WP36 – Browser Integration

D36.1 – Requirements Report

Document Identification	
Date	22.11.2013
Status	Final
Version	1.0

Related SP / WP	SP3 / WP36	**Document Reference**	D36.1
Related Deliverable(s)	D22.1	**Dissemination Level**	Public
Lead Participant	TUD	**Lead Author**	Christoph Busold
Contributors	G&D, SK, AG	**Reviewers**	ATOS, DTU

This document is issued within the frame and for the purpose of the FutureID project. This project has received funding from the European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 318424

This document and its content are the property of the FutureID Consortium. All rights relevant to this document are determined by the applicable laws. Access to this document does not grant any right or license on the document or its contents. This document or its contents are not to be used or treated in any manner inconsistent with the rights or interests of the FutureID Consortium or the Partners detriment and are not to be disclosed externally without prior written consent from the FutureID Partners.

Each FutureID Partner may use this document in conformity with the FutureID Consortium Grant Agreement provisions

1. Abstract

The browser integration work package provides a link between the FutureID client and programs running inside the browser. This allows service providers to integrate the advanced authentication and identification mechanisms of FutureID into their web applications. This document provides an analysis of the technical and security requirements for the browser integration of the FutureID client. Furthermore, we present existing techniques for browser integration, evaluate them with respect to our requirements and show their advantages and disadvantages.

2. Document Information

2.1 Contributors

Name	Partner
Christoph Busold	TUD
Jon Rios	TUD
Dr. Frank-Michael Kamm	G&D
Daniel Albert	G&D
Maili Keskel	SK
Florian Gruner	AG

2.2 Reviewers

Name	Partner
Juan Carlos Perez Baun	ATOS
Omar Almousa	DTU

2.3 History

Version	Date	Author	Changes
0.1	03.07.2013	Christoph Busold	Initial Version
0.2	10.07.2013	Christoph Busold	Distribution of Work
0.3	22.07.2013	Christoph Busold	Added Draft Content for Sections 6.1 and 7
0.4	01.08.2013	Frank-Michael Kamm	Added security requirements
0.5	26.08.2013	Christoph Busold	Added Section 5.2 and 8.3
0.61	09.09.2013	Christoph Busold	Extended 6.2.2
0.62	29.10.2013	Maili Keskel	Added Section 8.2
0.63	29.10.2013	Christoph Busold	Added Introduction
0.64	30.10.2013	Florian Gruner	Added Section 8.1
0.7	31.10.2013	Christoph Busold	Added Conclusion and Abstract
0.81	11.11.2013	Jon Rios	Included DTU comments for TUD sections
0.82	11.11.2013	Christoph Busold	Included ATOS comments for TUD sections
0.83	15.11.2013	Maili Keskel	Included ATOS and DTU comments for SK sections
0.84	21.11.2013	Florian Gruner	Included ATOS and DTU comments for AG sections
1.0	22.11.2013	Christoph Busold	Final Version

2.4 Table of Figures

2.5 Table of Tables

2.6 Table of Acronyms

APDU	Application Protocol Data Unit
API	Application Programming Interface
BHO	Browser Helper Object
BSI	Bundesamt für Sicherheit in der Informationstechnik (German Federal Office for Information Security)
FIdM	Federated Identity Management
HTML	Hyper-Text Markup Language
MIME	Multipurpose Internet Mail Extensions
NFC	Near Field Communication
NPAPI	Netscape Plugin Application Programming Interface
PKCS	Public-Key Cryptography Standards
SAML	Security Assertion Markup Language
SOP	Same Origin Policy
SSO	Single-Sign-On
SSL	Secure Sockets Layer
TLS	Transport Layer Security
XSS	Cross-Site Scripting

2.7 Glossary

ActiveX	Proprietary framework for components embedded in Microsoft Internet Explorer
AusweisApp	Client for the German eID card developed by OpenLimit SignCubes AG
CryptoAPI	Proprietary API for access to cryptographic tokens on Microsoft Windows, now called Cryptography API: Next Generation
OpenMobile API	Open API for access to smartcards and secure elements (especially SIM cards) on mobile phone operating systems
PKCS #11	Standard for an open API for access to cryptographic tokens supported by many platforms and applications
WebCrypto API	JavaScript API for access to cryptographic objects and algorithms from within the browser

3. Table of Contents

4. Introduction

The FutureID project develops a comprehensive, flexible, privacy-aware and ubiquitously usable identity management infrastructure for Europe and beyond, which integrates existing eID technologies and trust infrastructures, emerging federated identity management services and modern credential technologies to provide a user-centric system for the trustworthy and accountable management of identity claims. One key element of the FutureID infrastructure is the client application, which is running on the local device and provides the connection point between the backend infrastructure, the user's credentials (e.g., in form of an external eID card) and the service provider. In case the service provider is a web application, the FutureID client has to be integrated into the web browser in order to provide a communication channel between them.

The browser and the FutureID client together form the frontend interface of the FutureID infrastructure. In order to increase the acceptance of FutureID, the browser integration solution should therefore be smooth, robust and user friendly. The experience with existing eID clients shows that problems with the browser integration can have serious impact on the whole system.

For example the client for the German eID card, called AusweisApp, only supports certain versions of the Firefox browser tested at release, which at the time of writing this report lag behind the newest Firefox version by more than half a year. This creates both usability and legal problems. First, requiring users to keep an older version of their favourite browser is one of the reasons why the German eID card never got a wide acceptance among citizens [1]. Second, the browser integration is part of the client application and thus part of the software certification process. If this component needs to be updated for a new browser version, then the whole eID client has to be certified again. As it does not make sense to do this every six months, until now the AusweisApp has not been officially certified by the German federal office for information security (BSI) at all [2], although this is explicitly required by the regulations governing the use of the electronic identity card (PAuswV §23 [3]).

4.1 Scope

The scope of this document is to analyse the technical and security requirements for integrating the FutureID client into various web browsers on different platforms and evaluate existing solutions for browser integration with respect to these requirements.

