HELSINKI UNIVERSITY OF TECHNOLOGY
Faculty of Information and Natural Sciences
Department of Computer Science and Engineering

Long Nguyen Hoang

SECURE ROAMING WITH IDENTITY METASYSTEMS

Master’s Thesis
Espoo, June 30, 2008

Supervisors: Professor Antti Ylä-Jääski, Helsinki University of Technology
Peeter Laud, Ph.D., University of Tartu

Instructor: Pekka Laitinen, Nokia Research Center
<table>
<thead>
<tr>
<th>HELSINKI UNIVERSITY OF TECHNOLOGY</th>
<th>ABSTRACT OF MASTER'S THESIS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Faculty of Information and Natural Sciences</td>
<td>Degree Programme of Security and Mobile Computing</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Author</th>
<th>Date</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Long Nguyen Hoang</td>
<td>June 30, 2008</td>
<td>8 + 80</td>
</tr>
</tbody>
</table>

**Title of thesis**

Secure Roaming with Identity Metasystems

**Professorship**

Data Communications Software

**Professorship Code**

T-110

**Supervisors:** Professor Antti Ylä-Jääski, Helsinki University of Technology

Peeter Laud, Ph.D, University of Tartu

**Instructor:** Pekka Laitinen, Nokia Research Center

The notion of identity metasystem has been introduced as the means to ensure inter-operability among different identity systems while providing a consistent user experience. Current identity metasystems provide limited support for secure roaming: by "roaming" we refer to the ability of an user to use the same set of identities and credentials across different terminals. We argue that in order to support different types of roaming, the identity selector system should be structured as a set of distributable components. We describe such distributed client-side software architecture and how that architecture is implemented by adapting Novell’s Bandit project. Our implementation shows how added security assurances can be gained from the fact that credentials are stored in one trusted device in the form of a mobile phone but can be used on less trusted terminals in the form of PCs. The main goal of this thesis is to provide an abstract mechanism for digital identity roaming, allowing end users to manage and use their digital identities with the power of identity metasystem while maximizing the portability and usability of the identity metasystem using mobile trusted devices.

**Keywords:** Identity metasystem, Mobility, Roaming Management, Design, Security, Human Factors
Acknowledgements

I am grateful to all those who gave me the possibility to complete this thesis. I would like to express my sincere gratitude to professor Antti Ylä-Jääski in Helsinki University of Technology and professor Peeter Laud in University of Tartu for giving me the permission to commence this thesis. I had the pleasure of being an intern at Nokia Research Center in Helsinki during the first half of 2008, working with the same topic with Pekka Laitinen and N.Asokan. I am indebted to them for their great support during my studies.

I want to sincerely thank all of my friends for their unflinching encouragement in various ways.

Especially, I would like to give my special thanks to my parents whose patient love enabled me to complete this work.

Espoo 2008

Long Nguyen Hoang
## Abbreviations and Acronyms

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>IdP</td>
<td>Identity Provider</td>
</tr>
<tr>
<td>ISS</td>
<td>Identity Selector System</td>
</tr>
<tr>
<td>OS</td>
<td>Operating System</td>
</tr>
<tr>
<td>PKI</td>
<td>Public Key Infrastructure</td>
</tr>
<tr>
<td>P-STS</td>
<td>Portable Security Token Service</td>
</tr>
<tr>
<td>PTD</td>
<td>Personal Trusted Device</td>
</tr>
<tr>
<td>RP</td>
<td>Relying party</td>
</tr>
<tr>
<td>SOAP</td>
<td>Simple Object Access Protocol</td>
</tr>
<tr>
<td>SSO</td>
<td>Single Sign-on</td>
</tr>
<tr>
<td>STS</td>
<td>Security Token Service</td>
</tr>
<tr>
<td>URL</td>
<td>Uniform Resource Locator</td>
</tr>
<tr>
<td>URI</td>
<td>Uniform Resource Identifier</td>
</tr>
</tbody>
</table>
Contents

Abbreviations and Acronyms iv

1 Introduction 1

2 Background Study and Problem Statement 3
  2.1 Identity Metasystem ................................. 3
    2.1.1 Principles of Identity Metasystem ............... 3
    2.1.2 How Identity Metasystem Works ................. 6
    2.1.3 Implementations of the Identity Metasystem .. 9
  2.2 Identity Technologies ............................... 13
    2.2.1 Liberty ........................................ 13
    2.2.2 SAML 2.0 ...................................... 14
    2.2.3 OpenID 2.0 .................................... 15
    2.2.4 Technology Convergence ......................... 17
  2.3 Security Considerations in Identity Metasystem .... 19
  2.4 Problem Statement .................................. 21

3 System Design 24
  3.1 System Requirements ................................. 24
  3.2 Design Options ...................................... 24
  3.3 The Delegated-Card Approach ......................... 25
  3.4 Identity Metasystem as Distributed Components .... 26
    3.4.1 System Architecture ............................ 26
3.4.2 Distribution of Identity Components ............... 31
3.4.3 Middleware Architecture ............................ 38

