D4.4 Smartphone feasibility analysis

Jonas Lindstrøm Jensen

Editors:	*Jonas Lindstrøm Jensen (Alexandra Institute AS)*
Reviewers:	*Fatbardh Veseli (Johann Wolfgang Goethe – Universität Frankfurt), Michael Østergaard Pedersen (Miracle A/S)*
Identifier:	*D.4.4*
Type:	*Deliverable*
Version:	*1.0*
Date:	*05/08/2014*
Status:	*Final*
Class:	*Public*

Abstract

This deliverable documents the work done on analysing whether it is possible to realize parts of the ABC4Trust reference implementation on mobile platforms.

The main contributions are three prototypes: A version of the User side of the reference implementation ported to the Android platform, an application that enables an NFC-enabled Android device to emulate a smart card in the reference implementation and finally an implementation of the User side of Microsoft's U-Prove in JavaScript for use in web applications.

This document describes the prototypes and how they were developed and analyses the performance and security of them.

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 257782 for the project Attribute-based Credentials for Trust (ABC4Trust) as part of the "ICT Trust and Security Research" theme.

Members of the ABC4TRUST consortium

1.	Alexandra Institute AS	ALX	Denmark
2.	CryptoExperts SAS	CRX	France
3.	Eurodocs AB	EDOC	Sweden
4.	IBM Research – Zurich	IBM	Switzerland
5.	Johann Wolfgang Goethe – Universität Frankfurt	GUF	Germany
6.	Microsoft Belgium NV	MS	Belgium
7.	Miracle A/S	MCL	Denmark
8.	Nokia Solutions and Networks GmbH & Co. KG	NSN	Germany
9.	Computer Technology Institute and Press "Diophantus"	CTI	Greece
10.	Söderhamn Kommun	SK	Sweden
11.	Technische Universität Darmstadt	TUD	Germany
12.	Unabhängiges Landeszentrum für Datenschutz	ULD	Germany

List of Contributors

Chapter	Author(s)
Executive Summary	Jonas Lindstrøm Jensen (Alexandra Institute AS)
Introduction	Jonas Lindstrøm Jensen (Alexandra Institute AS)
First Chapter	Jonas Lindstrøm Jensen (Alexandra Institute AS)
Second Chapter	Jonas Lindstrøm Jensen (Alexandra Institute AS)
Third Chapter	Jonas Lindstrøm Jensen (Alexandra Institute AS)
Fourth Chapter	Jonas Lindstrøm Jensen (Alexandra Institute AS)
Fifth Chapter	Jonas Lindstrøm Jensen (Alexandra Institute AS)

Foreword Executive Summary

One of the main objectives of the ABC4Trust project is to develop a reference implementation of a Privacy-ABC architecture. The reference implementation has been developed with laptop and desktop computers in mind as the User's platform, and the contribution of this deliverable is an analysis of the feasibility of realizing parts of the reference implementation on mobile devices. The question of feasibility is relevant since mobile devices have limited performance compared to laptops and desktops.

We have focused on the User-side of the reference implementation, since it in most use cases will be the only entity for which it makes sense to use a mobile device, and it turns out that it is indeed possible to get an acceptable performance by simply moving the User-side of the reference implementation to a smart phone.

The deliverable also presents and analyses two additional prototypes: One that enables a NFC-enabled Android device to emulate the smart cards used in the reference implementation and another that is an implementation of Microsoft's U-Prove in JavaScript for use of privacy preserving ABCs in web applications.

This intended audience of this document is anyone interested in what work the ABC4Trust project has done on mobile platforms, and privacy on mobile devices in general. Note that the document does not contain a discussion of what a Privacy-ABC technology is, neither does it contain a full description of the ABC4Trust reference implementation, so readers not familiar with these should consult the deliverables D2.1 and D4.2 respectively.

Table of Contents

Index of Figures

Index of Tables

1 Introduction

The ABC4Trust reference implementation (see D4.2) has been developed with computers and laptops in mind as the intended platform of the User. However, during recent years, Users have embraced smart phones and tablets on massive scale, and expect the same functionality on these devices as they have on their laptops and desktops. In particular they expect the same privacy features so it is natural to consider how the ABC4Trust reference implementation can be used on mobile devices.

Another reason why we consider integration with mobile devices is that the User is carrying a smart phone at all times, and can potentially use it as an *identity hub,* keeping credentials on the device and use it to authenticate towards different verifiers while at all time staying in full control of what information is revealed to whom. The mobility of the device opens for a range of new use cases not available to laptops and desktops since the User is not limited to the physical restrictions of a laptops and desktops, and can e.g. use her smart phone to prove that she is old enough to buy alcohol at a shop or that she is allowed access to a building, both without revealing any other information than absolutely necessary.

However, mobile devices do not offer the same computational performance as laptops and desktops, and since the cryptographic computations done when using Privacy-ABCs are computationally intensive, we need to analyse whether it is possible to use mobile devices as a platform for Privacy-ABC's and still get an acceptable performance. This analysis has been conducted by implementing several prototypes where mobile devices are used in setups involving Privacy-ABCs. We also discuss what security issues should be considered when working on a mobile platform.

In chapter 2 we describe how the ABC4Trust reference implementation, which was developed for laptop or desktop computers, has been ported to the Android platform. We describe what issues were faced and analyse the performance and security of the implementation.