4.2 Outline

This document is structured as follows. Section 5 gives an overview of the platforms, which should be supported by the FutureID client. Section 6 provides a description of secure bindings. Section 7 evaluates existing browser integration methods. Section 8 defines the technical and security requirements for the browser integration solution. Finally, Section 9 concludes this document with a recommendation, how the browser integration for FutureID can be designed.

5. Targeted Platforms

This section lists the client platforms, which are targeted by the FutureID project and therefore should be considered in this document. A platform in this context is a combination of the operating system and the web browser of the user and his device.

D37.1 *Requirements Report for the Client Testbed* defines the platforms to be tested with the FutureID client in Section 5.1. The FutureID client cannot officially support platforms, which will not be tested, and tested platforms have to be supported. Therefore the targeted platforms for the browser integration exactly correspond to the tested ones. Support for untested combinations of supported browsers and operating systems is optional. However, the browser integration should generally be designed to run on as many platforms as possible.

5.1 PC-based Platforms

The main operating systems for PCs are Microsoft Windows, Apple MacOS and various distributions of Linux. In the last years almost all desktop systems switched to the new 64-bit (x64) architecture. Most of these operating systems, however, are still available in a version for 32-bit (x86) processors.

The most popular web browsers on PCs are Google Chrome, Microsoft Internet Explorer and Firefox, which is developed as open source project by the non-profit organization Mozilla. Apple provides MacOS with its own default browser Safari.

According to D37.1 *Requirements Report Client Testbed* Section 5.1.1 the FutureID client will be supported on Windows 7 (in its 32-bit version) and Windows 8 (64-bit) together with the web browsers Internet Explorer, Firefox and Chrome. Windows 8 will also consider the Metro version of Internet Explorer. Support for Linux focuses on the Ubuntu distribution (64-bit) with Firefox and MacOS will be supported in version 10.8 together with Safari.

5.2 Mobile Platforms

Over the last years Android, an open source project initiated by Google, has clearly become the most popular operating system for mobile devices like smartphones and tablets. Second in line is iOS, the operating system for Apple's iPhone and iPad. The remaining market is split between RIM (BlackBerry), Microsoft (Windows Phone) and other minor competitors like Nokia (Symbian).

Android by default ships with an open source browser while RIM uses a proprietary implementation for BlackBerry. Apple and Microsoft each preinstall their own browsers on their devices, Safari and Internet Explorer. Common third-party browsers for smartphones and tablets include the mobile versions of Google Chrome and Mozilla Firefox.

The only mobile platform, which will be tested with the FutureID client, is Android in version 4.1 together with Chrome (see D37.1 *Requirements Report Client Testbed* Section 5.1.1). Therefore this will be the only combination officially supported by FutureID and targeted in this work package.

6. Secure Bindings for Federated Identity Management

Federated Identity Management (FIdM) [4] is used to manage identities and users across different IT systems or even organizations. In such a system with multiple independent services, the Single-Sign-On (SSO) property means that a user needs to authenticate only once in order to get access to all those services. Examples for FIdM frameworks and standards are the Liberty Alliance [5], Microsoft Cardspace [6], SAML [7] and OpenID [8].

In such systems, a secure binding between an authentication session (with the identity provider) and the corresponding session with the service provider is required. Otherwise the authentication session could be misused by an attacker in another context. The following sections describe the security problem of the underlying FIdM protocols in more detail and provide different possible solutions.

6.1 Web-based Federated Identity Management Protocols

A Federated Identity Management protocol is browser-based, if it can be implemented with the standard functionality of web browsers. This allows using the browser as a platform-independent client and integrating FIdM seamlessly into web applications.

Figure 1 shows the communication between the three involved parties in browser-based FIdM protocols, the service provider (SP), the identity provider (IP) and the browser (or user agent):

1. The browser connects to the service provider.
2. The service provider determines the user's identity provider and redirects the browser to its login page.
3. The user authenticates to his identity provider.
4. The identity provider returns an access token.
5. The browser sends this access token to the service provider.
6. Based on this token, the service provider either grants or denies access.

The authentication token is used to access the service. This means, if an adversary manages to steal this token, he is able to impersonate the user. As the token is sent through the client, it has to be stored inside the browser. Modern browsers restrict access to data by the Same Origin Policy (SOP), which means that browser objects (like cookies) can be accessed by other objects (especially scripts) only if they come from the same domain.

However, this security mechanism is vulnerable to several attacks. Since the Same Origin Policy relies on the underlying Domain Name System (DNS), an attacker can circumvent it by spoofing his domain name or IP address. Another possibility for attacks is Cross-Site Scripting (XSS), where an attacker injects malicious JavaScript code into a different session. As the Same Origin Policy does not apply, this code can access all objects of this session.

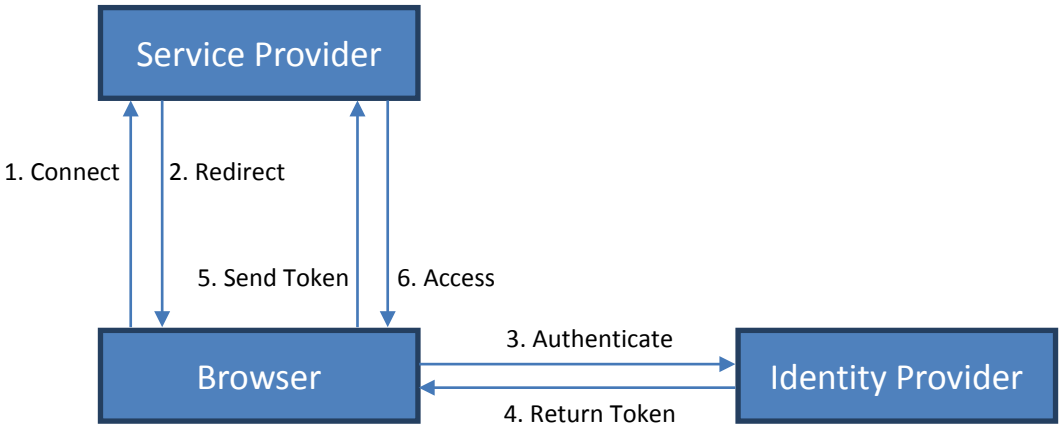

Figure 1: Protocol Steps in a Browser-based FIdM System

Furthermore, the specific Federated Identity Management protocols can be attacked. Groß [9] for example analysed the security of the Security Assertion Markup Language (SAML) protocol implementation and discovered an attack, through which an adversary could intercept the authentication token contained in a URL. Therefore, extensions to SAML try to bind the authentication token to a specific browser session, so that it cannot be used by an adversary in a different context. The following sections give a short overview over SAML and the methods for secure session binding that have been proposed so far.