4 Implementation ............................................. 45
   4.1 Brief description of work flow ....................... 45
   4.2 Technical Issues .................................... 47
   4.3 Adapting Bandit Implementation .................... 50
   4.4 Implementation Details ............................. 53
       4.4.1 Use cases .................................... 53
       4.4.2 Class diagram ................................ 58
       4.4.3 Package diagram .............................. 59
   4.5 Application ........................................... 59

5 Discussion .................................................. 63
   5.1 System Evaluation .................................. 63
       5.1.1 Known Limitations ............................ 64
       5.1.2 Performance Analysis .......................... 64
   5.2 Potentials of Identity Metasystem ................... 65

6 Conclusions and Future Work ............................ 69

Appendices .................................................. 75
   A.1 Middleware Request/Response Format ............... 75
   A.2 Delegated Card Format .............................. 76
   A.3 Operation Specifications in VDM++ .................. 77
   A.4 Application Package ............................... 80
List of Figures

2.1  Identity metasystem authentication process  .................................. 7
2.2  Request for Security Token Response with proof key  ..................... 8
2.3  Microsoft CardSpace architecture [11]  ..................................... 10
2.4  Novell Bandit architecture [31]  ............................................. 12
2.5  Liberty Alliance architecture  ................................................. 13
2.6  Liberty example scenario  ..................................................... 14
2.7  OpenID protocol flow  .......................................................... 16
2.8  Relationship between technology standards  ................................ 18
2.9  Dimension of identity metasystem[29]  .................................... 22
2.10 Basic user experience  .......................................................... 23

3.1  System components and interfaces  .......................................... 27
3.2  Configuration 0 - all in terminal  .......................................... 32
3.3  Configuration 1 - mobile storage  .......................................... 33
3.4  Configuration 2 - mobile identity provider  ................................ 34
3.5  Configuration 3 - mobile identity selector  ................................ 35
3.6  Configuration 4 - all in trusted device  .................................... 36
3.7  Configuration 5 - external service operator  ................................ 37
3.8  Middleware High-level Architecture  ........................................ 39
3.9  Connection set up in active mode  .......................................... 42
3.10 Connection set up in listen mode  .......................................... 43
3.11 Communication between two identity components in two devices  ...... 44
3.12 Communication between two identity components within one device ........................................ 44
4.1 Interaction between identity components ................................................................. 46
4.2 Bandit implementation .................................................................................................. 50
4.3 Sequence diagram of middleware message flow ....................................................... 52
4.4 Use case diagram ......................................................................................................... 53
4.5 Class diagram ............................................................................................................... 58
4.6 Package diagram ........................................................................................................... 59
4.7 Deployment diagram .................................................................................................... 60
4.8 DigitalMe2J2ME application ....................................................................................... 60
4.9 Pairing between terminal and phone ........................................................................... 61
4.10 Issue token .................................................................................................................. 61
4.11 Device disconnected .................................................................................................... 62
Chapter 1

Introduction

As more and more transactions are carried out over the Internet, the need for easy-to-use and secure authentication mechanisms becomes increasingly evident. Now that the web is used for everyday tasks, from shopping, banking, and paying bills to consuming media and entertainment, people must keep track of multiple accounts/passwords and various methods of authenticating to sites. This results not only in user frustration, known as "password fatigue", but also insecure practices such as reusing the same account names and passwords at many sites. The root of these problems is that the Internet was designed without a system of digital identity in mind. In efforts to address this deficiency, numerous digital identity systems have been introduced to save the users from having to create, manage, and use various passwords for different service providers. But no one single system meets the needs of every digital identity scenario. Furthermore, each of them uses its own protocols and requires incompatible types of user interaction.

More recently, the notion of Identity Metasystem [19, 20] has gained ground. The main purpose of the identity metasystem is to put an abstract identity management layer on the Internet to allow existing identity systems based on various technologies to inter-operate with each other while providing a consistent user experience regardless of which identity system is used. The identity metasystem introduces the concept of information card modeled after a business card, driver license, ID badge, etc. An information card, in general, can be seen as a digital representation of user identity. User can obtain an information card from an identity provider (managed card) or create it by himself (self-issued card). Each information card describes its owner’s selected identity attributes like name, age, email address, credit card number, or membership number. However, the actual values of identity
CHAPTER 1. INTRODUCTION

Information are not in the card (unless it is self-issued) or even in user PC but they are kept by an identity provider specified in the card, VeriSign or local bank for example. Whenever the user accesses a site that requires authentication, the identity agent in the local system show a consistent user interface to let the user to select one of information cards he has. Once an information card is selected, it retrieves identity information from the identity provider and passes it to the site the user wants to access.

From end-user’s point of view, the innovation provided by the identity metasystem is the metaphor of information card. However, it leads to the problems concerning digital identity which is called identity roaming across security domains. More specifically, the question is how users can safely carry and use their information cards anywhere even in untrusted terminals such as some kiosk PCs. At the time this thesis was written, the most well-known implementations of identity metasystem, namely Microsoft CardSpace [11], Higgins [40], and Bandit project [31], are unable to support roaming. The main reason is their low portability. They both allow the possibility to export information cards from one device and import them into another but that results in the transfer of the entire information cards, including the secret credentials if those cards are self-issued. There are research studies [32, 23, 3] about how to protect authentication transaction on a less trusted device but they propose their own protocols or frameworks, which makes it difficult to interoperable with other identity systems.

