MobCom
A Mobile Companion

D.3.1 Development of Architecture
D.4.1 Applications I
D.5.1 Validation Requirements

Faysal Boukayoua (editor), Italo Dacosta, Bart De Decker, Jorn Lapon, Luc Martens, Milica Milutinovic, Vincent Naessens, Bart Preneel, Andreas Put, Stefaan Seys, Dave Singelee, Kris Vanhecke (editor), Jan Vossaert

December 2012

KATHOLIEKE UNIVERSITEIT LEUVEN

UNIVERSITEIT GENT

KAHO

Contents

Chapter 1

Introduction

1.1 Background

Belgium is an early adopter of smart card based electronic identity cards. It is also one of the few countries worldwide where eID cards are mandatory. This has opened the door to new applications for government and industry, but has also resulted in the discovery of shortcomings in the current system. One major limitation is the limited availability of smart card readers. Now that advanced mobile devices are becoming ubiquitous, there is a new opportunity for applications that leverage government issued electronic identities.

The main objective of the MobCom project is to transform the mobile device into a powerful, flexible and user friendly tool for managing identities in a privacy friendly way and to act as the interface to personalized services.

Identities and user profiles are an integral part of many types of services such as e-government, e-health, e-commerce, social networking and banking. The goal of this project is the creation of an open and trustworthy infrastructure that builds on existing authentication methods, such as the electronic identity card. We will focus on developing the technological building blocks and infrastructure that will bring mobile identity management to a wide range of application domains:

- **rich mobile identities** consisting of traditional identity data already present on electronic identity cards, but extensible with user profiling information that can enable personalized service delivery, in spirit of the mobile 2.0 concept.

- a **distributed, secure architecture** supporting the exchange of identity data between devices, and between devices and services under the control of the user. Identities will be easily accessible and will be managed by a trusted third party.

- **privacy-enabled policy management** will let the user decide what data can be accessed by each specific application. Users will be able to manage and fine-tune their identity data. The user interface for this policy management will be easy to use and guarantee full transparency.

- user friendly methods for user **authentication**, and device and user **attestation** based on biometrics and advanced cryptographic protocols.

- an easy to use **application programming interface** for the creation of new applications by third party developers. This API will allow developers to build new applications on top of our mobile identity infrastructure. Powerful APIs contributed much to the success of web services such as Google Maps, YouTube or Flickr.

1.2 Summary of Contents

This document reports on the current state of work packages *Architecture*, *Applications* and *Validation*. We present an identity framework and several concrete prototypes that bring mobile identities to a wide variety of application domains. Each chapter presents the rationale and requirements for one particular development track. After discussing application features and implementation details, each prototype is validated against the requirements.

- A **privacy-preserving identity framework** is presented in chapter 2.
- Chapters 3, 4 and 5 encompass the **advanced access control** domain.
- The **context-aware services** domain is covered by chapter 6.
- An application that demonstrates **loyalty cards and payment vouchers** is discussed in chapter 7.

Chapter 2

Framework and Software Architecture

2.1 Introduction

The increasing capabilities of application and service providers to collect information about their users pose a serious threat to privacy. For example, authentication mechanisms such as passwords and X.509 certificates reveal personal information and allow providers to track and profile their users. As a result, anonymous credential systems such as Idemix [21, 49] and U-Prove [13, 71] have been proposed. Such systems allow users to have more control over the information released during a transaction with a service provider, thus, offering better privacy guarantees. Unfortunately, the adoption of anonymous credentials systems have been slow, despite the fact that they have been around for more than a decade and free development libraries are available. One of the reasons for this slow adoption is that, compared to traditional credential systems, anonymous credentials systems rely on more complex cryptographic concepts and are more challenging to configure and deploy. Therefore, developers face a steeper learning curve when evaluating anonymous credential systems. In addition, anonymous credential systems are more computationally expensive than traditional credential systems, requiring more careful implementation and configuration, particularly in platforms with constrained resources such as mobile devices (e.g., smartphones). Thus, it is important to design tools that simplify the development of privacy-preserving applications based on anonymous credentials and other privacy-enhancing technologies.

In this chapter we present PriMan, a privacy-preserving framework that hides the complexity of anonymous credential systems and facilitates the development of applications with better privacy guarantees. PriMan's reusable software architecture offers a uniform and technology agnostic interface to traditional and anonymous credentials systems and other supporting technologies (e.g., secure storage, user and server policy languages and secure communication channels). With PriMan, developers can quickly prototype applications, predict performance problems, analyze security and privacy properties and, hence, steer basic research to solve problems and gaps in this area. In addition, the framework is easily adaptable and extensible, allowing developers to easily try different privacy-enhancing technologies and switch to new ones as they become available without changes to the application code.

The PriMan framework is based on work from the ADAPID ("Advanced Applications for the Belgian eID card") project [1]. PriMan not only extends the ADAPID framework by implementing a more lightweight design and support for new technologies, but also it has been optimized for mobile platforms such as the Android operating system. PriMan introduces a thinner middleware layer between the application code and the different privacy-enhancing technologies supported, resulting on lower performance overheads.

2.2 Requirements

In this section we describe the general and functional requirements used to design the PriMan framework.

2.2.1 General Requirements

- **Usability.** The framework must provide a uniform, technology agnostic interface that hides the complexities of the different credentials systems and supporting technologies. Thus, it should be easier and faster for developers to build privacy-preserving applications using the framework than using PETs technologies directly.
- **Performance.** The computational overhead introduced by the framework must be small and should not impact significantly the overall performance of the application in both desktop and mobile platforms.
- **Extensibility.** The framework must support the addition of new technologies without requiring changes to its core components and the interface exposed to the developers. In particular, new technologies should be dynamically available without requiring the recompilation of the framework (i.e., a software plug-in approach)
- **Portability.** The framework must be developed in a language that is portable to both desktop/sever and mobile platforms. The framework should support the functionality required to build the server and client components of the targeted application.

2.2.2 Functional Requirements

- **Support for different credential technologies.** The framework must support the issuance and verification of different credential technologies, both traditional (e.g., passwords, X.509 certificates, BeID, etc.) and privacy-preserving (e.g., Idemix and U-Prove). All the functionality provided by a particular credential type must be available via the framework.
- **Support for server and user policies.** The framework must support the creation and use of server and user policies for the different technologies available. For this purpose, the framework must support standard policy languages for traditional credentials such as SAML, XACML and WS-Trust and for anonymous credential systems such as CARL.
- **Support for secure storage of credentials and related data.** The framework must support the use of different storage technologies for credentials and related data. These should include local (e.g., disk, SD card) and remote storage (e.g., cloud storage service) as well as secure (e.g., encrypted XML, PKCS #12) and non-secure (e.g., plain XML, object databases, etc.) storage formats. Support for TPM chips and other secure elements (e.g., JavaCard) is also required.
- **Support for different type of network protocols.** The framework must support different network protocols for the communication between the user, providers and any other participant of the application protocol. This includes support for protocols with no security (e.g., raw sockets, TCP, HTTP), with security (e.g., SSL/TLS, IPsec) and with anonymity (e.g., Tor, I2P).
- **Support for dispute handling.** The framework must offer support to detect misbehavior by users abusing their anonymity and the possibility must be given to take appropriate measures against such users.
- **Support for anonymity metrics.** The framework must be able to monitor and estimate the consequences of disclosures of personal properties on the user's anonymity and to provide advice.

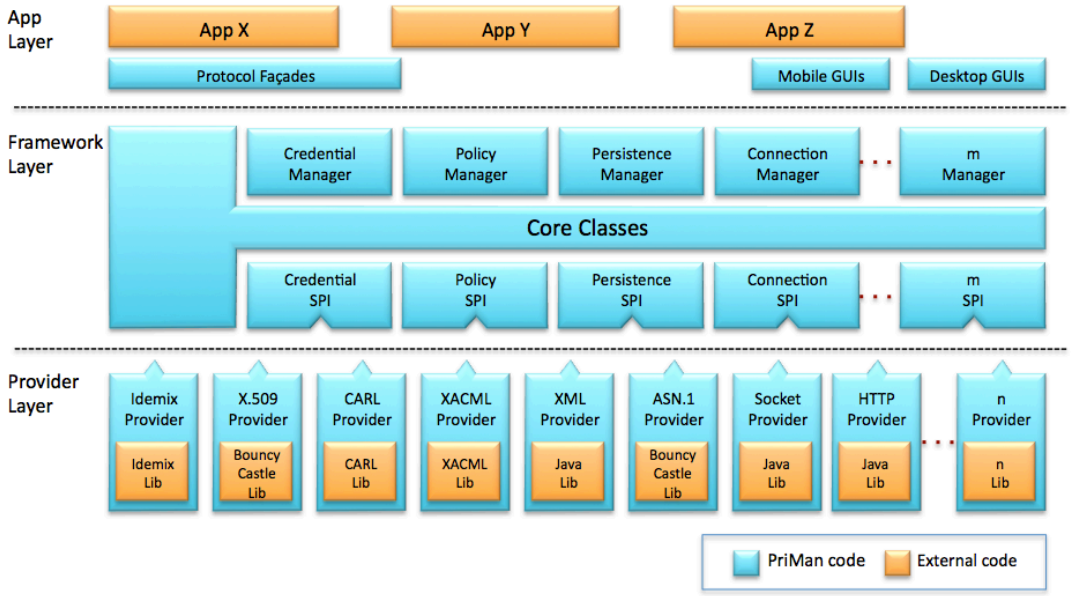


Figure 2.1: High level view of the PriMan architecture.

- **Support for client side profiles.** The framework at the client side has to keep track of the properties disclosed to other parties. Secondly, the possibility must be given to the user to maintain profiles with transactional data and user preferences.

2.3 PriMan Architecture

Figure 2.1 presents a high-level view of the PriMan framework architecture. The PriMan's design is based on two well-known design patterns: the service provider framework pattern [11] and the capability pattern [12]. The light-blue blocks represent PriMan components and the orange blocks represent external code components (e.g., application's code, technology providers libraries). Three layers can be identified: the application, the framework and the provider layers.

The *application layer* contains the code of the applications (orange rectangles) that rely on the functionality offered by PriMan. Applications can use PriMan's API directly to build authentication protocols or use *protocol façades*, which offers a simplified interface to authentication protocols already implemented with PriMan (e.g., Idemix's credential issuance, Idemix authentication, X.509 authentication). In this layer, PriMan also offers to developers with ready to use *graphical user interfaces (GUIs)* to display and manage authentication objects such as credentials and policies.

The *framework layer* contains the main components of PriMan: managers, core classes and service provider interfaces (SPIs). The *managers* and *core classes* form the service access API of the framework, which applications use to access the functionality provided by PriMan. The managers expose methods for high-level operations (e.g., issue credential, get connection) while core classes provide access to more specific operations according to the type of object (e.g., list credential's attributes, send data). References to core objects can only be obtained through their corresponding managers. Configuration files are used to define the type of technology and required parameters for such technology. Thus, no changes to the application code are required when switching technologies (e.g., using X.509 instead of Idemix). The *SPIs* define a uniform interface that providers implement to support the operations requested by the managers. In general, the managers, core classes and SPIs do not implement any logic (i.e., abstract classes and interfaces).

The *provider layer* includes the different technology providers supported by the framework.

The providers are not part of the framework but can be plugged in using platform-dependent plugin mechanism (e.g., dependency injection using java.util.ServiceLoader). Providers implement the interface defined by SPIs, acting as the glue between the framework and technology implementations. In addition, providers extend and instantiate classes that implement or extend core classes and pass references of these instances to the corresponding managers.

To use the PriMan framework API directly, a developer only needs to instantiate the PriMan class. The PriMan object will then provide access to the different managers. To guarantee correctness, the PriMan class can be instantiated only once (i.e., singleton pattern). As mentioned before, most classes in the framework layer are abstract classes or interfaces that are extended or implemented by the providers. This design reduces the performance overhead introduced by the framework. The uniform API exposed by the framework allows developers to try different credential technologies using in most cases the same code. To change technologies and configurations, the developers will only need to modify configuration files without having to recompile the application. In the next sections, we present more details of the different components of the PriMan framework.

2.3.1 Managers

As stated before, managers and core classes implement the service access API of the framework – a uniform interface to different credential systems and other privacy-enhancing technologies. Managers are organized according to their functional area. The framework relies on four basic manager classes: credential, policy, persistence and connection managers. Additional managers can be added to the framework to provide additional functionality and extend the framework's API. Each manager keeps track of the different technologies it has access to, allowing developers to determine and select the technologies that are more appropriate for their applications. By offering a uniform interface to the different technologies supported, the managers effectively reduce the developer's learning curve, as developers only need to get familiar with one API.

Managers expose high level methods (e.g., issueCredential(), verifyProof(), getConnection()) that use core objects (Section 2.3.2) and return references to such objects (e.g., credential object). Only managers can return references to core objects; developers cannot instantiate core classes directly. In addition, managers do not communicate directly with each other, instead, they rely on core objects to exchange data and functionality. Each manager produces and consumes core objects. For example, the policy manager produces a policy object that is used to create a claim object. This claim object can then be used by the credential manager to generate a proof object. To access the functionality offered by different technology implementations (e.g., Idemix library), managers rely on the interface defined by the SPIs (Section 2.3.3) and implemented by the providers (Section 2.3.4. Moreover, managers offer methods for both client and server side applications. The next sections present more details about each manager.

2.3.1.1 Credential Manager

The credential manager provides high-level operations for controlling the life cycle of user authentication credentials. Standard credential technologies (e.g., X.509 certificates, BeID and shared secrets) as well as anonymous credential systems (e.g., Idemix and U-Prove) are supported by the credential manager using a common set of public methods. In general, the credential manager offers the following functionality:

- *Issuance and receipt of credentials.* Different options can be set depending on the credential technology: the validity period, whether signing or authentication is allowed, the type of credentials that can be issued, the number of times the credential can be shown, whether or not attribute values are updatable and the credential attribute values. The latter can, if allowed by the credential technology, be partly chosen by the issuer, partly by the receiver or jointly. The other options can be set by the issuer. Depending on the credential technology, a credential can be issued to a pseudonym.

- *Issuance and creation of pseudonyms.* Some credential technologies allow issuing and showing credentials under a certain pseudonym agreed with the issuing or verifying entity. It is also possible that a pseudonym is locally created and that the well-formedness of it is proven as part of an authentication or signature protocol.

- *Authentication and signing.* The signatures can be interactive or non-interactive. Other features are dependent on the credential technology. For example, fine grained and flexible selective disclosure of credential values can be offered. Also, several credentials, commitments and verifiable encryptions can be involved in this selective disclosure. Some technologies allow for authentication or signing under a pseudonym.

- *Credential expiration and revocation.* The credential manager also takes care of determining if a credential has expired and needs to be removed from the client application. In addition, it also offers different mechanisms for revoking active credentials in case the credentials are compromised.

The credential manager relies on several core classes (Section 2.3.2) for its operations. For example, to create a new credential object, the client application sends a request to the issuer's server. The credential structure is defined in a credential specification document published by the issuer. The client, thus, uses the credential specification object to send a credential request to the issuer. In the server side, the issuer uses the credential specification object to generate a new credential object, which is then sent to the client application. Finally, the credential manager registers the new created credential object and passes it to the persistence manager (Section 2.3.1.3) for storage.

2.3.1.2 Policy Manager

Policy support is important for the development of more advanced authentication protocols. A policy is a mechanism that puts constrains on the behaviour of a system. For example, a policy could require certain properties on communication channels, restrict the use of credentials or regulate access to different user profiles. In general, policies shift most of the burden of operational decision making from people to the technology.

The framework supports both *user and server policies* through the policy manager. In general, the client will receive an authentication policy from the server describing the properties the user must (and may) disclose in order to access a particular resource or service. The policy manager takes this server policy as an input and matches it with all the user's credentials to determine the credentials that can be used to fulfill the server policy. To decide what credentials and attributes can be used to satisfy the server policy, the policy manager also takes into consideration the user policy. The user policy defines what credentials and attributes should be used and what actions (e.g., proof of ownership, revealing attributes) are allowed to fulfill the policy of a particular service provider. Different combinations of credentials can be used to satisfy a particular server policy. Each combination of credentials that satisfy a server policy is known as a *claim*.

After receiving the server policy, the policy manager returns a reference to a policy object that contains the parsed policy and at least one claim. If more than one claim is possible, then the policy object will also include all the possible attribute options that can be combined to generate a claim (i.e., attributes from different credentials that can be used to satisfy a particular policy requirement). At this point, the application can request the user to build a claim based on the available options or could select a default claim. The claim object is then used by the credential manager to build a proof object.

The policy manager has support for several policy languages. For example, for traditional credentials such as X.509 attribute certificates, the policy manager should support policy languages such as SAML, XACML and WS-Trust. For anonymous credentials, the policy manager should support CARL [22] (Credential-based Authentication Requirements Language).

2.3.1.3 Persistence Manager

The framework uses many different data objects (e.g. credentials, parameters, policy files, etc.). To make them available the next time the user starts the application, the persistence manager can be used by developers to save and load data objects. This manager keeps track of the locations of all these data objects. For instance, credentials with confidential information (e.g., medical health records) can be stored in off-line devices (e.g., encrypted USB token, smart card, etc.) and credentials that are used more frequently can be stored in a key store in a PC's hard disk. Moreover, some data objects can also be kept in cloud storage (a popular alternative nowadays). The persistence manager keeps metadata for each data object. This allows searching for the correct data object without having to load the objects first (which would be difficult or inefficient since it can be stored on remote or off-line storage locations).

If the user runs the framework on different devices (e.g., desktop, laptop, smartphone, etc.), the persistence manager has to take care of the synchronization of data objects and corresponding metadata information. For some data objects (e.g., frequently used credentials, settings, etc.), it may be required to cache these data locally to improve performance or allow for off-line transactions (an alternative to these type of scenarios is to used cloud storage services).

The persistence manager has access to different storage technologies through the interface defined by the persistence SPI and associated core classes. The persistence providers and the corresponding technology implementation are responsible implementing the logic for saving and loading data objects. Each persistence provider defines a location type (e.g., smart card or server), an encoding structure (e.g., XML, ASN.1) and the appropriate protection mechanism (e.g. password, biometric template). Also, in the case of remote storage technologies, each provider implements its own communication mechanism (connection parameters are defined in a configuration file). Non-secure storage technologies (e.g., XML, text files, PEM) and secure storage technologies (e.g., smart cards, PKCS11 keystore, PKCS12) are supported.

2.3.1.4 Connection Manager

The connection manager allows developers to establish communication channels with external entities (and, thus, other instances of the framework). Basically, this manager creates connection objects that send and receive data via different communication technologies. The data can be in any format (e.g., XML file, serialized Java object, etc.). Developers can use the connection manager to get references to connection objects that can be used to exchange data with remote entities (e.g., client-server communication). For example, the credential manager requires of a connection object to send requests and received responses with the issuer server to obtain a new credential. The connection manager also allows the framework to send an error message to the other entity when something goes wrong (authentication exception, general Java exception, etc.).

Developers can access several communication technologies through the connection manager and connection objects: network sockets (e.g., TCP, UDP, raw IP), application layer protocols (e.g., HTTP), secure protocols (e.g., SSL/TLS, IPsec), anonymity networks (e.g., Tor, I2P), personal area network protocols (e.g., Bluetooth) and near field communications (NFC) protocols. By using a similar interface, switching between communication protocols is straightforward for the application developer. Only the initialization parameters are technology specific and are included in configuration files.

