

- [What is SConnect ?](#)
- [What is a Smart card ?](#)
- [How does a web application authenticate a user via credentials stored in a Smart card ?](#)
- [So what are the problems with classical smart card based authentication and access over the web?](#)
- [How does SConnect help solving the aforementioned problems ?](#)
- [What are the currently supported browser and operating system combinations ?](#)

### **What is SConnect ?**

SConnect is a smart card agnostic browser extension for major web browsers for Windows, Mac OS X and Linux operating systems. The primary purpose of SConnect is to provide a connectivity bridge between JavaScript that runs in a rendered web page in browser and a smart card.

---

### **What is a Smart card ?**

A Smart card is a standards (ISO-7816) based security device which is used to store users' credentials (such as digital certificates) and perform cryptographic operations using these credentials. Modern smart cards are programmable using high level languages such as Java and C#. Learn more about Smart cards [here](#).

---

### **How does a web application authenticate a user via credentials stored in a Smart card ?**

Web applications delegate X509 certificate based authentication to the web server in which they are hosted. For authentication, browsers and web servers interact to establish the identity of the user via the appropriate cryptographic interface available in the browser. Following is a list:

- Internet Explorer : CAPI
- Firefox : PKCS11
- Safari : CDSA

Smart card vendors implement these cryptographic interfaces which provide access to X509 credentials. These implementations (compiled native binaries)

must be deployed on users' computer.

---

### **So what are the problems with classical smart card based authentication and access over the web?**

The problems with classical smart card based web authentication (described above) are as follows:

- o Ubiquity
- o Usability
- o Limitations

Let's tackle them one by one.

#### **Ubiquity :**

This is the most important aspect of the world wide web, i.e. enable the usage of applications across browsers and platforms. Unfortunately this ubiquity is challenged in the context of smart card based authentication for web applications. This is because a user needs to have smart card specific implementation (generally known as middleware) corresponding to the browser/operating system combination on his machine. In other words a user would need to install the browser/OS specific implementations on all the machines that he/she uses, hence while the credentials are stored in a device which you can carry in your wallet, they require host components installed on the machine to be useful. Additionally, since middleware for all the browser/OS combinations are *not* typically made available simultaneously, this approach destroys the mobility aspects of both smart cards and web applications.

#### **Usability :**

Smart card functionality is accessed via the cryptographic interfaces of the browser. As mentioned above cryptographic interfaces are agnostic to the credential store (smart card, PC, etc) and hence the abstractions they provide come in the way of efficiently accessing credentials stored in specific devices. Abstractions can seldom be rich *and* well written so that they can address all the specificities of target devices. Because of this most of the time security mechanisms are abandoned by the application as they come in the way of using the application efficiently and pleasantly. Here is an example -

## FAQ

1. You go to a website that requires certificate based authentication.
2. The browser displays a list of certificates propagated from your smart card.
3. The user is asked to select the appropriate certificate (each certificate has a specific usage attached to it) which in of itself is a challenge
4. Once the user selects the certificate he is prompted for a PIN.
5. The user enters the PIN and authenticates successfully.

While this sounds simple, Steps 2, 4, and 5 present a user interface challenge. Specifically, steps 2, 4 and 5 present the user with a UI that is specific to the browser and smart card middleware. More importantly the web application does not have any control on the way the user interacts with these UI elements. They are outside of the realm of the browser rendered web application. For example canceling the certificate selection, requesting smart card insertion, removing the smart card or abandoning the PIN entry yields an inconsistent response.

This is why smart cards suffer from the hard to achieve marriage of security and usability. While having a notion of carrying your credentials in a secure device is a fascinating idea it capitulates when it comes to the usage of credentials itself.

### **Limitations :**

The cryptographic interfaces of browsers/OS are mostly tied to the digital certificate based authentication (X509). However, there are a variety of other authentication/signature mechanisms besides X509 that are available using Smart cards such as One Time Passwords, PGP, Shared key authentication, etc that are not accessible via the standard cryptographic interfaces. Smart cards also need some management features (user and administrator) such as changing the PIN, unblocking the PIN, downloading a new application, provisioning, updating and deleting credentials. etc. that are not accessible via the cryptographic interfaces of all browsers.

---

### **How does SConnect help solving the aforementioned problems ?**

Coming up with abstractions in software has always been a challenge. Some abstractions work really well and some abstractions *leak*. The cryptographic abstraction used by browsers to access smart card functionality unfortunately leak in the context of web applications. They were primarily conceived to be used with client applications and find themselves very stretched in a different context. SConnect addresses this problem of leaky abstractions by moving down the the abstraction stack and to use an abstraction that targets smart cards. This

## FAQ

abstraction stack fortunately is the same across operating systems and is an industry standard. It is known as [PC/SC](#). The implementation of this standard is installed by default on all operating system images and exposes the same API as mandated.

What SConnect does is that it helps the JavaScript that runs in the browser access the PC/SC layer of the operating system. This enables programming the logic of accessing applications in a smart card through a cross browser consistent and simple interface via JavaScript. Using this approach one can write the logic that was earlier implemented as compiled binaries, partially in JavaScript and partially in a server side language of your choice (ASP.NET, PHP, Java, Python, etc). Not only does the end user experience get enhanced multi-fold, this technique also does not have the limitations of the classical architecture.

You can now utilize or access any custom functionality offered by smart cards over the web, using web based methods.

---

### **What are the currently supported browser and operating system combinations ?**

1. Internet Explorer 5.5+ on Windows 2000, XP and Vista
  2. Firefox 1.5+, 2.0+ on Windows 2000, XP, Vista, Mac OS X 10.4+, 10.5+, Linux
-