Advanced Applications for e-ID Cards in Flanders

ADAPID Deliverable D9A

Framework I

C. Diaz, Bart De Decker, Steven Gevers, Mohamed Layouni, C. Troncoso, H. Van Es, K. Verslype (Ed.).

September 2007
# Contents

1 Intro ........................................ 4  
1.1 Background .................................. 4  
1.2 Problem Domain ............................. 4  
1.3 Challenges ................................. 4  
1.4 Content of this deliverable ................. 5  

2 Applications ................................ 7  
2.1 E-health .................................... 7  
2.1.1 Description ............................... 7  
2.1.2 Functionality ............................. 7  
2.2 E-government ............................... 8  
2.2.1 Description ............................... 8  
2.2.2 Required functionality ................... 8  
2.3 Trusted Archival Service .................... 8  

3 Building Blocks ............................. 9  
3.1 Commitments ................................ 9  
3.2 Verifiable Encryptions ...................... 9  
3.3 Group Signatures ........................... 9  
3.4 Anonymus Credentials ...................... 10  
3.5 Mix networks ................................ 10  
3.6 Belgian eID card ........................... 11  

4 Framework General Definition and Principles 12  
4.1 Definition .................................. 12  
4.2 General Framework Requirements ........... 12  
4.3 Providers ................................... 14  
4.4 Sensitive Data Representation ............... 15  
4.5 Patterns .................................... 15  

5 The Adapid Framework ....................... 16  
5.1 Requirements ............................... 16  
5.2 Approach ................................... 16  
5.3 Components ................................ 17  

6 Interfaces .................................... 20  
6.1 Communication ............................. 20  
6.1.1 Description ............................... 20  
6.1.2 Functionality ............................. 20  
6.1.3 Interface ................................ 20  
6.1.4 Usage Examples ......................... 21  
6.2 Credential Handler .......................... 23  
6.2.1 Description ............................... 23  
6.2.2 Functionality ............................. 23  
6.2.3 Interface ................................. 24  
6.2.4 Interface Usage Example ................. 24  
6.3 Trusted Archival Service ................... 24
6.3.1 Description ................................................. 24
6.3.2 Functionality .............................................. 26

7 Implementation .............................................. 28
   7.1 Communication ............................................ 28
   7.2 Credential Handler ....................................... 29
   7.3 Trusted Archival .......................................... 29

8 Existing Identity Management Systems ...................... 31
   8.1 Microsoft Cardspace ..................................... 31
      8.1.1 General introduction ................................ 31
      8.1.2 Detailed description .................................. 31
      8.1.3 Interaction ............................................ 32
      8.1.4 Identity Providers concept ............................ 34
      8.1.5 Requirements ......................................... 34
      8.1.6 Technology Used & Components ...................... 35
      8.1.7 Evaluation ............................................ 35
   8.2 Higgins Project .......................................... 36
      8.2.1 Introduction ......................................... 36
      8.2.2 Requirements ......................................... 38
      8.2.3 Components .......................................... 38
      8.2.4 Evaluation ............................................ 38
   8.3 Liberty Alliance Project ................................. 39
      8.3.1 The Liberty Identity Federation Framework: ID-FF .... 40
      8.3.2 Liberty Identity Web Services Framework: ID-WSF .... 43
      8.3.3 Liberty Identity Services Interface Specifications: ID-SIS 44
      8.3.4 Evaluation of Liberty Alliance ....................... 44

9 Conclusions and Future Work ............................... 46
1 Intro

1.1 Background

The ADAPID project is a project of the Flemisch government (IWT-Vlaanderen) aimed at:

• developing a framework for secure and privacy-preserving applications based on the Belgian eID card, focussing mainly on e-government, e-health, and trusted archiving applications, and taking into account both technical and legal aspects; and

• investigating technologies for future enhanced generations of the eID card.

As an important part in the project, a framework has to be developed that is aimed at facilitating the development of privacy-supporting applications, based on the Belgian eID card. In a first phase, planned to be finished by september 2007, a first embryonic prototype will be provided. Based on the validation of the first phase, a more elaborate framework will be developed and delivered by september 2009.

1.2 Problem Domain

A credential is a piece of information attesting to the integrity of certain stated facts: properties about or rights of its owner. Because the credential is certified by some issuer, the facts are believed to be true if the issuer is trusted. For instance, a driving license is issued by the government administration and proves to other parties trusting that administration that the owner is able to drive a car. An identity card is a credential as well: the government guarantees properties of the user such as date of birth and nationality. As another example, a banknote gives the user the right to spend a certain amount of money. It is clear that a modern society cannot function well without credentials. Moreover, the importance of credentials is steadily increasing.

More and more, digital credentials are introduced into our lives: loyalty cards, e-cash, e-tickets, etc. Electronic identity (eID) cards are typically smart cards containing one or more digital credentials. A growing number of countries is switching towards eID cards. The evolution towards digital credentials will continue. We can expect that traditional membership cards of sports clubs, political parties, libraries and other subscription based membership cards will be replaced by digital credentials. The classical concept of single-use tickets will be digitalized as well. Imagine buying online your tickets for a rock concert by getting issued a digital credential enabling you to enter the concert. Also train tickets, cinema tickets and other ticketing scenarios are subject to this change.

1.3 Challenges

The use of digital credentials poses serious risks w.r.t. identity theft, which occurred less frequently in the era before the digital credential.

In combination with current sensor and data processing techniques, companies are now able to harvest a huge amount of data about the user’s behaviour
and are able to link this to the user’s identity. The storage of the resulting user’s profile is done on a computer. Even if the company’s policy and behaviour is 100% in accordance with legislation, it is still susceptible to theft of this data (by an external hacker or internal employee). This can cause harm to the user and can seriously damage the company in terms of reputation, profit, etc. Also, the user will not always want that the profile is linkable with his identity.

The user’s identity can as well be stolen in a more direct way by obtaining the user’s credentials (e.g. credit card), for instance by hacking his computer, by eavesdropping, by stealing a smart card containing a credential, etc.

Indeed, identity theft is becoming a serious issue in nowadays economy. In the USA for example, although a decrease in the number of identity thefts happened from 2003 to 2006, the average fraud per person rose from $5,249 to $6,383[^1], resulting in an increase of identity fraud to US $56.6 billion. The Identity Theft Resource Center [1] found out that the average time spent by victims of identity theft resolving the problem is about 40 hours and that the emotional impact is similar to that of victims of violent crimes. The threat for theft of user’s profiles in company databases results in extra costs for the companies in terms of preventive measures (education of responsibles, security management, etc.). Banking experts think electronic fraud will slow the growth of e-commerce.

Another problem related to identity theft is the management of all the credentials (i.e. the user’s identity) by a user. A user will have to manage dozens or hundreds of credentials in the future: a credential for his sporting club, a digital driving license, e-tickets for the cinema, etc. If no management tool is available, this will result in leakage of personal data as the user will not be able to control all his/her credentials in a thorough manner; the user will not immediately detect loss of a credential, does not have an overview of what personal data has been released to whom. Different types of credentials make their management and use more difficult as support to interpret the content and use the credentials is required.

1.4 Content of this deliverable

This deliverable is part of a work package within the Adapid project. The work package aims to deliver a proposal for a software framework that allows application developers to easily create privacy preserving applications. The framework should counter the challenges summed up in the previous section. This deliverable is a first step in the development of this framework.

The framework prototype will be developed in two phases. In a first phase, a prototype with limited functionality is implemented. A critical evaluation of that framework will be made. This evaluation is also based on input from the application development (D5, D6, D7). The shortcomings and comments will be covered as much as possible in the second phase, where a more enhanced implementation of the framework will be built. In the first phase, support for the Belgian eID card is indispensable. In the second phase, support for the future eID card can be provided as well.

In the first phase, Idemix Anonymous credentials are supported, as well as the Belgian eID card. For the latter, two implementations are provided: the Godot and the PKnet (Intesi) one. However, in the first prototype, not the full
power of Idemix can be offered. In the enhanced prototype, this should become possible. Other credential system implementations should be pluggable in the framework as well (e.g. the U-Prove anonymous credential system developed by Brands, or the Swedish eID card).

In the next section, the implementation of use cases in different application domains provides us with input about the required functionality by the framework. Section 3 gives an overview of the building blocks that will need to be supported by the framework. Section 4 presents the main framework design principles. Section 5 gives an overview of the design of the Adapid framework in its current state. Section 6 zooms in on the different component interfaces whereof a first iteration has been completed, while section 7 zooms in on the current prototype implementations already present. Section 8 gives a description and analysis of existing identity management systems. We conclude and outline future work in section 9.
2 Applications

As part of the ADAPID project, different use cases in different application domains where examined (cfr. D5: Electronic Archival, D6: e-Health and D7: e-Government) and partial prototypes where implemented. This provides us with input on what functionality should be provided by the framework for these applications.