The main goal of this thesis is to provide an abstract mechanism for digital identity roaming, allowing end users to manage and use their digital identities with the power of identity metasystem while maximizing the portability and usability of the identity metasystem using mobile trusted devices. Our system can be seen as a way of safely organizing digital identities into a "digital wallet". The result, in simplest saying, is that the roaming of digital identity is possible.

The following outlines the remaining chapters that form parts of this thesis. Chapter 2 studies the principals of identity metasystems and specifies the scope of our research. Chapter 3 focuses on system design. Based on the design, Chapter 4 describes technical implementation details on how Bandit identity metasystem has been adapted in order to make secure roaming possible. System evaluation is done in Chapter 5 which is followed by discussion of current issues in identity metasystem in general. Finally, conclusions are presented in Chapter 6.
Chapter 2

Background Study and Problem Statement

2.1 Identity Metasystem

2.1.1 Principles of Identity Metasystem

One of the problems with existing identity systems, even with very large scale ones, is that they are built upon the assumption of managing a single domain of influence. To solve the issue of cross-domain identity, they tend to choose one-to-one communication solutions instead of networking on top of an abstract identity management layer. It leads to increasingly complicated projects with very complex and detailed implementations for each particularized system to understand the others. The result is that those systems become more and more tightly coupled to each other. The identity metasystem was designed to address this problem. The identity metasystem is conceptually defined as follow:

"The Identity Metasystem is an interoperable architecture for digital identity that assumes people will have several digital identities based on multiple underlying technologies, implementations, and providers. Using this approach, customers will be able to continue to use their existing identity infrastructure investments, choose the identity technology that works best for them, and more easily migrate from old technologies to new technologies without sacrificing interoperability with others." [19]
CHAPTER 2. BACKGROUND STUDY AND PROBLEM STATEMENT

There is no definitive identity metasystem architecture, implementations of identity metasystem in general are expected to follow the principles set out by the Laws of Identity [24]. The Laws are stated as follows:

1. **User Control and Consent:** *Technical identity systems must only reveal information identifying a user with the user's consent.* This means the user must be informed (visually or implicitly) whenever her identity information is acquired. The user must have the ability not only to decide which parts of information to be revealed but also to be aware of the purposes for which any information is being collected or to verify the identity of the parties asking for information as well. In other words, the system must be designed to put the user in control of his/her digital identities and information being released [4].

2. **Minimal Disclosure for a Constrained Use:** *The solution which discloses the least amount of identifying information and best limits its use is the most stable long term solution.* In a given context, identity systems do not disclose more information than necessary, and they use identifiers that are designed specifically for that context. For example to verify whether one is mature enough to buy wine, it's better to prove that he is "age>21" than showing his birthday. Another example is that, to identify an user, the local library requires its own unique identifier instead of his Social Security number. This is the most important constraint in the Laws. Its purpose is to minimize identity information so that it become less attractive to identity attacks.

3. **Justifiable Parties:** *Digital identity systems must be designed so the disclosure of identifying information is limited to parties having a necessary and justifiable place in a given identity relationship.* This is an extension of the first rule User Control and Consent. Depending on the context, an user must be able to select among capable parties disclosing identity information. For example, it is possible to show passport or driving license to prove one's identity to a government department but it's wiser to show business card to a email web site rather than the passport. It is similar in digital justifiable parties. Every party acquiring identity information must state in advance the respective disclosing party(ies) in a policy statement which clearly describe what happens to disclosed information.

4. **Directed Identity:** *A universal identity system must support both "omnidirectional" identifiers for use by public entities and "unidirectional" identifiers for use by private entities, thus facilitating discov-
very while preventing unnecessary release of correlation handles. This comes from an analogy with the physical world, a subject possesses one or multiple identities and they can be either public or only available in a certain community. Entities that are public and willing to reveal their existence are said to have omni-directional identifiers while ones that show no correlation handle being shared between sites to assemble profile activities and preferences into super-dossiers are said to have unidirectional identifiers. The point is to maximize the reachability and discovery of the system.

5. **Pluralism of Operators and Technologies**: A universal identity system must channel and enable the inter-working of multiple identity technologies run by multiple identity providers. The identity metasystem provides a consistent user interface but it has to deal with different operators with different technologies being used. This laws implies that the universal identity metasystem itself must not be a standalone identity system but a result of gentle cooperation between identity systems running by different operators with different technologies. However, in order to enable that, we need a simple encapsulating protocol and a way to display different information semantics through a unified user experience.

6. **Human Integration**: The universal identity metasystem must define the human user to be a component of the distributed system integrated through unambiguous human-machine communication mechanisms offering protection against identity attacks. The meaning of this law is that the system in design must take into account the human factor. It must guarantee its reliability as much as possible in the communication with its human users. Its behavior must also be predictable. System communication is not just between cyber systems but includes human users.