Some smart phones contain a NFC-chip, and can use it to emulate a smart card. In chapter 3 we describe how we used this feature to make an Android phone emulate the smart cards that were used in the reference implementation.

The smart phone world is very fragmented and there are several widely used platforms, most notably Android, iOS, Windows Phone and BlackBerry. It is, however, possible to create applications that can run on all these platforms by implementing them as web applications in JavaScript. This also has the added benefit that the User does not have to install anything on her device, but can execute the application in a web browser. In chapter 4 we describe how we have implemented a Privacy-ABC technology, Microsoft's U-Prove [PZ13], where the application running on the User's device is developed in JavaScript and is running in a web browser.

2 Reference implementation on Android

2.1 Introduction

In order to analyse the possibility of realising parts of the reference implementation on smartphone platforms, we have implemented the part of the reference implementation that is supposed to run on the User's device as an app for the Android platform. We focus on the User's device since the only entity using a mobile device in a setup involving Privacy-ABC technologies in most use cases will be the User. However, the implementation could easily be adapted to make the mobile device act as other entities, e.g. Issuer or Verifier, should this be relevant in some use cases. The functionality of the implementation is discussed in section 2.2.

The Android platform was chosen over other smart phone platforms for convenience − the reference implementation is developed in Java, which can also be used to create Android applications, so realising the reference implementation on the Android platform did not involve writing a lot of new code. However, the Android platform is slightly different from the Java platform, so the task mainly consisted of making sure that the situations where the differences between the Android API and the Java API could cause trouble was addressed and making sure that the third-party libraries used by the reference implementations were available on Android. These issues are explained in more detail in section 2.3.

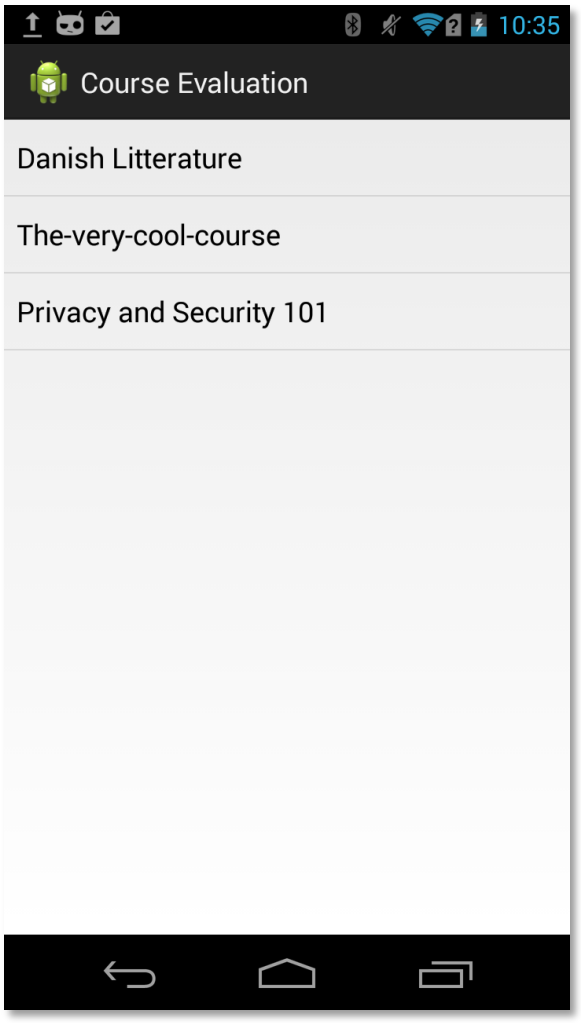

Figure 1: The UI of the demo client application, which is an app for evaluating courses at a university.

Successfully porting the reference implementation to the Android platform certainly implies that it is possible to integrate the User's side of the reference implementation on mobile platforms, but a remaining question is whether the performance and security model are acceptable in this setup. Mobile devices are in general somewhat slower than laptops and desktops, to whom the reference implementation was originally intended, so we have performed some benchmarks to see if we can get an acceptable performance with the Android version. This is discussed in section 2.4. Further, mobile devices are vulnerable to some attacks that are not as significant for laptops and desktops, e.g. the possibility of having a device stolen. This is discussed in section 2.5, where we analyse the security implications, both of the Android implementation and, more generally, the possibility to create a secure implementation of Privacy ABC-technology for mobile devices.

2.2 Overview of the prototype

The high-level idea behind the Android implementation is to create an application (*ABC4Trust app*) that stores the User's credentials and acts as a service for other applications, performing authentications on their behalf on demand.