6.2 Security Assertion Markup Language

SAML defines a standard for exchanging XML-encoded authentication and authorization messages between a service provider and an identity provider. The syntax and semantics of those messages are defined in [10]. SAML consists of several profiles defined in [11]. The most interesting for us is the *Browser SSO Profile*, which defines the protocol for browser-based SSO. SAML specifies two types of access tokens, SAML assertions and SAML artefacts.

The XML encoding of an assertion looks like the following:

```
<saml:Assertion Version ID IssueInstant>
   <saml:Issuer>
   <ds:Signature>?
   <saml:Subject>?
   <saml:Conditions>?
   <saml:Advice>?
   <saml:Statement>*
   <saml:AuthnStatement>*
   <saml:AuthzDecisionStatement>*
```

```
    <saml:AttributeStatement>*
</saml:Assertion>
```

The attributes of a `saml:Assertion` element are the SAML `Version`, the assertion `ID` and the issuing time in `IssueInstant`. The `saml:Issuer` field specifies the authority which makes the claims in this assertion. The contents of the assertion are protected by a digital signature. The usage of other elements depends on the SAML profile. User-defined statements may be placed in the `saml:Statement` field.

After the client authenticates to his identity provider, he receives an assertion which he forwards to the service provider. Based on the information contained in the assertion, the service provider will decide whether to grant access to the client or not, after checking the integrity of the assertion by verifying the included signature. In case an assertion cannot be sent through the browser, for example due to size constraints, an artefact can be used to indicate where and how to retrieve the corresponding assertion.

6.3 Binding to Client Certificate

One possibility for secure session binding is to bind the SAML assertion to the public key of a client certificate in the browser [12]. This method has already been standardized in the SAML Holder-of-Key Web Browser SSO profile [13].

This binding requires a client certificate in the browser. However, this certificate does not have to be part of a public key infrastructure. In fact, it may be self-signed. This method only relies on two important properties. First, the public key of the certificate must be unique, so that it can be used to securely identify this browser. This can be achieved by generating a new key pair with a good random number generator. Second, access to the corresponding private key is managed by the browser, so that JavaScript code cannot steal it. It may even be protected against software attacks by storing it inside a hardware security module such as a smartcard.

The protocol then works as follows: When the browser is redirected to the identity provider, it uses TLS with client authentication for this connection. The identity provider then extracts the public key of the certificate presented by the client in this TLS session and adds it to the assertion. The authors in [10] propose to place it either into the `saml:AuthnStatement` or into the `saml:Condition` element. As both of those elements are included in the signature, the public key is automatically protected against modifications.

While sending the assertion (or the artefact) to the service provider, the client has to use TLS with client authentication again together with the same certificate. Upon verifying the assertion, the service provider then additionally checks that the public key in the current connection's certificate matches the public key encoded in the assertion. If this is true, the service provider can be sure that it is dealing with the correct browser, because it had to be in possession of the corresponding private key during the TLS session establishment.

This method prevents attackers from using a stolen SAML assertion by binding it to a client certificate. Its main advantage is that this can be implemented with a standard browser. Software modifications are only required for the identity and service provider.

6.4 Strong-Locked Same Origin Policy

Another way to protect SAML assertions is to refine the Same Origin Policy and bind it to a cryptographic identifier instead of a domain name [12]. This extension to SOP is called Strong-Locked Same Origin Policy (SLSOP).

SLSOP extends the syntax of the `secure` attribute for the `set-cookie` command by adding a list of public keys of those servers who are allowed to access this cookie. In this case, the traditional `domain` value cannot be used.

After the client authenticates to his identity provider, the SAML assertion is stored inside the browser as a cookie, which is bound to the public key of the service provider. This way, the cookie is protected against DNS spoofing and similar attacks and will only be accessible if the session is secured by TLS and the public key in the server certificate matches the binding. Therefore, a malicious website would need the private key of the service provider in order to retrieve the assertion. Such cookies can also be protected against XSS by isolating them, so that they can only be transmitted to servers but not accessed directly by JavaScript code anymore. This is supported through `httpOnly` cookies on most common web browser platforms.

The required changes to the client browser are minimal and could be implemented as a plugin, depending on the browser platform. Further, the identity provider must know the public key of the service provider to protect the cookie. This can be implemented by registering services with their public keys in advance, so that an attacker cannot request a cookie for an arbitrary public key.

The advantage of this method is that assertions and their validation remain unchanged. On the other side, however, it requires modifications to the browser and it cannot make use of security hardware to protect against software attacks. Furthermore, this method might present issues for server farms, where multiple different servers process requests for the same service. In that case, either all of them need to have the same TLS certificate and key pair, or all of their public keys must be whitelisted in the cookie.

6.5 Binding to TLS Session

Another approach for secure SAML session binding similar to the first one is to bind the assertion to a specific TLS session instead of a client certificate [14].

Again, the browser first establishes a TLS session with the service provider. The underlying TLS protocol stack will then create a shared secret master key for this session on both sides. When he is redirected to the identity provider, the client puts a key derived from this TLS master key into the assertion. As this key

is protected by the signature, the service provider can check if this matches the derived key of his current TLS session upon receiving the assertion. This way the assertion is only valid for one single session and cannot be used in a different TLS session, because this will generate a different TLS master key.