2.1 E-health

2.1.1 Description

The e-Health deliverable focuses on how a prescription is issued by the doctor, used by the patient and handled by different entities (pharmacist, Medical Prescription Administration, Health Insurance Institute, etc.). The scenario is kept very close to current practice in Belgian health care. Liesje Demuyck examined how the patient’s health records can be centralized and at the same time protect the patient’s privacy (see [16]).

2.1.2 Functionality

The e-health applications needs the following functionality:

- Anonymous authentication by proving properties.
- Authentication by identification
- Functionality provided by non-interactive verifiable encryptions; i.e. asymmetric encryption and decryption where certain properties of the resulting cipher can be verified by everyone who knows the corresponding public key.
- A subset of the functionality provided by anonymous credential systems: proving in zero knowledge properties of credentials in an unlinkable way. Also, commitments and verifiable encryptions need to be supported. It must as well be possible to prove properties involving two different credentials.
- The functionality provided by traditional digital signatures: protecting the integrity and authentication of data.
- Anonymization of statistical data according to a policy.
- Communication providing anonymity, as well as identification. Integrity and confidentiality are required as well.
- Deanonymization in case of misbehaviour.
- Confidentiality of data objects.

The eID card can be used to:

- obtain access to the patient’s credentials (prescriptions, social security credential, etc.)
• obtain a new credential from a credential issuing party, by identifying first to that party.
• store credentials, and, in the long term, manage and use these credentials.
• authentication by identification towards another party.

2.2 E-government

2.2.1 Description

The e-petition demo is an e-government web application which lets citizens with a valid Belgian eID card sign petitions. The demo is implemented in a way that addresses security, privacy and other requirements of an ideal e-petition application. A single e-petition application cycle needs the involvement of two parties: the citizen and the e-petition web server. The e-petition web server publishes the available e-petitions and verifies the validity of citizens’ identity before accepting a citizen’s signature. When the citizen proves to be the owner of a valid eID, the web server lets the citizen sign the petition in a privacy preserving and secure manner.

The basic mechanism to achieve this level of privacy and security is as follows. First, the user obtains an anonymous credential from a credential issuer by authenticating using the Belgian e-ID. This anonymous credential, which is signed by the credential issuer, certifies that the user has a valid e-ID card. Later on, the user can present this credential to the petition server to sign petitions. The cryptographic protocols underlying the credential issuing and credential show ensure that double signing of a petition is detectable, and that credential shows cannot be linked to each other or to the credential issuing procedure.

2.2.2 Required functionality

The functionality that this application requires from the framework is the following: (1) e-ID PKI-based authentication; (2) credential issuing; and (3) credential show functionalities. Due to the specific requirements of petition applications (the user must be able to sign as many petitions as she wishes, but each petition can only be signed once), some special options need to be enabled in the credential issuing and show protocols. In particular, the name of the petition needs to be encoded in the credential show protocol, such that two signatures of the same petition result in the same transcript.

2.3 Trusted Archival Service

Although trusted archival is considered as an independent application in the Adapid project, we consider it in this document as a component that will be used by other applications or software components. In the Adapid framework for instance, the trusted archival can be used to store evidence of transactions for a long time. This storing of evidence is just a part of what can be expected from the framework. Therefore, Trusted Archival is described as a separate component in the framework.
3 Building Blocks

This section sums up the different building blocks that can be used to provide the functionality required by the use cases described in the previous section. We thus focus on the functionality provided by these building blocks. The framework will have to be able to support these building blocks and their functionality.

Classical building blocks are digital signatures [32, 28, 11], symmetric [29, 13] and asymmetric [32, 17] encryptions, cryptographic hash functions [22] and X.509 certificates [24, 25, 21]. Also, some less commonly known techniques will be used in this deliverable: commitments, anonymous credentials and verifiable encryptions.

3.1 Commitments

A commitment [30, 14] can be seen as the digital analogue of a “non-transparent sealed envelope” [20]. It enables a committer to hide a set of attributes (non-transparency property), while at the same time preventing her from changing these values after commitment (sealed property). The values contained in a commitment can be seen if one knows the commitment opening info, which is initially only known by the one who created the commitment.

3.2 Verifiable Encryptions

Verifiable encryptions [2, 7, 8] have all the characteristics of regular encryptions. Additionally, they enable their creator to demonstrate properties of the encrypted plaintext to an entity that knows and trusts the public key. Both interactive and non-interactive verifiable encryptions schemes exist.

3.3 Group Signatures

A group signature scheme is a method for allowing a member of a group to anonymously sign a message on behalf of the group. The concept was first introduced by David Chaum and Eugene van Heyst in 1991 (see [10]). For example, a group signature scheme could be used by an employee of a large company where it is sufficient for a verifier to know a message was signed by an employee, but not the particular employee who signed it. Another application is for keycard access to restricted areas where it is inappropriate to track individual employee’s movements, but necessary to secure areas to only employees of the group.

Essential to a group signature scheme is a group manager, who is in charge of adding group members and has the ability to reveal the original signer in the event of disputes. In some systems the responsibilities of adding members and revoking signature anonymity are separated and given to a membership manager and revocation manager respectively. Many schemes have been proposed, however all should follow these basic requirements:

- **Soundness and Completeness**: Valid signatures by group members always verify correctly, and invalid signatures always fail verification.
- **Unforgeable**: Only members of the group can create valid group signatures.
• **Signer ambiguous:** Given a message and its signature, the identity of the individual signer cannot be determined without the revocation manager’s secret key.

• **Unlinkability:** Given two messages and their signatures, we cannot tell if the signatures were from the same signer or not.

• **No Framing:** Even if all other group members (and the managers) collude, they cannot forge a signature for a non-participating group member.

• **Unforgeable tracing verification:** The revocation manager cannot falsely accuse a signer of creating a signature he did not create.

The ACJT 2000 [3] group signature scheme is a state of the art scheme. Classical group signatures require collaboration of each group member (possible signer of the placed signature) and signatures can be deanonymized. A simpler variant on group signatures are ring signatures, introduced by Rivest, Shamir and Tauman [33]. These allow the composition of the group at the moment of signing by the signer without collaboration or even consent of the other group members. The anonymity in ring signatures is absolute, meaning that deanonymization is not possible.

### 3.4 Anonymous Credentials

Anonymous credentials [4, 6] allow for anonymous yet accountable transactions between users and organizations. Classical credentials only allow the release of all the contained attributes. Anonymous credentials are computationally more demanding, but can offer more privacy enhancing properties:

- **Unlinkability.** Multiple shows of the same credential can not be linked. This assumes that no unique (combination of) claims is disclosed.

- **Selective disclosure.** The user chooses what properties of the attributes are disclosed and what is hidden in a credential show. Complex properties involving constants and multiple credentials can be shown; e.g. \(\text{cred1}.\text{wage} + \text{cred2}.\text{bonus}/2 < 2000\). Properties can be proved in plaintext, in verifiable encryptions or in commitments. Possession of a credential can also be proved.

- **Conditional anonymity.** When a certain, predetermined, condition is fulfilled (e.g. abuse), the user can be deanonymized.

- **Pseudonymity.** Different actions can optionally be linked by the use of the same, agreed user pseudonym.

### 3.5 Mix networks

Research on anonymous communications [15] started in 1981 with Chaum’s seminal paper “Untraceable electronic mail, return addresses, and digital pseudonyms” [9]. Since then, a research community has concentrated on building, analyzing and attacking anonymous communication systems.
Data communication networks use addresses to perform routing which are, as a rule, visible to anyone observing the network. Often addresses (such as IP addresses, or Ethernet MACs) are a unique identifier which appear in all communication of a user, linking of all the users transactions. Furthermore these persistent addresses can be linked to physical persons, seriously compromising their privacy.

Anonymizing the communication layer is thus a necessary measure to protect the privacy of users, and protect computer systems against traffic analysis. They also support anonymization techniques at the application layer, such as anonymous credentials, elections and anonymous cash.

3.6 Belgian eID card

For citizens, probably the most visible e-government initiative is the introduction of the eID [27], opening up a host of opportunities. After a successful pilot project in which the inhabitants of 11 municipalities were provided with their eID, the Council of Ministers gave the go-ahead for the national roll-out on 31 March 2004. By the end of 2009, 8 million Belgians over the age of 12 will possess an eID.

With the eID, Belgium has created a card that allows the cardholder to identify and authenticate himself and to place a qualified electronic signature. In this way the eID is a very important means for efficient e-government applications. The appearance of those applications combined with the legal recognition of the electronic signature will rapidly and safely replace a part of the paper documents by their electronic equivalent. It helps to simplify the administration and to modernize the public services. It must be noticed that the eID is not only used in the relation with the public sector, but that citizens and private companies can also use the electronic identity card for the same purposes as part of their own relationships.

As mentioned above the present eID card has three functions: visual and electronic identification of the cardholder, electronic authentication of the cardholder using asymmetrical cryptography and PKI and generation of digital signatures with legal power (non-repudiation). The Belgian eID card thus does not function as a purse for electronic currency.