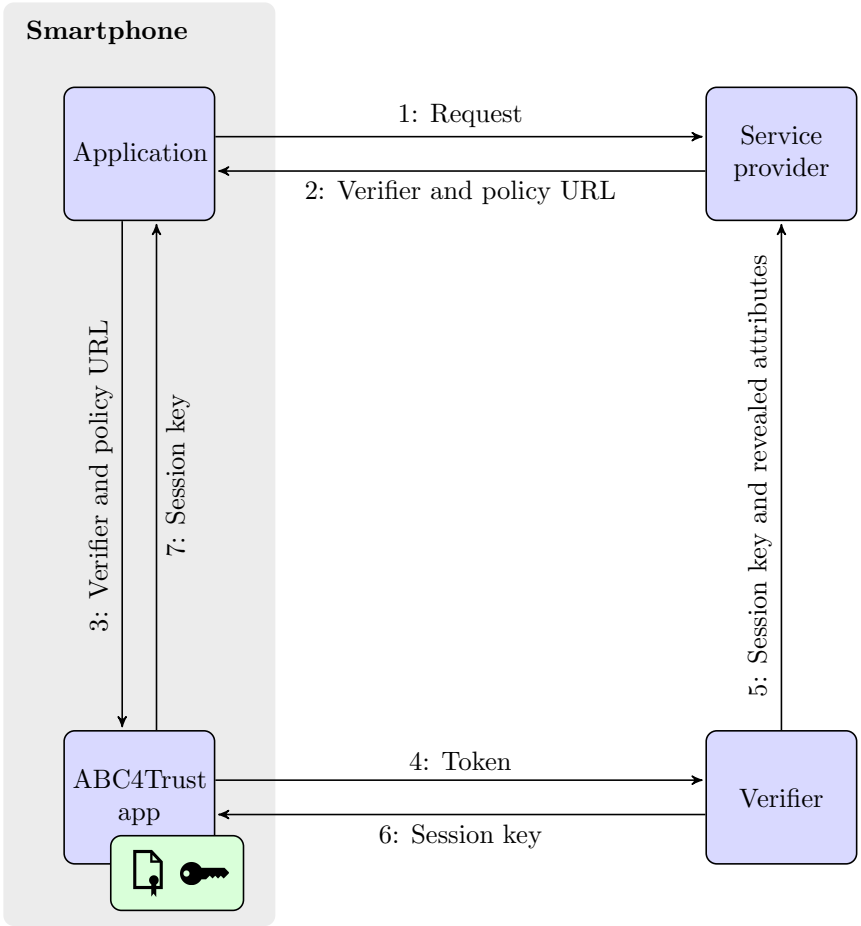

Figure 2: The communication flow when performing a presentation.

The communication flow when performing a presentation is shown in Figure 2. To illustrate it, we will run through a demo of an app that enables the User to evaluate a university course he has attended. For this purpose, the User uses the ABC4Trust app to prove towards the university, that he is a student at the specified course and that he only evaluates the course once.

1. The User chooses to evaluate a course through an app (*Application*) (see Figure 1).

2. The university (S*ervice provider*) needs the User to prove that he possesses a credential showing that he is a student at the chosen course, as well as a credential showing that the User is a student at the university. Finally the User also needs to present a scope exclusive pseudonym to make sure that it is only possible to evaluate the course once. The service provider responds with an URL to a policy describing these demands and an URL to a *Verifier* to whom the presentation token should be sent. The connection between the User and the Service Provider should be secure (e.g. using SSL), such that the URL cannot be altered in transit.
3. The client application forwards this information to the *ABC4Trust app*.
4. If the ABC4Trust app has the needed credentials, it presents the User with a graphical user interface (see Figure 3) asking him if he wants to reveal the attributes the service provider requests – that is showing that he is a student at the university and is following the course, as well as a scope exclusive pseudonym for the course. If the User accepts, the ABC4Trust app creates a presentation token containing this information, and sends it to the Verifier, and meanwhile the User is presented with another UI showing that this a presentation is currently being performed.
5. If the presentation token is OK, the Verifier sends the revealed attributes to the university along with a newly generated session key.
6. The session key is also send to the ABC4Trust app.
7. The ABC4Trust app forwards the session key to the client application, which can now continue the interaction with the university.

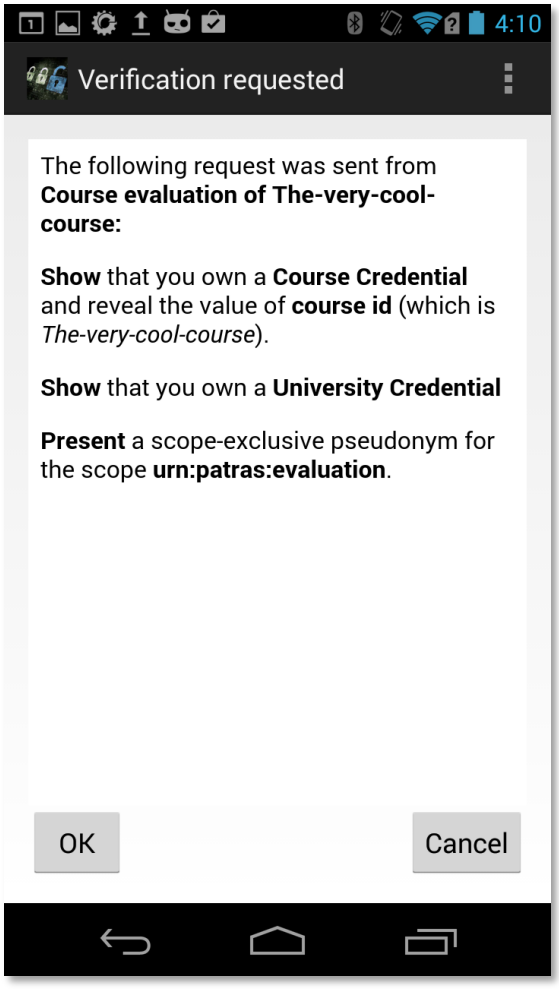

Figure 3: The ABC4Trust app presents the User with a UI showing what information she is about to reveal.