2.3.1.5 Other Managers

Future versions of the framework will also include additional managers that extend the framework's functionality and allow the design and implementation of more advanced authentication scenarios. Examples of these additional managers (and their corresponding core classes, SPIs and providers) are:

- *Privacy manager.*

When using privacy-preserving credentials, the properties that are selectively disclosed will affect the users level of anonymity. Also, if the same pseudonym is used several times in credential shows (signatures or authentications), the different credential shows can be linked, which will further reduce the users anonymity towards the service provider. The privacy manager estimates the level of anonymity of pseudonyms towards other parties and the impact of disclosing certain properties. This manager will rely in different technology implementations to keep concrete metrics of the user anonymity level.

- *Dispute manager.*

  In order to prevent abuse and fraud, anonymity can be made conditional, which makes dispute handling harder. In more complex protocols, the evidence required to accuse or to exonerate someone could be spread over different parties. The dispute manager offers the means to file complaints in case of abuse. Complementary, evidence to (e.g., to protect against false accusations) can be stored and later be disclosed to trusted third parties. These parties may do a de-anonymization when certain conditions are fulfilled. Disputes are application specific and a different provider handles each type of abuse. The dispute manager supports all the disputes supported by the underlying dispute providers.

- *Profile manager.*

  Profiles can be locally maintained by the users framework to keep track of the personal properties that have been disclosed to service providers and of transactional data and user preferences. The latter may be accessible to service providers. The profile manager main task is maintain these profiles. Each profile provider keeps track of a single profile. Depending on the framework policy, authorization can be given to external entities to make certain types of queries on one or more profiles: adding or requesting for data that can be application or context specific (e.g., books bought). Also, a user can add data to a profile (e.g., books he is interested in). The profile manager determines to which profiles data are added. A second type of profile provider maintains personal data that has been disclosed to the service providers.

- *Biometric manager.*

  Biometric authentication technologies (e.g., face recognition, fingerprints and voice recognition) provide a more robust and usable alternative to user-to-device authentication, particularly for mobile devices. In addition, the binding of credentials with biometric templates can prevent the non-authorized sharing of credentials (i.e., transferability). The biometric manager will provide developers with access to different biometric technologies, in both desktop and mobile platforms. Thus, developers will be able to integrate biometric authentication technologies in their applications.

2.3.2 Core Classes

As previously stated, managers and core classes provide a uniform API to developers to access all the technologies supported by the framework. Core classes are abstract classes that are extended and instantiated by their corresponding providers. For example, the IdemixCred subclass in the Idemix provider extends the Idemix core class. Providers pass references to instances of these classes to the respective managers, which make them available to developers and other managers. Each core class provides developers and managers a set of methods common to all the technologies associated with the class.

Each manager relies on a set of core objects to perform its operations. For example, the credential manager uses the credential, claim, proof and nonce classes. Figure 2.2 depicts a simplified class diagram of the credential class and associated classes: Attribute and Issuer. The credential class exposes a set of generic methods (e.g., getAttributes(), getIssuer()) that can be used by the managers and the developers. Each technology provider extends the credential class with a technology specific subclass (e.g., IdemixCred, X.509, BeID). In this way, the provider

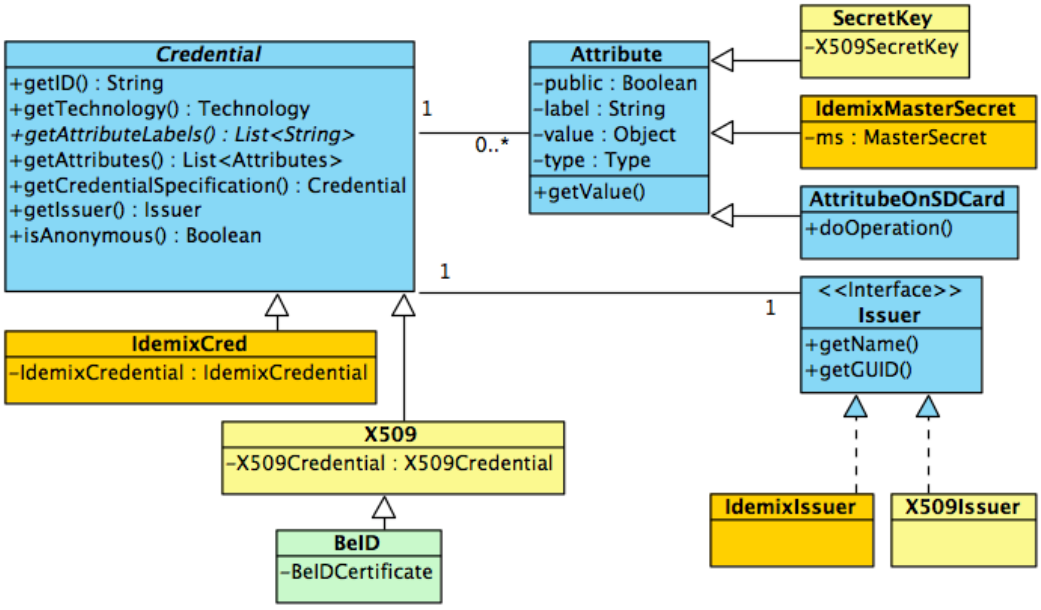


Figure 2.2: Credential class structure and its associated classes

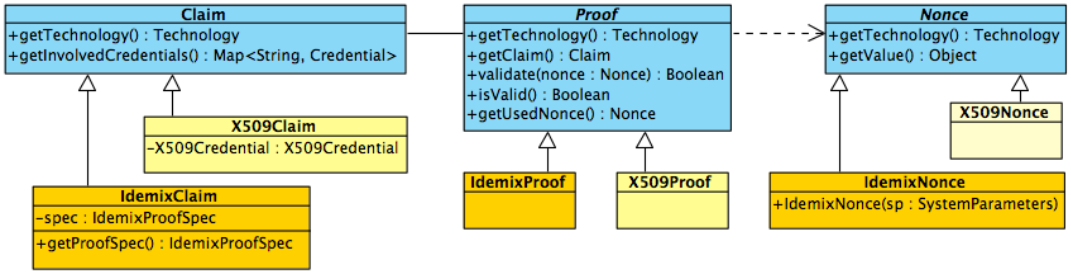


Figure 2.3: Claim, Proof and Nonce classes

has access to all the functionality of each technology implementation, while the managers and developers only has access to the generic functionality offered by the superclass (i.e., credential class). A similar design can be observed in Figure 2.3, where the claim, proof and nonce classes are shown.

2.3.3 Service Provider Interfaces

The Service Provider Interfaces (SPIs) allows managers to access the different technologies supported by the framework. Each manager has a corresponding SPI (e.g, credential manager and credential SPI). The SPI defines an interface with all the methods that the manager requires. Different technology providers then implement this interface. (e.g., Idemix and X.509 providers). Thus, the SPI defines a uniform interface that the different providers should implement to add new technologies to the framework. In this way, new technologies can be added to the framework without requiring changes to the managers and core classes.

2.3.4 Providers

Providers are the code used to translate technology implementation interfaces (e.g., Idemix library API) into the interfaces defined by SPIs (e.g., credential SPI). In addition, providers also extend

and instantiate core classes (e.g., credential class is extended by IdemixCred class in the Idemix provider). In general, providers act as wrappers for adapting third-party technology implementations to the framework (e.g., Idemix, U-Prove and Bouncy Castle libraries). However, providers not always need a third-party library; providers could also implement a particular functionality directly without external code. For example, a provider could implement new mechanisms that have not been made available yet in a third-party library. As Figure 2.1 shows, providers are not part of the framework. They are added dynamically to the framework through a plug in mechanisms (e.g., dependency injection using java.util.ServiceLoader). Thus, new technologies can be added to the platform, according to the platform used, without modifying other components of the framework (i.e., no recompilation of the framework is required). For this purpose, each manager keeps track of the different technologies available under its control. Developers can select a particular technology by modifying configuration files.

2.3.5 Protocol Façades

In general, developers can use the rich functionality offered by the framework API to implement different type of authentication scenarios and protocols. The different technologies supported by the framework can be viewed as building blocks that developers can use to build privacy-preserving applications. In addition, the framework offers different protocol "templates", ready to use authentication protocols (e.g., Idemix and X.509 standard authentication protocols) that developers can add directly into their applications. These protocols templates are available through the protocol Façades, a simplified interface that unifies a set of PriMan's managers and core classes to implement well-know authentication protocols (i.e., Façade design pattern). The protocol Façades are not part of the framework, they are an auxiliary API to help developers to evaluate standard authentication protocols in a faster and easier way. In addition, new protocol Façades can be added as needed, without modifying the framework.

2.3.6 GUIs

To accelerate development, the framework also offers a set of default GUIs for displaying and managing different types of core objects such as: user and server policies, credentials and claims. These GUIs take as input core objects and display them to the users. These GUIs are based on design guidelines from the research and industry communities, focusing on usability. Developers can use these GUIs directly into their applications or can use them as a guide to design and implement GUIs that match the look and feel of the application. PriMan offers two set of GUIs: desktop GUIs and mobile device GUIs (for devices with small screens such as smartphones). Similar to protocol Façades, these GUIs are not part of the framework and new GUIs can be added as needed.

2.4 Initial Performance Evaluation

We performed a preliminary experimental evaluation of PriMan to estimate the performance overhead introduced by the framework. The experiments focused on Idemix proof of ownership using a mobile device (smartphone) as a client. Three basic scenarios were evaluated:

- Proof of ownership using a credential with no attributes.
- Proof of ownership using a credential with three integer attributes and hiding all the attributes.
- Proof of ownership using a credential with three integer attributes and revealing all the attributes.

Each scenario was evaluated using three different sizes of the SRSA modulus: 1024, 1536 and 2048 bits. In the server side, we used a Apple MacBook Pro (Intel Core i5 @2.5 GHz, 4GB

msec (stdev)	1024		1536		2048	
(a_t, a_r)	Prove	Verify	Prove	Verify	Prove	Verify
(0,0)	58,5 (2,1)	21,1 (6,0)	138,6 (2,7)	39,6 (8,7)	285,6 (9,2)	59,5 (10,6)
(3,0)	83,4 (6,6)	25,1 (3,9)	191,3 (12,7)	46,8 (6,4)	360,0 (4,9)	78,4 (5,0)
(3,3)	59,0 (2,8)	17,9 (3,0)	137,7 (4,5)	32,6 (3,0)	285,2 (9,0)	57,7 (2,5)

Table 2.1: Total execution time for Prove and Verify Idemix operations using the PriMan framework and a smartphone as a client. Three types of proof were evaluated: ownership with credential with no attributes (0,0), ownership with credential with three hiden attributes (3,0) and ownership with credential with three revealed attributes (3,3).

msec (stdev)	1024		1536		2048	
(a_t, a_r)	Prove	Verify	Prove	Verify	Prove	Verify
(0,0)	0,15 (0,36)	0,05 (0,33)	0,09 (0,29)	0,11 (0,40)	0,56 (0,36)	0,04 (0,24)
(3,0)	0,19 (0,44)	0,02 (0,14)	0,40 (1,10)	0,03 (0,17)	0,16 (0,39)	0,04 (0,14)
(3,3)	0,21 (0,48)	0,02 (0,14)	0,19 (0,39)	0,03 (0,17)	0,17 (0,38)	0,01 (0,10)

Table 2.2: Framework only estimated execution time for Prove and Verify Idemix operations. Three types of proof were evaluated: ownership with credential with no attributes (0,0), ownership with credential with three hiden attributes (3,0) and ownership with credential with three revealed attributes (3,3).

RAM, Mac OS X 10.8.2 64 bits, JavaSE 1.6). In the client side, we used a Samsung Galaxy S III smartphone (Quad-core 1.4 GHz Cortex-A9, 1 GB RAM, Android 4.0.4). A isolated WiFi network was used for the communication between the server and the smartphone.

For the experiments, a small client and server application were developed using the framework. We used code instrumentation to measure the total execution time required to generate a proof (i.e., client operations) and to verify such proof (i.e., server operations). The instrumentation code allowed us to measured not only the total execution time, but also the time that the technology implementation (i.e., Idemix) required on each task. Thus, by computing the difference between the total time and the Idemix time, we were able to estimate the execution time of the framework (i.e., framework overhead). Each experiment was executed at least 100 times to guarantee the statistical significance of the experiments.

Table 2.1 shows the total execution time (i.e., Idemix + PriMan) for proof generation (Prove) and proof verification (Verify) for each experiment. The results show that, even for a 2048 bit modulus, executing Idemix operations in a mobile device is practical (less than 300 msec to generate a proof). Also, as expected, proof generation takes significantly more time than proof verification, mainly because the former is executed in the mobile device and the latter in the server (i.e., MacBook Pro). The values in Table 2.1 represent mean values and their corresponding standard deviation.

Table 2.2 shows the estimated execution times of the PriMan framework only (i.e., PriMan overhead). As mentioned before, we estimated these values by computing the different between the total execution time and the execution time due to Idemix only. Mean values and their corresponding standard deviation are reported. Note that these values include not only the execution time of the framework layer components but also of the Idemix provider. The results show that the overhead introduced by the framework is small – less than 1 msec. This is expected as the framework contains little logic; it consists mainly of abstract classes and interfaces. In addition, the standard deviation values are higher than the average values. The reason for this outcome is that the time resolution of the counters used is only in milliseconds, thus, inadequate to measure PriMan's small overhead. Nevertheless, these results show that the overhead introduced by the framework is insignificant and unlikely to affect the overall performance of applications using the framework.

2.5 Conclusions and Next Steps

In this chapter we have described the overall architecture of PriMan, a privacy-preserving framework aimed to help developers to evaluate different privacy-enhancing technologies in a faster and easier way. PriMan achieves this by introducing a thing middleware layer that exposes a uniform API to developers to access a variety of credential systems and other privacy-enhancing technologies. Our initial experimental evaluation demonstrates that the overhead introduced by PriMan is small and unlikely to affect the overall performance of an application. We are currently working on the addition of more technology providers and defining the first stable of the API exposed by PriMan to developers and technology providers (i.e., PriMan 0.1). We plan to evaluate PriMan with more advanced authentication scenarios, including the use of more complex credentials and server and user policies.

Chapter 3

Claim-based Access Control

3.1 Rationale and Motivation

In this chapter, we show how mobile devices may facilitate the use of anonymous credentials, as an electronic identity, for data-minimizing authentication. We envision scenarios where users authenticate to terminals using their mobile, with the requirement that it is ensured that the user at the terminal is the one performing the authentication. We denote these scenarios *user-to-terminal authentications*.

Examples in the real world are: the age verification of customers or buyers in a bar when selling alcoholic beverages, access control to the premises of one's employer, and a broad variety of electronic ticketing solutions. For the age check, an important property is that no third party can perform the authentication instead of the user at the terminal.

We present the requirements (Sect. 3.2) and protocol constructions (Sect. 3.3), based on short-range QR channels between a user's handheld device and a terminal to establish an authenticated channel between the user's handheld and the terminal. As key property, the user is authenticated based on her properties instead of identifying attributes. We therefore use the Identity Mixer credential system on the mobile device. Through the properties of the short-range channels, we acquire a higher level of trust in that the device executing the protocol is the one interacting with the terminal, and vice versa. As a particularly important security feature, the prototype employs a Secure microSD card as a secure element for handling secret cryptographic tokens throughout their life cycle. That is, storing them and performing all computations related to them. We have implemented a prototype system on top of an Android mobile phone. Implementation specific details can be found in Sect. 3.4. In Sect. 3.5, we present measurements of the key metrics that determine protocol runtimes. We found the results encouraging for a practical application of anonymous authentication technologies on standard mobile devices. We discuss the properties of our prototype. Finally, we conclude in Sect. 3.6.

Related Work. Chari et al. [25] presented a taxonomy of different m-commerce scenarios with mobile devices. Our user-to-terminal scenarios fit in their *Kiosk-Centric* and *Full Connectivity* model respectively. Similarly, our solution also fits in the model of Claessens [27], in which the mobile is combined with a workstation in order to have a higher degree of security and mobility.

Next to manual authentication [44] or authentication based on image comparison [37,63], many schemes have been presented that use physically constrained channels in order to obtain a secure communication channel between two nearby devices. Although they are often related to device pairing, they also apply to our user-to-terminal authentication. Examples are based on physical contact [5,69], motion [58], infrared [5], audio [45,68], bar- and QR codes [57,59,70], Bluetooth [54] and radio profiling [73].

In contrast to the schemes above, where privacy is often of a lesser concern (e.g., for device

pairing), our solution combines the short-range communication channel with a more privacy-friendly authentication mechanism, namely anonymous credentials. In our prototype, we apply a QR based communication channel. Nevertheless, our solution also supports other short-range channels, as mentioned above.

3.2 Requirements

The requirements can be summarized as follows:

1. authentication should only happen with the user's consent;
2. authentication should only be possible by the user in front of the terminal;
3. sharing of credentials (e.g., with friends) should be impossible;
4. when the smart phone is lost or stolen, the credentials should no longer be usable.

The *first requirement* demands that for every authentication, at least some **user interaction** is necessary (e.g., at least a click on an OK-button).

The *second requirement* is more difficult to realize. The authentication process needs to **involve a short-range communication channel** between the terminal and the mobile device. An ideal channel would be an NFC-channel. However, most smart phones do not yet support this kind of communication and terminals even less. Bluetooth is not really short-range, and WiFi certainly not. Therefore, our scenarios will use a **visual channel**: QR-codes will be displayed on the screen and scanned by the camera of the device at the other end of the channel. Since the camera must be positioned a few centimeters in front of the displayed QR-code, this way of exchanging information certainly realizes a short-range communication channel.

The *third requirement* can be realized by introducing a **secure element**. In our scenarios, we will assume that the smart phone has a free slot for a μSD card. A (secure) smart μSD card is used to store the user's secrets of the credentials. Every credential is associated with a secret; when using a credential, the user needs to prove knowledge of the corresponding secret. Since the user has no longer direct access to these secrets, the original Identity Mixer library that we use as anonymous credential system, had to be modified as follows:

- the user secret is generated on the smart μSD card and never leaves the card;
- the credential issue protocol and the credential show protocol have been adapted so that all computations (exponentiations) that involve the user secret are delegated to the smart μSD card.

Since one needs the user secret for showing a credential, and user secrets are kept in a secure element, it becomes impossible to share credentials.

For the *fourth requirement*, the use of the smart μSD card is further restricted: the card will only perform an operation if the correct **PIN-code** is given. Hence, when the mobile device is lost or stolen, the new owner will not be able to use the stored credentials since she cannot "activate" the card via the correct PIN-code.

Note that in this solution we do not prevent relay attacks such as *mafia fraud attacks* and *terrorist attacks* [6]. In a mafia fraud attack, the attacker forwards messages between an honest prover and an honest verifier. Whereas in a terrorist attack, the prover is dishonest and collaborates with the attacker. The latter could allow an adult, to help a youngster to prove that she is older than 18, based on the credentials of the adult. Mafia fraud attacks may be prevented using *distance bounding* [2, 3, 14, 47, 55, 67], a mechanism mostly discussed in the context of RFID. Distance bounding in combination with the tamper resistance of the smart card also makes the terrorist attacks harder to achieve, as the prover has less control on the actual protocol running on his device. Unfortunately, in contrast to radio based channels such as NFC, when using a QR-based visual channel or an out-of-band channel, distance bounding is hard to achieve.

