일하다면 해당 웹 사이트가 정상적인 웹 사이트라 판단하고, 함께 전달받은 키를 암·복호화에 사용한다. 이 과정을 통해 클라이언트는 현재 웹 사이트에 대한 검증이 가능하고, 서버는 로컬스토리지 파일에 대한 무결성을 보장할 수 있다.

위 과정을 통해 본 논문에서 지적했던 세 가지 취약점에 대한 보완이 가능하다.

오프라인에서의 사용은 연구 대상에서 제외한다. 이는 실제 로컬스토리지가 오프라인에서 사용되는 경우가 매우 드물고, 오프라인에서 사용되는 데이터의 경우 데이터의 중요도가 떨어진다 판단하여 연구범위에서 제외했다.



Database in server

ID	hash	key
id_1	$h(Dv_1 \| \| salt_1)$	key_1
...	...	...
id_n	$h(Dvn \| \| salt_n)$	key_n
id	$\boldsymbol{h(Dv \| \| salt)}$	**key**

(그림 3) 초기 등록
(Figure 3) Initialize

4.3 변수 설명

본 논문의 제안 기법을 설명하기 위한 변수는 표 3과 같다.

(표 3) 변수 설명
(Table 3) Parameter Description

Item	Detail
Lk/Lv	로컬스토리지의 평문 key/value 쌍
Dv	디바이스의 고유 값
id	서버 데이터베이스에 키 위치를 나타내는 값
$salt$	난수 salt
pk_{server}	서버의 공개키
sk_{server}	서버의 비밀키
key	로컬스토리지의 암호화 키
r	난수

4.4 세부설명

본 논문에서 제안하는 기법은 총 4개의 알고리즘(Initialize, Request Key, Save, Load)으로 구성되어 있으며, 각 알고리즘에 대한 상세 설명은 아래와 같다.

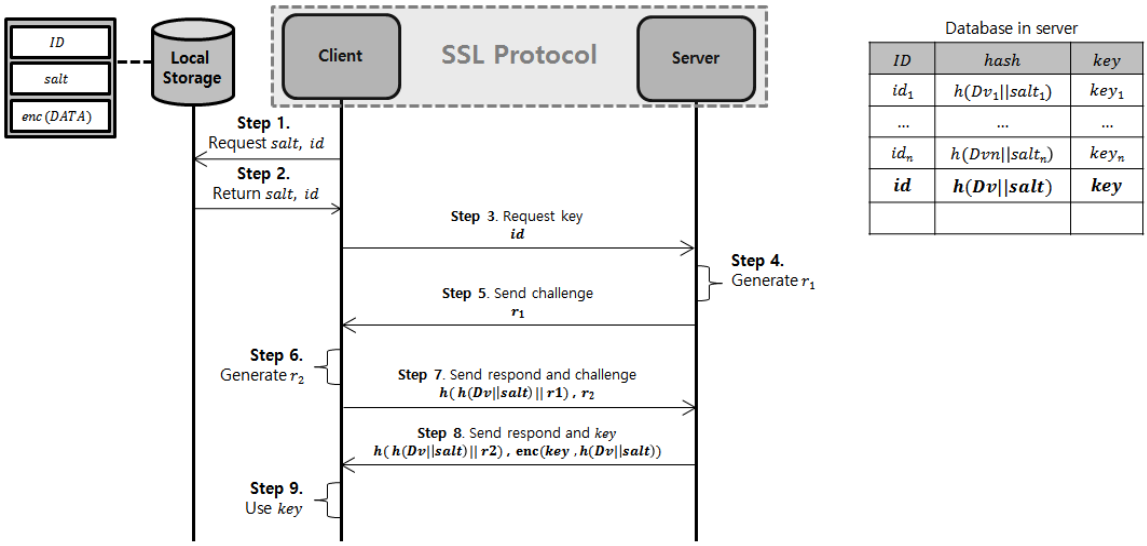
4.4.1 Initialize

웹 사이트에서 최초로 로컬스토리지가 생성될 때의 과정이다. 클라이언트는 $salt$ 를 생성하고 $salt$ 와 디바이스 고유 값을 이용하여 해시값을 생성한다. 서버에 등록 요청과 함께 해시값을 전송하면 서버는 키를 생성하고 해시값과 함께 저장한다. 자세한 과정은 그림 3과 같다.