One of the key ideas behind the original reference implementation is that it should be easy for developers to integrate, and this idea is preserved here as the client application's role is very simple: it forwards the received URL's to the ABC4Trust app and gets a session key in response.

Starting the ABC4Trust app will show a list of the credentials currently stored on the device as shown in Figure 4.

2.3 Implementation issues

The reference implementation has been developed in Java, which can also be used to create apps for Android, so realising the reference implementation on Android did not involve much additional development. It did, however, require some work to make the code workable on Android, since the Android API and Java API are not completely similar, and not all third-party libraries available in Java are also available on Android.

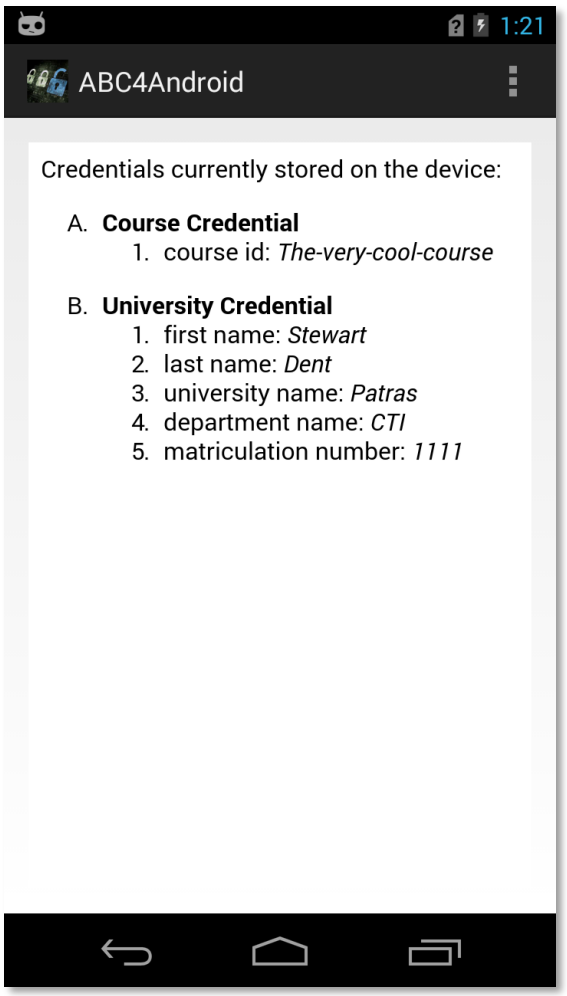

Figure 4: The ABC4Trust app can show the User what credentials are currently stored on the device.

One third-party library that caused problems was Java Architecture for XML Binding[1] (*JAXB*), which is part of the Java SE platform, but not available on Android. Luckily, we were not the first to face this problem, and an unofficial Android version has been made[2], which we included in the ABC4Trust app. However, Java does not allow you to add classes in a number of packages, for example in the `javax.xml.*` package which is the one used by JAXB, so this unofficial version can not use the same package name as JAXB, and some small changes had to be made to the original source code to include this library, namely changing the import statements, and repacking the unofficial JAXB such that the classes are in another package, for example `ae.javax.xml.*`. More specifically, statements of the form

```
import javax.xml.*;
```

have to be changed to

```
import ae.javax.xml.*;
```

The functionality of the unofficial JAXB library is limited, and does for example not allow canonicalization of XML because the functionality depends on *Apache Xerces*[3] and *Apache Santuario*[4], which are two libraries that are also not officially available on the Android platform. Apache Xerces has been unofficially ported[5], but Apache Santuario has not. The Java source code for this library is, however, available and creating an Android version can be done by simply interpreting the Java source code as an Android library. As in the case for the unofficial JAXB library, the package names for these two unofficial Android versions had to be renamed.

Further, file handling is a little bit different in Android, where an application only has access to its own internal storage, and each class that needs to access these files needs to know the path to the internal storage, which is available by calling `getFilesDir()` on the applications `Context`-object[6]. The `Context`-object is only available from the actual Android application and not from imported libraries, so the path has to be distributed to the classes in the reference implementation that needs to access files.

Note that the modifications needed to make the source code of the reference implementation run on Android are minimal and mainly involve changing package imports, but the modifications are nonetheless necessary, making it impossible to completely reuse the original source code in its current state on Android.

2.4 Performance

The CPUs and memory in mobile devices are in general slower than the ones in laptops and desktops. This could be an issue when implementing privacy ABC technologies on mobile platforms, since these technologies involve some rather complicated cryptographic computations. So in order to determine whether it is feasible to use mobile devices, we need to check whether this gap in performance is big enough to give a notably worse User experience.

However, our benchmarks show that creating a presentation token on an Android device takes about 6 seconds when revealing attributes from one credential. In the demo application this is the time from the User accepts the presentation policy to the token is sent to the verifier, and for most use cases this should be acceptable.