The advantage of this method is that the master key changes with every TLS session and thus provides anonymity for the browser, whereas the client certificate uniquely identifies it and therefore allows service providers to track the user.

The most important disadvantage is that the TLS master key is a secret managed by the TLS implementation. This has to be exposed through a special interface, requiring a modification to the client browser as well as the web server of the service provider.

6.6 Comparison between Binding Methods

Table 1 shows a comparison between the three session binding methods discussed in the previous sections.

	Client Certificate	SLSOP	TLS Session
Standard Browser Support	✔	✘	✘
Standard Web Server Support	✘	✘	✘
No SAML Extension Required	✘	✔	✘
Untraceability for Client	✘	✔	✔
Hardware-based Security Support	✔	✘	✘

Table 1: Comparison between Methods for Session Binding

7. Existing Solutions for Browser Integration

The following section presents existing solutions for browser integration, namely using a browser plugin, using a local web service running on the client and integration with PKCS #11. Each solution is briefly introduced and then evaluated regarding its advantages and disadvantages.

7.1 Browser Plugin

7.1.1 Introduction

According to the technical specification TR-03112 any eID-clients represent a middleware platform, which realizes and manages the communication between eID-card, eID-terminal and eID-server in order to simplify (trans-)national identity management. Taking this into account, an eID-Client is a term, describing the complete client-software necessary to perform Online-Authentication [15]. Thus, an eID-client for FutureID is not just only a piece of software realizing the exchange of sensitive user information, it is more like a kind of a security backbone of the FutureID infrastructure. Such a client for FutureID strongly depends on the trust of the European citizens.

At the moment an eID-client should not be realized as a plugin for browsers. By developing such a sensitive part of the eID infrastructure developers and governmental responsibilities would have to solve some major difficulties. These challenges can be divided in three sections – security, user acceptance and managing a variety of different web browsers.

7.1.2 Security

In the last years, browsers were getting more and more complicated. That means in consequence, that bugs could be used by hackers and thieves to get into a computer [16]. The same applies to plugins, as encapsulated pieces of software, adding new capabilities into browsers. They can be also be used to get control of chosen functionalities of the browser or even the whole system by utilizing malicious code to get in [16]. But not only manipulated code is dangerous, but also bugs of interfaces or the plugin architecture. That's why vendors of browsers and plugins must be carefully during coding and afterwards during a period of maintenance. Not all users are aware of these facts and probably they do not even know that a secure systems requests maintenance by the owner, by updating any safety-related systems.

In many cases user do not know, for what plugins are there for and what they are doing. They would not be able to realize, that where are undesirable hidden function calls between different plugins or keyloggers, recording any keyboard input in order to get security PINs or other passwords. Such criminal functionalities could he hidden in extended browser plugins, such as Java, Active X, PDF-Viewer and other well-known plugins which could be affected by third parties for fraud reasons to manipulate an browser or even the operating system. Furthermore, plugins could offer unknown and manipulable access rights within browsers, so that it could be possible, that other browser plugins are able to get different rights.

In some cases, users are not entitled to install plugins for their browsers for security reasons by enterprise guidelines. Especially in security sensitive enterprises, such as banks and insurances, IT departments could be concerned about any plugins. Besides that, in many cases, users are only utilizing a thin client, there they are not able to install any programs.

7.1.3 Managing a variety of different web browsers

There is a huge variety of different web browsers – consequently there should be specialized FutureID plugins to address all of them, due to increase user acceptance in the long term. Beside the variety of different web browsers, users are not always able or even willing to make sure, that they use the latest web browser version. In some cases, users are not allowed to update their system and other applications by their administrators. Consequently, current browsers in use are not always up to date, so that published eID-plugins must be backward compatible in order to guarantee a widespread coverage irrespective of the published version.

Taking the section above into account, developers have to handle two challenges; they have to produce code for at least the most common browsers, like Chrome, Firefox, IE or Safari. Table 1 shows the development of popular browsers in the European Union for the last 12 month, starting from October 2012 (please see http://gs.statcounter.com/#browser-eu-monthly-201210-201309). Developers have to guarantee backward compatibility of their plugins, due to different versions of used browsers, as you can see in Table 2 (please see http://gs.statcounter.com/#browser_version-eu-monthly-201210-201309). Beside these challenges, developers of a plugin have to test new versions of the FutureID-client additionally on the most common operation systems within the European Union. Currently there are about seven operation systems, which have to be supported in order to provide convenient experience (see Table 3, please see http://gs.statcounter.com/#os-ww-monthly-201210-201309).

Date	Chrome	Firefox	IE	Safari	Opera	Other
2012-10	32,39	29,29	26,12	7,8	3,31	1,1
2012-11	33,23	29,57	25,34	7,96	2,71	1,19
2012-12	34,16	29,55	24,36	7,99	2,54	1,4
2013-01	34,58	28,64	24,47	8,35	2,36	1,59
2013-02	35,02	28,69	23,64	8,6	2,39	1,66
2013-03	35,51	27,91	23,95	8,59	2,29	1,75
2013-04	36,66	27,68	22,89	8,78	2,15	1,84
2013-05	37,86	27,3	21,67	9,1	2,14	1,93
2013-06	38,36	27,15	21,15	9,25	2,12	1,96
2013-07	38,34	27,21	20,88	9,09	2,27	2,22
2013-08	38,46	26,05	21,76	9,09	2,31	2,33
2013-09	37,33	25,28	23,64	9,17	2,3	2,27

Table 2: European Marketshare of common web browsers (in %)