Step 1) 클라이언트는 $salt$ 값을 생성한다.
Step 2) 로컬스토리지에 생성한 $salt$ 를 저장한다.
Step 3) 서버에게 공개키 pk_{server} 를 요청한다.
Step 4) 서버에게 공개키 pk_{server} 를 받아온다.
Step 5) 클라이언트는 해시값 $h(Dv\|salt)$ 를 생성하고 서버의 공개키 pk_{server} 를 이용하여 암호화한다.
Step 6) 웹 사이트 서버에 키 생성 요청과 함께 암호문 $enc(h(Dv\|salt), pk_{server})$ 를 전송한다.
Step 7) 서버는 암호화에 사용될 암호키 key 를 생성하고, 전달받은 암호문을 비밀키 sk_{server} 로 복호화하여 해시 값 $h(Dv\|salt)$ 를 획득한다. 그 후 획득한 해시 값과 암호 키 key 를 데이터베이스에 저장한다.
Step 8) 서버는 완료되었다는 상태 메시지와 함께 저장된 데이터베이스 위치확인을 위한 값 id 를 클라이언트에게 전송한다.
Step 9) 전달 받은 id 를 로컬스토리지에 저장한다.

4.4.2 Verification & Request Key

웹 사이트 서버에게 클라이언트의 암호키를 요청하는



(그림 4) 키 요청 및 검증
(Figure 4) Verification and Request Key

과정이다. 클라이언트는 로컬스토리지에서 저장된 $salt$와 디바이스 고유 값을 이용하여 해시값을 생성한 후, 로컬스토리지에 저장된 값 중 id를 서버에 보내 암호화에 사용될 키를 요청한다. 서버는 id를 통해 해당되는 데이터를 탐색한 후, 난수를 통한 상호 인증을 진행한다. 이 단계에서 로컬스토리지 생성원 무결성에 대한 검증이 이루어진다. 인증이 완료되면 서버는 클라이언트에게 암호키를 서버 데이터베이스에 저장된 해시 값으로 암호화하여 전송한다. 자세한 과정은 그림 4와 같다.

Step 1) 클라이언트는 로컬스토리지에 $salt$ 와 id 를 요청한다.

Step 2) 로컬스토리지에 저장되어있는 $salt$ 와 id 값을 받아온다.

Step 3) 클라이언트는 웹 사이트 서버에게 id를 전송하며 암호 키를 요청한다.

Step 4) 서버는 id에 해당하는 데이터를 탐색하고, 탐색이 완료되면 인증에 사용될 난수 r_1 을 생성한다.

Step 5) 서버는 클라이언트에게 생성된 난수 r_1 을 전송한다.

Step 6) 클라이언트는 전달받은 난수 r_1 와 앞서 생성한 해시 값 $h(Dv\|salt)$ 을 이용하여 해시 값 $h(h(Dv\|salt)\|r_1)$ 를 생성하고, 난수 r_2 를 생성한다.

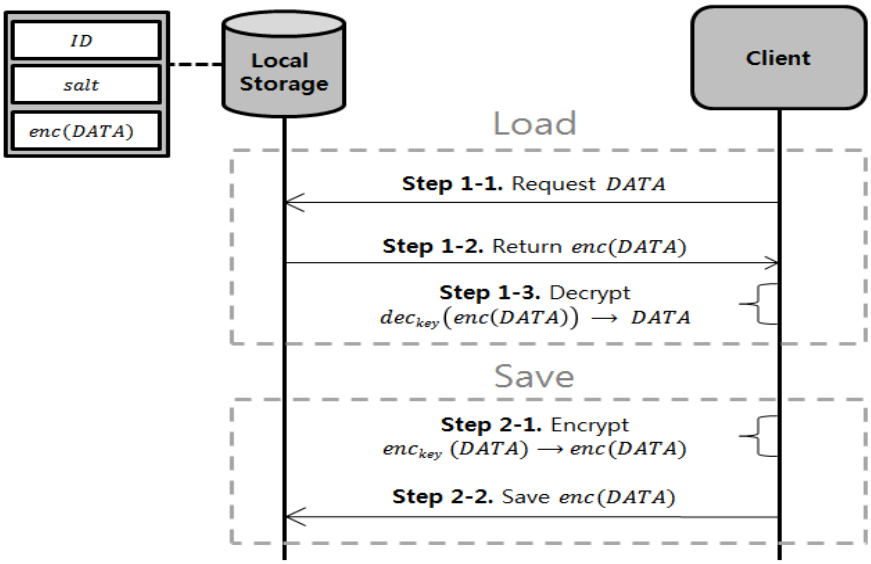
Step 7) 클라이언트는 서버에게 해시값 $h(h(Dv\|salt)\|r_1)$ 와 생성된 난수 r_2 를 전송한다.