3.3 Construction

We now discuss the general architectural concepts used, as well as protocols for realizing our mobile authentication application.

Roles and setting. Figure 3.1 illustrates the parties in our setting. A user U is the holder of a mobile device M. An issuer IP provides credentials to the user U. U keeps the credentials on the mobile; they can later be used when U authenticates towards a terminal T. In order to allow a more flexible setup, with multiple terminals, verification is not performed at the terminal itself. The terminal is in direct connection with a trusted authorization server AS, who performs all verifications. In this case the terminal simply acts as gateway between the user and the authorization server. We denote the relying party RP as consisting of both T and AS. Note that communication between the terminal and the authorization server, is SSL/TLS protected, in order to obtain a higher level of security.

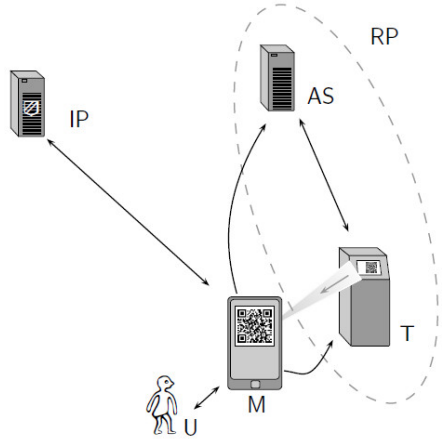


Figure 3.1: Parties in User-To-Terminal authentication. The User U, the Mobile M, the Issuer IP, the Authorization Server AS and the Terminal T. The Terminal T and Authorization Server AS together Form the Relying Party RP.

To address the *proximity* requirement, it is sufficient to have the first request sent through a short-range channel and cryptographically bind the response to that event, leaving open how the response is returned (e.g., using an out-of-band channel).

System Setup. The issuer IP publishes the public key information used in the anonymous credential-issue and credential-show protocols. To ensure entities that the public key is indeed the one of the trusted issuer, this information can be certified in a PKI-infrastructure, using standard X.509 certificates [28]. IP can then use this certificate for server authentication. Similarly, AS has a standard X.509 certificate for server authentication.

Setting Authentication Requirements. The authentication requirements are defined using the CARL policy language. Multiple policies may be defined for a single resource being protected. For instance, the user is allowed to prove, based on the date of birth included in a credential, that she has actually reached the age of majority, without revealing her date of birth. On the other hand, she is also allowed to prove possession of a valid driver's license, implying age majority. In addition, the relying party may also specify a *response channel*, and a destination address (e.g., URI or Bluetooth address), over which and to which the response should be returned.

Registration of U with IP. The user first registers with the registration server IP. The system can be bootstrapped by the user when getting issued an anonymous credential on M in a variety of ways. A guard protects the issuing of credentials at IP, specified in a policy on the requirements for obtaining a credential. Note that guards for multiple authentication schemes may be used. For instance, a practical use case for Belgium, where eID cards are issued to all citizens as of 12 years and older, is that the guard requires an attribute statement based on the user's eID card. The attributes obtained in such way can then be issued as attributes of an anonymous credential. IP, relying on the correctness of attributes originating from the Belgian eID card, then acts as a (re-)certifier of these attributes. The guard-based and policy-based architecture of the issuer leaves the authentication of U with IP open to a concrete deployment of our system, thereby ensuring flexibility.

Authentication of U at T. Figure 3.2 illustrates how the user may authenticate towards the terminal, based on whether or not an out-of-bound channel is used. In both cases, the terminal receives an *authentication request* (linked to the HTTP session) from the authorization server (1), and hands it over to the mobile over the short-range channel (2).

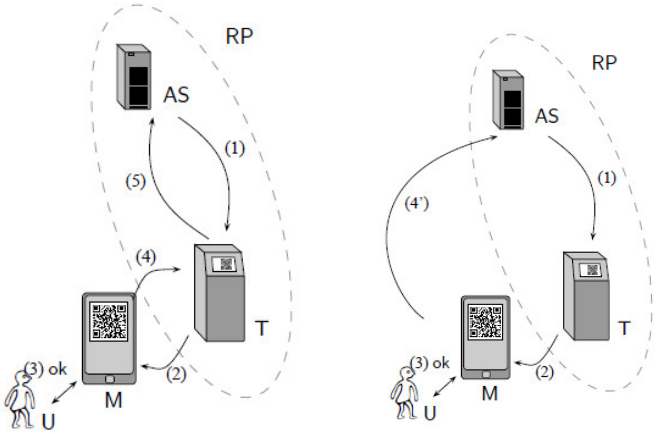


Figure 3.2: Route (a) over T to AS, and (b) Directly to AS.

M parses the received authentication request, containing a challenge and the server policies. It may also include the relying party's certificate to encrypt the response, if it is to be kept secret. The applicable policies are selected, based on the availability of credentials, and rendered by the application to a for humans easy-to-understand presentation format, which is then displayed on the screen of M. Each policy comprises a data request that needs to be disclosed and proved with the Identity Mixer or other technologies in our framework. U is required to choose how to fulfill the policy and to give her consent to the information release (3). If the master secret is protected by a secure element, the user is also challenged to enter her PIN code.

The privacy and security framework computes a credential proof and the *authentication response*, which includes this proof and a reference to the chosen policy, is returned to AS. Now, depending on the response channel, as specified in the authentication request, the response may follow a different path.

(a) From the user's perspective, the most straightforward scenario is to reply to the terminal from which the user received its authentication request (4) (see Fig. 3.2(a)). This could be over another or the same short-range channel as used in (2). Then, the terminal forwards the authentication response to the authorization server over a secure channel (5). The latter parses

the response and verifies the correctness of the included proof. Finally, the authorization server notifies the terminal about the result.

(b) In the second case, illustrated in Fig. 3.2(b), the reply is sent directly to the authorization server over an out-of-band channel (4'). Next, the server notifies the terminal (of the corresponding HTTP session) about the verification result.

3.4 Implementation

We have designed and implemented a prototype for validating our constructions and showing the practical feasibility of our ideas.[1]

Specifications. The mobile application was written in Java running on an Android mobile (i.e., Samsung Galaxy i9000: 1 GHz ARM Cortex-A8 with 512MB RAM, a 480x800 WVGA Super AMOLED screen and as 2592 x 1944 Camera) and the relying party was implemented as a Java EE web service combined with a GWT web application running on a desktop (i.e., DELL E4300: Intel Core2 Duo P9600 @2.54GHz with 4GB RAM running Windows 7(64) with a HD 1280x720 Webcam). We refer to [56] for details on the implementation.

In addition, we present how, in the case of Identity Mixer anonymous credentials, the authentication is implemented using existing standards.

Standard Authentication Protocol. The authentication protocol based on anonymous credentials, is realized according to the HTTP/1.1 standard [40]. In short, whenever a request is made (01) and client authentication is required, the server replies with a [401 UNAUTHORIZED] response (01a). The header of this response contains a [WWW-AUTHENTICATE] header field, in which it passes the *authentication info*, containing challenge information in order to allow proper authentication. If the client decides to authenticate, she includes an [AUTHORIZATION] header field into the new request (01b), containing the *proof claim*. The server may then parse the response and verify the proof.

In the case of authentication based on anonymous credentials, the [WWW-AUTHENTICATE] header field contains the policies accepted by the server, and a challenge, allowing the client to authenticate correctly. The client may then compute a proof based on a chosen policy. The proof and a reference to the chosen policy is included into the [AUTHORIZATION] header field.

For our mobile to terminal authentication application, there is a slight difference between the two scenarios (see Fig. 3.2(a)). In the first scenario (a), the terminal plays the role of the client. It therefore sends a request to AS, obtaining the [401 UNAUTHORIZED] response, and converts this into a QR code. In order to limit the size of the QR code, only the [WWW-AUTHENTICATE] header field (i.e., the authentication info) is converted into a QR code, which is displayed on the screen and should be scanned by the mobile device's camera. Next, the mobile displays the policy to the user, waits for the user's consent, computes the proof and displays its proof and choice in a QR code, which is then scanned by the terminal's webcam. This QR code is parsed by the terminal and its content is included into the [AUTHORIZATION] header field of a new HTTP request to AS. In this case, the terminal is immediately notified about the verification result.

In the second scenario (b), after having scanned the QR code displayed on the terminal, and processing the user interaction, the mobile itself includes the proof and choice into the [AUTHORIZATION] header field of an HTTP request and sends it directly to AS. Note that in order to get notified, T has to frequently poll AS for the verification result related to a specific authentication request.

[1] Lines Of Code: ≈19000 (incl. middleware, mobile app, web services, terminal website and widgets).

3.5 Results and Analysis

Using the implementation of our prototype, a number of measurements were obtained. In this section we present and discuss these measurements. We looked at three different metrics that have a major effect on the overall system performance:

1. The execution time for a cryptographic Identity Mixer proof, that is, the time for constructing a proof and for verifying the proof;

2. The additional overhead introduced by the use of the secure element;

3. The encoding size of the Identity Mixer proof, which is relevant in the context of the bandwidth-limited QR channels we employ.

3.5.1 Identity Mixer Proof Execution Times

We present measurements for a spectrum of different variants of Identity Mixer proofs by using three sizes of the RSA modulus used for the protocol computations, credentials with zero and with three attributes, and proof specifications which hide or reveal all attributes, which need an inequality proof over one attribute, and a proof over one enumeration-type attribute. For these tests, the protocols are entirely computed on the mobile device (without a smart card). This leads to the proofs (a), (b), (c), (d), and (e) that are each constructed with the different modulus sizes 1024 bits, 1536 bits, and 2048 bits. We use the following encoding triplets as a shorthand notation for the structure (i.e., attributes and used features) of a proof: A_t, A_r, F with A_t the total number of integer attributes contained in the credential, A_r the number of revealed attributes, and F a feature to be proved or $\emptyset$ for no feature.

Proofs (a) and (b) perform a basic credential show in which only the fact that the user has a valid credential, is revealed. In case (a), the credential contains no attributes $(0, 0, \emptyset)$, while in (b) it contains three $(3, 0, \emptyset)$, none of them being disclosed. The proofs in (c), (d) and (e) use the same credential with 3 attributes as in (b), though, compute different proofs on them. In proof (c), all attributes are revealed $(3, 3, \emptyset)$. In (d) and (e), the attributes remain hidden and additional proofs are performed in addition to the basic credential show, resulting in more complex cryptographic protocols.

Table 3.1 summarizes the median values we have measured for the prover and verifier side of the protocol and the overall runtime for the proof variants (a) - (e) with the used modulus sizes, without considering communication overhead, nor user-interaction. We may notice that for (c) the resulting timings closely resemble those of (a), which can be explained by how the cryptographic proof is computed as shown in [50] Sect. 6.2.3: *Revealed attributes are realized with modular exponentiations with rather small exponents corresponding to the actual attribute sizes and thus have no major influence on the overall protocol runtime.*

Besides the above mentioned cases, without attributes and with three attributes, more extensive tests have been run with different numbers of attributes, and they clearly show a linearity in the computational overhead with respect to the number of attributes in the credential. Overall, the figures show the expected dependencies of the proof runtime on the key size and the number of non-released attributes of the credential.

Proof (d) illustrates the overhead caused by an inequality proof $(3, 0, \text{ineq})$. Proofs like this allow, for instance, to convince the verifier that the user's date of birth is more than 18 years back in the past. For such predicate to be shown with the Identity Mixer library, a substantial protocol runtime overhead is incurred, which confirms the expectations deduced from the protocol specification [50].

Similarly, in proof (e) $(3, 0, \text{enum})$, additional overhead results from the proof that an attribute of type enumeration (e.g., Drivers License Category $[A, B, EB]$) contains a specific value (e.g., B).

The modulus size of 2048 bits as recommended for high-security applications leads to a credential show having a runtime of at least about 0.9 s. Note that the computation time on the mobile

(ms)	1024			1536			2048		
(A_t, A_r, F)	prove	verify	total	prove	verify	total	prove	verify	total
(a) $0, 0, \emptyset$	103	78	181	240	187	427	495	375	870
(b) $3, 0, \emptyset$	139	125	264	323	265	588	634	515	1149
(c) $3, 3, \emptyset$	102	78	180	243	187	430	495	375	870
(d) $3, 0,$ ineq	481	436	917	1182	1077	2259	2358	2184	4542
(e) $3, 0,$ enum	247	213	460	617	510	1127	1259	1014	2273

Table 3.1: Timing Results (median over 100 runs), in Milliseconds, for Proving and Verifying a Credential Show with a Modulus of 1024, 1536 and 2048 bits

A_t: number of attributes in the credential
A_r: number of revealed attributes
F: feature to be proved

phone is comparable to that on the server, although the prover side of the protocol needs to perform more computations [50]. This is an unexpected result as the CPU of the phone has a lower clock frequency and less RAM than the one of the PC we used. However, further investigations related to the implementation of the BigInteger class in the Android environment showed that it invokes native code, while on the PC, the class is entirely implemented in Java. This explains the excellent performance of the mobile phone with respect to arbitrary precision arithmetic.

3.5.2 Overhead through the Secure Element

We have measured the overhead incurred during a credential show, because of the use of the optional Secure microSD card as a secure element for protecting the user's master secret key.

The figures in Table 3.2 show a substantial additional overhead compared to the timing results in Table 3.1. The overhead for each key length is fixed and independent of the proof specification. Moreover, it has no influence on the performance of the verification of the proof. As an example, a basic proof, case (a) with a 1024-bit modulus, now lasts about 1.44 s, of which about 0.18 s comes from the software and 1.26 s from the secure element fraction of the prover's protocol, compared to the 0.18 s without the secure element.

Compared to the full Javacard implementation of Bichsel et al. [7], taking 7.4 s, and the DAA implementation of Sterckx et al. [72] (which is partially run on the host) requiring 4.2 s, our protocol is substantially faster. Note that the DAA implementation provides protection against corrupted TPMs when issuing, which is not the case in our scheme in which we need a trusted setup of the card, during which a credential is issued to the card. In addition, the full card implementation offers enhanced privacy properties with regard to the host.

Table 3.2 also shows that a significant share of the overhead amounts to communication between the host and the secure element. This is partially explained by the current implementation requiring four rounds of communication. This can be reduced to only two cryptographic protocol requests of the Identity Mixer library, by combining the PIN verification and protocol selection rounds (i.e., issue or prove) with the first of those requests.

Note that for the 1024 and 1536-bit keys the delay due to the communication is the same, while for the 2048-bit modulus, the communication takes longer. The reason for this is that communication with the secure element happens in message blocks of 254 bytes. In case of 2048-bit keys, it does no longer fit into a single message block, resulting in an extra block and, hence, additional overhead.

3.5.3 Size of QR Codes

Proofs generated by the Identity Mixer library are formatted in XML, a verbose syntax. As the QR code-based channels are severely limited in bandwidth, and the Identity Mixer proof is the largest part of the content to be transferred back to the terminal, we created a customized space-efficient

(ms)	1024	1536	2048
build proof	1262	1606	2082
communication	310	310	375
computation	952	1296	1707

Table 3.2: Overhead, in Milliseconds, Incurred by the Secure Element, for a Modulus of 1024, 1536 and 2048 bits. The Overhead is Split Up into the Overhead Due To the Communication, and the Overhead Due To the Computation in the Secure Element.

(bytes)	1024	1536	2048
(a) $0, 0, \emptyset$	793	878	1005
proof	589	675	802
header info	147	148	148
response info	57	56	56
(b) $3, 0, \emptyset$	1053	1138	1267
proof	811	897	1024
header info	186	185	187
response info	56	57	57
(d) $3, 0, ineq$	3243	4031	4855
proof	2867	3657	4488
header info	319	317	311
response info	57	57	56

Table 3.3: Average Size of the Authentication Response (in bytes), for a Modulus of 1024, 1536 and 2048 bits, for Proofs with Credentials without Attributes (a), with Three Attributes (b, d) and with an Inequality Proof (d). The Total Proof Size is Divided into the Theoretical Proof Size, the Size of Header Info (e.g., names of attributes) and Response Info (e.g., session information).

binary format for representing Identity Mixer proofs. The format is based on an ordered length-value encoding. Note that it is straightforward to replace the formatting with a more standardized formats such as ASN.1 [38]. Table 3.3, presents the average size of the authentication response, containing as its major part the proof generated on the mobile. In the table, the message size is decomposed into: the *proof* size, being the theoretical number of bytes of the Identity Mixer proof; the *header info* size, being additional information required to encode the Identity Mixer proof in our custom format such as attribute names and lengths of proof values; and the *response info* size, being additional information such as a reference to the chosen policy. Table 3.3 also shows that different proof specifications result in quite different proof sizes.

For the more complex proofs, such as proof (d), the size of the proof becomes too big to be encoded in a single display-readable QR code: as defined by the QR standard, only about three kilobytes of binary data may be included in a single QR code. To work along those constraints, one solution is to use the out-of-band channel and send the proof through the radio channel to AS. Another solution, also implemented, is splitting the message into multiple chunks and cycle through the resulting QR codes until the reader has scanned them all.

Note that the generation of the QR codes on the mobile device currently takes a substantial fraction of the overall protocol runtime. When showing a credential without attributes, the QR code is generated in about 0.8 s. For the case of an interval proof with a 2048-bit modulus size, it requires two (larger) QR codes generated in about 2.5 s.

3.6 Discussion

For increased security and assurance, our system architecture and implementation comprises an optional secure element based on a Java Card microSD token. This achieves not only sharing prevention and theft protection for the user's master secret, but also a stronger binding between a user and her device through the PIN-based authentication. Those properties give the relying party a stronger assurance of the authenticating person having the claimed properties.

The short-range communication channel offers a higher level of security, in that the required *proximity* decreases the chance that another party is communicating with the terminal. However, the terminal is still not fully ensured about this, as the mobile could simply forward the messages to another mobile that performs the authentication instead.

Mobile devices are carried along most of the time, and therefore are an ideal target as a deployment platform. Though, today's mobile devices suffer from vulnerabilities that may make the software-based computations or the I/O between the user and her device untrusted (e.g., captured or provided by a virus). Trusted Execution Environments [34, 41, 42, 66] allow certain processes to be executed with a higher level of assurance, thereby ensuring that no malicious software can change computations or intercept the I/O of this process to the user. Developments on this are ongoing and can be employed as an orthogonal mechanism in our system architecture once they will be deployed on mainstream platforms.

3.7 Conclusion

We provide a solution to the authentication dilemma of users being required to identify themselves in most of their authentications today. We have brought anonymous credential systems to mobile devices as a privacy-preserving authentication solution and defined protocols for establishing secure channels between a user's mobile device and a terminal, based on short-range channels. This allows us to handle user-to-terminal authentication solutions through an easy-to-use system. While our protocol constructions apply to a range of short-range channels, we employ QR code technology to establish visual short-range channels in our prototype.

Our system is applicable to a wide range of practically-relevant authentication scenarios which users come across on a daily basis, ranging from user-to-terminal authentication such as age verification in a bar, over access to premises, to authentication to Web services from a home computer.