7. **Consistent Experience Across Contexts**: The unifying identity metasystem must guarantee its users a simple, consistent experience while enabling separation of contexts through multiple operators and technologies. It means that the process done by users on any context must be consistent and predictable. Typically, digital identities can be visually translated into icons on the desktop, making it easy for the users to manage/select/use their digital identities and to control what information is released.
2.1.2 How Identity Metasystem Works

The concept behind identity metasystem is relatively simple; it is based on the identification process we experience in the real world when using physical identification cards. In general, an information card is a digital representation of user identity modeled after business card, driver license, ID badge, etc. It can be either a managed card or a self-issued card. Managed cards are issued by trusted authorities and they contain only metadata describing how to get security tokens. Secret credentials needed when generating authentication token are managed by the authority. Self-issued cards, on the other hand, are managed by user with the help of the identity agent installed on the local machine. Unlike managed cards, self-issued cards contain both meta-data and credential information that can be used to generate security tokens.

On top of the information-card concept, the identity metasystem defines a model [28] where different parties can participate in with one or many roles. Any system capable of authenticating users based on information card technology is called a relying party (RP). It can be a program, a web server, or any other service that requires its users to prove their identities before allowing them to access its resource. The second role is identity provider (IdP) which issues information cards and provides authentication service. In user machine there is a user identity agent called identity selector system (ISS) that provides a secure, consistent mechanism for managing information cards, and operates as a self-issued IdP.

Figure 2.1 describes a simple authentication process in the identity metasystem. In order to use information card technology the user must first enroll with an IdP, which is the out-of-band step shown by a dashed line in the figure. It could be as simple as creating a self-issued card on user’s own machine using the identity selector’s GUI, in which case the IdP is right on the local system. Alternatively, the user might obtain an information card from a managed IdP via any protocol such as file transfer or email. When the user runs an application that makes a request of an information card-enabled service, i.e. RP, (1), the application invokes the local ISS with the security policy retrieved from the RP (2,3). The ISS shows the user a secure user interface with a collection of cards that satisfy the type of token and the type of claims stated in the RP’s policy (4,5). If the user has never authenticated towards this RP before, the ISS will display information such as the RP’s certificate, its issuer authority, logos to give the user a chance to visually verify that he’s talking to the right service, not to an imposter.
Figure 2.1: Identity metasystem authentication process

Once the user selects a card to send (6), the identity selector initiates a behind-the-scene conversation with the IdP specified in the card. First there’s a metadata exchange phase (7, 8), where the IdP’s security policy is discovered so that the identity selector knows how the user is supposed to authenticate himself to the IdP to get the security token for the RP. Depending on the IdP’s policy, the user can be authenticated in several ways including a password, a certificate, Kerberos [36] credentials, or a self-issued information card. For example, if the provider requires a password, the identity selector will prompt the user for his password. Then the identity selector composes a
request to the IdP (11) which comprises the user’s credential (the password in this case), the identifier of the selected card for the provider to recognize which account is being accessed, and a list of requested claims.

Assuming the user’s password included is valid, the IdP constructs a security token with all the claims requested by the service. The token is then signed and encrypted before it is sent back to the identity selector (12). Upon receiving the token, the identity selector decrypts the token and displays identity data included in it. This allows the user to see exactly what identity data, age for example, is going to be sent to the RP. If the user does not give his final confirmation by pressing ’Send’, the whole authentication process is terminated. Otherwise, the identity selector gives the token to the application to conclude the authentication with the RP (15). The RP now gets the claims it requested and uses those claims to make authorization decisions (16, 17).

Figure 2.2: Request for Security Token Response with proof key

To protect the token from being captured on-the-wire and to ensure that the client application has obtained a legitimate token from the IdP (11, 12, 15, 16), the identity metasystem offers an extra protection mechanism known as proof of possession. The use of proof of possession gives superior guarantees for demonstrating evidence of the sender’s knowledge of information that should only be known to the sender of the security token. If a man in the middle would be able to capture the token, this fact alone would not allow him to perform any authenticated calls to the RP unless he could break the
secret of the proof-of-possession. Technically, the key used to demonstrate
the sender’s knowledge is called the \textit{proof key}. A proof key used can be
symmetric, asymmetric or non-proof-key (i.e. the token with no proof key
attached), and its type is specified in RP’s policy. For better security, the use
of asymmetric proof keys is strongly recommended. Figure 2.2 demonstrates
the outline of the responses from the IdP in which symmetric/asymmetric
proof keys are used respectively.

If the proof key is required to be symmetric, it is derived from entropies
contributed by the client and/or the IdP (step 11 and 12 in Figure 2.1).
The client first sends its entropy to the IdP. If that entropy is accepted, it
is used as the negotiated proof key and part A of the response is skipped.
Otherwise, the IdP calculates a new proof key, encrypts it with the client’s
public key and puts the result to part A so that the client knows about the
new proof key. After that, the token and the negotiated proof key are signed
by the IdP forming part B of the response (step 15). Once receiving the IdP
response, the client can forward part B signed with the proof key to the RP.
Using this protocol the RP can trust the message because the proof key and
the token are signed by the IdP and the secret of proof key is unknown to
anyone else than the client.

If the proof key is required to be asymmetric, a asymmetric keypair is gen-
erated by the client and the public part is included in the request (step 11).
The IdP, using its private key, signs the token together with the attached
public part of the proof key. Part A is skipped because it is unnecessary.
After that the whole message is sent back to the client (step 12). Similarly,
the client can prove its knowledge of possession by signing part B with the
private key it has kept for itself (step 15).