Step 8) 서버는 데이터베이스에 저장되어있는 해시 값 $h(Dv\|salt)$ 과 앞서 생성한 r_1 를 이용하여 해시값 $h(h(Dv\|salt)\|r_1)$ 를 생성하고 전달받은 값과 비교한다. 만일 값이 동일하다면 함께 전달받은 r_2 를 이용하여 $h(h(Dv\|salt)\|r_2)$ 를 생성한다. 클라이언트에게 $h(h(Dv\|salt)\|r_2)$ 와 $enc(key, h(Dv\|salt))$ 를 전송한다.

Step 9) 클라이언트는 앞서 생성한 난수 r_2를 이용하여 해시 값을 생성하고, 전송받은 $h(h(Dv\|salt)\|r_2)$ 의 값과 동일하다면, 신뢰 가능한 웹 사이트로 판단하고 함께 전달받은 $enc(key, h(Dv\|salt))$를 복호화한 후 키 key를 사용한다.

4.4.3 Load & Save

서버로부터 암호키 를 전송받게 되면 키를 사용한 암·복호화를 통해 로컬스토리지부터 데이터의 입출력을 수행한다. 자세한 과정은 그림 5 와 같다.



(그림 5) 저장 및 불러오기
(Figure 5) Load and Save

(1) Load

Step 1-1) 클라이언트는 로컬스토리지에 $enc(DATA)$를 요청한다.

Step 1-2) 로컬스토리지에 저장되어있는 $enc(DATA)$를 값을 받아온다.

Step 1-3) Verification & Request key 단계를 통해 획득한 key 를 사용해 복호화하여 $DATA$ 를 획득한다.

(2) Save

Step 2-1) Verification & Request key 단계를 통해 획득한 key 를 사용해 $DATA$를 암호화한다.

Step 2-2) 암호화한 $enc(DATA)$ 를 로컬스토리지에 저장한다.

5. 검 증

본 논문에서 설명하는 로컬스토리지 생성원에 대한 기밀성 및 무결성 검증을 위해, 실제 환경을 적용하여 외부 디바이스에서 로컬스토리지의 사용가능 여부를 검증해 보았다.

검증에 사용된 PC는 3.5 GHz intel core i3 CPU, DDR3 8GByte RAM, 윈도우 운영체제7를 사용했다. 사용한 브라우저는 Internet Explorer 11을 사용했다.

암호화 및 해시 값 생성을 위해 CryptoJS[16] 자바스크립트 라이브러리를 사용했다. 암호화에는 대칭키 암호화 방식인 AES[17]를 사용했고, 운영모드는 CBC를 사용했다. 해시방식은 SHA3[18] 해싱 방식을 사용했다. 디바이스 고유 식별 값으로 사용되는 Dv 는 현재 PC에 128 비트를 만족하는 디바이스 고유 값이 존재하지 않기 때문에, 여러 디바이스 정보 중 MAC address 와 BIOS 버전 정보 그리고 OS 설치시간을 패딩하여 128비트의 길이의 값을 생성해 주었다. 키 보관을 위해 사용한 데이터 베이스로 데이터베이스 관리도구 MySQL[19]을 사용하였다. 키 생성을 위해 math[20] 자바스크립트 라이브러리를 사용하여 구현했다. 최초 키 생성 요청 시 해시값 암·복호화를 위해 JSencrypt[21] 자바스트립트 라이브러리의 RSA[22] 암호화 방식을 사용했다.