Future extensions on the protocol level may comprise including the introduction of the user accountability property [4, 23] through the use of verifiable encryption [20], or the support of credential revocation mechanisms, e.g., based on dynamic accumulators. Those features are not conceptually changing the constructions or architecture which are the main focus of this paper, but rather require some additions, like for key management.

Chapter 4

Network-based Access Control

4.1 Rationale and Motivation

Over the past couple of years, a multitude of authentication and access control technologies have been designed and implemented. Although not always clearly, a distinction can be made between two major categories: claim-based and network-based technologies. Each of them offers interesting advantages, yet also poses a number of drawbacks.

In both technologies, three important actors can be identified: the user, the service provider and the identity provider. In a network-based system, the user is typically redirected to an identity provider, which authenticates the user and supplies a *token* upon successful authentication. This token is then verified by the service provider and access is granted if it is valid. The procedure also includes the provisioning of any required user attributes, from the identity provider to the service provider. In a claim-based system, on the other hand, the service provider informs the user about the prerequisites to fulfill. If the user consents, he returns a *claim*. The latter can be regarded as a response to the server's challenge, along with a verifiable statement about required user attributes. A claim may be endorsed by one or multiple issuers. The user sits in between the identity providers and the service provider, thereby controlling the information exchange between the two sides. This leads to less trust assumptions and better privacy properties, in comparison to a typical network-based system, where the identity provider's role is more pervasive. The latter is also capable of transaction monitoring and linking, collusion with other identity or service providers, and user impersonation. In addition, the concentration of multiple security-critical responsibilities makes network-based identity providers a high-value (single) point of attack. However, in terms of interoperability and standardisation, network-based systems have an advantage over their claim-based counterparts. For instance, deployment of eID technology is typically country-wide, thereby imposing additional efforts at best, or excluding international users altogether. Moreover, there is a need to facilitate the adoption of new privacy-friendly credential technologies. The reuse of existing infrastructure and expertise is important in this respect. The design of this application synthesises a hybrid counterpart of network-based and claim-based identity management architectures, striving to leverage the strengths and to mitigate the weaknesses of both.

4.2 Application Description

4.2.1 Requirements

Prior to devising a suitable architecture, we list the requirements that should be fulfilled.

4.2.1.1 General requirements

G_1 Security issues in the network-based authentication protocol, are resolved or mitigated. The specific problems and the extent to which they are treated, is specific to the network-based system that is considered.

G_2 Pluralism of operators, claim-based credential technologies, as well as network-based authentication protocols, is facilitated.

4.2.1.2 User requirements

U_1 User trust in the network-based identity provider is decreased. Depending on the incorporated claim-based and network-based systems, this requirement ideally results in reduced capabilities for transaction linking and profiling.

U_2 The user has control over which personal attributes are released.

U_3 Switching to a different claim-based technology is facilitated.

4.2.1.3 Service provider requirements

SP_1 Service providers can obtain reliable user information for service personalisation.

SP_2 The service provider is agnostic towards credential technologies and authentication protocols: he needs to support only few implementations to reach the majority of users.

4.2.2 Components

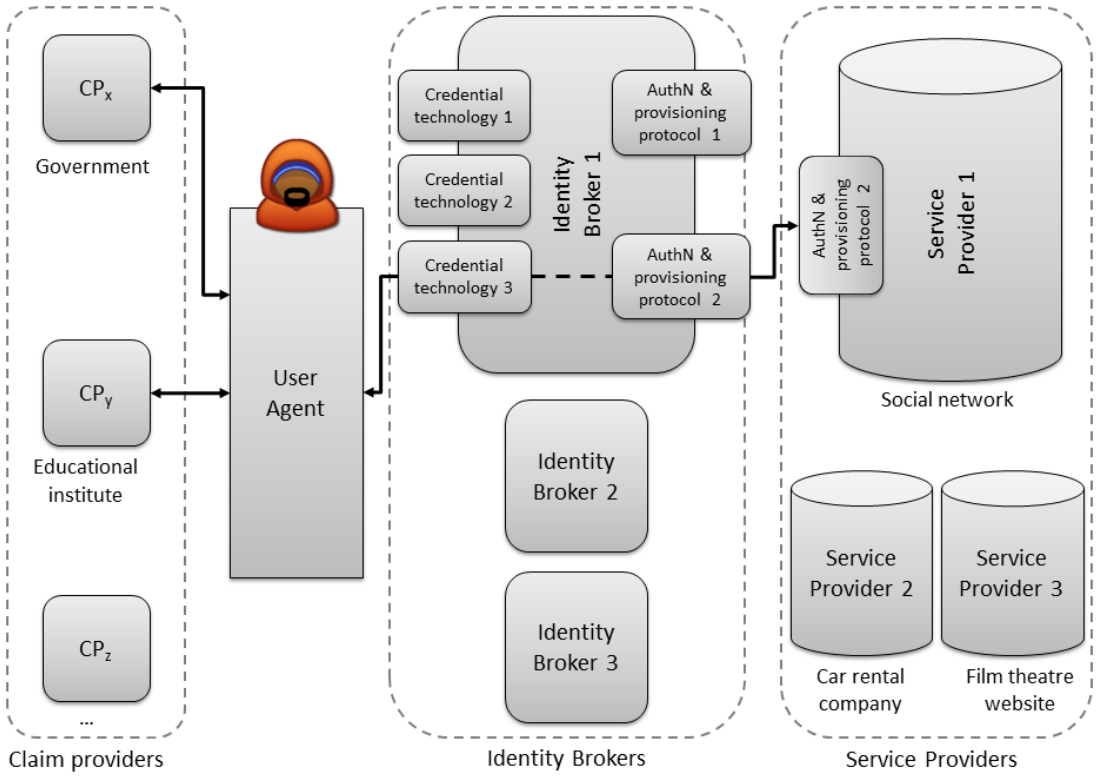


Figure 4.1: The proposed identity management architecture.

The proposed architecture is depicted in figure 4.1. There are four components: the *service provider*, the *identity broker*, the *user agent* and the *claim provider*.

4.2.2.1 Service provider

Service providers offer personalised services to users. To enable personalisation, the release of a set of user attributes is required during authentication. Service providers interact with an identity broker (see 4.2.2.2) to authenticate users and obtain personal information from them. Hence, their role and implementation is unchanged, compared to original network-based service providers.

4.2.2.2 Identity broker

Similarly to claim-based systems, the user presents to the identity broker a claim proving the fulfillment of the service provider's requirements. However, in the proposed architecture the identity broker extracts the user attributes or their properties, and provisions them to the service provider. This is performed according to a the network-based authentication protocol. Hence, brokers can be regarded as a conversion layer. They no longer store and maintain user information.

An identity broker implements credential-specific connectors to authenticate the user and obtain required user attributes. Each connector supports a credential technology. Once proven, the required attributes are transferred to service provider. This step can be achieved using any network-based authentication protocol that is supported by the identity broker: OpenID, SAML,...

4.2.2.3 Claim provider

The claim-based identity provider, or claim provider for short, corresponds with the issuer of a claim-based system. It is henceforth the only responsible for user attribute storage and maintenance, whereas the network-based identity provider is transformed to an identity broker. Each claim provider can support a different technology.

4.2.2.4 User agent

Similarly to the ABC4Trust project terminology, the user agent is responsible for managing the user's claims. Multiple technologies may be incorporated. Some necessitate no online presence of the claim provider during authentication, while others may require the user agent to fetch additional credentials or attributes on the fly. This component is typically implemented on a (portable) device, like a smartphone or a smartcard.

Apart from managing credentials and authenticating the user to service providers, the user agent also implements other useful functionality. This may include a connection component, providing support for different communication types: TCP/IP-based socket connections, HTTP, Bluetooth, etc. Furthermore, a privacy component can display the service provider the user is authenticating to, along with the personal attributes that are about to be released. In addition, the user agent allows the user to select the claims he wants to authenticate with. User preferences can be automated as policies. They comprise restrictions that can be posed on communication channels, credentials, attribute disclosure,... Lastly, the user agent can also give feedback about the anonymity impact of an ongoing authentication.

4.2.3 Trust relationships

This subsection shortly reflects on the changes in trust relationships.

4.2.3.1 User → identity broker:

The identity broker attests the user's identity to the service provider. The user no longer needs to entrust the broker with the storage of his personal data, as this is now assigned to claim providers,

of which the corresponding credential technology often provides better privacy properties than purely network-based approaches. For instance, user profiling is far less trivial in an anonymous transaction where only a common attribute (e.g. *gender = "F"*) or a property thereof (e.g. *age > 18*) needs to be attested.

4.2.3.2 User → claim provider:

The user trusts the claim provider to securely store a justifiable subset of his personal data. Note that the user has multiple partial identities that are distributedly stored across claim providers. Additionally, this actor is also trusted to only perform authenticated provisioning to the user agent. The claim provider cannot monitor or link the user's transactions.

4.2.3.3 Service provider → claim provider:

The service provider trusts the claim provider to only enforce genuine user attributes.

4.2.3.4 Service provider → identity broker:

The service provider trusts the identity broker to only assert verified attributes. Furthermore, the service provider may request from the identity broker to only accept credentials from a *trusted set of claim providers*. This allows the service provider to maintain a clear overview of trusted actors in the system.

4.3 Validation

4.3.1 Prototype Description

A proof of concept of the architecture in section 4.2 is implemented using Shibboleth as a network-based identity management system. The Shibboleth service provider remains unchanged upon its integration in the proposed architecture. This implies that the network-based authentication protocol between the service provider and the identity broker is unaltered, compared to a standard Shibboleth configuration. The user agent is implemented on a smartphone for two significant reasons. The device is nearly always in the user's presence *and* most often bound to the user exclusively. An additional advantage is the widespread adoption of smartphones. The user agent interacts with the identity broker. Modifications to Shibboleth to support this, consist of integrating authentication and provisioning support for two claim-based credential technologies. The first claim provider is an Idemix issuer, whereas the second one is part of Msec's internally developed identity management architecture. Both systems allow for selective disclosure and support pseudonymous and anonymous transactions.

4.3.1.1 Interactions

Two distinct setups are implemented. In the first one, a service is consumed on the user's smartphone. In the second configuration, the Web service is accessed on a device other than the user agent (e.g. a workstation and a smartphone, respectively).

In both cases, the user's browser is typically redirected to the Shibboleth Discovery Service. Next, the user selects his preferred identity broker. The latter is responsible for endorsing the attributes that are provided by the user agent. The next steps are discussed separately, due to the differences between both scenarios.

First setup: the device on which the service is consumed, also contains the user agent. In the first setup, the user agent on the smartphone has direct access to the session context of the transaction that was initiated with the identity broker. The user agent can, hence, authenticate the user through this session. This is similar to a standard Shibboleth setup that implements advanced authentication mechanisms, such as PKI-based authentication.

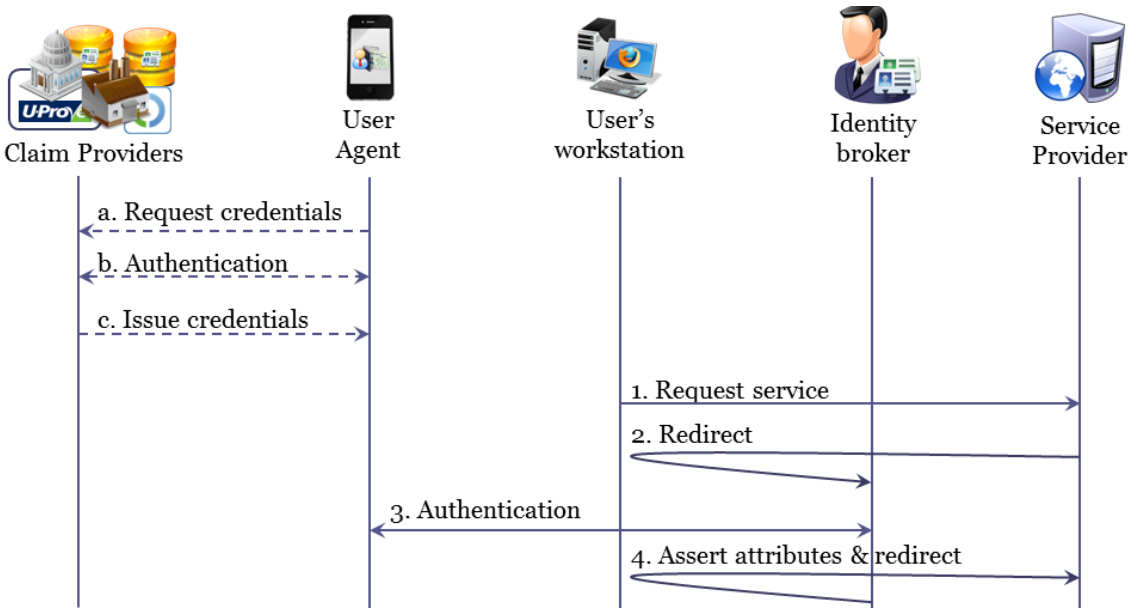


Figure 4.2: The hybrid access control procedure

Second setup: the device on which the service is consumed, does not contain the user agent. This setup is depicted in figure 4.2. It is slightly more complex, as the authentication and the service request do not share the same session context. Therefore, a binding needs to be established between the two. This is carried out as follows:

1. the identity broker generates a QR barcode, containing a credential-specific authentication challenge. In addition, it also contains a policy that must be satisfied for the authentication to succeed. For example, a policy may demand a proof that the user is over the age of 18, and that the authentication response be sent back only over HTTP. The barcode also contains the Id of the session that is to be authenticated.

2. the barcode is displayed by the device on which the service is consumed (e.g. a workstation's browser). The user subsequently scans the QR code with his the smartphone.

3. the user agent generates an authentication response and transfers it back to the identity broker, using a secure and authentic out-of-band communication channel.

4. The authentication response is verified. If valid, the identity broker extracts the user's personal attributes (or properties thereof) from the response and generates a SAML response, which is then transferred to the service provider, using the standard Shibboleth protocol.

4.3.1.2 Implementation details

The prototype builds upon Shibboleth v2. Section 4.2.3.4 mentions the ability of the service provider to specify a trusted list of claim providers. The implementation at hand attains this by incorporating the trusted list in the Shibboleth SAML Metadata. The latter is exchanged when service providers and identity providers join a Shibboleth trust federation.

Furthermore, the user agent is implemented on an Android-powered smartphone, running Idemix 2.3 and Msec's IdM architecture alongside each other.

Authentication Support for the two included claim-based credential systems is added by implementing a *Shibboleth LoginHandler* for each credential technology. This component authenticates the user, also obtaining required personal attributes along the way. During this procedure, the

identity broker transfers information about the service provider to the user agent. This allows the latter to inform the user about the authentication details.

Attribute provisioning The *standard Shibboleth approach* to impose restrictions on attribute disclosure, consists of the identity provider configuring *attribute filter policies*. Therefore, decisions about the disclosure of personal information are taken by the identity provider on behalf of the user.

The *proposed approach* entrusts the user with the decision of which attributes to disclose. The introduction of this notion, renders Shibboleth Attribute Filter Policies obsolete and leads to a new question. Namely, how can the user be informed of the service provider's policy? The prototype stores this policy in the Shibboleth Metadata, similarly to the trusted set of trusted providers. This allows the identity broker to obtain user attributes on behalf of the service provider.

Secure credential storage Secure credential storage is implemented on the Giesecke & Devrient Mobile Security Card 1.0. The storage requirements differ, depending on the credential technology being used. Msec's IdM architecture, for instance, relies on the tamperproofness of the card. Idemix, on the other hand, can either be run entirely on the smartphone if speed is an important factor or with the master secret securely stored in the secure micro SD card.

4.3.2 Evaluation

The identity broker is only responsible for user authentication and attribute provisioning. Storage of users' personal data is moved towards the claim providers, thereby reducing the identity broker's value and exposure as a target of attack. Additionally, some claim providers need not be online permanently, by which their issuance key is less susceptible to attack. Moreover, phishing is less likely to be successful, due to the authentication feedback that is displayed by the user manager. In addition, the latter could be extended to allow the user or administrator to preconfigure preferred identity brokers. Therefore, the architecture positively contributes to requirement G1.

Since users are no longer bound to the same party for personal data storage on one hand and authentication and attribute provisioning on the other, they can more easily switch identity brokers. The latter are now generic actors that support a variety of claim-based credential technologies and network-based authentication protocols, thereby giving users a broader choice and facilitating interoperability with service providers. Hence, requirement G2 is satisfied.

The identity broker remains a trusted party in the system. However, drawing from the properties of some claim-based credential technologies, the user's privacy is now better protected. This is, for instance, the case in transactions where non-uniquely identifiable information is released. In addition, some authentications are anonymous, making it harder to monitor and link transactions and to compile profiles. Therefore, the architecture positively contributes to requirement U1.

The user agent shows information about the service provider and the requested user attributes during authentication. This allows the user to give his informed consent prior to the disclosure. This satisfies requirement U2.

Requirement U3 is trivially satisfied, since an identity broker in the architecture supports multiple claim-based credential technologies.

A typical Shibboleth identity provider maintains his own data storage, while claim providers in the proposed architecture endorse their provisioned data and thereby convince the identity broker of its correctness. As previously mentioned, service providers can specify a *set of trusted claim providers*. Therefore requirement SP1 is fulfilled.

Furthermore, the identity broker implements a variety of network-based authentication protocols, making it interoperable with many service providers and fulfilling requirement SP2.

Chapter 5

Biometric Access Control

5.1 Rationale and Motivation

This chapter presents the design of a strategy for misuse prevention of credentials. The system impedes credential sharing and theft and it realizes informed consent to users. This prevents malware from abusing the credentials by misinforming the user about the transaction or authentication being processed.

The approach is applied to a scenario in which an individual accesses a remote and protected resource via a workstation. The user's credentials, stored on his mobile device, are bound to the biometrics of the user. This biometric binding is verified on the workstation, in a secure execution environment trusted by both the user and the service provider. This strategy avoids leaking biometric information to, for instance, service providers. This secure execution environment is realized by using the secure virtualization technologies currently available in many workstations and, hence, requires no additional hardware infrastructure to be rolled out. This environment is generic, hence, the user is not restricted to his personal workstation but can also use public workstations.

Related Work. Some credential systems such as Idemix [24] provide all-or-nothing non-transferability to discourage users from sharing their credentials. All the credentials of the user are tied together. Hence, assuming that the user owns at least one valuable credential, he will not be willing to share his credentials with other users. A similar approach is PKI-assured non-transferability where the credentials are bound to a valuable secret outside the system (e.g. credit card information). Where these systems focus on the prevention of credential sharing, the system in this chapter also focuses on theft prevention and informed consent.

In [10,51] the *wallet with observer* architecture [26] is extended to include biometric authentication towards the observer. The user is issued a tamper-resistant card containing the credentials and the biometric data of the user and a biometric scanner. Before the card will participate as prover in any protocol, it requires the holder to scan his biometric data which is subsequently matched with the data stored inside the card. In [30, 31] similar principles are applied in the design of a privacy-preserving identity card. These system are further improved in [9] where *Fuzzy extractors* [35, 36, 48] are used to generate a cryptographic key based on the retrieved biometric features. Subsequently, the key can be used during credential issuance and show protocols. The cryptographic key value is not stored with the credential, the tamper-resistant device is trusted to erase the value after authentication. Hence, fresh biometric readings are required to reconstruct the cryptographic key. These systems focus on theft and sharing prevention but do not realize the aspect of informed consent. Moreover, the tamperproof devices on which the credential operations are executed are typically resource constrained, impeding the performance of privacy friendly credential technologies [15, 24]. These tamperproof devices also decrease the flexibility of the system

with respect to using new biometric or credential technologies.