Browser	Version	2013-09
IE	10.0	11,13
IE	9.0	5,04
IE	8.0	6,76
IE	7.0	0,56
IE	6.0	0,11
Chrome	29.0	33,06
Chrome	28.0	1,29
Chrome	27.0	0,39
Chrome	26.0	0,17
Chrome	25.0	0,22
Chrome	24.0	0,18
Chrome	23.0	0,23
Chrome	22.0	0,16
Safari	iPad	4,27
Safari	6.0	2,94
Safari	5.1	1,38
Safari	5.0	0,42
Firefox	23.0	19,23
Firefox	22.0	0,62
Firefox	21.0	0,31
Firefox	20.0	0,23
Firefox	19.0	0,16
Firefox	18.0	0,15
Firefox	17.0	0,24
Firefox	16.0	0,27
Opera	12.1	1,52
Opera	12.0	0,1
Opera	11.6	0,06
Other		8,8
Summe		100

Table 3: Browser Version in use in September 2013 (in %)

Date	Win7	WinXP	WinVista	MacOSX	Win8	iOS	Linux	Android	Other
2012-10	53,24	24,23	9,41	7,46	0,4	2,9	1,36	0,47	0,54
2012-11	53,15	23,71	9,05	7,5	1,07	3,06	1,37	0,55	0,53
2012-12	53,39	22,86	8,85	7,23	1,78	3,28	1,34	0,75	0,53
2013-01	52,84	22,29	8,35	7,45	2,63	3,55	1,36	0,97	0,57
2013-02	52,5	21,94	7,99	7,45	3,27	3,85	1,4	1,06	0,53
2013-03	52,38	21,48	7,95	7,29	3,9	3,98	1,4	1,07	0,55
2013-04	52,48	20,41	7,7	7,57	4,61	4,05	1,48	1,13	0,57
2013-05	52,23	19,54	7,48	7,85	5,25	4,26	1,59	1,21	0,58
2013-06	52,17	19,04	7,19	7,81	5,81	4,47	1,62	1,32	0,58
2013-07	51,94	18,93	6,92	7,44	6,39	4,57	1,7	1,5	0,62
2013-08	51,28	18,68	6,97	7,34	6,88	4,72	1,77	1,72	0,64
2013-09	51,2	18,67	6,89	7,51	7,28	4,54	1,55	1,76	0,6

Table 4: Operating System market Share (in %)

A very rough calculation could provide involved partners of the FutureID project with a slight idea of the effort of publishing a new version of the eID client:

- 5 types of different browsers in about 11 different versions (at least 1 per cent market share);
- 6 types of different operating systems (running on typical personal computer devices);
- a new version of an eID client must be tested on **66 different variations** of web browser version and running operating system.

Not only, that plugins mean a massive effort for developing, programming and testing – involved producers of such an eID-client highly depend on browser publishers. So they always have to check the latest version. And in case of a new version, they have to adapt the client to the new conditions of the latest browser plug in. A newly developed or adapted eID client must be provided to users by loading it to web sites or browser specific stores.

The following xml –snippets taken from an extension for Firefox (Private Browsing Proxy 1.3) will clarify the problem of compatibility. It is a plugin to support users with a possibility for private browsing ("Automatically switches to the specified proxy type when entering Private Browsing. Reverts to previous setting when all PB windows are closed.") (see https://addons.mozilla.org/de/firefox/addon/private-browsing-proxy/?src=cb-dl-recentlyadded)

```
<!-- Firefox -->
 <em:targetApplication>
   <Description>
     <em:id>{ec8030f7-c20a-464f-9b0e-13a3a9e97384}</em:id>
     <em:minVersion>19.0</em:minVersion>
     <em:maxVersion>20.*</em:maxVersion>
   </Description>
 </em:targetApplication>
```

According to the disclosures of the target application-tag it only could be used by about 1.8 per cent of all Firefox users.

In order to build up a widespread acceptance it is not possible to publish an important part of the FutureID environment with such close constraints. Consequently, the development and maintenance of a functional eID-client offer would be very time-consuming and costly.

7.1.4 User acceptance

The following facts are directly related to efforts for developers:

- User only accept secure, stable and reliable software;
- Look and feel of software and applications should be similar across different platforms.

Providers are forced to supply users with stable and reliable software. Users do not want to install the latest version of their web browser or even change their well-known browser in order to meet the requirements of a plugin. This means for developers, that they have to meet users at their own ground; even it means much more programming effort and expenditures. Furthermore, many users are concerned about security and privacy protection in the internet. They want to know what is going on in their computer system and many of them know, that the usage of plugins could be critical.

Beside reliability and data security, users want to have a similar look and feel over different platforms. So they want to use identification and authentication clients not only on non-mobile devices but also on notebooks, tablets or even smartphones. The objective is to simplify and secure registration and login for users at internet providers by replacing username and password-authentication with eID. In future it should be possible to utilize NFC-enabled mobile devices for this process [17]. In order to fulfill governmental requirements for mobile authentication it would not be possible, to implement such a client as a plugin. A mobile client has to implement required protocols, functions and processes [18]. Please see MONA as an example for mobile authentication [18].

There is a noticeable trend towards the usage of mobile devices. In the long term, users want to use every device for every reason. They do not want to change their device for different tasks.

In a mobile environment it is not possible to work with plugin, as it would be necessary for online identification and authentication. Mobile devices and their operating systems are following a different philosophy. Users have to download and install so called apps (short term for application, being used on mobile devices), which are providing specialized functionalities in a very close manner. This means for eID publisher additional platforms (iOS, Android, Windows) they have to take care about.

7.1.5 Technical implementation and system performance

To ensure a strong authentication, the eID client must be activated through the web browser in order to build up a connection to the eID-Server. The eID-Server is used to fulfill the actual authentication. According to "TR-03124-1 eID-Client" an eID client should provide its services at a specific local interface, which should not be available to external callers. There are two mechanism to call an eID-Client during a session:

- MIME-Type
- Embedded Link

It is possible to utilize the approach of embedded links, if there are no components installed in the browser, such as plugins or other kinds of extensions. The main advantage is, that all platforms are supported independently, as long as they allow to install software and to open or register port 24727 locally. MIME-Type should not be used anymore, because it requests browser specific extensions, which are hard to maintain [19].