\subsection*{2.1.3 Implementations of the Identity Metasystem}

In the identity metasystem, one party can take one or many different roles
due to its business needs. Some parties choose to go into the IdP business
while others provide certification services for identities. Some implement
RP or IdP server software whereas others implement client software. More-
ever, device manufacturers and mobile telephone vendors can host identities
on their platforms. By introducing a set of specifications \cite{28}, the iden-
tity metasystem allows implementations of its components to independently
inter-operate. Many projects have been set up to realize those specifications
including xmlidap \cite{25} and Ping Identity \cite{33}. Here the three most famous
representatives of implementations of identity metasystem are introduced.
Overview of Microsoft CardSpace

Microsoft CardSpace[11] is the first commercial implementation of the identity metasystem concept and the one with the most widespread identity selector since it is shipped as part of Windows Vista platform. Figure 2.3 depicts the system architecture of CardSpace. Services or applications on the platform trigger the CardSpace system via activators. The core component, called ISS, handles all management requests and messages, and invokes system services such as secure user interface or low-level storage. Compared to other identity selectors, CardSpace offers unique security features such as secure desktop to prevent the interception of malicious processes during authentication phase or double-encrypted local storage that can only be opened by corresponding user account. Unfortunately, CardSpace is a closed-source product and the APIs provided are really limited.

![CardSpace Architecture Diagram](image)

Figure 2.3: Microsoft CardSpace architecture [11]
Overview of Higgins framework

Higgins[40] is an open source Internet identity framework designed to integrate identity, profile, and social relationship information across multiple sites, applications, and devices. Higgins itself is not a protocol, it is software infrastructure with identity components that works with all popular digital identity protocols and contexts. Because Higgins provides the means to make the identity metasystem cross-platform, Higgins is usually said to be an implementation of identity metasystem although its scope is much larger.

Technically, Higgins focuses on three types of identity solutions. First, it implements the Higgins Global Graph (HGG) data model. The Higgins Global Graph is a foundation for achieving portability, interoperability and unification for identity-related data. The main purpose is to build a common model for social network on the Internet including identity, profile, preference and social relationship data about people, things or concepts. On top of that, Higgins implements the Higgins Identity Attribute Service (IdAS) to combine identity and social network data in different formats across data sources such as directories, relational databases, and social networks. Using adapters called Context Providers, IdAS can be extended to connect various systems or data stores. Second, Higgins provides implementations of identity web services. In particular they are implementations of identity providers and relying parities. Finally, Higgins provides implementations of multi-platform identity selectors that bring consistent and secure user experience to authenticate to services using information-card technology.

By unifying all identity interactions regardless of protocol and data format under a common model, Higgins makes it easier to integrate identity information scattered over across multiple systems. In summary, Higgins constitutes an excellent open-source platform for further development of identity metasystem.

Overview of Bandit project

Bandit[31] is an open-source, cross-platform project of the identity metasystem concept. The Bandit project has a very strong relationship with Higgins in a way that some Bandit objectives and Higgins objectives are highly identical, some Bandit components make use of Higgins components to accomplish their goals. In general, Bandit provide implementation identity selector and a common framework for easily integrating any digital identity management system. Figure 2.4 shows the high-level system architecture of Bandit. On the top is the management user interface, called DigitalMe. The main mod-
ule ISS handles all requests and operates as an identity selector. It heavily makes use of Higgins IdAS architecture to ease the communication with different identity providers. Also multiple Store Providers are implemented to read and write identity data from local or remote data sources such as file system, Bluetooth, relational database management system (RDBMS).

In summary, Bandit project, built on open source, contributes to the development of other community partners and projects to develop an open identity framework that simplifies the creation of identity-enabled applications.

![Novell Bandit architecture](image)

Figure 2.4: Novell Bandit architecture [31]
2.2 Identity Technologies

2.2.1 Liberty

Liberty Alliance is a project aiming to build open standard-based specifications for federated identity, to provide interoperability testing and solutions to identity theft. Liberty specifications are divided into three frameworks (Figure 2.5). The first phase, referred to as Identity Federation Framework (ID-FF [21]), is to enable federated network identity management and mostly known for single sign-on and linking accounts in the set of service providers in the boundary of the trust circle. The second phase, referred to as Identity Web Services Framework (ID-WSF [22]), extends the Phase 1 and provides an open, standards-based platform for delivering identity-based web services that can exchange identity attributes and offers personalized security services. However, in Liberty's latest version, ID-FF becomes obsolete and gets replaced by SAML 2.0. The third phase called Service Interface Specifications (ID-SIS) is in the process of developing, which aims to define concrete XML schemas for representing identity data used in ID-WSF applications such as personal profile, employee and contact information.

Figure 2.5: Liberty Alliance architecture
Figure 2.6: Liberty example scenario

Figure 2.6 demonstrates a sample case of how Liberty works. In the scenario, the Identity Provider (IDP) provides authentication service for the service provider (SP) with the help of SAML 2.0 mechanisms. After the authentication has been done, the user can use services provided by web service providers (WSPs) without having to re-login. The role of the Discovery Service (DS) in ID-WSF framework is to maintain references to services that the user has registered to on behalf of the user. In this example, the service provider queries user’s Discovery Service to locate the GeoLoc service. Using this model, ID-WSF services can cooperate in business processes to offer personalized services even when the user is not online. Liberty’s open specifications especially ID-WSF with its flexible security model receive a wide scale deployment from vendors and service providers because they offers solutions not only from technical aspect but also from business aspect (through best practices and business guidelines).