Other approaches prevent digital credentials from easily being copied and shared by embedding them in a tamper-resistant device. The system proposed in [19] leverages the DAA [17] protocols implemented in Trusted Platform Modules (TPMs) to bind the user's anonymous credentials to a TPM and, hence, a workstation. In [8, 61] implementations of anonymous credential systems on smart cards are presented. These systems have similar weaknesses, related to the use of tamperproof devices, as the systems described in the previous paragraph.

5.2 Application Description

This section first lists the different actors in the system. Subsequently, the requirements are listed.

5.2.1 Actors

5.2.1.1 Credential carrier

The Credential Carrier (C) stores the biometric template and the authentication credential(s) of the user. These credentials are only released to a predefined trusted application (i.e. the application maps to a known digest value $\overline{T}$) running in the secure environment. The credential carrier also contains a set of public keys of CAs trusted by the user. These are used to verify the authentication certificate of the service provider.

5.2.1.2 Service Provider

The Service Provider (P) provides services to users. Users are required to authenticate to prevent unauthorized users from gaining access. Moreover, the service provider requires that users prove ownership of the used credentials via biometrics. Ownership verification is done by an application on the workstation of the user. The service provider is assured that the verification was executed by a trusted application (i.e. the application maps to a known specified digest value $\overline{T}$) running in a secure environment. The service provider further owns a certificate Cert_P and private key SK_P to establish a secure authenticated channel with the workstation of the user.

5.2.1.3 Workstation

The Workstation is equipped with TPM based hardware technologies for running applications in a secure execution environment. The workstation is also equipped with a biometric scanner.

Legacy operating system. A regular operating system is running on top of the workstation of the user. The user can interact with service providers using an application (R), such as the browser, running on the operating system. If the user needs to authenticate or sign a transaction, the application leverages the authentication functionality of the *trusted authentication application* running in the secure environment to handle the authentication/signing.

Trusted authentication application. The secure environment capabilities of the workstation are leveraged to run a *trusted authentication application* (T). This application maps to a digest value $\overline{T}$ which is known to the credential carrier and service provider.

This application executes three main tasks. First, the credentials of the user are obtained from his Credential Carrier. This is done over an authenticated secure channel (i.e. C checks the asserted $\overline{T}$ against the known value). Second, it is verified that the user is the owner of the obtained authentication credentials. T therefore scans the biometric traits of the user via the biometric sensors attached to the workstation and verifies the biometric binding. Third, if the biometric binding verification was successful, T uses the authentication credentials of the user to satisfy the service provider's access control policy.

5.2.1.4 User

The User (U) wants to gain access to the services provided by the service providers via a workstation. The Credential Carrier is used to transfer the authentication credential and biometric certificate to the trusted authentication application on the workstation.

5.2.2 Requirements and adversary model

5.2.2.1 Functional requirements

F_1 The system for biometric binding is credential technology agnostic.

F_2 The solution should not require that authentication needs to be performed on one particular workstation.

F_3 The system can be realized using a commodity infrastructure.

F_4 The system is extensible, allowing for new biometric systems or algorithms and credential technologies to be included.

5.2.2.2 Security requirements

S_1 The misuse through credential sharing and theft is impeded.

S_2 Malicious software cannot mislead the user into approving malicious signing transactions or authentication attempts.

5.2.2.3 Privacy requirements

P_1 The system does not require releasing information that can be used to uniquely identify users or link user profiles of different service providers.

5.2.2.4 Adversary model

A_1 Regarding the secure execution environment, we make the same assumptions as the Trusted Computing Group [46]. This means that attackers can launch simple hardware attacks such as attaching a hardware debugger but sophisticated attacks such as monitoring the high speed bus that links the CPU with memory are not allowed.

A_2 T is assumed to be formally verified.

A_3 The biometric system used is secure (i.e. sufficiently low false acceptance and false rejection rate).

A_4 The credential carrier correctly verifies the integrity of the secure application running on the workstation before transferring the credentials of the user.

5.3 Implementation

5.3.1 Technologies

5.3.1.1 Secure execution environment

Where the TPM [46] is embedded in workstations for several years now, a more recent evolution is the adoption of secure execution environment technologies such as Intel's Trusted Execution Technology (TXT) [29] and AMD's Secure Virtual Machine (SVM) [32]. These technologies are nowadays being embedded in commodity workstations and allow the execution of code independently of previously executed software. The TPM specification has been extended with additional

capabilities dedicated to software running in the secure execution environment. Before these technologies can be used, they need to be activated through the BIOS of the workstation.

Recent work [60] implements a framework that allows developers to isolate security critical code from applications and run it in the secure execution environment. It, therefore, temporarily suspends the main OS and subsequently uses the secure execution environment technologies to securely execute the sensitive code. When execution of the sensitive code is completed, the OS resumes execution. The framework also allows passing data to the secure environment application and obtaining the resulting response. Moreover, in conjunction with the TPM *data attestation* and *secure session establishment* can be realized. This is further discussed in the following subsections. The framework supports both Windows and Linux systems. In [16] the framework is extended to allow secure user I/O. The system proposed in this chapter uses the framework and its extension for the implementation of a secure environment application. The execution of a secure environment application is denoted as shown below.

$$R \leftrightarrow T\text{: } output := \mathsf{invokeSecureAp}(input)$$

Data attestation. When the secure environment application has finished executing, it passes the *output* data it generated based on the provided *input* back to the operating system. Applications running on top of the operating system can now obtain the data and use the *attestation* capabilities of the TPM to ensure a remote party (V) that the *output* was produced by a specific (trusted) application. Each secure environment application is uniquely represented by a digest, denoted as $\overline{T}$. This digest can be mapped to the implemented functionality. The protocol is denoted as shown below. It can also be extended to include *input* attestation (i.e. assert that *output* was generated based on *input*). However, this property is not required for the system presented in this chapter and is, hence, not reflected in the protocol description below.

$$R \leftrightarrow V\text{: } \mathsf{assertData}(\overline{T},\ output)$$

Secure session establishment. The *secure session establishment* protocol allows setting up a secure channel between an application running in a trusted environment (T) and a remote party (V). The remote party is ensured that a fresh session key (K_s) is established with the secure environment application represented by $\overline{T}$.

$$T \leftrightarrow V\text{: } K_s := \mathsf{secureSession}(\overline{T})$$

5.3.1.2 Credential technologies

In this chapter, abstraction is made of the different properties achieved by the credential technologies that can be used for user authentication. Two credential primitives are defined. The first operation generates an authentication response (*auth_resp*) using an authentication credential of the user ($Cred_U$) based on the authentication request (*auth_req*, typically containing the access control policy and a challenge) and the X.509 certificate (Cert_P) of the service provider. Including the certificate of the service provider prevents the authentication response from being used to authenticate towards other service providers than the one intended by the user. The access control policy states the requirements for gaining access to the service. Subsequently, the authentication response can be verified using the public key of the credential issuer (PK_I).

$$U:\ auth\_resp := \mathsf{genAuthResponse}(Cred_U,\ auth\_req,\ \mathrm{Cert}_P)$$

$$P:\ true/false := \mathsf{verifyAuthResponse}(auth\_resp,\ PK_I,\ auth\_req)$$

5.3.1.3 Binding credentials to biometric data

Biometry can be used to uniquely identify a person [52, 53]. Commonly used biometric traits include a fingerprint, iris, face and voice. A sensor *scans* the biometric trait of the user. During enrollment, a distinguishing feature set is extracted from the biometric data and stored as a *Biometric Template* (BT_U) of the user. During authentication, the user *scans* his biometric trait. The resulting feature set is *matched* to the feature set contained in the template of the user. Based on the similarity between the two sets the authentication is either accepted or rejected. These biometric templates can be bound to the authentication credentials of the user. This allows for verifying that the user of the credentials is also the owner of these credentials. The binding verification procedure is denoted as shown below. In section 5.3.3, a more elaborate description on the realization of the binding between credentials and biometrics can be found.

$$V: \textit{true/false} := \mathsf{verifyBioBinding}(\mathit{Cred}_U, BT_U, \mathsf{scan}(U))$$

5.3.2 Protocols

5.3.2.1 Using the user's authentication credentials

The workstation initiates the trusted authenticated application and passes the access control policy and the certificate of the service provider. The trusted authentication application retrieves the credentials of the user and verifies the biometric binding. If the binding is successfully verified, it generates an authentication response and passes it back to the workstation as output data. The protocol is described in more detail below and in Table 5.1.

After receiving the service provider's access control policy and certificate (1), T initiates a secure session with the credential carrier (C) of the user (2). C is hereby assured that T is an instance of the trusted authentication application. Subsequently, the certificate of the service provider is transferred to C where it is verified using the set of CAs trusted by the user (3-4). C now transfers the authentication credential(s) (Cred_U) and biometric template (BT_U) of the user to T (5). Subsequently, T notifies the user about the service provider and his access control policy (6). If the user wants to carry forward with the authentication, he scans his biometric data with the biometric reader attached to the workstation (7). T now verifies if the scanned biometric data matches with the biometric template bound to the credentials of the user (8). If so, an authentication response is generated using the authentication credentials (Cred_U) of the user and the authentication request and certificate of the service provider (9). When the secure authentication application is terminated, the *auth_resp* is passed back to the workstation (10).

	$T \leftrightarrow R$: *auth_resp* := invokeSecureAp(*auth_req*, Cert_P):	
(1)	$R \rightarrow T$:	*auth_req*, Cert_P
(2)	$T \leftrightarrow C$:	$K_s :=$ secureSession($\overline{T}$)
(3)	$T \rightarrow C$:	$\{\mathrm{Cert}_P\}_{K_s}$
(4)	C :	**if** (! verify (Cert_P)) abort()
(5)	$T \leftarrow C$:	$\{\mathit{Cred}_U, BT_U, PK_I\}_{K_s}$
(6)	$U \leftarrow T$:	*auth_req*, Cert_P
(7)	$U \rightarrow T$:	*bio_features*
(8)	T :	**if** (! verifyBioBinding(Cred_U, BT_U, *bio_features*)) abort()
(9)	T :	*auth_resp* := genAuthResponse(Cred_U, *auth_req*, Cert_P)
(10)	$R \leftarrow T$:	*auth_resp*

Table 5.1: Using T to satisfy the access control policy.

5.3.2.2 Accessing remote resource

The user wants to gain access to a remote protected resource via his workstation. The service provider enforces an access control policy to ensure that only legitimate users access the resource. Moreover, the service provider requires assurance that the owner of the credentials that are used during authentication is in the proximity of the workstation. The protocol is described below and in Table 5.2.

The user requests access to a remote protected resource via an application running on his workstation (1). Subsequently a secure session is established during which the service provider authenticates (2). In the context of Web applications, this is typically a HTTPS connection with server authentication that is established. The service provider now sends the authentication request to the workstation over the secure channel (3). The workstation now invokes the secure authentication application that verifies the credential binding and generates an authentication response (4). See previous section for a detailed description of the internals of the secure authentication application. The workstation now uses te assertion capabilities of the TPM to assure the service provider that *auth_resp* was generated by the trusted application (5). This ensures the service provider that the response was generated using credentials whose owner was physically present at the workstation. Subsequently, the service provider verifies the authentication response (6). If the response satisfied the access control policy, access is granted to the protected resource over the secure channel (7).

		$U \leftrightarrow P$: accessRemoteService():
(1)	$U \rightarrow R \rightarrow P$:	`access_request`
(2)	$R \leftrightarrow P$:	$K_s :=$ authKeyAgreement(Cert_P, SK_P)
(3)	$R \leftarrow P$:	$\{auth\_req\}_{K_s}$
(4)	$T \leftrightarrow R$:	$auth\_resp :=$ invokeSecureAp($\{auth\_req, \text{Cert}_P\}$)
(5)	$R \leftrightarrow P$:	assertData($\overline{T}$ $auth\_resp$)
(6)	P :	**if** (! verifyAuthResponse($auth\_resp$, PK_{UI}, $policy$)) abort()
(7)	$U \leftarrow R \leftarrow P$:	$\{resource\}_{K_s}$

Table 5.2: The user gains access to a remote resource via his workstation.

5.3.3 Binding credentials to biometric data

Several mechanisms can be used to bind the biometric template of the user (BT_U) to the user's credentials ($Cred_U$). This section further discusses some of these mechanisms.

Attribute based binding. The user registers with a biometric certificate issuer (e.g. the government) which subsequently provides the user with one or more biometric certificates. These certificates contain user information (e.g. name, data of birth, address, NRN) and the biometric template of the user.

To verify the binding between the user's biometric template and the user's credential(s), the trusted environment application matches the attributes contained in the credential with the information contained in the biometric certificate(s). If a matching set of information is found which (sufficiently) uniquely identifies the user, the biometric template contained in the biometric certificate can be used to identify the owner of the credential(s).

This approach is very scalable. Once the biometric certificates are issued, no additional overhead is introduced when issuing credentials. While the binding verification procedure can be straightforward when the same unique attribute (e.g. the NRN) can be found in both the biometric certificate and the credential, it can also be complex if unique subsets of user information need to be matched. Moreover, the credentials need to include a set of attributes that sufficiently uniquely identifies a user and which is also contained in one of the user's biometric certificates.

This approach is especially useful for integrating the system proposed in this chapter in an existing credential infrastructure.

Credential based binding. In credential based binding, a cryptographic hash of the user's biometric template is included in the user's credentials.

Before using a credential, the trusted environment application verifies that the cryptographic hash of the biometric template corresponds with the value included in the credential and that the scanned biometric data matches with the biometric template.

This system allows a straightforward binding verification procedure. It, however, introduces additional overhead as for each new credential the biometric template needs to be included. The overhead, could, however, be reduced by extracting the template from an existing credential.

Hardware based binding. In hardware based binding, the credentials of the user are stored on a tamperproof device (embedded in C), together with the biometric template of the user. The trusted application running on he tamperproof device is certified, allowing it to authenticate to a third party.

The trusted environment application on the workstation initiates an authenticated secure channel with the tamperproof device. The tamperproof device subsequently transfers the credentials and biometric template to the trusted environment application. The credentials and biometric template are, consequently, bound under the tamperproofness assumption.

This system allows a straightforward binding verification procedure and, moreover, provides a secure environment for storing the credentials of the user. However, it also requires an additional hardware infrastructure to be rolled out and it increases the complexity of the credential issuance phase as the credentials need to be stored on the tamperproof device over an end-to-end encrypted channel with the credential issuer. Moreover, it does not allow easy integration of existing credential infrastructures. This approach is especially useful in settings where a tamperproof hardware infrastructure is already available such as bank or SIM cards.

5.4 Evaluation

5.4.1 Security and privacy analysis

5.4.1.1 Requirements review

This section discusses how the requirements presented in section 5.2.2 are realized in the design.

Functional requirements. A credential technology independent notation was used during the system design. Furthermore, as discussed in the privacy requirements review section, the system does not require the disclosure of any uniquely identifiable information to the service provider. This allows privacy preserving credential technologies to be used to their full extent, satisfying requirement F_1.

The only component in the system that is uniquely dedicated to one user is the credential carrier. This component can be implemented on a mobile device such as a smart card or a smartphone. T is user independent, hence, not restricting the user to his own workstation(s), satisfying requirement F_2. The workstation does need to meet certain requirements. It needs to be equipped with a biometric scanner and provide a secure environment for running T.

The technology for secure environment applications is being embedded in commodity off-the-shelf workstations, satisfying requirement F_3. However, the used technologies are not by default enabled. Hence, some BIOS configuration changes need to be made and some additional software deployed (i.e. the trusted environment binary and some middleware).

The trusted environment application can easily be updated by distributing a new binary to the workstation and updating the digest of the application with the credential carrier and service providers. This can be managed by a trusted third party that certifies and revokes these

digests. This allows the integration of additional biometric technologies or credential technologies, satisfying requirement F_4.

Security requirements. The authentication credentials of the user are bound to his biometrics. The trusted environment application requires a reading of the user's biometrics to verify the biometric binding. The service provider is assured that the credential proof was generated by a trusted application, ensuring that the biometric verification was performed, satisfying requirement S_1.

The credential operations are executed in the secure environment on the workstation. Its integrity is verified by the service provider and the credential carrier of the user. Hence, the user can trust the information provided by the application and, hence, give an informed consent to the transaction. Since the authentication response is bound to the certificate of the service provider, it cannot be abused for authentication towards other parties, satisfying security requirement S_2.

Privacy requirements. To protect the privacy of the user, the system should avoid leaking additional information, apart from what is leaked by the used credential technology. If *DAA* is used to assert the data from the trusted authentication application, the system only leaks the group information in which the *DAA* credential was issued and the digest of the trusted authentication application ($\overline{T}$). If sufficient TPMs are included in the *DAA* group, this information cannot be used to identify users or workstations. As $\overline{T}$ is also released to the remote party, it must not be uniquely identifiable. This can be realized by embedding the platform dependent code (i.e. the drivers for the biometric reader(s)) for all the supported platforms in T. This results in an identical $\overline{T}$ for all users but is less flexible (e.g. adding new drivers) and increases the trusted computing base. A second option is deploying a limited set of T binaries, each targeting a single or limited set of hardware configurations. This decreases the anonymity set of the user but is more flexible and decreases the trusted computing base compared to the previous option. Hence, privacy requirement P_1 is satisfied.

5.4.1.2 Adversary model discussion

Assumption A_1 is required to ensure a trustworthy attestation of T. If the attestation is not trustworthy a malicious user can use credentials obtained from a legitimate user (i.e. stolen or shared) to satisfy the access control policy of a service provider. It also allows malicious users to steal credentials from legitimate users as C can be wrongfully assured that a trusted application is running. To mitigate this risk, the keys of compromised TPMs should be revoked.

Assumption A_2 is required for similar reasons as A_1 since implementation flaws could be exploited to produce undesired behavior of T. The functionality of the trusted authentication application is kept minimal (i.e. obtaining user credentials, biometric verification and user authentication). The code contained in the trusted application is therefore minimal, decreasing the chance of bugs and suggesting that formal verification is possible. Moreover, as mentioned in the functionality requirements review section, the trusted environment application can be easily updated after which the previous version can be revoked by blacklisting the digest (i.e. $\overline{T}$).

Assumption A_3 ensures that the *false acceptance* rate of the biometric technology used does not allow malicious users to use the credentials of other users. This could by achieved by using multiple biometric traits.

Assumption A_4 ensures that the credentials of the user are not leaked to an attacker. This would impact the privacy of the user as private information may be contained in the credentials and provides opportunity for abuse if service providers do not require biometric binding verification.

5.4.1.3 Discussion

Although the system provides additional protection against credential sharing and theft, there exist attack vectors that can be used to exploit the system. These attack vectors are further discussed in this section.

Biometric verification. Ideally, the biometric reader used for obtaining the biometric data of the user is bound to the workstation on which the user is working. However, most biometric scanners are plug-and-play and can, hence, be easily removed and replaced with other hardware devices. If a fingerprint reading of the user is obtained, it can be replayed as a fresh reading. This risk can be mitigated by implementing a protocol between the trusted environment application and the reader that endorses the biometric reading. To prevent relay attacks, the trusted environment should also be able to verify that the used biometric reader is, in fact, attached to the workstation on which the trusted environment is running.