---

[1] https://jaxb.java.net/
[2] http://www.docx4java.org/blog/2012/05/jaxb-can-be-made-to-run-on-android/. This website also contains a discussion on why JAXB is not working on Android.
[3] https://xerces.apache.org/
[4] https://santuario.apache.org/
[5] https://code.google.com/p/xerces-for-android/
[6] https://developer.android.com/reference/android/content/Context.html

As discussed in the introduction of this section, the ABC4Trust reference implementation was not created with mobile devices in mind, which leaves some room for potential performance improvements on mobile platforms. Note also that mobile devices have developed rapidly over the last few years, so performance will also depend on the brand and age of your device.

2.5 Security analysis

The intended platform for the User in the ABC4Trust reference implementation is a laptop or desktop with an attached smart card reader. The User is given a smart card which is used both to store the User's credentials, such that she can use it with any laptop that has the reference implementation installed, but an important feature of a smart card is also that it is tamper resistant, meaning that it is difficult to extract the key from the smart card.

In the ABC4Trust app, both keys and credentials are stored in the ABC4Trust app's internal storage in the device's memory. In the Android security model, this data is only accessible by the app it self, and should hence be secure from other apps. However, an adversary with physical access to the device can easily extract this data, and if Android's data encryption is broken, the adversary can retrieve both keys and credentials and hence create presentation tokens on behalf of the User. The User herself can also use this to share her credentials with other Users.

In most scenarios the security model where keys and credentials are stored in the device's memory are acceptable, but in high-risk scenarios one could improve the security by utilizing a hardware module in the mobile device, for example a SIM card or a TPM-module, for storing the User's private key. However, direct interaction with such hardware modules is in general not available through the standard API's, and the API's that does allow some access, e.g. Androids KeyStore[7], do not support the functionality needed for our purpose, so we have not been able to test it nor implement it in the prototype.

[7] https://developer.android.com/reference/java/security/KeyStore.html

3 Smart card emulation

3.1 Introduction

In the two ABC4Trust pilot projects in Patras and Söderhamn (see D7.1 and D6.1 respectively for more details), the Users were equipped with smart cards. In Patras, university students had to present their smart card every time they attended a course, and at the end of the semester they were allowed to do an anonymous evaluation of the course if their course attendance was high enough.

In Söderhamn, pupils and teachers at an elementary school used the smart cards to store their personal credentials, and using their smart card they could log into a school chat forum from any computer. The chat forum had different chat rooms that were only accessible for certain pupils, e.g. of a certain age or pupils following a specific course.

Basically the smart cards in these pilots have two purposes:

1. To act as a tamper resistant second factor during issuance and presentation, performing cryptographic operations involving the User's private key without revealing it.
2. For storing the User's credentials, making her able to perform both issuance and presentations from any laptop with a card reader.

So the reason for using smart cards is both security (1) and better user experience (2). A custom smart card application was developed for the pilots by CryptoExperts SAS [BDP12] and deployed on the smart cards, which were then handed out to the Users.

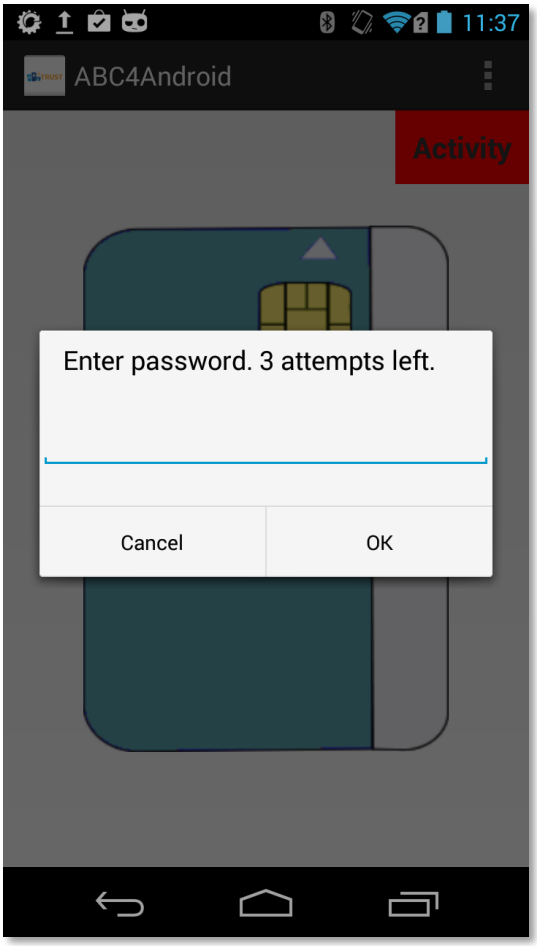

Figure 5: The smart card is protected by a password, which the User has to enter when starting the app.

Some mobile devices have a Near Communication Field chip (*NFC*). This chip can be used to communicate with other NFC-enabled devices, but it can also be used to interact with smart card readers. This allows the mobile device to act as a smart card (known as *Smart card emulation*).

Smart cards have a few limitations. They contain CPUs and are able to perform computations on the card, but they do not contain a power supply and must use the power from the smart card reader. This causes the smart card to have rather limited performance, and the performance bottleneck in the pilot projects was in some cases the smart card. Furthermore, the memory of a smart card is limited, which is an issue when a User wants to store many credentials on one card.

A mobile device does not have these limitations. Furthermore, a User is carrying a mobile device with her at all time, so using this device instead of requiring her to carry a smart card with her too, will also give a better User experience. However, as discussed in section 2.5, a mobile device does not have the same security advantages as a smart card, unless a tamper resistant hardware module, for example a SIM card (which is actually a smart card) is used.