The Belgian eID takes the form of a processor chip card. On the one hand data are printed on the card, and on the other, data are stored on the card’s processor chip.
4 Framework General Definition and Principles

In this section, we explain what a framework exactly is, and what principles must be fulfilled.

4.1 Definition

“A software framework is a reusable design for a software system (or subsystem). This is expressed as a set of abstract classes and the way their instances collaborate for a specific type of software.” - Johnson and Foote 1988; Deutsch 1989 [26]

According to Pree [31], software frameworks consists of frozen spots and hot spots. On the one hand, frozen spots define the overall architecture of a software system: its basic components and the relationships between them. These remain unchanged (frozen) in any instantiation of the application framework. On the other hand, hot spots represent those parts where the programmers using the framework add their own code to enhance/extend the functionality specific to their own project.

A framework should allow the developers to spend more time concentrating on the business-specific problem rather than on the code. Also a Framework will limit the choices during development, so it increases productivity, specifically in big and complex systems. A framework should eliminate the effort of continuous re-discovery, re-invention of concepts and their re-development.

A software framework can be geared toward building graphical editors for different domains like artistic drawing, music composition, and mechanical CAD. Another software framework can help build compilers for different programming languages and target machines. Yet another might help build financial modeling applications or decision support systems. There are frameworks for multimedia, web applications, and even communicating between different systems.

4.2 General Framework Requirements

In this subsection, general guidelines for a good framework are given.

The primary benefits of Object Oriented application frameworks stem from the modularity, reusability, extensibility, and inversion of control they provide to developers, as described in [18] and repeated below:

- **Modularity.** Frameworks enhance modularity by encapsulating volatile implementation details behind stable interfaces. Framework modularity helps improve software quality by localizing the impact of design and implementation changes. This localization reduces the effort required to understand and maintain existing software.

- **Reusability.** The stable interfaces provided by frameworks enhance reusability by defining generic components that can be reapplied to create new applications. Framework reusability leverages the domain knowledge and prior effort of experienced developers in order to avoid re-creating and re-validating common solutions to recurring application requirements and
software design challenges. Reuse of framework components can yield substantial improvements in programmer productivity, as well as enhance the quality, performance, reliability and interoperability of software.

- **Extensibility.** A framework enhances extensibility by providing explicit hook methods [Pree:94] that allow applications to extend its stable interfaces. Hook methods systematically decouple the stable interfaces and behaviors of an application domain from the variations required by instantiations of an application in a particular context. Framework extensibility is essential to ensure timely customization of new application services and features.

- **Inversion of control.** The run-time architecture of a framework is characterized by an “inversion of control.” This architecture enables canonical application processing steps to be customized by event handler objects that are invoked via the framework’s reactive dispatching mechanism. When events occur, the framework’s dispatcher reacts by invoking hook methods on pre-registered handler objects, which perform application-specific processing on the events. Inversion of control allows the framework (rather than each application) to determine which set of application-specific methods to invoke in response to external events (such as window messages arriving from end-users or packets arriving on communication ports).

Another requirement is **usability**, which is indeed a broad term which involves multiple elements. This means that the framework should be sufficiently easy to learn by application developers: learning to use the the framework must not be harder than learning to use the individual technologies that are required to develop the applications. Learning how to work with the framework must not require a huge amount of effort, otherwise it will not be used at all. Therefore, the API must be made as intuitive as possible. To quote Einstein: “Things should be made as simple as possible, but not any simpler”. The framework should be easy to configure, deploy and maintain once deployed. It is unavoidable that the framework will need to be updated. This must be possible with minimal changes by the application developers. Usability also implies that the framework does not involve a big computational or storage burden. Usability also means that a good, detailed and clear documentation should be available. Although this is often seen as lower-priority task, it is indispensable in gaining popularity amongst application developers.

**Testing and validation** is an indispensable part of framework development. Due to the high level of abstraction, testing the framework apart will be a though job. It can as well be tested using concrete use case implementations, however if a bug is found, it can be located in either the framework or the application built upon the framework. However, **validation** of the framework using concrete use cases is necessary to test the completeness of the functionality of the framework, to test its user-friendliness, etc. Based on the feedback, a next iteration is possible.

Indeed, the framework’s functionality are abstractions derived from concrete use cases. Based on those concrete examples, common functionality is grouped and the corresponding methods are refactored to generic methods. This generalization is an iterative process and is based on the feedback given by the
experts of the different use cases. Analyzing and implementing those example use cases is a large fraction of the cost of the project.

4.3 Providers

The framework as such cannot be used by the application developer. Therefore, an implementation of one or more components is necessary. Three properties must be satisfied: technology independence, implementation independence and multiple instances:

- **Technology independence.** It must be possible to add or remove technologies to the framework in a transparent and easy way for the application developer. E.g. if the framework only offers support for the Belgian eID card, it must be possible to change the used technology to the Swedish eID card.

- **Implementation independence.** The possibility must be offered to easily change the implementation of a technology supported by the framework. E.g. if implementation A is plugged in offering the possibility to show the Belgian eID to others, it must be possible to change to another implementation B doing the same. This can be useful e.g. if security breaches are detected, or if more efficient implementations are offered. This allows lightweight software implementations on mobile phones, more heavy weight software implementations on desktop computers and fast hardware implementation on servers.

- **Multiple instances.** In one framework, it must be possible to use multiple technologies for the same interface at the same time. This way, the support for the Belgian and Swedish eID card is possible at the same time. One must be able to choose which technology to use (e.g. to select the credential type to issue). If multiple implementations can be plugged into the framework, one must be able to choose the implementation (the one that is considered as the most trustworthy one, the most efficient one, etc). One must be able to add or delete and choose technologies and implementations in an easy way, preferably at runtime.

Multiple instances allow the Adapid framework to support classical X.509, Idemix and U-Pove credentials and multiple eID cards at the same time. Hundred percent technology independence is impossible in our framework as different technologies have different properties. At least the different technologies must be usable by the application developer in a similar way.

To obtain these properties, so called providers can be used. A provider contains one or more implementations of one or more technologies that fit into the framework. Thus, at the one hand, we have the framework, offering the API, and at the other hand, we have the providers, offering the implementations. Different providers must be pluggable at the same time into the framework without conflicting.
4.4 Sensitive Data Representation

In a security framework, it makes sense to have two representations of sensitive data (e.g. secret keys) outside the framework. Opaque representations allow one to access only the non sensitive data (e.g. name), but disables the possibility to use the sensitive data outside the framework. The full functionality can only be used inside the framework. This is the more secure and default representation within the framework context. Transparent representations on the other hand do not have these restrictions and allow to export the sensitive data. This representation will typically be a global standard representation and thus allows interoperability.

In the ADAPID framework, the sensitive data consists of two types of sensitive data: secret keys and personal data. The former is needed to use credentials while the latter is contained in credentials or in evidence of actions (transcripts). By default, an opaque representation will be used to enhance security, while the possibility of transparent representations must evidently be offered to enhance interoperability with other systems able to handle the data (e.g. X.509 certificates).

4.5 Patterns

In software engineering (or computer science), a design pattern [19] is a general repeatable solution to a commonly occurring problem in software design. A design pattern is not a finished design that can be transformed directly into code. It is a description or template for how to solve a problem that can be used in many different situations. Object-oriented design patterns typically show relationships and interactions between classes or objects, without specifying the final application classes or objects that are involved. Algorithms are not thought of as design patterns, since they solve computational problems rather than design problems. In designing frameworks, the use of patterns is of utmost importance in obtaining maximal flexibility and extensibility. In this section, a few examples of how pattern can be used in the design of frameworks are given.

A potentially important design pattern is the Bridge Pattern. It decouples an abstraction from its implementation, so that the two can vary independently. This is important if a framework has to offer uniform APIs, while the underlying algorithms and implementation can vary.

The Observer pattern offers a mechanism to keep related components consistent, without having a close coupling between these components, as this increases complexity and decreases reusability.

As a final example, setting up a connection to a server can be rather tedious as complex interfaces and network operations are involved. The Service Locator pattern hides this complexity.

Other patterns exist as well. We refer to [19] for an overview of useful patterns.
5 The Adapid Framework

This section describes how the current framework is developed, as well as its resulting high level architecture. We start by giving the specific requirements, besides the ones described in the previous section. We continue by sketching how the current API was developed and how the framework is structured in components.