Untrusted OS. The service is consumed via the untrusted workstation, this can have implications on the security and privacy of the system.

To violate the privacy of the user, the service provider can use application fingerprinting [39] or network traffic analysis to link different sessions of the user. Hence, techniques need to be used to anonymize network traffic [64, 65] between the workstation and the service provider. This is not enforced by the system, hence, the workstation needs to be trusted to correctly apply these techniques to protect the privacy of the user.

Malware on the untrusted OS can hijack and transfer active sessions the user established to a remote workstation. Similarly, a user could exploit this to authenticate to a service on behalf of a friend. Especially data provisioning services (e.g. news sites) are vulnerable to this type of attack. Services such as eBanking where transactions are signed are less vulnerable to this type of attack as they can be entirely processed in the trusted environment.

5.4.2 Applicability

This section discusses two possible application domains in which the system presented in this chapter can be used to realize increased security compared to currently deployed systems, namely *eID systems* and *online banking*.

5.4.2.1 eID Systems

typically allow the user to access a wide range of personal services. This can go from very privacy sensitive services such as online tax submission to less privacy sensitive services such as pay-per-view news site. While these privacy sensitive services will typically only be accessed from a trusted home computer, other services might be accessed on workstations not fully trusted by the user. However, when using the same authentication credentials for the privacy sensitive and the other services, malware on an untrusted workstation could use the authentication credentials to access other services then requested by the user. The system presented in this chapter prevents this type of abuse by correctly informing the user about the service which will be accessed.

5.4.2.2 Online banking

enables a wide variety of banking services (e.g. viewing the status of your bank accounts, loans and execute bank transactions) via a workstation connected to the Internet. The user owns a credential with which he can log in and authorize transactions (e.g. wire transfer).

In current eBanking systems this secret is stored in an smart card (i.e. the bank card of the user). To authorize transactions, the details of the transaction are transferred to the bank card that subsequently generates an authorization response. This approach protects the credentials of the user but is vulnerable to phishing attacks. Our approach also tackles the latter as the secure execution environment application correctly informs the user about the pending transaction.

Chapter 6

Context-Aware Services

6.1 Rationale and Motivation

As the user becomes swamped with more and more information, filtering this information becomes ever more important. This is especially true for mobile devices where small screens and limited input capabilities form an obstacle when searching for information. Recommender systems can offer a solution because they learn the preferences of the user and filter out the most relevant information. On mobile devices, this recommendation process can be further improved by taking into account the user's current context and even their current activity.

Smartphone and mobile internet usage is growing at a fast pace. For some time now, websites have been using context in a limited way to deliver a personalized experience. A typical use case is the delivery of advertisements tailored to the user's language and general location. Connected mobile devices can derive a much more detailed model of the user context by using various physical sensors and online information services. The GPS receiver provides a precise location. Accelerometer data can hint at the current method of transportation (e.g. walking, cycling, riding the train). Online information services can tell us if the user is in a busy city center, in an urban space or in a rural environment. All of this information can be used to anticipate the user's needs and personalize various applications.

6.2 Application Description

6.2.1 Requirements

The primary goal is to develop a software framework that is capable of detecting the activity and context of the user of the mobile device. Secondly, a proof of concept application must be developed that leverages this framework to deliver a personalized service. This application must recommend the most relevant and interesting information to the user based on their current activity and context, without losing sight of the need for privacy and transparency.

6.2.1.1 Activity Detection Framework Requirements

- Detect basic activities such as standing still, sitting down, walking, running, cycling.
- Detect complex activities such as sitting down on a moving train.
- Easy and efficient integration into other applications.
- Limit battery drain.

6.2.1.2 Proof of Concept Requirements

- Integrate the activity framework so the user context can be determined.
- Create a personal interest profile for the user based on the locations they visit, the activities they take part in and the events they attend.
- Assist the user in quickly finding the information that is most relevant under the current circumstances through a context-aware recommendation system.
- Design and implement a modular, context-aware recommendation framework that combines multiple techniques. A modular approach allows for specific recommendation techniques to be turned off in the interest of privacy and transparancy.
- Provide a graphical user interface for presenting points of interest and collecting feedback.

6.2.2 Components

This section describes the components of the entire proof of concept application. Like the client application, the activity detection framework runs entirely on the mobile device itself, but its description is kept separate from the (graphical) user client.

6.2.2.1 Activity Detection Framework

The detection framework runs as a background service so that it can still collect sensor data when the client application is sent to the background. Activity detection occurs through three distinct steps, where the output of each phase is used as the input for the next phase. At the end of this process, raw sensor data from a multitude of sources will have been converted into a complex activity.

- Monitor sensors, physical or otherwise.
- Process the sensor data.
- Analyse the data to determine the current context.

6.2.2.2 Server

The server-side part of the proof of concept has the following components:

- **Recommender system:** a hybrid recommendation system that combines multiple techniques to determine what type of activities fit a particular user context.
- **Information retrievers:** components that retrieve data from external services such as point of interest data or event information.
- **Support functions:** various methods for creating and managing user profiles.
- **Connection:** server functions are exposed through a webservice.

6.2.2.3 Mobile Application

The mobile application is made up of five main components:

- **ApplicationManager**: creates an instance of the application according to the *model-view-controller* pattern and manages access to the other application components.
- **ServerManager**: used for communicating with the server.
- **RecommendableManager**: retrieves a list of recommended activities or events.

- **ContextManager**: integrates the activity detection framework and notifies other components when the context changes.
- **FeedbackManager**: collects and stores user feedback.

6.3 Validation

6.3.1 Prototype Description

6.3.1.1 Activity Detection Framework

The implementation details of the three phases of activity detection (sensing, processing, analysis) are discussed here.

Monitor sensors. Mobile platform have many kinds of sensors. Some sensors only deliver new data when a particular value has changed. An example of such a sensor is the proximity sensor, which is only triggered when the user raises the phone to their ear. These sensors can be monitored all the time. Other sensors, like GPS, deliver new data constantly. This type of sensor will use up a lot of battery power, so the framework tries to monitor them only when needed. In addition to hardware sensors on the phone itself, the framework provides custom sensors that use data from online services. An overview is given here:

- Acceleration
- GPS location
- GSM Cell ID
- Proximity to ear
- Battery status
- Urbanization degree
- Weather

Process sensor data. Before it can be analyzed, the raw sensor data must be processed into structured data. The framework offers processing implementations that can each be attached to a particular sensor. A few interesting Processors are described here.

- **MovingToProcessor**: takes the output of the GPS sensor and generates a list of interesting hotspots (locations) that the user is currently moving towards.
- **SpeedProcessor**: uses the GPS sensor output to determine the user's current velocity.
- **HotSpotProcessor**: using either GPS or CellID data, this processor determines whether the user has arrived at a particular hotspot.
- **CellIDProcessor**: receives a list of recent locations as determined by the CellIDMonitor and computes the number of cell handovers and approximate distance traveled.
- **AccelerationProcessor**: determines the basic user activity (sitting, walking, running) from the X-, Y- and Z-axis acceleration data provided the AccelerationMonitor. A Support Vector Machine model was trained with motion data from 11 test users, allowing the framework to quickly classify the user's current movement type.

Analyse data. In this final step, the output of the Processing phase is analysed to determine the actual user context. A number of *Goals* may be triggered when framework is reasonably sure that the processed data meets certain requirements. To decrease the chance of incorrectly classifying the context, multiple iterations may be required before the framework will conclude that a particular Goal was reached.

The basic principle is that a complex activity such as *sitting down on the train* or *walking around in the city center* can be defined as *goals* between which there exists some temporal relationship, such as the activities taking place simultaneously. This way, a developer that is using the detection framework can easily define what actions the application should perform for a given user context.

A few Goals are defined here:

- **BasicActivityGoal**: determines whether the user is standing still, walking, running or cycling.
- **CityGoal**: triggered when the user is in a city.
- **TrainCellIDGoal**: activated when the battery is being charged.
- **TimeGoal**: determines the time of day, such as morning, evening, office hours, weekday, weekend.
- **HotspotGoal**: triggered when the user reaches a particular location.
- **WeatherGoal**: decides on weather conditions, such as sunny, (partially) clouded, rainy.

Two kinds of temporal relationships are currently supported:

- **ParallelCondition**: this condition checks whether two or more Goals are met simultaneously.
- **TimeCondition**: checks whether two goals were met independantly of eachother within a given timeframe.

6.3.1.2 Context-Aware Recommendation Framework

Recommending context-appropriate events or activities is a three-phase process. Figure 6.1 gives a schematic overview of the process.

First, four models determine which categories are most appropriate for the given user context. Events or activities that are inappropriate under the given context are discarded.

- **ActivityModel**: a knowledge-based approach that consists of basic rules. Users that are walking towards a train station are likely interested in train schedules. Users that are in a new city in the evening may be interested in restaurants. This model requires no prior knowledge of the user's interests and helps to solve the cold start problem of recommendation systems.
- **HistoryModel**: keeps track of the user's history and explicit feedback and thus learns what type of events or categories have proven relevant for this user under a given context.
- **PopulationModel**: uses the same data as the HistoryModel, but from all users in the system to identify general trends in the population.
- **PreferencesModel**: lets the user set explicit preferences for what type of activities or events to suggest for a particular context.

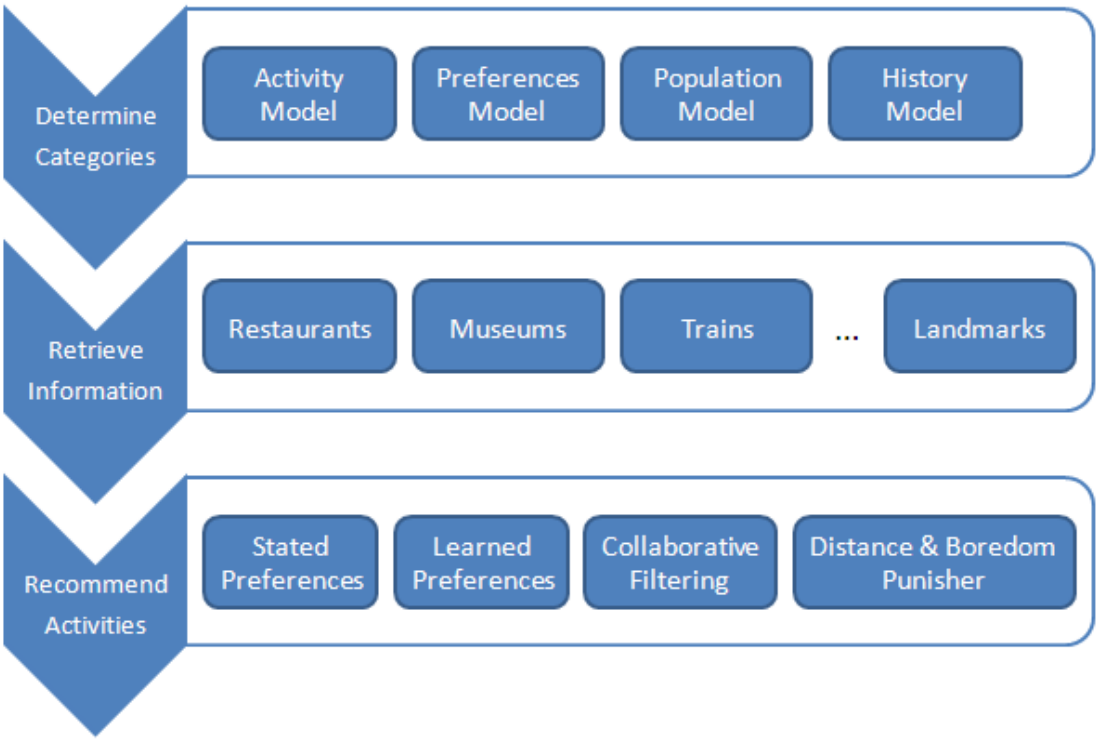
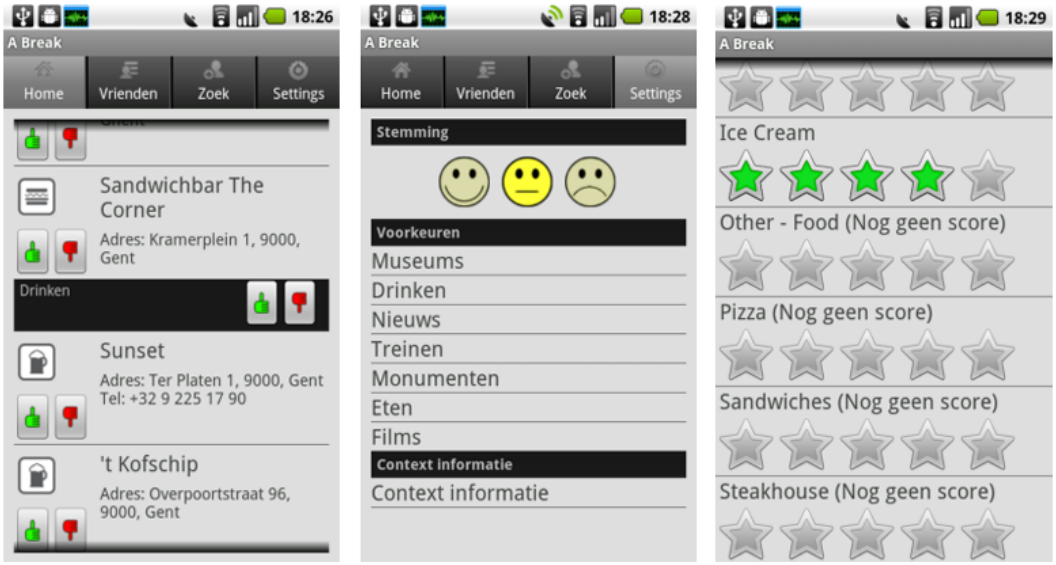


Figure 6.1: Context-aware recommendation framework.

After determining which categories are best suited, the framework retrieves activity or event data from online services. This way, the framework does not spend power and network traffic retrieving items that will not be recommended anyway.

Finally, the framework assigns a score to each event or activity based on the user's own interests based on five recommendation models. Some models again incorporate the user context, while other models may use general trends from the entire user base.

- **CollaborativeFilteringModel**: a straightforward collaborative filtering approach using Pearson correlation that does not consider the current context.
- **StatedPreferencesModel**: uses explicitly stated preferences to rank the activities or events.
- **LearnedPreferencesModel**: favors activities or events with metadata that is similar to items that the user has rated positively in the past under similar contextual circumstances.
- **DistancePunisherModel**: discourages activities or events that are further away. This is a function of both current method of transportation and weather circumstances; users will be willing to walk a little longer if it is sunny.
- **BoredomPunisherModel**: attempts to increase serendipity and novelness of the recommended activities or events. This has a stronger influence on some categories than others. The user will not be recommended a movie he has already paid to see. A restaurant that was visited previously and rated favorably may again appear in the list of recommendations, but behind a new venue.



(a) (b) (c)

Figure 6.2: Screenshots of the mobile application showing the Settings screen (a), stated preferences (b) and recommendation (c).

6.3.1.3 Mobile Client

After starting the application, the user is asked to either provide his credentials or create a new account. The user is then presented with four application tabs that each serve a specific purpose. For the purpose of this demonstrator, we will focus on the *Home* and *Settings* tabs. The *Friends* and *Search* tabs allow the user to manage his friends list and indicate who is currently in their company. This is additional information that can enrich the context even further, allowing for recommendations that take the entire group's interests and context into account. This functionality will be available in a future iteration.

Home The Home tab, seen in 6.2(a), presents a list of nearby points of activities, sorted by relevance given the current user context. The items are grouped by category. The user can give explicit feedback in the form of a thumbs up or thumbs down rating. The application tries to show as much background information as is available. In the case of a train, this includes the departing time, platform number and estimated delay. Movies include a brief description and a list of screening times.

Settings The Settings tab, shown in 6.2(b) and 6.2(c), lets the user assign an explicit preference for the various categories. This solves the cold start problem of recommendation systems as well as giving more insight into why a particular activity is being recommended. It also lets the user set their mood, since this is not something that can be detected automatically.

6.3.2 Evaluation

In this section we evaluate if the activity detection and proof of concept meet the stated requirements. We also let a small panel of test users evaluate the application in a real-world setting.

6.3.2.1 Activity Detection Framework

The activity detection framework is able to detect basic activities such as walking or standing still as well as more complex contexts like *walking towards the station in the rain*. The proof of concept that we have developed illustrates how applications can integrate the framework. It is sufficient to start a background service, activate the desired sensors and register listeners that will be notified when the context changes. By only monitoring desired sensors, the framework reduces battery drain. Other techniques such as falling back on GSM Cell ID rather than GPS further reduce power consumption.

6.3.2.2 Proof of Concept

The proof of concept consists of a mobile application and a context-aware recommendation framework. This recommendation framework combines multiple techniques to find the most appropriate content. The PopulationModel uses data of all users, reducing transparency and posing a risk to the privacy of the user. If the user were to choose to disable this particular model, no data would have to be sent to a central server and the user would still get recommendations based on the other three models. The ActivityModel, PreferencesModel en HistoryModel approaches can all be performed on the mobile device itself. The framework can thus offer better recommendations to users that choose to disclose some information, while still adding value for users that wish to keep their usage data entirely private. The mobile application is able to deliver a personalized experience to users, whether they are in their hometown or an entirely new location.

6.3.2.3 User Test

To test whether the application creates any value for the end-user, we asked a test panel of sixteen users to install the proof of concept on their smartphone and try it out in their daily activities. The users, all between the ages of 20 and 30, had a lot of experience with smartphones and frequently used their mobile device in their daily life. They were asked to use the application at least once per day, and at least three times outside their home. After one week, they were given a survey consisting of nine multi-point (*Likert-scale*) questions and three open questions. Figure 6.3 shows the response to three of the multi-point questions, indicating users generally felt the application is useful and provides interesting recommendations.

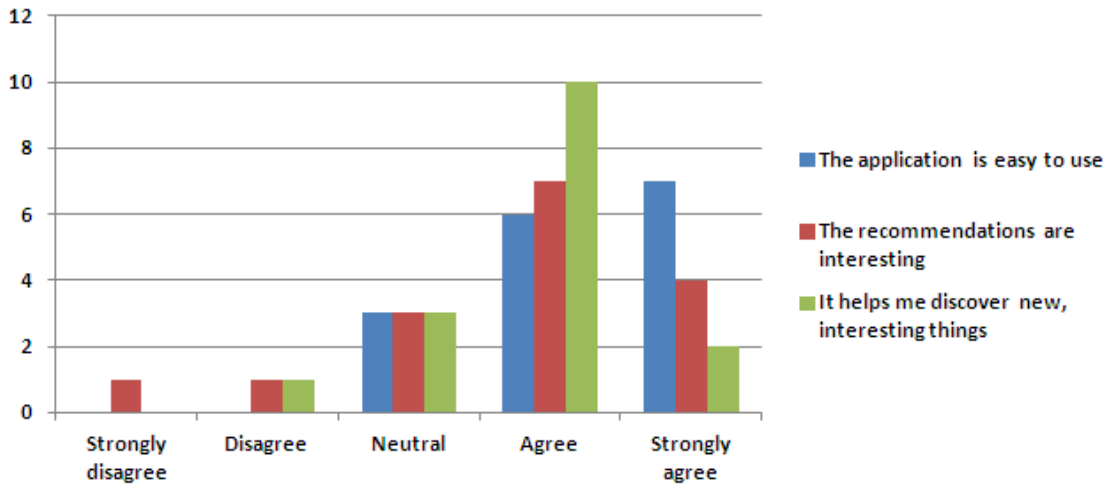


Figure 6.3: User test survey response.