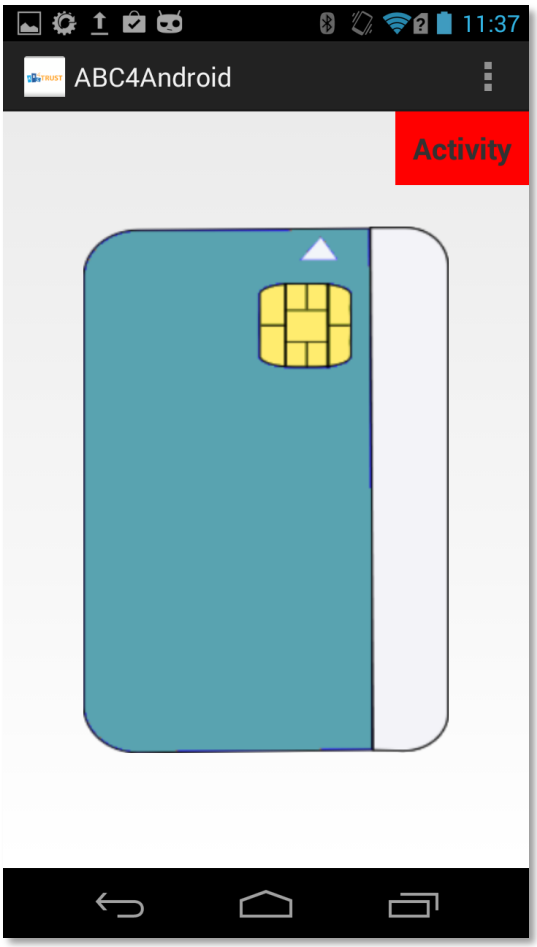

Figure 6: The app only has one User interface, which displays a picture of a smart card and an activity LED in the top-right corner.

As a proof-of-concept, we have implemented a prototype, which is the smart card application, ABC4Trust Lite [BDP12], implemented as an Android application, which uses smart card emulation to interact with a smart card reader.

3.2 Overview of the application

The idea behind the prototype is to create an application that the User should be able to install on an NFC enabled mobile device, and then use this device instead of a smart card. Furthermore, the User service running on the laptop with an attached smart card reader should not require any changes, allowing the use of mobile devices instead of smart cards in existing systems.

The application itself is a very simple Android application. When the application is started for the first time, the User is requested to create a new virtual smart card. The User is then requested to enter a password that is used to encrypt the virtual smart card when it is stored on the mobile device, and should be entered every time the application is started as shown in Figure 5.

When the application is running, the UI displayed in Figure 6 is shown and this is the only UI the User is ever presented to. Through the menu in the top right corner, the User can request *Info,* which causes the application to display some information about what is currently stored on the 'smart card'. Through this menu the User can also wipe all information from the smart card.

Figure 7: An NFC enables mobile device running the app can communicate with smart card readers that support contactless communication.

When the User wishes to use her mobile device as a smart card, the application is started and the mobile device is put on a smart card reader that supports contactless communication (see Figure 7). When the application interacts with the smart card reader, the indicator in the top right corner flashes green.

3.3 Implementation issues

Smart card emulation has been used on Android by Google's app Google Wallet[8] for quite a while, but this feature has only been available through the standard Google API since Android KitKat 4.4. We developed the prototype before this was released (in September 2013), and so we had to use the alternative Android platform, CyanogenMod[9], where smart card emulation was available at the time.

The concept of the prototype is that the mobile device should be able to be used interchangeably with a real smart card in the ABC4Trust reference implementation. This was almost fulfilled, since the NFC chip embedded in the Android device, which is used to communicate with the smart card reader, does not support extended APDUs (the ability to receive more than 255 bytes from the card reader in one response), so in order not to decrease the performance for regular smart cards, the User service has to ask the device it is currently communicating with, whether it is an Android device emulating a smart card, and if so, not use extended APDUs.

Even though we had to discard extended APDUs for mobile devices, benchmarks show an increase in performance of about a factor six, compared to regular smart cards.

3.4 Security analysis

One of the reasons for using smart cards is that they offer tamper resistant storage of credentials and keys. As discussed in section 2, the memory of an Android device, where this prototype stores its keys and credentials, is not tamper resistant, so the prototype does not have the same security advantages.

The keys and credentials are encrypted under a password, which the User has to enter when starting the app. If the User enters the password incorrect three times, the content of the smart card is deleted. However, if the file containing the encrypted virtual smart card data is extracted from the device, an adversary could launch an offline attack on the encryption without being limited to three attempts. One way to enhance the security could be to encrypt the virtual smart card under a key stored in the Android KeyStore, which on some platforms is hardware based. This would allow keys with higher entropy than the ones generated using a User's password, and the key would be protected in tamper resistant hardware.

[8] http://www.google.com/wallet/
[9] http://www.cyanogenmod.org/

4 Privacy ABC's in web applications

4.1 Introduction

The world of mobile devices is very fragmented in the sense that there are multiple widely used platforms, most notably Android, iOS, Windows Phone and Blackberry. A way to create applications that can run on all platforms is to implement it as a web application using JavaScript instead of as a native application.