5.1 Requirements

Based on the functionality required by the applications in section 2, an overview of what is needed by the Adapid framework is given. Evidently, the general guidelines described in section 4.2 still apply. The framework needs to offer at least:

- The possibility to easily set up connections with certain properties (anonymity, identification, confidentiality, etc.).
- In general, support for both client and server is required. This functionality will often be complementary.
- Handling of credentials: issuing credentials and receiving newly issued credentials from an issuer on the one hand, and, on the other hand, showing credentials to a verifier and verification of credential shows by a verifier.
- The possibility to use credentials of different types in a uniform way: X.509 certificates, Idemix, U-Prove, eID cards, etc.
- Together with the support for anonymous credentials, support of pseudonyms is needed. Although not explicitly stated in the application section, a user needs to known what personal data has already been released to whom under what nym or identity. In general, easy management of identities and nyms is required, combined with the ability to get an overview of the personal data released by the user.
- Complementary to the previous bullet, the server side may want to keep an overview of the received personal data and nyms (profiles).
- As a user will potentially have dozens of credentials, easy management of these credentials (with their nyms and or identities) must be provided.
- Collection, management, storage and use of evidence of certain actions.
- In case of abuse or in case one is accused, a mechanism must be present such that these disputes can be handled automatically or at least semi-automatically. To correctly handle disputes, deanonymization must be possible (and thus supported by the framework) as well.

5.2 Approach

This section explains how the API of the framework was gradually developed.

In Deliverable 2: Requirements, multiple use cases where described in the e-governement, e-health, trusted archival and financial domains. These where
converted into interaction diagrams. A simplified example of a patient visiting a doctor is given in figure 1.

The doctor logs into his doctor’s computer (1). This can happen by the use of the Belgian eID card, once or a few times a day. When the patient consults the doctor, he also needs to authenticate to her trusted device (2), in order to obtain access to this device. Now, the doctor proves his qualifications to the patient (3). The patient identifies himself to the doctor (4), which can as well happen with the help of the Belgian eID card. After having examined the patient, the doctor issues a prescription to the patient.

The methods seen in this small example are mostly application specific. For instance, an `issuePrescription()` method will likely not be used in the e-government or e-finance domain. Therefore, these remote, application level methods (interactions) are converted to generic methods towards the underlying, locally running framework, as shown in figure 2, where the `issuePrescription()` method is converted to an `issueCredential()` call by the doctor to the doctor’s framework and a `getCredential()` call by the patient to the patient’s framework.

Note that the conversion from application level to generic methods is not always unique. This is due to the fact that the generic methods do not allow the same level of abstraction. In the given example, the prescription can be a credential (X.509, Idemix, etc.), but as well a message signed by using a private key corresponding to such a credential. This will have its effect on the translation of the application specific methods to generic ones. The different options need to be examined and need to be provided by the framework. Note that the more complex interactions between the framework instances (denoted as `credentialProtocol()` in figure 2) are completely hidden for the application developer.

The framework should not offer an implementation for these generic methods as such, but need to offer a local interface that can handle these interactions.

5.3 Components

In this section, the different components of the Adapid framework and their functionality are discussed. An overview of the different components is given in table 3. Each component is a logical grouping of functionality and contains an interface for that functionality. Most of these components are the result of a logical grouping of the functionality requested by the applications in section 2 and the generic methods derived as explained in the previous sections.

Four layers can be distinguished in the framework. At the highest level the presentation layer provides a GUI to the user. At the functional and administrative layer, APIs are offered to the application (developer). While the func-

![Figure 1: Example of a simplified interaction diagram derived from a use case, showing the application level interactions between different parties.](image-url)
Figure 2: Example of how a remote, application specific method is translated to local calls to the framework.

Figure 3: Overview of the framework and its components.

The functionality of the different components is now explained. We elaborate on the component API in section 6.

Presentation Layer

- **User Interface.** The User Interface (UI) provides a uniform interface to the user. This interface can be independent or can be incorporated in the application’s user interface. The UI is a GUI (Graphical User Interface), but can in principle as well be a command line interface, which is still common for server applications. This UI is not compulsory, an application developer can still build his/her own UI, but a uniform, well-thought-

...
out interface increases both user-friendliness and security and speeds up application development.

**Functional Layer**

- **Communication.** This component is responsible for establishing connections with external entities.

- **Credential Handler.** Component offering the functionality to handle credentials: showing, verifying, issuing, and receiving newly issued credentials.

- **Credential Manager.** The credential manager allows the user to manage his/her credentials. It helps the user/application to find an appropriate credential when properties have to be shown and facilitates browsing the present credentials.

- **Privacy Manager.** The privacy manager manages the user’s privacy. It allows the user at the client side to manage his/her different profiles (identities); i.e. it allows the user to keep track of what personal data has been disclosed to whom. It allows the server application to manage profiles of the different persons/nyms.

- **Dispute Handler.** Even by the use of the latest technologies, not all abuse can be prevented. Therefore, evidence during the interactions between different parties must be collected and stored. In case of abuse, one must be able to submit a dispute claim to some dispute handling authority; i.e. a server to which one can submit complaints. The dispute handler component offers the logic to handle disputes: collecting evidence, submitting complaints and handling them.

- **Deanonymizer.** Using anonymous credentials, deanonymization might be necessary, for example, only the winner of an anonymous auction is deanonymized to the manager of the auction or the deanonymizer can be contacted when abuse is detected.

- **Trusted Archival Service (TAS).** This component is responsible for storing data for a long time while at the same time guaranteeing the integrity.

**Administrative Layer**

- **Policy Manager.** A policy can be associated with one or a set of credentials, or with the framework as a whole. These policies are managed and enforced by the policy manager. Both clients and servers can have policies.

- **Logger.** Actions and events related to the framework can be logged (by default or as described in policies). This is handled by the Logger component.

**Storage Layer**

- **Storage Manager.** The Storage Manager contains the logic to store data, not the storage itself. A client Storage Manager could for instance contact a remote storage server to store his/her credentials while logs may be stored locally.
 Interfaces

In this section, we zoom in into the component interfaces that are already developed. These components are the Trusted Archival component, the Communication Component and the Credential Handler component. We emphasize that no validation of the framework has yet been done and that thus the interfaces will likely be modified in the future, based on the results of the validation phase.

6.1 Communication

6.1.1 Description

This component is responsible for establishing connections with external entities. It is used by the client to connect to another entity in the network, but also the server uses (complementary) functionality offered by this component to set up a server offering services that are listening for incoming connections. Different options such as integrity and network layer anonymity can be offered by the component.

6.1.2 Functionality

The Communication component must be able to:

- Establish a connection to another party. This party will be typically addressed by its IP address or domain name.
- Listen for and accept incoming connections. The acceptance will only be done if the incoming connection has certain properties (e.g. integrity preserving in both directions). Listening happens on specific ports, which can be associated with specific services.
- Allow interactions between the two parties in order to agree/know the properties of the connection.
- Provide the following properties in each of the two directions of the communications link: anonymity of an entity, network level identification (classical authentication) of entities using classical certificates (e.g. X.509 certificates), integrity, confidentiality. Extra attributes need to be provided in order to obtain these properties. For instance, a gateway to an anonymity providing network must be given, or a certificate used to do the authentication. Evidently, configurable defaults must be provided by the framework
- Set other communication related properties such as connection timeout, maximum numbers of incoming connection a single entity can handle, etc.

6.1.3 Interface

An overview of the communication interface is given in the communication packet shown in figure 9.
The abstract class Connection represents a connection to another entity. A Connection object can be used to send or receive objects. Different implementation of this class can implement different technologies (sockets, RMI, etc). However, the user of the framework should not necessarily be aware of the used technology. A concrete Connection object is generated by either a Connection-Factory or a Server implementation. In the former case, a connection is started, while in the latter case, an incoming connection is accepted.

A server application needs to implement a Service class. This class must contain the logic of how to handle incoming connections. For instance, sending over the connection a request to show a certain credential, followed by a verification of the credential show when the credential is shown; finally, the user could be given access to e.g. vote on a petition. The Server class listens for incoming connections and creates new service instances to handle the incoming requests. A Server instance can contain different service implementations.

The properties a connection are described in an implementation of the abstract ConnectionProperties class. A ConnectionProperties object contains data such as the connection timeout, the local portnumber. Two subclasses are ClientConnectionProperties and ServerConnectionProperties. The former contains client specific connection properties, such as the address of the computer to connect to. The latter contains the maximum number of connections the server will accept simultaneously.

6.1.4 Usage Examples

Two examples are given of how to use the current Communication component interface. We emphasize that this interface is a first iteration and might thus have serious problems. These, however, will be discovered by the validation and tackled in the next iteration.
String technology = "SOCKETS";
String service = petition;
String host = www.petition.com;
int port = 12233;
Object message = ...;
Object reply;

// Generate clientConnectionProperties
ClientConnectionProperties cprops = new ClientConnectionProperties();
cprops.setTechnology(technology);
cprops.setServer(host);
cprops.setService(service);
cprops.setPort(port);

Connection conn = connectionFactory.establishConnection(cprops);
    // Open the connection
connection.send(message);  // send message through connection
reply = connection.receive(); // receive message through connection
connection.close(); // close connection

Figure 4: Setting up a connection to a client

Client connection establishment
In figure 4, example code is given of how a connection can be established by a client to some other (listening) party. First, the client properties are set; i.e. the technology that will be used to establish the connection with, the service name, the host and an optional port number. In the example, the technology is ‘sockets’. Evidently, this is only possible if a provider offering an implementation therefore is plugged in the framework.