Chapter 7

Mobile Electronic Payments

7.1 Rationale and Motivation

Mobile electronic payments are rapidly becoming a reality. There is no doubt that users of mobile phones are willing and even asking to be able to use their phone as a convenient and always available payment method. In May 2012, Gartner [43] predicted a total of $171.5 billion in mobile payment transactions for the year 2012, a rise of over 60 percent compared to 2011. They estimate a total of 212.2 million people (an increase of 32 percent from 160.5m in 2011) using some form of mobile payment service. They predict mobile payments will keep increasing to a volume of $617 billion by 2016.

For the moment, SMS based payment schemes are dominant in developing markets, while in more mature markets, about 80% of payments are made via mobile web portals. NFC adoption is still rather low and Gartner predicts it will take until 2015 before NFC payments become mainstream. This is due to the fact that NFC payments require changes in business case and better collaboration between banks, mobile carriers, mobile device manufacturers, card networks and merchants.

Not only require NFC payment schemes a change in behavior by the stakeholders, they are also more difficult to secure than pure web portal based solutions. Therefore, we have selected to different approaches to mobile payments applications and developed them in two proof of concept demonstrators.

7.1.1 Related work

Mobile payment systems and wallets are rapidly growing and new initiatives are being placed in the market every week. We will briefly discuss the most prominent players that are active today:

Google Wallet The Google Wallet App for Android phones is a Cloud based mobile credit card solution. When the user registers for this service, he can connect one or more real credit cards to his Google Wallet account. The card details are stored on the servers at a Google data center and not on the phone. Instead, a virtual prepaid Mastercard issued by Bancorp is stored on the embedded SE inside the Android phone. When you pay in-store by tapping your phone, Google Wallet passes the virtual card to the merchant for payment, and charges your selected credit or debit card for the purchase.

Mastercard Paypass MasterCard PayPass is an EMV compatible, contactless payment feature based on the ISO/IEC 14443 standard that provides cardholders with a simpler way to pay by tapping a payment card or other payment device, such as a phone or key fob, on a point-of-sale terminal reader rather than swiping or inserting a card.[1] Currenlty, Mastercard is in

[1] Citation from Wikepedia.

the process of certifying NFC mobile devices for approval to be used for mobile payments. Approved devices will be allowed to carry the *MasterCard PayPass Ready* identifier. The certification entails the mobile device, the secure element and the wallet application.

As far as we know, Google Wallet is actually using a Mastercard Paypass account.

Online Wallets There are many players in the field of Online Wallets. In these systems, the user's account is stored in an online service. The user needs to "charge" his account with virtual money. This charging process depends on the actual wallet application (using debit/credit card transactions, through the mobile operator, etc.). In order to pay, the user just needs to identify and or authenticate himself to the mobile service and the amount is debited from his account. This identification or authentication can take many forms. Popular examples include sending an SMS (identification through phone number), using an RFID tag (identification through serial number of tag), or using a mobile App (authentication through username/pasword or more advanced techniques).

7.2 Application Description

7.2.1 Actors

The two mobile payment applications described here involve 3 different actors the User, the Merchant, and the Bank. Both the User and the Merchant have a "Wallet" (in the case of the User this is stored on his mobile phone) and an account at the Bank.

These actors interact with each other using 3 protocols:

Withdraw. The User uses this protocol to Withdraw a certain amount from his account in the Bank and transfer it to his Wallet on the mobile phone.

Spend. The User pays the merchant. This transaction is "stored" in both the User's and Merchant's Wallet.

Deposit. The Merchant presents his Wallet to the Bank and deposits the money present in the wallet to his account.

7.2.2 Requirements

7.2.2.1 Security Requirements

S_1 Payments only happen with the user's consent.

S_2 Private credentials used in the payment protocol are stored and used in such a way that unauthorized access is impossible, even if the phone is stolen.

S_3 It is possible to revoke payment credentials in the case the smart phone is lost or stolen.

S_4 It is impossible to forge or insert new "digital cash" in the system. This means that digital cash, coins, stored value, vouchers, etc. can only enter the system through a valid run of the Withdraw protocol. Similarly, the Deposit protocol ensures that only valid digital coins are accepted by the bank.

7.2.2.2 Privacy Requirements

P_1 The payment system should not release personal user data unless strictly necessary for that particular transaction.

7.2.2.3 Functional Requirements

F_1 Transactions are atomic.

F_2 The system can be realized using commodity hardware.

F_3 The payment scheme allows offline transactions between the merchant and the user, i.e. no online connection to the bank or any other third party is required during the payment phase.

7.2.3 Emulation of contactless smartcards with stored value

The first mobile payment application we consider is emulation of a contactless smartcard with stored value. In this setting, the user's smartphone presents itself as a contactless smartcard to the merchants payment terminal.

7.2.3.1 User Wallet

The user has a smartcard embedded in his mobile phone. This smartcard contains a stored value counter field that can be incremented or decremented with a specific value. The smartcard enforces certain access rules on writing and or consulting the stored value. The access controls and authentication mechanism depending on specific capabilities of the smartcard that is used. This will be explained more in detail in Sect. 7.3.1.1.

The User interacts with the smartcard through a mobile App that is running on the smartphone. This App allows the user to consult the stored value and to approve or deny charging the stored value from an account (Withraw money) or Spending money towards the merchant.

There are three types of smartcards that can be embedded in a mobile device. We will refer to embedded smartcards as Secure Elements (SE). In order to add Applets (functionality) to these smartcards, the issuer of the Applet needs to authenticate himself to the SE with valid authentication credentials. The party controlling these credentials depends on the type of SE.

1. Embedded by the OEM in the mobile device itself. In this case, the SE is integrated within the hardware of the mobile device. The OEM (or a designated partner) controls the access rights to manage Applets on the SE. The Google Wallet (controlled by Google) is an example of this type of SE.

2. Embedded in the SIM. Here, the SE is part of the SIM. As the SIM card is owned by your service provider, it is also this service provider that has control over the SE.

3. Finally there are also SE's embedded in SD memory cards. In this case, the SE can be purchased by anybody and the owner of the SD card has control over the SE. Obviously, any third party that is has build a business case on top of this type of SE is free to replace the credentials in order to keep control over the SE after selling it to the end customer.

As only the third option allows easy and independent research on SE Applet development, our payment application is running on an SE embedded in a SD memory card that is plugged in the micro SD slot of the phone.

7.2.3.2 Merchant Wallet

Then Merchant's payment terminal should at least be able to perform the following actions:

1. Execute a debit transaction on the User's SE. For this, the terminal has to authenticate to the SE using the necessary credentials. These credentials can either be stored locally in the terminal or the terminal can act as a "bridge" or proxy between the SE and an online payment service.

2. Generate a tamper proof log of the executed transactions. This log can be used by the Merchant to claim money from the Bank.

7.2.3.3 Payment Protocols

Withdraw The Withdraw protocol is executed between the User and the Bank. This protocol *credits* the digital counter value that is stored within the SE and removes the equivalent amount from the User's bank account. The identification of the user is usually carried out in some out of band mechanism, unrelated to the SE. For example, a user can use his debit or credit card to authenticate to the Bank. In order to carry out the *credit* transaction, the Bank requires access to the *credit credentials* related to this SE.

Spend The Spend protocol is executed by the User and the Merchant. Using the necessary *debit credentials*, the Merchant decreases the counter value on the SE with the desired amount. The Merchant's terminal or online service should generate a tamper proof log of this *debit* transaction. An example of how this works in practice with DESFire cards is shown in Sect. 7.3.1.1.

Deposit The Merchant shows his transaction log and obtains the monetary value of all transactions on his account.

7.2.4 Mobile anonymous e-cash system

The second mobile payment application we have developed within the MobCom project is an anonymous e-cash system. In order to fulfill the security and privacy requirements specified above, an e-cash system should at least have the following properties:

- Forgery of e-coins is hard,
- duplication should be either prevented or detected,
- multiple runs of the Spend protocol should be unlinkable (anonymity).

As described in Section 7.1.1, many different e-cash systems have been proposed in the literature. Some of them are focussed on efficiency, while others try to achieve more advanced properties such as the possibility to transfer money from one user to the other, while still others aim at maximizing the privacy protection of the users. For the anonymous e-cash system in the MobCom project we were particularly interested in a scheme that allows to share the necessary computations between the smartcard in the phone and the cpu of the mobile phone itself. Without this load sharing it is currently not possible to implement the demanding cryptographic operations, as the smart card does not have sufficient resources (RAM and cpu).

7.2.4.1 User Wallet

The user has a smartcard embedded in his mobile phone. This smartcard contains the user's private key that is used to authenticate to the Bank during the Withdraw protocol. Not only is the private key stored in the smartcard, all computations in which this private key is involved are also carried out on the smartcard. This way, the private key never has to leave its secure environment.

The User interacts with the smartcard through a mobile App that is running on the smartphone. This App allows the user to execute the Withdraw and Spend protocols.

7.2.4.2 Merchant Wallet and Bank implementation

Both the Merchant and Bank also require credentials that they need to store securely in order to avoid unauthorized access. The actual security mechanisms used in order to secure the confidentiality, integrity and potential recovery of these credentials is not within the scope of this work.

For a Bank it is probably practical to store the required credentials in some network-enabled HSM (Hardware Security Module). For the Merchant the solution of choice depends on the setup

of the sales channel and the scale (e.g., a large distributed online store might also invest in an HSM, while a small local merchant might use a similar setup as the User).

7.2.4.3 Payment Protocols

In the e-cash system selected for this application [18], all three parties in the system have a public/private key pair $(sk_{\mathcal{P}}, pk_{\mathcal{P}})$ where $\mathcal{P}$ can be the User ($\mathcal{U}$), the Merchant ($\mathcal{M}$) or the Bank ($\mathcal{B}$). The private key is only known to the party that owns it, a certified version of the public keys is known by all parties. To explain the e-cash system without going into all the mathematical details, we will show how a single-use electronic cash system could be designed.

Withdraw The Withdraw protocol (see Fig. 7.1) is executed between the User and the Bank. By means of this protocol, users are able to convert money from their account into e-cash coins. The user is only able to perform the Withdraw protocol by using his private key that is verified by the bank against his registered public key. At the end of a successful protocol run, the user is in possession of a set of e-cash coins and the bank has lowered the account with the monetary value of these coins. A coin consists of the digital signature by the Bank on the set of values $(sk_{\mathcal{U}}, s, t)$. Here s is a random serial number of the coin and t is a value used to blind the identity of the user during the Spend protocol.

Spend The Spend protocol (see Fig. 7.2) is executed by the User and the Merchant. In order to protect the anonymity of the user (i.e., the e-cash coins themselves should not reveal the identity, for example the public or private key, of the User), the user does not provide the actual coins to the Merchant, but a cryptographic commitment C on the coin, together with a cryptographic zero-knowledge proof that the coin has been signed by the Bank. In response to this "blinded" coin, the Merchant returns a random value R. Finally, the User reveals the serial number s and the value $T = sk_{\mathcal{U}} + Rt$. The user also produces a second zero-knowledge proof that the values s and T correspond to the values inside the earlier commitment C.

Deposit The Deposit protocol (see Fig. 7.3) is executed between the Merchant and the Bank. The Merchant submits the values (s, R, T, π_1, π_2), where $\pi_{1,2}$ are the two zero-knowledge proofs provided by the User during the Spend protocol to the Bank. The Bank stores all the submitted coins in a database and verifies the zero-knowledge proofs.

After the Deposit protocol, the Bank will check for double spending. If the database already contains the same coins (all five values are the same), than the Merchant is accused of double spending. If the Bank sees that its database already contains a coin with serial number s, but with different values for the other elements of the coin, then the user is accused of double spending. By combining the different values of s, R and T, the Bank is able to solve the set of equations $T_1 = sk_{\mathcal{U}} + R_1 t$ and $T_2 = sk_{\mathcal{U}} + R_2 t$ for the values t and $sk_{\mathcal{U}}$ and identify the cheating user by its private key $sk_{\mathcal{U}}$.

7.3 Validation

7.3.1 Prototype Description

7.3.1.1 Emulation of contactless smartcards with stored value

The prototype implementation was primarily focussed on the User's SE in combination with a mobile phone running the Android 4.0 operating system. The User's implementation consists of two components. First, the DESFire [62] specifications have been used to write a complete DESFire emulator in software that is running inside the SE in the phone. The SE itself is a Giesecke & Devrient Mobile Security Card (MSC) SE 1.0 [33] that is running the Java Card Operating System Sm@rtCafé Expert 5.0. The DESFire emulator is written as a Java Applet for Java Card and is installed on the User's MSC. The Second part is an Android App of which the

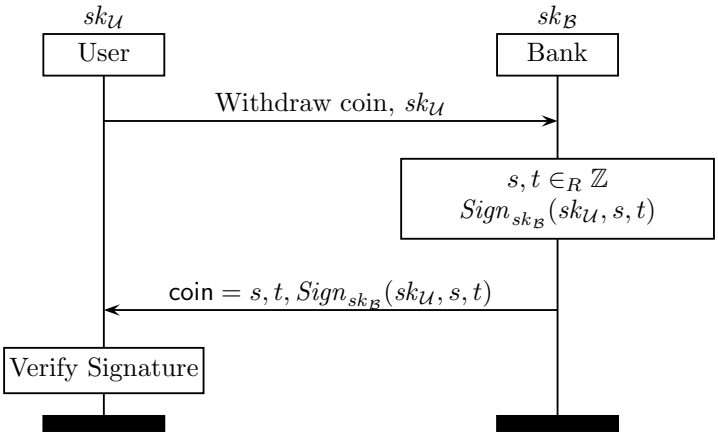


Figure 7.1: Single-use e-cash Withdraw protocol.

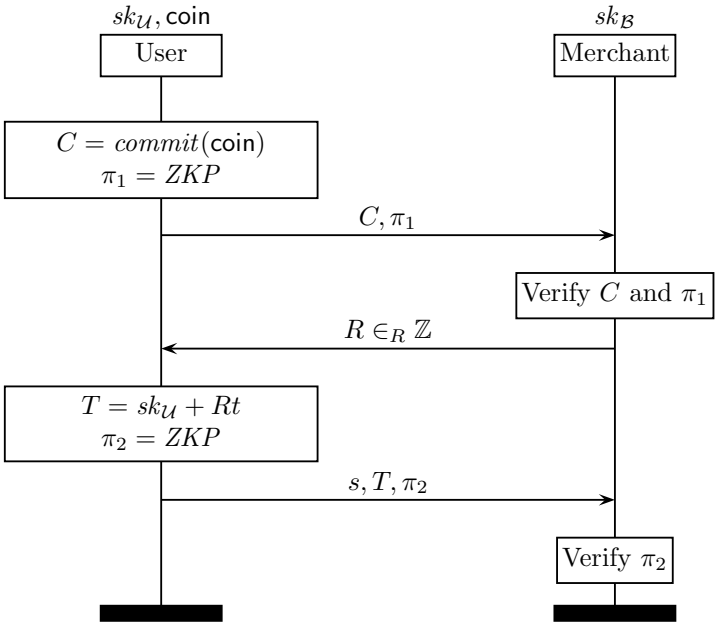


Figure 7.2: Single-use e-cash Spend protocol. C is a cryptographic commitment to the values in the coin, while π_1 is a zero-knowledge proof of the fact that the coin "inside" the commitment is signed by the Bank. Finally, π_2 is a zero-knowledge proof of the fact that s and T correspond to the values earlier committed to in C.

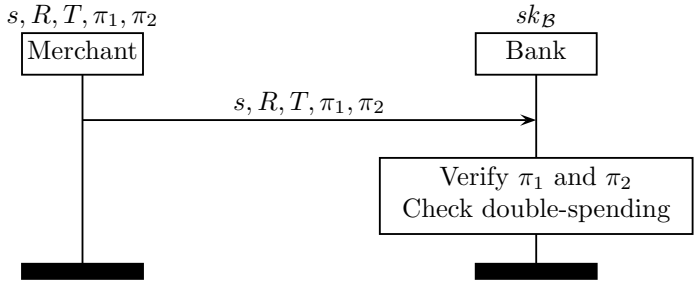


Figure 7.3: Single-use e-cash Deposit protocol.

primary role is to channel messages (APDU's) between the the Android NFC system and the MSC that is residing in the phone's SD memory slot. As a secondary role, the Android App also offers some functionality to the User: approve transactions and consult transaction logs.

The security of DESFire transactions is based on a shared symmetric key. Figure 7.4 shows how this key is used to provide mutual authentication between the Terminal and the Tag according to the DESFire standard. Once this authentication is completed successfully, commands can be given from the Terminal to the Tag. This commands can be either in the clear or encrypted/authenticated with the shared key k_i.

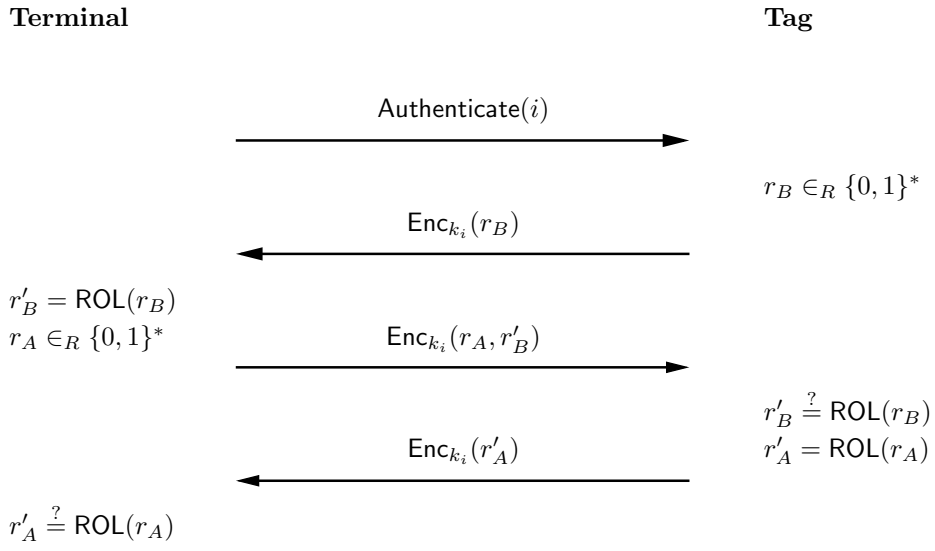


Figure 7.4: DESFire implementation of the Spend protocol in the stored value scenario. "ROL" is the bitwise rotate-left operation, while k_i is a shared cryptographic key between the two parties.

The Android App in combination with the DESFire emulator running in an NFC enabled Android phone is capable of emulating a DESFire card towards any reader, without requiring any changes to the reader hardware or software. Unfortunately, this requires the NFC system of the phone to run in "Card Emulation Mode", which is not supported by stock Android versions. For prototyping purposes, this has been resolved by installing a tweaked version of Android (CyanogenMod 9.0) on the phone.