A web application is executed in a web browser, and if implemented in a standardized format, such as JavaScript, it is able to run on any device that has a web browser, including smart phones and tablets. The downside is that a web application has very limited access to the device's hardware and memory and that the performance is not as good as when running a native application. Furthermore, the web application has to be downloaded from some provider, e.g. the Issuer, every time it is needed, giving a more complicated security scenario.

4.2 Overview

We have created a prototype implementation of Microsoft's U-Prove [PZ13] in JavaScript, to study whether it is feasible to use Privacy ABC technologies with web applications. U-Prove is one of the Privacy-ABC technologies used in the ABC4Trust reference implementation, the other being IBM's Identity Mixer[10]. U-Prove was chosen over Identity Mixer for this prototype because it is simpler and has more limited functionality, but it still has the key features of Privacy ABC's, e.g. selective disclosure.

As in the other prototypes presented in this deliverable, we have focused on the User, since it in many use cases is the only entity for which it is relevant to use a mobile device.

The prototype is able to perform the cryptographic operations needed to get credentials from Issuers and use these to create presentation tokens and send them to Verifiers. Note that in the U-Prove terminology this is translated into that we have implemented the *Prover* entity, which is able to receive *Tokens* from *Issuers* and use these to create *Presentation proofs* and send them to *Verifiers*.

A web application that wishes to use the library first has to import the libraries JavaScript files and then create a new Prover-object:

```
var prover = new Prover();
```

This object can be used to get credentials and create presentation proofs. The following command request three tokens (in U-Prove a token can only be used once, so it is practical to issue multiple tokens at a time) from an issuer with URL 'http://127.0.0.1:8080' with the four attributes 'Jonas', 'Jensen', 'Aarhus' and 'Denmark', and stores them in the browsers local storage under the key 'token1':

```
prover.issueToken('http://127.0.0.1:8080',
      ['Jonas', 'Jensen', 'Aarhus', 'Denmark'], 'token1', 3);
```

The following command creates a presentation proof based on the credential stored under the key 'token1', revealing attributes 1 and 2, committing to the value of attribute 3, using attribute 4 as a pseudonym and 'ABCD' as scope. This proof and the token is then sent to a verifier with URL 'http://127.0.0.1:8080'.[11]

[10] http://www.zurich.ibm.com/idemix/

[11] For more information about these terms, see the U-Prove specifications at http://www.microsoft.com/u-prove.

```
prover.presentToken('http://127.0.0.1:8080', 'token1', [1,2], [3], 4,
      'pseudo', 'ABCD')
```

Note that here the Issuer and Verifier has the same URL, since they in our example are running on the same server. This is only the case for this example, and is not a general requirement – they can run on different servers.

The functions described above interact with issuers and verifiers which exposes the functionality of Microsoft's own C# implementation of U-Prove [UPSDK] as web services, as shown in Table 1.

Relative URL	Web service description
`/Issuer.asmx/GetParameters`	Get the Issuer parameters for this Issuer.
`/Issuer.asmx/GetFirstMessage`	Request first issuance. This request should be sent along with the following parameters: `parameters` An array of the desired attributes represented as strings. `noOfTokens` The number of tokens the Issuer should create.
`/Issuer.asmx/GetThirdMessage`	Request the third issuance. The following parameter should be sent along with the request. `secondMessage` The second issuance message as JSON.
`/Verifier.asmx/PresentToken`	Send a presentation proof and token to a verifier. `token` A token encoded as JSON. `proof` A presentation proof encoded as JSON.

Table 1: This table shows the methods of the U-Prove web service. For details about the contents of the presentation tokens and proofs and the issuance messages, see the U-Prove documentation.

4.3 Implementation issues

The prototype has to perform calculations with arbitrarily large integers, which is not supported in JavaScript as standard, so we had to find a library to do this. Most available third-party libraries lack important features and proper documentation or are simply buggy, and we ended up choosing *JSBN*[12], because it is documented, and because it supported some features that we needed, e.g. elliptic curves and conversion to and from Base64 strings. A missing feature is hashing with SHA-256 (JSBN supports SHA-1). Again several JavaScript implementations were available, and we ended up using one made by Chris Veness[13] due to its non-restrictive license (CC by 3.0). We also had to implement a few additional features, for example an algorithm for calculating modular square roots.

A lot of effort was put into making the implementation compatible with Microsoft's own implementation of U-Prove, especially in making sure that hashing is done in the same way.

[12] http://www-cs-students.stanford.edu/~tjw/jsbn/
[13] http://www.movable-type.co.uk/scripts/sha256.html

4.4 Benchmarks

The prototype has been used to study whether it is possible to get a descent performance when using privacy ABC's in JavaScript – recall that application in JavaScript are much slower than native applications.

So far only the elliptic curve construction has been tested. This is because one can use smaller key sizes when using elliptic curves than when using subgroups of $\mathbb{Z}_p$.

For the issuance (*Precomputations* and *Create second message*[14]), a token with four attributes is issued. For the presentation proof (*Create presentation proof*), two of the attributes are disclosed, and none is committed.

Device	Model	CPU	Clock Freq.	RAM	OS
Laptop	MacBook Pro	Intel Core i7	2.8 GHz	16 GB	OS X 10.8.5
Android	Samsun Galaxy Nexus	ARM Cortex-A9	1.2 GHz	1 GB	CyanogenMod 10.2.0
iPhone	iPhone 5	Apple Swift Dual Core	1.3 GHz	1 GB	iOS 7.0.3

Table 2: A list of the devices used in the benchmark.