These properties are used as input to the ClientConnectionFactory. This factory tries to establish a connection with the requested properties. once this succeeded, the newly established connection is returned. send() and receive() are methods used to send and receive messages. These messages can be everything. Finally, the connection is closed.

Server connection establishment

In figure 5 example code is given of how a server can handle an incoming connection. After setting up the server connection properties, a new server with that properties is created. A service is added. In our example, the service is an e-petition server. Writing such a service is writing an extension of the Service class that contains an implementation for the run() method. The run method contains the logic of how to respond on message an/or how to execute higher level protocols. More services can be added to the same server and within the same framework, multiple servers listening on different ports can be set up. Finally the server is started.
ServerConnectionProperties props = new ServerConnectionProperties();
// Set properties
int port = ...;
props.setPort(port);
...

Server server = new Server(props);
server.addService("petition", new PetitionService(...));
server.startServer();

Figure 5: Handling an incoming connection by a server

If the (service, port) tuples are known by the framework, the port number does not have to be given if the service name is given to the (server or client) communication properties object.

6.2 Credential Handler

6.2.1 Description

The credential handler component offers the functionality required to use all types of credentials (X.509 certificates, Idemix credentials, eID cards, etc.). This means that it must offer the possibility to show, receive, issue and verify credentials in a similar way. Credentials usually serve to proof properties.

6.2.2 Functionality

The credential handler must offer the possibility to support classical certificates such as X.509 certificates and SAML tokens, liability enhanced certificates, but as well more advanced credential types such as the U-Prove and Idemix credential systems. Also, support for eID cards and other smart cards must be possible.

In the case of classical certificates and eID cards, the owner must be able to sign and authenticate using these physical certificates.

One must be able to set all the properties that will be shown using one or more anonymous credentials. This means that not only the properties w.r.t. the attributes must be given, but as well the way under which these will be disclosed: plaintext, commitment or verifiable encryption. In the case of verifiable encryption, the public key of the party able to decrypt the cipher must be given as well. The connection and pseudonym (if any) that will be used in the show protocol must be set as well. Because of the necessity to use nyms in many show protocols, support is provided to issue, receive, use and verify nyms as well, both signed an unsigned nyms. When showing a credential, it must be possible to sign a message (i.e. anonymous/pseudonymous signing). The deanonymization conditions must be set as well before showing a credential.
6.2.3 Interface

In the credential handler component in figure 10, the structure of the credential handler interface is given. CredentialHandler is the most important interface. It offers an interface to request, issue, show and verify credentials of potentially different types (X.509, Idemix credentials, etc.), depending on the provided technologies. If also allows to (anonymously or not) sign messages and to agree pseudonyms. Revocation and checking for revocation is provided as well. When a credential is shown, issued, etc., a myriad of options can be set (e.g. what properties to show). This can be set in the Properties classes; ProverProperties describe what will be shown, VerifierProperties describe how a credential will be verified, IssuerProperties describe how a credential will be issued, ReceiverProperties describe the credential that is expected to be received. NymProperties describe the nym to issue or to receive.

6.2.4 Interface Usage Example

Two examples of how the framework interface can be used are given. First we show the code required to show a credential, secondly, we show how a credential can be received and verified by a server.

Figure 6 illustrates how a credential can be shown. Therefore, an instance of the Credential Handler is created. A connection over which the credential will be shown must be set as well as the assertion describing the properties to prove, the credential (or credentials) that will be shown and the nym under which the credential will be shown. Now, the properties object can be created. The assertion, credential and nym are added to is. The credential is now shown by calling the showCredential() method in the credential handler object. the connection and properties are given as argument.

In case the credential is a classical X.509 certificate or a Belgian eID, the nym, and assertion can be null values The interfaces are a first version and will be further improve. E.g. it is not yet possible to choose the implementation to use. It is not logic that the credential and nym are put in the properties object. Also, the interface is too much directed towards anonymous credentials.

The example in figure 7 illustrates what is needed in order to receive and verify a credential show by a server. First a connection between a client and the server is established. The server listens in this example to the connection till it receives the properties that the client wants to show. Now, a new Credential Handler instance is created and the proof protocol between the client and the server is performed in the last line. The connection and the properties are given as input by the server. As you can see, the interface does not yet provide full technology transparency as the received properties are casted to an Idemix properties class.

6.3 Trusted Archival Service

6.3.1 Description

The primary goal of storage security is, traditionally, to ensure the CIA (confidentiality, integrity and availability) properties. In order to provide the first
CredentialHandler cHandler = new CredentialHandler();
Connection conn = ...;
String assertion = ...;
Credential cred = ...;
Pseudonym nym = ...;

IdemixProverProperties props = new IdemixProverProperties();
    props.setAssertion(assertion);
    props.setCredentials(cred);
    props.setPseudonym(nym);

credentialHandler.showCredential(connection, props)

Figure 6: Proving properties of a credential

Connection conn = ...
IdemixVerifierProperties props =
    (IdemixVerifierProperties) conn.receive();
CredentialHandler credHandler = new CredentialHandler();
credHandler.verifyCredential(conn, props);

Figure 7: receiving a credential show by a server
two, we usually rely on cryptographic tools, like encryption for obtaining confidentiality or digital signatures in the case of integrity. On the other hand, availability cannot solely rely on cryptography and needs additional measures, like replication or error correction.

Even when an archival service provides these properties, they will be degraded in the course of time. A Trusted Archival Service (TAS) aims to go a step further than the traditional storage systems and provide mechanisms to preserve these properties over time, such that they can be proven valid well in the future even when the cryptographic references (e.g. Certification Authorities, Certificate Revocation Lists, . . .) are no more available.

The document [5] describes the specific requirements that a Trusted Archive Service should fulfill in order to provide a secure storage service. These requirements consider the three main aspects of security: availability (for submission, retrieval, deletion. . .), integrity (integrity should be provable), confidentiality (against third parties and the archive itself). Furthermore, there should exist a long-term policy, operations on groups of documents should also be supported. . .

In the frame of the ADAPID project we have designed a TAS that focuses on long-term integrity that:

- proves the existence in a bounded time in the past of a document well into the future when, most probably, the certificates related to its digital signature are no longer available (or no longer valid) and
- proves that the content of the signed document has not changed during the storage period.

The designed scheme allows users to sign documents using the e-ID card, store them in the archive and retrieve them long time after. The TAS guarantees the maintenance of the integrity of the document and a proof of the time when the signature has been generated is given. An in-depth explanation of the theory behind it can be found in Deliverable 5 “Trusted Archival I”, together with the description of a demonstrator which serves as a proof-of-concept of our design.

We consider a client-server architecture, with a trusted third party, a Time Stamp Authority (TSA) which issues timestamps (see Fig. 8). In the framework context, the TAS would act as a server, and the rest of the framework components (like the Logger, the Policy Manager, the Access Controller, . . .) would be seen as clients. The main functionality of the TAS is to store any document coming from the rest of the framework components. Beside storing the documents in a secure way, the TAS ensures their integrity over time and keeps a proof of their existence at a certain point in time.

6.3.2 Functionality

The next sections go through the functionalities offered by the TAS:

Archiving a document. When a client has a document to archive in the TAS, creates a packet and sends it to the TAS. The TAS checks the validity of the signatures and collects the information necessary to recreate this operation in the future (CLR, OCSP). Then, the TAS will request a timestamp, which proves that checking and archiving the data took place after the signing and
before the time given by the timestamp.

Re-timestamping. The main purpose of a Trusted Archival Service, is to extend the reliability of the cryptographic primitives used to sign a document. In our scheme, we re-timestamp the document and all the reference information at regular intervals. The TAS checks that the integrity of the current packet has not been violated and requests a new timestamp over all the information. We assume that the Time Stamping Authority uses always the latest signing technology, ensuring protection on the document until the next scheduled refreshing.

Retrieval of documents. By storing all the timestamps along with their certificates and proofs of correctness, a verifiable path is created. This path can be checked later on and it can be proved that there has been no modification on the stored document.

When a client requests a file, a proof of ownership will be and, assuming this proof is correct, the client receives the whole packet, with the original document, and the whole chain of timestamps. With this information, the client can perform the validation.

Deleting a document. A client (or the storage itself) may want to eliminate a document (and its correspondent reference information) from storage. In order to do that he needs to proof that he has rights to perform the operation and it will be done by the TAS in one of the ways described in Sect. 3 of Chapter 3 in Deliverable 5.
The current adaptors are interwoven with the framework, making it extremely inflexible. This separation will be done in the coming months.