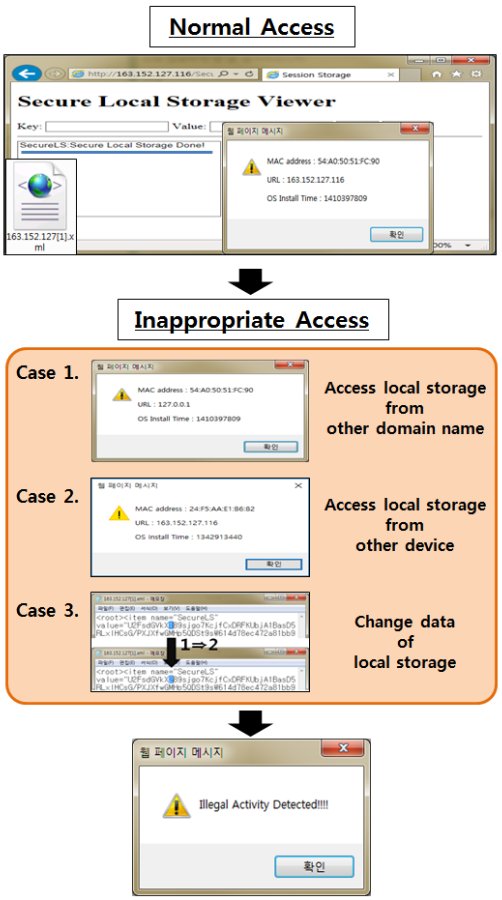
위의 설정대로 구현한 환경에 대해 다음 세 가지 상황에 대해 실험을 진행했다.

(1) **Case 1** : 로컬스토리지 파일명을 변경하여 다른 도메인에서 로컬스토리지에 대한 참조를 시도한다.

(2) **Case 2** : 로컬스토리지 파일을 생성원 디바이스가 아닌 다른 디바이스로 이동한 후 로컬스토리지에 대한 참조를 시도한다.

(3) **Case 3** : 로컬스토리지 파일에 저장되어 있는 값을 임의로 수정한 후 로컬스토리지에 대한 참조를 시도한다.

비정상적인 세 가지 접근에 대해 실험을 진행한 결과 그림 6 과 같이 잘못된 접근을 탐지해냈음을 확인했다.



(그림 6) 제안 사항에 대한 검증
(Figure 6) Verification of purposed method

6. 결 론

본 논문에서 제안 하는 방법은 값을 암호화하고 해시 값을 생성하는 모든 과정에서 사용자에 의해 어떠한 입력 없이 스크립트를 통해 자동으로 처리되기 때문에 키로거 공격 및 리다이렉션 공격 등 여러 공격자 모델에 대해서 안전하다. 암·복호화에 사용되는 키 관리 또한 서버에서 관리되기 때문에 비교적 안전하다. Initialize 과정과 디바이스 인증 과정에서 디바이스 고유 값에 대한 노출이 없기 때문에 서버에 보관된 정보가 유출된다 하더라도, 다른 웹 사이트의 해시 값을 알아낼 수 없고 이차적인 피해에 대해 안전하다. 인증 과정에서 매번 다른 난수를 생성하여 인증하기 때문에, 인증과정에서 교환되는 값이 노출 되더라도 디바이스 고유 값을 알지 못한다면 인증 프로토콜을 통과할 수 없다.

본 논문에서는 디바이스 고유 값이 제안사항의 안정성에 가장 큰 영향을 미친다. 현재 PC에서는 본 논문에서 제안하는 요구사항에 만족하는 단일 값이 존재하지 않으므로, 검증 단계에서는 여러 디바이스 정보를 조합하여 사용하였다.

향후 인터넷 브라우저가 설계될 때 디바이스에 설치된 브라우저 각각에 대해 고유의 serial 혹은 설치 번호 등이 부여된다면, 본 논문에서 언급한 기법을 좀 더 쉽게 활용될 수 있을 것이라 기대한다.