The Merchant and Bank prototypes are both implemented in Java SE and can run in any environment that support Java and the necessary drivers for the USB NFC reader. The Bank is able to carry out Credit transactions, while the Merchant is able to carry out debit transactions.

Table 7.1: Response times of the e-cash implementation

Protocol	Host (s)	MSC (s)	total
Withdraw	1.020	17.804	18.824
Spend	0.148	23.340	23.488

7.3.1.2 Mobile anonymous e-cash system

Again, the prototype implementation effort was primarily targeted at the User side. Again, the G&D MSC SE 1.0 was used in combination with an Android mobile phone. The e-cash payment system was chosen for prototyping in order to investigate how fast an actual implementation would run within the limited resources of the Java Card Operating System in combination with the more powerful capabilities of an Android mobile phone.

For this reason, the Compact e-cash protocol developed by Camenish et al [18] was adapted to split the load of the demanding computations between the cpu within the MSC and the CPU in the mobile phone (the host). During the redesign of the protocol, special care was taken to ensure that no information about the secret key of the User leaks from the MSC to the mobile phone.

7.3.1.3 Evaluation

Although the calculations were offloaded to the host, a large part of the calculations still have to be carried out by the MSC. Because of the low CPU speed, EEPROM and RAM restrictions (1600 kB and 75MB respectively) and the very restricted Java RE on the MSC, the implementation on the MSC was not trivial and is still very time consuming. Table 7.3.1.3 shows the response times of the host and MSC for both the Withdraw and Spend protocols. This table clearly shows that the response times are unacceptable for practical use in real life applications. The table also shows that it is mostly the MSC that is just to slows to perform the demanding computations. But even the host performance for the Withdraw protocol is borderline (> 1 second).

Although there probably still are some software optimizations possible, an acceptable implementation would need to be 20 times faster. We don't think this is possible with software improvements only and can conclude that a practical implementation of e-cash systems requires dedicated hardware co-processors to carry out the calculations.

Chapter 8

Conclusions

We conclude the first 24 months of the project with a report on our progress creating the privacy-preserving identity framework and the domain-specific application prototypes that were layed out in the requirements document. The applications described here serve as validation of the requirements study and the basic research that has been performed so far. They will also be presented to the usergroup as a summary of the current state of the project, allowing us to gather additional input for the direction and focus in the months and years to come.

Bibliography

[1] adapID Project. advanced applications for electronic IDentity cards in Flanders. `http://www.cosic.esat.kuleuven.be/adapid/`, 2009.

[2] Gildas Avoine, Muhammed Ali Bingöl, Süleyman Kardaş, Cédric Lauradoux, and Benjamin Martin. A framework for analyzing RFID distance bounding protocols. *Journal of Computer Security*, 19(2):289–317, April 2011.

[3] Gildas Avoine and Aslan Tchamkerten. An efficient distance bounding RFID authentication protocol: Balancing false-acceptance rate and memory requirement. In Pierangela Samarati, Moti Yung, Fabio Martinelli, and Claudio Ardagna, editors, *Information Security*, volume 5735 of *Lecture Notes in Computer Science*, pages 250–261. Springer Berlin / Heidelberg, 2009.

[4] Michael Backes, Jan Camenisch, and Dieter Sommer. Anonymous yet accountable access control. In *Proceedings of the 2005 ACM workshop on Privacy in the electronic society*, WPES '05, pages 40–46, New York, NY, USA, 2005. ACM.

[5] Dirk Balfanz, Diana K. Smetters, Paul Stewart, and H. Chi Wong. Talking to strangers: Authentication in ad-hoc wireless networks. In *Proceedings of the Network and Distributed System Security Symposium*. The Internet Society, 2002.

[6] Samy Bengio, Gilles Brassard, Yvo G. Desmedt, Claude Goutier, and Jean-Jacques Quisquater. Secure implementation of identification systems. *Journal of Cryptology*, 4:175–183, 1991.

[7] Patrik Bichsel, Jan Camenisch, Thomas Groß, and Victor Shoup. Anonymous credentials on a standard java card. In *Proceedings of the 16th ACM conference on Computer and communications security*, CCS '09, pages 600–610, New York, NY, USA, 2009. ACM.

[8] Patrik Bichsel, Jan Camenisch, Thomas Groß, and Victor Shoup. Anonymous credentials on a standard java card. In *Proceedings of the 16th ACM conference on Computer and communications security*, CCS '09, pages 600–610, New York, NY, USA, 2009. ACM.

[9] Marina Blanton and William M. P. Hudelson. Biometric-based non-transferable anonymous credentials. In *Proceedings of the 11th international conference on Information and Communications Security*, ICICS'09, pages 165–180, Berlin, Heidelberg, 2009. Springer-Verlag.

[10] Gerrit Bleumer. Biometric yet privacy protecting person authentication. In *In Proceedings of 1998 Information Hiding Workshop (IHW 98*, pages 101–112. Springer-Verlag, 1998.

[11] Joshua Bloch. *Effective Java*. Addison-Wesley, 2 edition, 2008.

[12] Tim Boudreau. The Capability Pattern - Future-Proof Your APIs. `http://weblogs.java.net/blog/timboudreau/archive/2008/08/the_capability.html`, 2008.

[13] Stefan Brands. *Rethinking Public Key Infrastructures and Digital Certificates: Building in Privacy*. MIT Press, Cambridge, MA, USA, 2000.

[14] Stefan Brands and David Chaum. Distance-bounding protocols. In Tor Helleseth, editor, *Advances in Cryptology - EUROCRYPT '93*, volume 765 of *Lecture Notes in Computer Science*, pages 344–359. Springer Berlin / Heidelberg, 1994.

[15] Stefan A. Brands. *Rethinking Public Key Infrastructures and Digital Certificates: Building in Privacy*. MIT Press, Cambridge-London, 1 edition, August 2000.

[16] Franz Ferdinand Brasser, Sven Bugiel, Atanas Filyanov, Ahmad-Reza Sadeghi, and Steffen Schulz. Softer smartcards - usable cryptographic tokens with secure execution. In Angelos D. Keromytis, editor, *Financial Cryptography*, volume 7397 of *Lecture Notes in Computer Science*, pages 329–343. Springer, 2012.

[17] Ernie Brickell, Jan Camenisch, and Liqun Chen. Direct anonymous attestation. In *Proceedings of the 11th ACM conference on Computer and communications security*, CCS '04, pages 132–145, New York, NY, USA, 2004. ACM.

[18] J. Camenisch, S. Hohenberger, and A. Lysyanskaya. Compact e-cash. In *Proceedings of the International Conference on the Theory and Application of Cryptographic Techniques: Advances in Cryptology*, EUROCRYPT '05, pages 302–321. Springer-Verlag, 2005.

[19] Jan Camenisch. Protecting (anonymous) credentials with the trusted computing groups tpm v1.2. In Simone Fischer-Hbner, Kai Rannenberg, Louise Yngstrm, and Stefan Lindskog, editors, *Security and Privacy in Dynamic Environments*, volume 201 of *IFIP International Federation for Information Processing*, pages 135–147. Springer US, 2006.

[20] Jan Camenisch and Ivan Damgård. Verifiable encryption, group encryption, and their applications to separable group signatures and signature sharing schemes. In Tatsuaki Okamoto, editor, *Advances in Cryptology - ASIACRYPT 2000*, volume 1976 of *Lecture Notes in Computer Science*, pages 331–345. Springer Berlin / Heidelberg, 2000.

[21] Jan Camenisch and Anna Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In Birgit Pfitzmann, editor, *Advances in Cryptology - EUROCRYPT 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 93–118. Springer Berlin / Heidelberg, 2001.

[22] Jan Camenisch, Sebastian Modersheim, Gregory Neven, Franz-Stefan Preiss, and Dieter Sommer. A card requirements language enabling privacy- preserving access control. In *Proc. of the 15th ACM Symposium on Access Control Models and Technologies (SACMAT)*, pages 119–128, 2010.

[23] Jan Camenisch, Dieter Sommer, and Roger Zimmermann. A general certification framework with applications to privacy-enhancing certificate infrastructures. In Simone Fischer-Hbner, Kai Rannenberg, Louise Yngstrm, and Stefan Lindskog, editors, *Security and Privacy in Dynamic Environments*, volume 201 of *IFIP International Federation for Information Processing*, pages 25–37. Springer Boston, 2006.

[24] Jan Camenisch and Els Van Herreweghen. Design and implementation of the idemix anonymous credential system. In *Proceedings of the 9th ACM conference on Computer and communications security*, CCS '02, pages 21–30, New York, NY, USA, 2002. ACM.

[25] Suresh Chari, Parviz Kermani, Sean Smith, and Ros Tassiulas. Security issues in m-commerce: A usage-based taxonomy. e-commerce agents. In *LNAI*, pages 264–282. Springer, 2001.

[26] David Chaum and Torben P. Pedersen. Wallet databases with observers. In *Proceedings of the 12th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '92, pages 89–105, London, UK, UK, 1993. Springer-Verlag.

[27] Joris Claessens. *Analysis and design of an advanced infrastructure for secure and anonymous electronic payment systems on the Internet*. PhD thesis, Katholieke Universiteit Leuven, 2002. Bart Preneel and Joos Vandewalle (promotors).

[28] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk. RFC 5280 - Internet X.509 public key infrastructure certificate and certificate revocation list (CRL) profile, May 2008.

[29] Intel Corporation. LaGrande technology preliminary architecture specification. Intel Publication no. D52212, May 2006.

[30] Yves Deswarte and Sébastien Gambs. A proposal for a privacy-preserving national identity card. *Trans. Data Privacy*, 3(3):253–276, December 2010.

[31] Yves Deswarte and Sébastien Gambs. Cryptography and security. chapter The challenges raised by the privacy-preserving identity card, pages 383–404. Springer-Verlag, Berlin, Heidelberg, 2012.

[32] Advanced Micro Devices. AMD64 architecture programmer's manual: Volume 2: System programming. AMD Publication no. 24594 rev. 3.11, Dec 2005.

[33] Giesecke & Devrient. Mobile security card se 1.0. `http://www.gd-sfs.com/the-mobile-security-card/mobile-security-card-se-1-0/`.

[34] Kurt Dietrich and Johannes Winter. Implementation aspects of mobile and embedded trusted computing. In Liqun Chen, Chris Mitchell, and Andrew Martin, editors, *Trusted Computing*, volume 5471 of *Lecture Notes in Computer Science*, pages 29–44. Springer Berlin / Heidelberg, 2009.

[35] Yevgeniy Dodis, Leonid Reyzin, and Adam Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology - EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 523–540. Springer Berlin / Heidelberg, 2004.

[36] Yevgeniy Dodis, Leonid Reyzin, and Adam Smith. Fuzzy extractors. In Pim Tuyls, Boris Skoric, and Tom Kevenaar, editors, *Security with Noisy Data*, pages 79–99. Springer London, 2007.

[37] Steve Dohrmann and Carl Ellison. Public-key support for collaborative groups. In *Proceedings of the 1st Annual PKI Research Workshop*, pages 139–148, 2002.

[38] Olivier Dubuisson and Philippe Fouquart. *ASN.1: communication between heterogeneous systems*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2001.

[39] Peter Eckersley. How unique is your web browser? In Mikhail Atallah and Nicholas Hopper, editors, *Privacy Enhancing Technologies*, volume 6205 of *Lecture Notes in Computer Science*, pages 1–18. Springer Berlin / Heidelberg, 2010.

[40] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. RFC 2616, hypertext transfer protocol – HTTP/1.1, 1999.

[41] Eimear Gallery and Chris Mitchell. Trusted mobile platforms. In *Foundations of security analysis and design IV*, pages 282–323. Springer-Verlag, 2007.

[42] Tal Garfinkel, Mendel Rosenblum, and Dan Boneh. Flexible OS support and applications for trusted computing. In *Proceedings of the 9th conference on Hot Topics in Operating Systems-Volume 9*, pages 25–25. USENIX Association, 2003.

[43] Gartner. Forecast: Mobile payment, worldwide, 2009-2016. `http://www.gartner.com/DisplayDocument?ref=clientFriendlyUrl&id=2010515`, 2012.

[44] Christian Gehrmann and Kaisa Nyberg. Manual authentication for wireless devices. *RSA Cryptobytes*, 7:2004, 2004.

[45] Michael Goodrich, Michael Sirivianos, John Solis, Gene Tsudik, and Ersin Uzun. Loud and clear: Human-verifiable authentication based on audio. In *26th IEEE International Conference on Distributed Computing Systems, 2006. ICDCS 2006*, pages 10–10. IEEE, 2006.

[46] Trusted Computing Group. TCG TPM specification. `http://www.trustedcomputinggroup.org/resources/tpm_main_specification`.

[47] Gerhard Hancke and Markus Kuhn. An RFID distance bounding protocol. In *First International Conference on Security and Privacy for Emerging Areas in Communications Networks, 2005. SecureComm 2005*, pages 67 – 73, September 2005.

[48] Feng Hao, Ross Anderson, and John Daugman. Combining crypto with biometrics effectively. *IEEE Trans. Comput.*, 55(9):1081–1088, September 2006.

[49] Specification of the Identity Mixer cryptographic library – version 2.3.2, 2010. IBM Research – Zurich.

[50] Specification of the Identity Mixer cryptographic library – version 2.3.2, 2010. IBM Research – Zurich.

[51] Russell Impagliazzo and Sara Miner More. Anonymous credentials with biometrically-enforced non-transferability. In *Proceedings of the 2003 ACM workshop on Privacy in the electronic society*, WPES '03, pages 60–71, New York, NY, USA, 2003. ACM.

[52] Anil K. Jain, Patrick Flynn, and Arun A. Ross. *Handbook of Biometrics*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.

[53] Anil K. Jain, Karthik Nandakumar, and Abhishek Nagar. Biometric template security. *EURASIP J. Adv. Signal Process*, 2008:113:1–113:17, January 2008.

[54] Wayne Jansen, Serban Gavrila, and Vlad Korolev. Proximity-based authentication for mobile devices. In Hamid R. Arabnia, editor, *Security and Management*, pages 398–404. CSREA Press, 2005.

[55] Chong Kim, Gildas Avoine, Franois Koeune, Franois-Xavier Standaert, and Olivier Pereira. The Swiss-Knife RFID distance bounding protocol. In Pil Lee and Jung Cheon, editors, *Information Security and Cryptology - ICISC 2008*, volume 5461 of *Lecture Notes in Computer Science*, pages 98–115. Springer Berlin / Heidelberg, 2009.

[56] Jorn Lapon. *Anonymous Credential Systems: From Theory Towards Practice (Anonieme credential systemen: van de theorie naar de praktijk)*. PhD thesis, July 2012.

[57] Kuan-Chieh Liao, Wei-Hsun Lee, Min-Hsuan Sung, and Ting-Ching Lin. A one-time password scheme with QR-code based on mobile phone. In *Fifth International Joint Conference on INC, IMS and IDC, 2009. NCM '09.*, pages 2069 –2071, August 2009.

[58] Rene Mayrhofer and Hans Gellersen. Shake well before use: Authentication based on accelerometer data. In Anthony LaMarca, Marc Langheinrich, and Khai Truong, editors, *Pervasive Computing*, volume 4480 of *Lecture Notes in Computer Science*, pages 144–161. Springer Berlin / Heidelberg, 2007.

[59] Jonathan McCune, Adrian Perrig, and Michael Reiter. Seeing-Is-Believing: using camera phones for human-verifiable authentication. *International Journal of Security and Networks*, 4(1):43–56, 2009.

[60] Jonathan M. McCune, Bryan J. Parno, Adrian Perrig, Michael K. Reiter, and Hiroshi Isozaki. Flicker: an execution infrastructure for tcb minimization. *SIGOPS Oper. Syst. Rev.*, 42(4):315–328, April 2008.

[61] Wojciech Mostowski and Pim Vullers. Efficient U-Prove implementation for anonymous credentials on smart cards. In Muttukrishnon Rajarajan, Fred Piper, Haining Wang, and George Kesidis, editors, *7th International ICST Conference on Security and Privacy in Communication Networks, SecureComm 2011, London, UK, September 7-9, 2011. Proceedings*, volume 96 of *Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Tele-communications Engineering (LNICST)*, pages 243–260. Springer-Verlag, August 2012.

[62] NXP. Mifare desfire. `http://www.nxp.com/products/identification_and_security/smart_card_ics/mifare_smart_card_ics/mifare_desfire/`.

[63] Adrian Perrig and Dawn Song. Hash visualization: a new technique to improve real-world security. In *International Workshop on Cryptographic Techniques and E-Commerce*, pages 131–138, 1999.

[64] Michael G. Reed, Paul F. Syverson, and David M. Goldschlag. Anonymous connections and onion routing. *IEEE Journal on Selected Areas in Communications*, 16:482–494, 1998.

[65] Michael K. Reiter and Aviel D. Rubin. Crowds: anonymity for web transactions. *ACM Trans. Inf. Syst. Secur.*, 1(1):66–92, 1998.

[66] Andreas Schmidt, Nicolai Kuntze, and Michael Kasper. On the deployment of mobile trusted modules. In *Wireless Communications and Networking Conference, 2008. WCNC 2008. IEEE*, pages 3169–3174. IEEE, 2008.

[67] Dave Singele and Bart Preneel. Distance bounding in noisy environments. In Frank Stajano, Catherine Meadows, Srdjan Capkun, and Tyler Moore, editors, *Security and Privacy in Ad-hoc and Sensor Networks*, volume 4572 of *Lecture Notes in Computer Science*, pages 101–115. Springer Berlin / Heidelberg, 2007.

[68] Claudio Soriente, Gene Tsudik, and Ersin Uzun. HAPADEP: Human-assisted pure audio device pairing. In Tzong-Chen Wu, Chin-Laung Lei, Vincent Rijmen, and Der-Tsai Lee, editors, *Information Security*, volume 5222 of *Lecture Notes in Computer Science*, pages 385–400. Springer Berlin / Heidelberg, 2008.

[69] Frank Stajano and Ross Anderson. The resurrecting duckling: Security issues for ad-hoc wireless networks. In Bruce Christianson, Bruno Crispo, James Malcolm, and Michael Roe, editors, *Security Protocols*, volume 1796 of *Lecture Notes in Computer Science*, pages 172–182. Springer Berlin / Heidelberg, 2000.

[70] Guenther Starnberger, Lorenz Froihofer, and Karl Goeschka. QR-TAN: Secure mobile transaction authentication. In *International Conference on Availability, Reliability and Security, 2009. ARES '09*, pages 578 –583, March 2009.

[71] Christian Paquin Stefan Brands. U-Prove cryptographic specification v1.0, 2010. Microsoft Corporation.

[72] Michaël Sterckx, Benedikt Gierlichs, Bart Preneel, and Ingrid Verbauwhede. Efficient implementation of anonymous credentials on java card smart cards. In *First IEEE International Workshop on Information Forensics and Security, 2009. WIFS 2009.*, pages 106–110, December 2009.

[73] Alex Varshavsky, Adin Scannell, Anthony LaMarca, and Eyal De Lara. Amigo: proximity-based authentication of mobile devices. In *Proceedings of the 9th international conference on Ubiquitous computing*, UbiComp '07, pages 253–270, Berlin, Heidelberg, 2007. Springer-Verlag.