### 7.1 Communication

As can be seen in figure 9, adaptors are provided to establish connections using sockets and SSL sockets. The latter provides identification of one or both of the parties and integrity and confidentiality of the sent data. Therefore, classical X.509 certificates are used. The existing socket and SSL socket implementations provided by Java are used as base implementations. No implementation/adaptor for anonymous connections is currently provided.
7.2 Credential Handler

Support for the Belgian eID card is present in the current prototype implementation (signing, authentication and reading the personal data file in the Belgian eID card). The Godot implementation [12] as well the PkSuite [23] implementation are supported, meaning that adaptors are written to transform the framework API calls to the Godot and PkSuite implementations. Besides support for the eID, the PkSuite also offers support to use classical certificates. An Idemix adaptor is written as well. Deanonymization, which is functionality that should be provided by another component in the framework, is not yet supported by neither the framework, nor the current Idemix implementation. Also, other functionalities are missing in the current Idemix implementation and can thus not yet been offered by an adaptor; e.g. anonymous signing of a message and verifiable encryptions.

Figure 10 shows how the adaptors and extensions are plugged into the framework. No adaptors have yet been written for U-Prove anonymous credentials (since no implementation is available), SAML tokens, etc.

7.3 Trusted Archival

The protocol designed to create a Trusted Archival Service achieves the objective of maintaining the integrity of documents over time, as it is shown with the demonstrator defined in Chapter 4 of Deliverable 5. The chain of evidence created by the protocol allows to check the validity of a digital signature over a document stored in the TAS, even when the certificates used to produce them are no more valid.

Our TAS also complies with the requirement of being able to give a proof of existence of a document at a certain point in the past. By requiring a timestamp before and after producing the signature, the signing time can be tightly bounded, and therefore can be used as evidence in court in case of dispute.
However, our protocol does not deal with format migration issues. It also does not address how to extend the validity period of an encryption algorithm. These, and other limitations of the protocol are also discussed in Sect. 3 of Chapter 3 of Deliverable 5.

Finally, the demonstrator we developed, is just a proof-of-concept of the protocol we designed. More advanced implementations need to be done in order to create a commercial/Governmental TAS, like not limiting the number of connections to one, increasing the available storage space, etc.

Currently, the Trusted Archival was developed as a stand-alone component. In the future, it will be integrated into the framework.
8 Existing Identity Management Systems

In this section, existing management systems are introduced, together with an analysis. These identity management systems are Microsoft Cardspace, Higgins Project and the Liberty Alliance Project. This offers us a basis for future comparison with the Adapid framework.

8.1 Microsoft Cardspace

8.1.1 General introduction

MS Cardspace is part of the operating system Windows Vista, and of the .NET Framework 3.0 which can be downloaded for Windows XP of Windows Server 2003. MS Cardspace was previously known as MS Infocard. MS Cardspace is offering an identity layer between the user and the website. Users can choose what to reveal to websites, using cards. This also means that MS Cardspace is a client side software, which distinguishes it from e.g. IBM TIM and TAM of Liberty Alliance specifications.

8.1.2 Detailed description

Cardspace (formerly known as Infocard) is Microsoft’s new identity management system. It allows users to manage a collection of digital identities from various identity providers, and employ them in different contexts where they are accepted to access online services. Digital identities are encoded as security tokens (credentials). Security tokens contain claims about a user (name, telephone number, ...). They can be issued by an Identity Provider (IP) or self-issued. They are presented to a service provider, referred to as a Relying Party (RP). To organize digital identities, Microsoft Cardspace uses infocards. An infocard represents a digital identity issued by a certain identity provider. Different digital identities are represented by different infocards. Therefore, users typically have a collection of infocards. An infocard only contains XML metadata about the digital identity. It only defines that a certain IP (e.g. the Belgian government) asserts certain claims (name, birth date, ...) about the owner of the infocard. The infocard does not contain the security token or any personal information of the user. The actual security token has to be requested from a security token service of the identity provider when needed. When a user tries to access a service of a RP, the RP will request certain claims about the user. He will also define the IPs he trusts and the kinds of security tokens he accepts. The Cardspace client running at the user’s computer will then search the infocards that fulfill the requirements of the IP. The user can then select the infocard to use. Since the infocard describes where to get a certain security token, Microsoft Cardspace is able to retrieve a security token from the IP and to access the service. In order to get a security token from the IP, some authentication method (username/password, smart card, ...) is required.

Microsoft Cardspace is very interoperable. For communication the open WS-* protocols are used. The structure of infocards is also open. Open source implementations are already in progress.
8.1.3 Interaction

Following example goes deeper into the interactions when a user authenticates to a RP. John is a citizen with an eID card. Suppose that a firm BelgianTrust? is an identity provider. If a citizen subscribes to BelgianTrust? with his eID, he receives an infocard. This card states that the user can retrieve several kinds of security tokens (X.509 certificates, SAML tokens, ...) certified by BelgianTrust?. These security tokens can include the name, address and birth date of the user. If the user needs such a security token, he sends a request to the security token service (STS) of BelgianTrust? at address www.belgiantrust.be/sts. This service is able to issue the different kinds of security tokens. To authenticate towards this STS, users have to use their eID card. Suppose Amazon.com starts supporting infocards. When buying a book, users have to prove their address. Following figure shows the interactions between the different components.

0 (out of band) Before accessing services, users have to subscribe to an IP/STS. After subscribing, the IP sends the user an infocard. Microsoft Cardspace does not define the way (mail, web, ...) in which this card has to be sent.

1. The user requests an identity based service.
2. The client requests the security policy of the RP.
3. The user client receives the policy. To do this WS-MetaDataExchange, an XML format designed to exchange policies and other kinds of data, is used. Two things are included in the policy:

- The types of security tokens that are accepted, the trusted IPs and the required claims are specified. For example:
  - Security token type: SAML 1.1
  - Identity Providers: TrustE or BelgianTrust
  - Requested Claims: address
- The security requirements about the communication channel are specified (e.g. SSL is needed). Microsoft CardSpace? does not dictate how communication channels have to be secured. The relying party can specify the required degree of protection.

4. Every user of Microsoft Cardspace has an identity selector installed on his device. The identity selector collects the different infocards of the user. When the IP sends his requirements, the identity selector checks every infocard and determines the infocards that can be used.

5. The identity selector shows the infocards that comply with the policy to the user. The figure below shows how the identity selector looks like.

6. The user selects the infocard that he wants to be used. Suppose the infocard from BelgianTrust? is selected.

7. The identity selector now knows the security token service from which he has to retrieve the security token. In the example this is www.belgiantrust.be/sts. The STS is contacted to receive a security policy (using WS-MetaDataExchange).

8. The security policy is sent to the user. The security policy, again, describes the security measures that must be taken into account. Also, in most cases, the user will need to authenticate towards the IP/STS. He must prove that he is the legitimate owner of the infocard. It is up to the IP/STS to choose the particular technology (X.509 authentication, username/password, ...).

9. The security token with the required claims is requested. To do this WS-trust, an XML specification for requesting security tokens, is used. The request also includes the authentication described in 8. Furthermore, it includes the ID of the infocard for the IP to know which account is accessed. Note that IP/STS does not need to know the identity of the RP.

10. A signed security token is sent back to the identity selector, together with proof-of-possession key. Only the required claims are included in the security token.

11. The security token is presented to the RP using WS-security. The WS-security standard describes how to attach signature and encryption headers to SOAP messages. It also describes how to attach security tokens.

12. The RP returns the requested service response.

13. The user has access to the identity based service.
8.1.4 Identity Providers concept

MS Cardspace uses the concept of ‘identity providers’. A Bank can offer you an identity, each organisation can offer you an identity, and also yourself can act as an identity provider for yourself. This is for example what you do if you use email verification to register as a new user to a web application. Using this identity provided by the identity provider, you can afterwards login to the web application of your choice, on the condition that this web application accepts the identity provider authenticity.

Even if MS Cardspace is seen as an identity selector, MS itself likes to refer to it as an Identity Metasystem. The main architect for Microsoft Cardspace is Kim Cameron, who is widely accepted as an expert in identity management (see also http://www.identityblog.com)

8.1.5 Requirements

Microsoft Cardspace tries to provide a uniform interface for users to manage their digital identities. The major focus is on security and on maintaining end-user control. Following list provides the main requirements.

- The system tries to achieve strong authentication by using claims in security tokens as authentication mechanism.
- Users are in control of the use of their identities. The user can always choose the way of disclosing information.
- Users should be sure of the identity of the service provider to which they disclose information. Therefore, the system achieves cryptographically verifiable but human-friendly identification of the recipients of a user’s
digital identities. When a user has never used a service before, Microsoft CardSpace displays the service’s certificate logos. Also, this logo can be signed by a trusted party. The logo of that party is also shown. This gives the user a chance to visually verify that he’s talking to the right web service.

- Interoperability is very important for Microsoft Cardspace to be successful. Therefore, it makes use of open protocols (WS-* protocols) for the different interactions. Also, the system tries to remain agnostic of specific security token types. Service providers can choose the security tokens they want to accept.

- The system tries to protect the privacy of the user. Security tokens that are disclosed to the service provider only contain the information asked for. The user also knows exactly what information is disclosed.