7.1.6 Conclusion

It is recommended to develop an eID-client as a closed system, separated from the operating system or other applications. This procedure would follow an approach of decomposition of systems in parts of different stages of trustworthiness [17]. Developers would be able to set up their stringent access according to the demand of security. Thus, developing an eID client as a proprietary software could reduce the risk of compromising data and code.

7.2 Local Web Service

7.2.1 Introduction

In this part Local web service solution as browser integration would be described.
Solution advantage is that one installed solution in user PC fits all browsers compared to traditional approach where every browser has to has own integration.
Solution works by directing for example digital signature related communication to local web service where smartcard (or similar) related commands are executed.

7.2.2 Solution explanation

The solution has to be binary compatible with supported platform, which needs more testing. Installed service has to have its own PIN handling dialogues.
Interaction with user might be tricky since service does not know which browser process or tab has initiated communication – problem is that PIN dialogue is unfocused.
Local web service solution has to have separate solution to secure channel between browser and service (SSL/TLS or similar)
Local web service more detailed:

- User has installed local web service component in PC;
- User has smartcard or similar attached to PC;

- User initiates communication with web server where authentication or digital signature is needed;
- Web server script (JavaScript or similar) directs all authentication and digital signature related high level queries to local web service hosted on user PC;
- Local web service upon receiving requests forms card related commands (APDU or similar) and communicates with smartcard or similar token;
- User enters PIN for smartcard operation;
- Local web service on successful operation returns positive feedback;
- Web server allows user access to content or forms digital signature.

7.2.3 Alternative solution

As an example Estonian ID-software browser integration solution is described. In Estonian browser integration solution plugins are used for digital signing only, authentication is done thorough browsers native SSL/TLS authentication support.
Native browser plugins for major browsers are:

- IE BHO plugin;
- NPAPI plugin for Firefox, Safari and Chrome;
- old IE ActiveX for legacy systems;
- old Java Applet for legacy systems.

Plugins interact with smartcard thorough OS native API. In Windows with CryptoAPI, OSX and Linux with opensc-pkcs11. Plugins are distributed with Estonian ID-card middleware which contains all needed software included plugins. Plugins are integrated into systems via unified JavaScript calls.

Conclusion
It is not recommended to use local web service solution since:

- Security model of the solution has to be implemented separately since operating systems and browsers are not supporting such solution;
- There is no good reference (open-source and widely used) solution which has been proven and can be considered as example;
- Installation and binary compatibility issues on different operating systems.

7.3 PKCS #11 Module

This section describes how browser integration with PKCS #11 could be implemented. This approach is needed in some cases like the Estonian eID card, which uses a client certificate on the card in order to establish a TLS channel with the service provider.

7.3.1 Introduction

PKCS #11 [20] is part of the Public Key Cryptography Standards (PKCS) family of standards published by RSA Laboratories and defines a platform independent API for accessing cryptographic tokens, namely hardware security modules and smartcards. The API defines abstract object types (like signature keys) and functions, which operate on these objects (like “sign input data”).

Many applications use PKCS #11 as platform independent cryptographic interface, including the Mozilla Firefox web browser. Figure 2 shows its cryptographic architecture (Source: [21]). Cryptographic operations are handled by the Network Security Services (NSS), which use PKCS #11 as backend interface for private key and certificate management. Next to its internal crypto module (and a FIPS 140-1 compliant version), NSS also supports additional third-party PKCS #11 modules, which can serve as interface to smartcards or other cryptographic tokens.

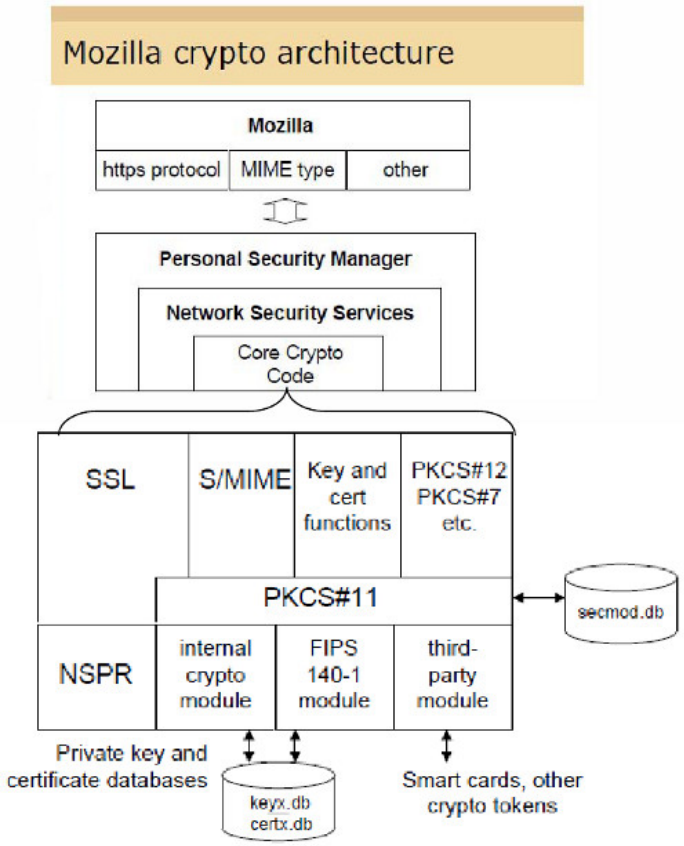

Figure 2: Cryptographic Architecture of Mozilla Firefox

7.3.2 Integration Technique

The FutureID client can be integrated into a browser using a PKCS# 11 module. This requires that it provides a library, which implements the PKCS #11 interface. As soon as this module is loaded into the browser, the FutureID client can be used to authenticate users to websites using TLS with client certificates. During session establishment the browser will then call the PKCS #11 module, which is connected to the FutureID client.