2.2.2 SAML 2.0

Security Assertion Markup Language (SAML) [30] developed by Oasis Security Services Technical Committee is an XML standard for exchanging authentication and authorization data between security domains. The most important problem that SAML is trying to solve is single sign-on where a
user who has signed on to one service can move directly to another without the need to log on again. To make it become possible, SAML defines standards essentially in three levels of services, each of which builds on top of the earlier. First, at its core, SAML defines mechanisms for user assertions which are about a format for security token. Second, it provides a number of protocol profiles and bindings for requesting and managing assertions. Finally, a metadata format for for exchanging service description metadata is formalized. The latest version of SAML is 2.0, in which makes use of a lot of work done by Liberty ID-FF and Shibboleth [8, 9].

SAML itself does not include notions of identity providers, relying parties like in identity metasystem; rather, it provides profiles describing many kinds of interactions that can occur when using the same protocols. SAML assertions and protocols describe only a method for exchanging information on authentication and authorization and identity information. Therefore, it is sometimes criticized for its heaviness - especially in terms of the complexity of specifications and the constraint of security rules. It is because SAML aims to support a variety of sign-on scenarios, many of which are largely identical to the objectives of the identity metasystem and Liberty ID-FF. The most well-known profile specified by SAML is the Browser-based single sign-on profile. The idea behind the Browser-based single sign-on profile is relatively close to that of identity metasystem, user signs on to an identity provider using whatever method the identity provider supports, which could be username and password, smartcards or anything else. After authenticating to the identity provider, the user receives an assertion that she can use to sign on to any service providers.

2.2.3 OpenID 2.0

OpenID[34] is a shared identity service which allows users to log on to many different web sites using a single digital identity. Unlike Liberty and the identity metasystem, OpenID takes a different approach: it provides users with unique URIs that serve as personal identifiers. These URIs may be used to login to diverse relying parties (RPs). Figure 2.7 illustrates the basic workflow of OpenID authentication process.

The process of logging an user into an external site (or RP) has seven key steps. User first provides an Identifier to the RP (1) using OpenID technology. This usually takes the form of a text box asking the user to enter an OpenID identifier which can be either an URI/URL or an XRI. Upon receiving the identifier from the user, the RP cleans up that identifier to
retrieve a normalized identifier URL or a normalized identifier XRI (2). That normalized identifier is used for discovering the OpenID provider (3). Then the RP and the OpenID provider may establish a security association in which a shared secret-key is exchanged if necessary (4). After that, the user is redirected to the OpenID provider’s interface to login with his registered account (5). If the credentials provided by the user is successfully verified by the OpenID provider, the OpenID provider redirects the user back to the RP with cryptographic proofs that the user claims the identifier and any profile data that the user choose to release. The RP examines the response and
CHAPTER 2. BACKGROUND STUDY AND PROBLEM STATEMENT

does the final verification (6) to decide whether the user is authenticated to
it (7).

OpenID's key distinctions are its lightweight protocol and its simple trust
model. OpenID is designed to provide portable, user-centric security services
without requiring any special capabilities of the user agent or other client
software. As a result, user does not need to carry any special device or token
or store any private data on the machine, everything is done in the browser.
In version 2.0, OpenID supports both public identifiers and privacy-protected
(unidirectional) identifiers (user provides merely the address of IdP and not a
URL that uniquely identifies her). Users are also allowed to switch OpenID
providers with their Identifiers can be preserved. These strong points make
OpenID the technology that draws most interest from service providers on
the Internet.

However, OpenID still has some disadvantages. One of them is that user
must keep track of all of her identifiers. Another problem is with usability,
typing in an OpenID is not very friendly for the average web user. Besides,
its simple protocol which is not based on trust model is vulnerable to several
phishing attacks [7].

2.2.4 Technology Convergence

The vision of federated identity management has attracted interest from
different communities. Therefore, there are different approaches to solve the
issue. The identity metasystem is designed to leverage Web Service standards
(SOAP [15], WS-Trust[26], WS-Mex[12] and WS-SecureConversation[27]) to
inter-connect individual identity systems while other approaches mentioned
above publish their own solutions. Despite of very different structures, they
share some common objectives. However, the fact that one system tries
to reach complete security solutions makes it more difficult to work with
other identity systems because of their overlapped domains. For example,
one way to obtain a security token is to use the mechanisms from ID-WSF
Identity Provider. Another possibility is to the WS-Trust [26] specification
and a security token service as described in Section 2.1.2. Currently, Web
Service standards uses their own protocols and metadata formats which are
not compliant with those from SAML 2.0 specifications. It is similar with
Liberty. OpenID is more likely an island compared to other technologies for
the reason that its protocol is too simple and lightweight to support heavy
SOAP-message processing.