- Microsoft Cardspace tries to provide a consistent user experience. If a user proves information to a service provider, the system always uses the same interface.

8.1.6 Technology Used & Components

Cardspace is an essential part of the .NET Framework 3.0 API’s. Standard included inside Windows Vista, but also available for Windows XP and Windows Server 2003. Microsoft Cardspace can be decomposed in three components.

- **Relying party**: The relying party requests a user to prove certain claims. By using the protocols of Microsoft Cardspace, a relying party can easily define the types of security tokens he accepts and what claims he needs to know.

- **The Identity selector**: The identity selector is used whenever the user needs to disclose information. The identity selector shows the cards the user can use. After that it contacts the STS of the identity provider to retrieve the appropriate security token. Last, this security token is shown to the relying party.

- **Security token service (STS)**: the security token service is able to issue one or more type of security token. Sometimes users will first have to authenticate towards the STS, for example by using a smart card. Every user will have his own STS that handles self-issued infocards.

8.1.7 Evaluation

**Evaluation on user friendliness user experience**

Cardspace offers the user a visible approach to his identities. The usage of course depends on:

- the number of web applications that actually support these technology
- the number of identity providers that provide identities for Cardspace

One of the major achievements of Microsoft Cardspace is to offer a consistent user experience.
Evaluation on user privacy

The user can choose which identity to use. This implies a lot of functionality. Possibilities to enhance the privacy of the user are:

- The user can choose to only disclose some attributes to authenticate himself.
- The user can choose to disclose a different identity to authenticate himself.

All these functionalities are of course limited by:

- The requirements of the web application that will be authenticated to
- The availability and compatibility of identity providers to offer just the identity you wish to disclose.

Although the system is already quite privacy friendly, several aspects can be improved. Users may only want to disclose information if, for example, they know what the data is used for and what the retention rate is. This is not handled by Microsoft Cardspace. This could be solved by including privacy policies. Another aspect is attribute disclosure. This system is not capable of disclosing properties on attributes.

By using the mechanisms introduced in Microsoft Cardspace, linkability is prevented in most cases. For example, if a user always requests a new security token every time he proves his year of birth, the relying party can not link them. However, if the identity provider and the relying party work together, it is still possible to link them. To obtain absolute unlinkability, anonymous credentials should be used.

Evaluation on usability

Currently the system is not widely supported. This limits a lot the functionality in any case. However, for further discussion, let’s assume the support is adequate. This because the same is true for a lot of other different identity management systems or identity selectors.

8.2 Higgins Project

8.2.1 Introduction

Higgins is an ongoing project, aimed at creating an common interface layer that will allow various existing identity management systems to interoperate with one another without having to adapt their respective architectures. This common interface layer is called the Higgins framework.

According to Higgins’s documentation, applications use Higgins to insulate themselves from the details and complexities of each distinct collaboration space, communications system, or enterprise directory. Higgins’s service architecture allows applications to use and access an ever increasing range of external systems without any changes to the application itself.

This interoperability is assured by means of a collection of contexts. A context is the surrounding environment and circumstances that determine the
meaning of digital identities and the policies and protocols that govern their interactions. The Higgins service interacts with a set of context providers that either adapt existing legacy systems to the Higgins framework, or implement new ones.

Figure 13 shows how different applications can plug into the Higgins framework to gain access to identity information across multiple contexts, and disparate systems and implementations.

As hinted to in figure ??, the idea behind Higgins’s design is to facilitate a service-oriented architecture. For a service-oriented architecture to be effective, each service (or context) has to be well-defined, self-contained, and should not depend on the context or state of other services. These context definitions are supplied by Higgins’s plugins (context providers.) Using context providers, existing and new systems such as directories, collaboration spaces, and communications technologies (e.g. Microsoft/IBM WS-*, LDAP, email, IM, etc.) can be plugged into the Higgins framework.

Users & Privacy

According to Higgins’s documentation, a key focus of Higgins is to provide a foundation for new "user-centric identity" and personal information management applications. It also adds that Higgins provides virtual integration, user-centric federation, management, trust brokering, linking, and search capabilities that are applied to identity, profile and relationship information across multiple contexts...

The documentation points out also the potential risk for privacy violations, and claims that Higgins will create a key part of the open source infrastructure required for an open, accountable, socially-searchable web while ensuring privacy and personal control over identity information. The documentation does not give details however on the approach to be taken to achieve this.
8.2.2 Requirements

The Higgins platform intends to achieve four requirements:

- **The ability to manage multiple contexts:** It is often desirable for users to be able to easily manage their identities, relationships, reputation and trust within and across multiple contexts.

- **Interoperability:** Rather than introducing yet another new identity system, instead Higgins introduces a new context abstraction and allows developers to create adapters to legacy systems. Systems operating above the abstraction layer have the potential to link identities across identity system boundaries.

- **Trust:** Allowing users and organisations to selectively share information on the web while protecting their privacy will lead to an open, accountable, and socially-searchable/responsible web.

- **Integration/Common Interface for developers:** By using a common API, developers who want to integrate different identity management systems, will not have to learn the peculiar complexities of each system.

8.2.3 Components

An overview of the Higgins components is given if figure 14.

- **Web services:** They provide services to users and organisations on the web. They may require identity-related information both for access control and profiling purposes.

- **Client users:** They are the data subjects. They should be able to decide how, when, and to whom their data can be released.

- **Higgins Trust Framework:** This is the engine that will allow all parties to securely interact.

- **Web service context providers:** These reflect the way information is interpreted locally at the web service. It specifies also what kind of information or assertions is required to gain access to a given functionality offered by the web service, as well as the policies, if any, that govern the retention/disclosure of that data.

- **Client-side context providers:** It contains the different pseudonyms and credentials of the user, as well as his/her privacy preferences.

8.2.4 Evaluation

Higgins stands out clearly as a very ambitious project. Unlike server-based identity management tools, Higgins will be implemented as a unified and universal trust platform where users can manipulate their identities across multiple domains and contexts as well as keep control on that data. Unlike Microsoft’s
Passport, it is the user who negotiates and decides the disclosure of his/her information.

The other big winner from the Higgins project is going to be software developers. In fact, it happens often that different identity management applications (e.g., Shibboleth, YADIS or OpenID) or standards (e.g., LDAP, WS-Trust, Liberty Alliance, SAML) need to be integrated together. So far, developers had to learn the intricacies of each individual system/standard to be able to do that. With the Higgins framework, both identity data and interactions with identity services have been abstracted away, so that developers can build their own identity management applications without worrying about possible incompatibility with other applications.

With respect to privacy, however, Higgins scores a bit low. While it points out the potential privacy risk that may result from moving personal information back and forth across multiple domains, it remains silent on the way this risk is going to be controlled.

It is also worth to note that Higgins cannot patch possible vulnerabilities stemming from participating Identity management systems. In that sense, Higgins will be as secure and privacy-preserving as the least secure and privacy-preserving IMS participating in the framework. This possible lack of security may discourage some IMS vendors/users from participating in/using Higgins.

8.3 Liberty Alliance Project

The Liberty Alliance [?, ?] project proposes an architecture that consists of three big components:

- The Liberty Identity Federation Framework (ID-FF) proposes an approach for implementing single sign-on with federated identities that enables identity federation and management through features such as identity/account
linkage, simplified sign on, and simple session management.

- Liberty Identity Web Services Framework (ID-WSF) provides the framework for building interoperable identity services, permission based attribute sharing, identity service description and discovery, and the associated security profiles.

- Liberty Identity Services Interface Specifications (ID-SIS) proposes an architecture that enables interoperable identity services such as personal identity profile service, contact book service, geo-location service, presence service, etc.

In this overview, we describe the Liberty Identity Federation Framework (ID-FF), and give a short introduction of the functionalities of the Web Services Framework (ID-WSF) and the Identity Services Interface Specifications (ID-SIS).

8.3.1 The Liberty Identity Federation Framework: ID-FF

The Liberty Alliance is a consortium of business leaders with a vision to enable a networked world in which individuals and businesses can more easily conduct transactions. To accomplish its vision, the Liberty Alliance established an open standard for federated network identity through open technical specifications. In essence, this open standard is a structured version of the Security Assertions Markup Language, commonly referred to as SAML, with the goal of accelerating the deployment of standard-based single sign-on technology.

The Liberty Alliance specification has two main components: the Liberty Identity Provider (Liberty IDP) and the identity consumer, referred to as a Liberty Service Provider (SP). A Liberty IDP is the central credential store for a user’s identity information, and it is the heart of the user’s identity federations, or account linkage information. The Liberty IDP also serves as the authentication authority, which is viewed as a trusted identity store by the Liberty SPs. Liberty SPs are the Web sites that the user wants to connect to. A “circle of trust” is formed between Liberty IDPs and SPs to provide the user a secure infrastructure for controlling his or her identity information, and to facilitate Web single sign-on.