참 고 문 헌 (Reference)

[1] W3C, "HTML 5.1" September 2015. http://www.w3.org/TR/html51/

[2] W3C, "Web Storage (Second Edition)" June 2015, http://www.w3.org/TR/webstorage/

[3] W3C, "HTTP Specifications and Drafts" March 2002, http://www.w3.org/Protocols/Specs.html

[4] OWASP, "Cross-site Scripting(XSS)" April 2014, https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)

[5] J.S. Park, R. Sandhu, "Secure cookies on the Web." IEEE internet computing 4.4 (2000): 36. http://dx.doi.org/10.1109/4236.865085

[6] M. Ter Louw, K.T. Ganesh, V. N. Venkatakrishnan, "AdJail: Practical Enforcement of Confidentiality and Integrity Policies on Web Advertisements." USENIX Security Symposium. 2010. http://static.usenix.org/event/sec10/tech/full_papers/TerLouw.pdf

[7] J.P. Yang, K.H. Rhee, "The design and implementation of improved secure cookies based on certificate." Progress in Cryptology－INDOCRYPT 2002. Springer

Berlin Heidelberg, 2002. 314-325.
http://dx.doi.org/10.1007/3-540-36231-2_25

[8] H. Wu, W. Chen, Z. Ren, "Securing cookies with a MAC address encrypted key ring." Networks Security Wireless Communications and Trusted Computing (NSWCTC), 2010 Second International Conference on. Vol. 2. IEEE, 2010.
http://dx.doi.org/10.1109/nswctc.2010.151

[9] M. Jemel, Mayssa, A. Serhrouchni, "Security assurance of local data stored by HTML5 web application." Information Assurance and Security (IAS), 2014 10th International Conference on. IEEE, 2014.
http://dx.doi.org/10.1109/isias.2014.7064619

[10] R. Zhao, C. Yue, "All your browser-saved passwords could belong to us: A security analysis and a cloud-based new design." in Proceedings of the Third ACM Conference on Data and Application Security and Privacy, ser. CODASPY,13. ACM, 2013, pp. 333-340.
http://dx.doi.org/10.1145/2435349.2435397

[11] H.W. Myeong, J.H. Paik, D.H. Lee, "Study on implementation of Secure HTML5 Local Storage" Journal of Korean Socieity for Internet Information, 2012, 4: 83-93.
http://dx.doi.org/10.7472/jksii.2012.13.4.83

[12] J Ruderman, "The Same Origin Policy" August 2001. http://www-archive.mozilla.org/projects/security/components/same-origin.html

[13] W3C, "Same Origin Policy" January 2010.
http://www.w3.org/Security/wiki/Same_Origin_Policy

[14] MDN, "Same-origin policy" July 2015.
https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy

[15] OWASP, "Top 10 2013-A10-Unvalidated Redirects and Forwards" June 2013.
https://www.owasp.org/index.php/Top_10_2013-A10-Unvalidated_Redirects_and_Forwards

[16] J. Mott, "crypto-js"
https://code.google.com/p/crypto-js/

[17] J. Daemen, V. Rijmen,"The design of Rijndael: AES-the advanced encryption standard" Springer Science & Business Media, 2013.

[18] P. Gauravaram, et al. "Grøstl – a SHA-3 candidate." Submission to NIST, 2008.
http://drops.dagstuhl.de/opus/volltexte/2009/1955/

[19] A.B. MySQL, "MySQL." (2001).

[20] J. Jong, "math.js"
http://mathjs.org/index.html/

[21] T. Wu, "JSEncrypt"
http://travistidwell.com/jsencrypt/

[22] M. Bellare, P. Rogaway, "The exact security of digital signatures-How to sign with RSA and Rabin." Advances in Cryptology—Eurocrypt'96. Springer Berlin Heidelberg, 1996.
http://dx.doi.org/10.1007/3-540-68339-9_34

◐ 저 자 소 개 ◑

김 지 수 (Ji-soo Kim)
2014년 Tsinghua University, Software Engineering 학과 졸업(학사)
2014년~현재 고려대학교 정보보호대학원 정보보호학과 석사과정
관심분야 : 시스템보안, 컨텐츠보안, 임베디드보안
E-mail : wwlrrhd@korea.ac.kr

문 종 섭 (Jong-sub Moon)
1981년 서울대학교 계산통계학과 졸업(학사)
1983년 서울대학교 대학원 계산통계학과 졸업(석사)
1991년 Illinois Insitute of Technology 전산학과 졸업(박사)
2002년~현재 고려대학교 전자 및 정보공학과 교수
관심분야 : 정보보호, 전자공학, 통신공학
E-mail : jsmoon@korea.ac.kr