Below are some benchmarks on different platforms. Each column represents different elliptic curves yielding different key size (256, 384 and 521 bits respectively). See Table 2 for specifications of the used devices.

Operation	P-256	P-384	P-521
Precomputations	100 ms	194 ms	376 ms
Create second message	36 ms	110 ms	230 ms
Create presentation proof	73 ms	189 ms	408 ms

Table 3: Benchmark for the Laptop running Google Chrome.

[14] The computations done by the User during an issuance is the sum of *Precomputations* and *Create second message*. For details aboud what these include, consult the U-Prove specifications.

Operation	P-256	P-384	P-521
Precomputations	2096 ms	5209 ms	10922 ms
Create second message	1033 ms	3293 ms	7162 ms
Create presentation proof	1831 ms	5675 ms	13093 ms

Table 4: Benchmark for the Laptop running Apple Safari.

Operation	P-256	P-384	P-521
Precomputations	101 ms	191 ms	343 ms
Create second message	44 ms	104 ms	223 ms
Create presentation proof	94 ms	181 ms	437 ms

Table 5: Benchmark for the Laptop running Mozilla Firefox.

Operation	P-256	P-384	P-521
Precomputations	1248 ms	2584 ms	4919 ms
Create second message	556 ms	1457 ms	3098 ms
Create presentation proof	903 ms	2537 ms	5528 ms

Table 6: Benchmark for the Android device.

Operation	P-256	P-384	P-521
Precomputations	10412 ms	25797 ms	59752 ms
Create second message	5198 ms	15740 ms	39783 ms
Create presentation proof	9068 ms	27705 ms	69396 ms

Table 7: Benchmark for the iPhone device.

From the benchmarks in the above tables, we see that the performance varies greatly from platform to platform. However, 9 seconds (which is the time spent on a presentation on iPhone on P-256, see

Table 7) would be acceptable in many use cases, so it is feasible to create an application that can perform acceptable on all the platforms we have tested.

4.5 Security analysis

Depending on the use case, using a curve with key size 256 bits, which is an acceptable security level, even for long-term protection, could give an acceptable performance (at most about 16 seconds for an issuance and 9 seconds for a presentation) but choosing higher security levels is not possible, as this will give a non-acceptable performance on some devices (28 seconds for a presentation on an iPhone, see Table 7).

The keys and credentials are currently stored in the browser's local storage, and are protected by a same-origin policy, meaning that the data is only accessible by scripts running from the same domain as the script that stored the data. However, the credentials are vulnerable to attacks on the devices permanent memory, e.g. from adversaries with physical access to the device or malware.

The User has to get the script from this domain every time she wants to use it. This goes against the philosophy of privacy ABC's, since the host of the domain now knows exactly when she is using it. The User should also somehow verify that she received a valid script (e.g. by checking its signature) each time it is downloaded.

4.6 Future work

Google has recently (June 3rd, 2014) released the source code to a project called *End-to-End*[15], which is a Chrome extension giving the possibility to do encryption within the browser, and a part of this is a newly created JavaScript crypto library. Also Microsoft has released a cryptographic library in JavaScript recently[16]. As discussed in section 4.3, we had trouble finding a well supported crypto library, but these new libraries now seem like obvious candidates for future projects. However, Microsoft has also recently released an implementation of U-Prove in JavaScript[17], which is good news since it gives a positive answer to the main question addressed in this chapter, namely that it is possible to use Privacy-ABC's in real world web applications.

[15] https://code.google.com/p/end-to-end/
[16] http://research.microsoft.com/en-us/downloads/29f9385d-da4c-479a-b2ea-2a7bb335d727/
[17] https://research.microsoft.com/en-us/downloads/1008a07e-5cc6-4a96-a6f1-4f26dadb317b/

5 Conclusion

They main task of this deliverable has been to study if it is possible to run parts of the reference implementation on mobile platforms. In chapter 2 we have described a prototype that does exactly this on the Android platform, so it must be concluded that it is indeed possible.

It has not been possible to test the possibility of using a secure element, such as a SIM-card or TPM-chip, since access to these is either not available through the standard APIs or do not have the specific functionality needed for our purpose.

This deliverable also contains descriptions of a prototype of an application that enables a NFC-enabled Android device to emulate a smart card as used in the reference implementation. In some scenarios this might be useful, especially because the User already is carrying a smart phone with her at all time.

A third prototype described is an implementation of the User side of Microsoft's U-Prove in JavaScript, showing that is indeed feasible to use Privacy ABC technology in web applications on the client side.

Possible future work includes realizing a secure element with the implementation, and a full demo of a web application using Privacy-ABCs technologies with some of the newly released libraries from Google and Microsoft.

6 Bibliography

[BDP12] Thomas Baignères, Cécile Delerablée and Pascal Paillier, Programming Privacy-ABCs on theABC4Trust Lite v1.0 Smart Card (To be published).

[PZ13] Christian Paquin and Greg Zaverucha ,U-Prove Cryptographic Specification V1.1 (Revision 3).

[UPSDK] U-Prove Crypto SDK V1.1 (C# Edition), http://uprovecsharp.codeplex.com/