The Liberty Alliance specification defines a set of protocols that collectively provide a solution for identity federation management, cross-domain authentication, and session management. This specification also defines provider metadata schemas that may be used for making a priori arrangements between providers.

The Liberty architecture contains three actors: Principal, identity provider, and service provider. A Principal is an entity (for example, an end user) that has an identity provided by an identity provider. A service provider provides services to the Principal.

Once the Principal is authenticated to the identity provider, the identity provider can provide an authentication assertion to the Principal, who can present the assertion to the service provider. The Principal is then also authenticated to the service provider if the service provider trusts the assertion. An identity federation is said to exist between an identity provider and a service provider when the service provider accepts authentication assertions regarding a particular Principal from the identity provider. This specification defines a
protocol where the identity of the Principal can be federated between the identity provider and the service provider.

Requirements and Functionalities

One of the main requirements for Liberty Alliance is interoperability. The system is designed to function with a variety of web browsers, and the protocol specification provides support for various operating systems, programming languages and network infrastructures.

The functionalities provided by Liberty Alliance are:

- Identity federation
- Authentication
- Use of pseudonyms
- Single-Sign-On
- Global log-out

The security requirements imposed on the channels include confidentiality, data integrity, and optional per-entity authentication (required for identity providers but not for service providers). Message security requirements include transaction and message integrity, data origin authentication, non-repudiation of messages, and optional confidentiality.

Components

The overall Liberty architecture is composed of three orthogonal architectural components:

- Web Redirection: Action that enables Liberty-enabled entities to provide services via today’s user-agent-installed base. This technique carries security vulnerabilities, as the communication can be intercepted (often, no encryption is used), or the user agent could leak the information through, for example, the cache. The use of cookies, particularly persistent ones, also introduces security concerns and it is not recommended.

- Web Services: Protocol profiles that enable Liberty-enabled entities to directly communicate with each other.

- Metadata and Schemas: A common set of metadata and formats used by Liberty-enabled sites to communicate various provider-specific and other information.

Identity Federation and Single Sign-On

The first time that users use an identity provider to log in to a service provider they must be given the option of federating an existing local identity on the service provider with the identity provider login to preserve existing information under the single sign-on. It is critical that, in a system with multiple identity
providers and service providers, a mechanism exists by which users can be (at their discretion) uniquely identified across the providers.

An explicit trust relationship, or chain, is created with the opt-in identity federation that occurs the first time a user logs in to a service provider using an identity provider. While multiple identities can be federated to each other, an explicit link exists between each identity. Providers cannot skip over each other in the trust chain to request information on or services for a user because user identity information must be checked at each step. Therefore, the only requirement is that, when two elements of a trust chain communicate, they can differentiate users.

Members of the circle of trust are not required to provide the actual account identifier for a user and can instead provide a handle for a particular user. Members can also choose to create multiple handles for a particular user. However, identity providers must create a single handle for each service provider that has multiple Websites so that the handle can be resolved across the Websites.

Because both the identity provider and service provider in such a federation need to remember the other’s handle for the user, they create entries in their user directories for each other and note each other’s handle for the user.

For example, consider a user Bob who is registered with an identity provider IPA with pseudonym Bob771. Bob is known to two service providers SPA and SPB by pseudonyms B.RGS and Bobby, respectively.

When Bob wants to federate his identities at IPA and SPA, the identity and service providers create entries for the user of the type shown in Fig. 15:

When Bob wants to federate his IPA identity to a second Service Provider SPB, IPA will add a new alias and name for Bob for the SPB security domain. SPB will create an entry for Bob similar to the one indicated for SPA (with different name and alias), as indicated in Fig. 16.

Single sign-on is enabled once a user’s identity provider and service provider identities are federated. From a user’s perspective, single sign-on is realized when the user logs in to an identity provider and uses multiple affiliated service providers without having to sign on again. This convenience is accomplished by having federated the user’s local identities between the applicable identity providers and the service providers.

Figure 15: Federation of Bob’s identities at the identity provider IPA and the service provider SPA
8.3.2 Liberty Identity Web Services Framework: ID-WSF

The Liberty Identity Web Services Framework (ID-WSF) is a set of specifications for creating, using, and updating various aspects of identities.

ID-WSF consists of a number of distinct elements that together form a framework of web services. There are several types of system entities: Web Service Providers (WSP), which host web services such as a profile service, Web Service Consumers (WSC), which, with appropriate authentication and authorization, can access a user’s web services by communicating with the WSP’s endpoint (the targeted entity that contains the resource), and Discovery Service (DS), which is a web service typically hosted by an identity provider that enables a WSC to determine which WSP provides the needed service.

### Discovery Service

The Discovery Service presents an interface for consumers of identity services to locate resource offerings. Entities place resource offerings information describing the location of different types of information about Principals’ in a discovery resource. Thus the Discovery Service is essentially a web service interface for “discovery resources,” each of which can be viewed as a registry of resource offerings.

### Interaction Service

An identity service may sometimes need to interact with the owner of the resource that it is exposing, for example, to collect attribute values or to obtain permission to share the data with a Web Service Consumer. The Interaction Ser-
vice specification is an ID-WSF specification that defines schemas and profiles that enable a Web Service Provider to interact with the owner of the resource that is exposed by that WSP. The Interaction Service (IS) allows its clients (services) to indirectly query a resource owner for consent, authorization decisions, etc. By definition, the IS is capable of interacting with the Principal at any time, for example, by using special protocols, mechanisms, or channels such as instant messaging, WAP Push, etc. The IS accepts requests to present some information and questions to a Principal and is responsible for “rendering” a “form” to the Principal; to do so, the IS must know about the Principal’s capabilities and preferences. The IS returns the answer of the Principal in a response that contains the parameters and values of the request.

Data Services

Web services provide data services to computers and networked devices. A data service is a web service that supports the storage and update of specific data attributes regarding a Principal. The Liberty Personal Profile Service and the Liberty Data Service Template are two examples of data services; the Personal Profile Service provides profile information regarding a Principal while the Data Services Template provides protocols for querying and modifying data attributes while implementing a data service using ID-WSF.

8.3.3 Liberty Identity Services Interface Specifications: ID-SIS

Liberty Identity Services Interface Specifications (ID-SIS) proposes an architecture that enables interoperable identity services. Liberty has issued specifications for the following identity services:

- Personal Profile Service Specification: Describes a web service that provides a Principal’s basic profile information, such as their contact details, or name.
- Employee Profile Service Specification: Describes a web service that provides an Employee’s basic profile information, such as their contact details, or name.
- Contact Book Service Specification: Specifies a web identity service that allows a Principal to manage contacts for private and business acquaintances, friends, family members, and even for the Principal.
- Geolocation Service Specification: Specifies a web service offering geolocation information associated with a Principal.
- Presence Service Specification: Specifies a web service offering presence information associated with a Principal.

8.3.4 Evaluation of Liberty Alliance

Liberty Alliance’s security and privacy heavily rely on the trustworthiness of Identity Providers. Identity Providers know the set of Service Providers a user is interacting within the circle of trust, and has therefore the ability to profile
the user (in theory, the user still could have the alternative to have different identity providers for different service providers; but in practice an insufficient number of identity providers trusted by the service providers would eliminate this option). By creating many local identities with many service providers and/or identity providers and then federating them, users possess many sets of local credentials that may be used as a basis to authenticate with many service providers via single sign-on. This situation constitutes a risk. For example, every identity provider that possesses reusable user credentials (e.g., username and password) can impersonate the user at every service provider federated with that account.

Liberty Alliance avoids to a certain extent easy linkability of user pseudonyms in different security domains (through the use of ‘handles’). However, any two entities federated by the user are able to link the local pseudonyms of the user. In this respect, the Liberty Alliance model is extremely vulnerable towards malicious providers.
9 Conclusions and Future Work

In this deliverable, the basis for a future framework offering support for credentials such as the Belgian eID card and Idemix credentials is given. This not only involves using a credential, but as well the management of identities and pseudonyms; i.e. keeping track of what personal data has already been released to what party under what nym. As Privacy is an important issue in the framework, anonymous credentials are supported as wel. In case of abuse, dispute handling and deanonymization might be necessary and must thus be supported by the framework as well. Evidence need to be stored to be used in case of dispute. therefore, trusted archival is provided by the framework as well. Management of the credentials is part of the framework as well. A study of existing identity management systems is started as well. This will help us in comparing the future framework with functionality provided by these identity management systems as part of the validation.

Currently, no separation between the framework and the providers is made. Thus, no real framework is available as no separation with the providers offering the adaptors and implementations is made. This is the first task in part two of the framework development. This framework should fulfill the requirements given in section 4.2 and 5 as good as possible. The framework interfaces will be extended and modified based on the input from the validation. Crucial components such as the deanonymizer will be worked out and prototype implementations and adaptors will be provided. The more enhanced framework will be compared with existing identity management systems. Other identity management systems such as PRIME also need to be considered.
References


[27] L-SEC. eid beyond face value. from e-government tool to lever for growth. 2005.