7.3.3 Evaluation

PKCS #11 can be used for those cases, where a client certificate is used to establish a TLS channel with the service provider. It further allows implementing secure session binding using client certificates (see Section 6.3). The integration with FutureID can either be done either by accessing the eID card directly with a suitable PKCS #11 module, or by implementing a generic PKCS #11 module, which acts as bridge to the FutureID client. The latter case is of course more flexible and would generally also allow FutureID to use secure bindings with other eID cards. For this case, the FutureID client could generate self-signed certificates with random keys on demand.

Third-party PKCS #11 modules are supported natively by some browsers like Mozilla Firefox and Chrome. These modules are, however, native libraries and thus dependant on the underlying operating system and hardware and therefore need to be adapted to each platform. Internet Explorer does not provide native support for PKCS #11, because Microsoft uses instead its own proprietary interface for cryptographic tokens called Cryptography API: Next Generation (CNG) [22], formerly CryptoAPI or CAPI. However, since the functionality is more or less the same, it should not be difficult to write a PCKS #11 wrapper. Mobile platforms also do not support PKCS #11 by default. This is addressed in the *SEEK for Android* project [23], which implements the OpenMobile API for secure element access on Android and also provides a service framework layer with PKCS #11 interface. However, this requires a security extension in the browser and it is not widely deployed on current Android systems by default.

7.3.4 Conclusion

Overall, PKCS #11 is not suitable as general browser integration method. Instead, it should only be used in special cases where this functionality is helpful, like with the Estonian eID card or in order to support secure bindings.

8. Requirements for Browser Integration

This section defines the requirements for the integration between the browser and the FutureID client. It is divided into technical requirements and security requirements.

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this section MUST be interpreted as described in RFC 2119 [24].

8.1 Technical Requirements

The following paragraphs present the mandatory and optional technical requirements.

8.1.1 Mandatory Requirements

No.	TR-BI-1 – Communication Channel
Description	The browser integration MUST provide a communication channel between the code of a website running inside the browser (JavaScript) and the FutureID client. The browser MUST be able to send commands to and receive corresponding responses from the client. A possibility to initiate communication from the client is not required and therefore OPTIONAL.

No.	TR-BI-2.1 – Platform Compatibility
Description	It MUST be possible to implement the browser integration solution of the FutureID client on all targeted platforms and browsers listed below (see Section 5): • Windows 7 32-bit: Internet Explorer, Google Chrome, Mozilla Firefox • Windows 8 64-bit: Internet Explorer (also Metro version), Chrome, Firefox • Ubuntu 64-bit: Firefox • MacOS 10.8: Firefox, Safari • Android 4.1: Chrome

No.	TR-BI-3 – Client Interface
Description	The syntax and semantics of the commands and responses exchanged between the browser and the FutureID client MUST be specified in a platform-independent interface. This will be done in task D36.2 *Interface and Module Specification and Documentation* of the FutureID project.

No.	TR-BI-4.1 – Efficiency
Description	The browser integration implementation of the FutureID client MUST be efficient. This means it MUST NOT introduce excessive delays or resource consumption to the FutureID client and the browser.

No.	TR-BI-4.2 – Reliability
Description	The browser integration implementation of the FutureID client MUST be reliable. This includes high availability.

No.	TR-BI-5.1 – Open Source License
Description	It MUST be possible to implement the browser integration solution of the FutureID client under the open source license as specified by the General Assembly. In particular it MUST NOT rely on proprietary or copyright restricted libraries or APIs.

No.	TR-BI-6 – Conflicts
Description	The browser integration solution of the FutureID client MUST NOT produce conflicts with other components in the browser. This includes for example another browser integrated application running at the same time.

8.1.2 Optional Requirements

No.	TR-BI-2.2 – Browser Version Compatibility
Description	The browser integration solution of the FutureID client SHOULD be compatible with new versions of the supported browsers. This means it SHOULD NOT be required to modify the client for each browser update. Ideally it SHOULD be completely independent of the underlying platform and require only standard web browser functionality.

No.	TR-BI-5.2 – Third-Party Software
Description	The browser integration solution of the FutureID client SHOULD not rely on any third-party software. Ideally it SHOULD only require the client application interface and a standard web browser to work. In case the browser integration does require third-party software, these components SHOULD be platform-independent. In accordance with requirement TR-BI-3, they MUST at least support all targeted platforms.

No.	TR-BI-7 – Client Startup
Description	The browser integration solution of the FutureID client SHOULD be able to start the client application from inside the browser. This can be used to start it automatically, in case the browser opens a website supporting FutureID login and the client is not already running.

8.2 Security Requirements

The following paragraphs present the mandatory and optional security requirements.

8.2.1 Mandatory Requirements

No.	SR-BI-1 – Security Compliance
Description	Independent of the chosen integration method, the browser integration solution MUST NOT violate or undermine any of the client security requirements as defined in deliverable D22.2 *Security Requirements Analysis*.

No.	SR-BI-2 – TLS Communication
Description	The browser MUST establish a secure TLS session to the service provider for handling the service login.

No.	SR-BI-3 – Integrity Check
Description	The browser integration solution MUST be able to verify the authenticity of the certificate and URI of the service provider and MUST deny any further action in case of a negative result.

No.	SR-BI-4 – Integrity Handling
Description	The browser integration solution MUST ensure the integrity of any data that is forwarded to the client for further processing. It MUST be able to detect any manipulation of data.

No.	SR-BI-5 – Attack Resistance
Description	The browser integration solution MUST be resistant against replay-attacks and man-in-the-middle attacks. Depending on the chosen integration solution, a secure connection between the browser and the client cannot be guaranteed. Therefore, the browser integration solution MUST NOT rely on a secure connection to realise this requirement.

No.	SR-BI-6 – Cross-Site Scripting
Description	Cross-site scripting (XSS) attacks MUST be prevented by the service provider website implementation.

8.2.2 Optional Requirements

No.	SR-BI-7 – HTML Verification
Description	The service provider SHOULD verify the signature of any HTML application used for FutureID login to ensure that it is not loaded from cache, offline storage or is an outdated version.