With the variety of identity management systems, it is obviously necessary
to integrate them. In approach for interoperability, Liberty has claimed to endorse SAML 2.0 as its identity federation solution. More specifically, ID-WSF 2.0 is based on SAML 2.0 rather than Liberty ID-FF specifications due to the fact that SAML 2.0 mostly likely replaces Liberty ID-FF. Recently, the Bandit project has developed an open source integration system between Liberty and CardSpace and OpenID. Although the source code is provided, it seems that there is no published specification of the integration system, which makes it difficult to discover exactly how the Bandit scheme works. There are also few research studies [2] but only one-to-one mapping solutions are proposed. In addition, there is attempt for creating a OpenID Information Cards draft 1.0 [16].

![Relationship between technology standards](image)

Figure 2.8: Relationship between technology standards

About the identity metasystem, there are endless discussions about whether it should embrace other protocols than the Web Service stack being used. With respect to a single, dominant protocol, one could argue that it is better to have a common identity infrastructure as a new identity management layer of the Internet. The infrastructure with a standardized protocol would allow new existing and new technologies to be easily plugged in, bringing increased value with each additional participant. The more protocols that the infrastructure tries to support, the more vulnerable it is to attack. Therefore, it is wiser to concentrate resources in strengthen a single protocol than fixing all the weak points of the supported protocols, not including the vulnerabilities caused during the integration.
CHAPTER 2. BACKGROUND STUDY AND PROBLEM STATEMENT

On the contrary, one could argue that, for security reasons, the identity infrastructure should support multiple, competing protocols for interconnection flows as this would allow multiple lines of defenses in case one protocol is attacked. In other words, interoperability leads to increased security as it brings reinforcements. Besides, it would afford flexibility and openness leading to increasingly better products being offered to the user.

No matter what argument is presented, it would make sense to note that the infrastructure should be open to participants and put user in the center of identity management. The use of Web Service stacks in current design of identity metasystem does not mean that they are the best or the only option, but that it they are increasingly available as a very viable option when we need protocol-neutrality, security, identity, management capability, and so on.

2.3 Security Considerations in Identity Metasystem

Compared to other identity systems mentioned in this chapter, the identity metasystem offers many distinct advantages such as phishing prevention and being technology neutral. However, it still has some drawbacks. Those advantages/disadvantages can be considered in different aspects.

- **User experience** (advantage): The most innovative features offered by the identity metasystem are the simple and consistent user interface with highly secure environment and the authentication metaphor *information card* which frees users from having to memorize username/password combination(s) for each service.

- **User acceptance** (disadvantage): The new authentication metaphor, *information card*, however, may not be easily accepted. Hundreds of millions of people have been taught to type their names, secret passwords, and personal identifying information into almost any input form that appears on their screen. The concept of username/password has been used for years, which makes it difficult to persuade users to switch to a new authentication method although it is easier to use and more secure.

- **Interaction between parties** (advantage/disadvantage): the main contribution of the identity metasystem is that it puts an abstract
identity management layer on the Internet where identity systems can inter-operate with each other. As a result, a trust network is built not only between well-known parties. Parties have to trust each other to interoperate. This is both a advantage and a drawback for the reason that trust in digital identity is not easy to be build. If a trust network is established between RPs and IdPs, the identity metasystem can significantly help reduce the overheads of duplicated identity information thus efficiently decrease the cost of maintaining multiple identity information repositories. However, if it can not be done, the identity metasystem is then only used as a common protocol for isolated identity systems to talk to each other, which is not new to us. Current sites still keep their own customer database and their own authorization logic; it’s hard to tell them to shift to a new infrastructure which may be impossible in short term.

- **Trustworthiness and reliability of parties** (advantage/disadvantage): The trust network also raises an issue about the trustworthiness and reliability of parties. User credential can be misused by relying parties and identity providers and identity selectors. Users must trust them to keep their identity information. Should the attacker breaks into the identity provider, the security of the whole system will be compromised. For example, he can easily issue any security token and access to any related relying parties. A malicious identity selector, when deployed in an terminal unknown to user, can inveigle the user into giving all his credentials including all information cards and PIN if possible. The result is that the attacker can learn user credentials and try to impersonate later. The existence of malicious identity selectors is unavoidable because identity management is pushed to client on behalf of the user for the need of a consistent user experience and each instance of identity selector must be installed in every local machines, which is hard for maintaining and administrating.

Fortunately, there are some fixes/solutions for the problem above. The need for initial trust between the user and the RP can be avoided by using zero knowledge proofs. More specifically, when user registers to the RP, he does not have to reveal personal information to the RP but demonstrate knowledge of that information [1]. Attacks to identity providers are considerably infeasible. Last of all, attacks to malicious identity selector can be thwarted by using personal trusted devices as an additional authentication method. The idea is to store secret credential information needed to generate security tokens on the devices so that the credentials can be safely used on less trusted terminals.
2.4 Problem Statement

One of the problems with current implementations of the identity metasystem is their portability. In order to use the same information cards on multiple machines, the end user has to export his information cards to a file and then import it into the system on another machine. Although import/export is conceptually simple, it involves moving all data associated with the card, including sensitive data like credential secrets of self-issued cards. The user has to remember to remove all the cards including the secret data from the machine before she leaves. Moreover, even if she remembers to do that, her cards and credentials could be copied without her consent during the session.