No.	SR-BI-8 – Secure Bindings
Description	The browser integration SHOULD support at least one of the secure binding methods for federated identity management protocols, which are discussed in Section 6. The implementation of secure bindings is OPTIONAL.

9. Conclusion

In this document we defined the technical and security requirements for integrating the FutureID client into web browser, starting with the platforms targeted and supported by FutureID. Then we discussed secure bindings for federated identity management protocols like SAML and how they can be implemented. Finally, we presented existing solutions for browser integration and evaluated their pros and cons.

Based on the evaluation we draw the following conclusion. The browser plugin method (see Section 7.1) is unpractical, because it highly depends on the platform, the browser and even its version. This is also the method that was chosen for the German eID client, called AusweisApp, and therefore caused the problems described in the introduction. The local web service method on the other hand is mostly independent from platform and browser, because it is based on standard web browser functionality. However, since anyone can open or connect to network ports, this communication channel requires mutual authentication before transferring security sensitive information. The third method, PKCS #11 module, has some interesting advantages, like support for secure bindings with client certificate, but it does not work with all platforms and eID cards.

Therefore we propose a hybrid solution. In general, we use the local web service method, as this basically runs on any platform. In special cases like the Estonian eID card, however, we try to use the PKCS #11 method, if it is available. Otherwise we fall back to the local web service method. Furthermore, in the course of this work package we want to investigate, if and how it is possible to combine secure session bindings with other eID cards, for example using the local web service for communication to the FutureID client and the PKCS #11 module for session binding only. As PKCS #11 modules are not supported by all targeted platforms, we also want to investigate, if we can replace PKCS #11, for example by managing certificates directly from JavaScript using the WebCrypto API.

10. Bibliography

[1] S. Asheuer, J. Belgassem, W. Eichhorn, R. Leipold, L. Licht, C. Meinel, A. Schanz and S. Maxim, "Akzeptanz und Nutzerfreundlichkeit der AusweisApp: Eine qualitative Untersuchung," Universitätsverlag Potsdam, Potsdam, 2012.

[2] Heise Online, "Rechnungshof rügt BSI für AusweisApp-Schlamperei," [Online]. Available: http://www.heise.de/newsticker/meldung/Rechnungshof-ruegt-BSI-fuer-AusweisApp-Schlamperei-1848234.html.

[3] Bundesministerium des Innern, *Verordnung über Personalausweise und den elektronischen Identitätsnachweis (Personalausweisverordnung PAuswV),* 2010.

[4] J. Camenisch and B. Pfitzmann, "Federated Identity Management," in *Security, Privacy, and Trust in Modern Data Management*, Springer Berlin Heidelberg, 2007, pp. 213-238.

[5] D. Pfitzmann and M. Waidner, "Analysis of Liberty Single-Sign-On with Enabled Clients," *IEEE Internet Computing,* vol. 7, no. 6, pp. 33-44, 2003.

[6] S. Gajek, J. Schwenk and X. Chen, "On the Insecurity of Microsoft's Identity Metasystem Cardspace," Horst Görtz Institute for IT Security, 2008.

[7] OASIS, "OASIS Security Services (SAML) TC," [Online]. Available: https://www.oasis-open.org/committees/security.

[8] OpenID Foundation, "OpenID Authentication 2.0 - Final," [Online]. Available: http://openid.net/specs/openid-authentication-2_0.html.

[9] T. Groß, "Security Analysis of the SAML Single-Sign-On Browser/Artifact Profile," in *Annual Computer Security Applications Conference*, 2003.

[10] OASIS, "Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) v2.0," [Online]. Available: http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf.

[11] OASIS, "Profiles for the OASIS Security Assertion Markup Language (SAML) v2.0," [Online]. Available: http://docs.oasis-open.org/security/saml/v2.0/saml-profiles-2.0-os.pdf.

[12] S. Gajek, L. Liao and J. Schwenk, "Stronger TLS Bindings for SAML Assertions and SAML Artifacts," in *Proceedings of the 2008 ACM Workshop on Secure Web Services* , 2008.

[13] OASIS, *SAML V2.0 Holder-of-Key Web Browser SSO Profile,* 2010.

[14] F. Kohlar, J. Schwenk, M. Jensen and S. Gajek, "Secure Bindings of SAML Assertions to TLS Sessions," in *International Conference on Availability, Reliability and Security*, 2010.

[15] Federal Office for Information Security (BSI), "Technical Guideline TR-03124-1 eID-Client – Part 1: Specifications".

[16] Q. Community, "https://community.qualys.com/docs/DOC-1542#s2_q1," 18 08 2010. [Online]. [Accessed 24 10 2013].

[17] G. Hornung, M. Horsch and D. Hühnlein, "Mobile Authentisierung und Signatur mit dem Personalausweis," *Datenschutu und Datensicherheit,* pp. 189-194, 3 2012.

[18] M. Horsch, Mobile Authentisierung mit dem neuen Personalausweis, Darmstadt, 2011.

[19] T. Hühnlein, "eID-Aktivierung," [Online]. Available: https://www.openecard.org/de/framework/eid-activation.

[20] RSA Laboratories, "PKCS #11: Cryptographic Token Interface Standard," [Online]. Available: http://www.rsa.com/rsalabs/node.asp?id=2133.

[21] R. Griffin, "Encryption and Key Management Tutorials, Part II: PKCS #11 – Enhancements and Opportunities," in *RSA Conference*, 2009.

[22] Microsoft, "Cryptography API: Next Generation," [Online]. Available: http://msdn.microsoft.com/en-us/library/windows/desktop/aa376210%28v=vs.85%29.aspx.

[23] SEEK for Android Team, "The SEEK for Android Project," [Online]. Available: https://code.google.com/p/seek-for-android/.

[24] IETF, *RFC 2119: Key Words for use in RFCs to Indicate Requirement Levels,* 1999.