Obviously, user credentials themselves should not be persistently stored in untrusted terminals. The question is whether it's better to keep the those credentials in the network or in mobile trusted devices. On one hand, a trusted device is the physical hardware that the user personally controls, it can represent its owner directly. It is crucial that the trusted device itself is considerably secure because it provides the security service. The downside to this is that people would loose their identity information should the device is lost. Mobile phones with their increasing capability are preferred as they have interactive function, ability to process heavy request, and large storage space which smartcard does not have. The network approach, on the other hand, provides ubiquitous solutions in which user credentials can be accessed anywhere at anytime. However, it requires connectivity and additional authentication to the remote credential repository which might limit usability. Most average users tend to trust their phones to keep their private information rather than putting it in a network server because user credentials stored on trusted devices give better privacy and usability and they can be transferred offline or peer-to-peer in many ways.In this thesis, user trusted devices (PTDs) in the form of mobile phones are used to store user credential.

Fundamentally, implementations of the identity metasystem can be appraised in seven different aspects: Card Store, Credential Store, UI Generation, Security Token Service, STS Authentication, Identity Selector, Browser (Figure 2.9). Here the aspect "STS authentication" denotes the technology used to authenticate to other STSs, while the other axes correspond to the locations of different stores or functionalities. The figure gives a comparison of state-of-the-art implementation of the identity metasystem and the "portable" identity metasystem (marked with blue line). Let us envision the user experience in our envisioned system. Initially, the user connects his PTD to a terminal and his information cards stored on the PTD becomes available on the terminal identity selector. The terminals can be trusted or untrusted. It is
possible that the user walks away from the client terminal during an active session, and in that case his information cards disappear, i.e. nobody else can take his session and send access requests on his behalf. In other words, the user is able to safely provision his credentials/identities even on untrusted terminals using his PTD.

Figure 2.9: Dimension of identity metasystem[29]

The real authentication process is started when a user is asked to prove his identity (i.e. possession of an authentication token) when going to a RP on the terminal (described in Figure 2.10). The user first selects one of his information cards on the terminal then confirms on the PTD. Assuming that the user has assigned and mapped a private RP identifier in the form of an image or company logo to the RP identifier in the certificate, the device can display the image on the screen when the certificate has been verified. Since the user chose the image in the first place, one can assume that the user is able to recognize the same image when authenticating the SP at a later stage. It is noted this model effectively eliminates the phishing attack threat. Even if the attacker is able to purchase a genuine certificate and inveigle the user into
accessing a phishing web site (e.g., user responds to a spam email message by clicking on a URL pointing to the attacker’s server, in the belief that it points to the genuine server), the certificate can not be mapped to anything and the device will give a warning saying that the web site is unknown. Modern browsers can also help limit phishing attacks but user had better rely on the verification on their own devices because the browsers are deployed in untrusted terminals. After receiving confirmation from the user, the PTD generates a corresponding security token to be transmit to the terminal and the RP.

Our envisioned system is a realization of the novel model proposed in [4]. Chapter 3 describes in details how to realize that in particular with two devices (one terminal and one trusted device) forming the functionality of identity selector system.

![Diagram](Figure 2.10: Basic user experience)
Chapter 3

System Design

3.1 System Requirements

To make the model proposed in Chapter 2 become possible, it requires the device and the terminal to cooperate in providing the client functionality in the identity metasystem model, i.e., identity selector. The functionality of the identity selector system needs to be split in a way that only the PTD carries user credentials and does crucial security operations (e.g., signing and issuing security tokens) while the terminal does the rest. Besides, the adapted identity selector system must be flexible in adapting itself to different use cases. For example, the user might want to select information cards on the terminal or on the PTD, the system must allow the user to do that in runtime. It should be noted that changes are only made in the identity selector side so that other agents (relying parties and identity providers) can work seamlessly with it as usual without having to be aware of the presence of the PTD.

3.2 Design Options

Roaming identities in the context of identity metasystem implies roaming of user information cards. The question is how information cards stored on user device can be safely used on untrusted terminals. Self-issued cards need more protection than managed cards as they contain user credentials. Currently, most identity metasystems system implicitly assume that self-issued cards are stored and used in the local machine. We need a way to specify where and how to get self-issued security tokens like managed cards do. This section
CHAPTER 3. SYSTEM DESIGN

discusses two possible methods to achieve that.

The first one is to create references of self-issued cards that point to the self-issued identity provider. Each original self-issued card stored on user's PTD has a representative in the form of a managed card (called delegated card) to be imported to the identity selector on terminal. Those delegated cards have the Issuer identifier of self-issued identity provider and service references pointing to the PTD's endpoint. When the user picks a delegated card, the identity selector processes it as a managed card, asking the self-issued IdP on PTD for self-issued security token. Now in the terminal there are only metadata and security token sent from the PTD. Those data can be safely removed in the end of user session because they are not sensitive personal information.

The second approach uses a different method to solve the issue. In practice user wants to use his device in different ways. For example, he might want to select information cards on his own device or he could use his device as a standalone identity selector. The whole system (terminal and device) should be flexible in providing the functionality of the identity selector. Typically, it can be done by implementing the identity selector as distributed identity components which can be distributed to multiple devices. Therefore, different ways of using the whole system mean to different way of distributing identity components of the identity selector system. Using this method, user credentials can be safely used in untrusted terminals but they always stay in personal trusted devices.

The advantage of the first approach is that it requires very little changes to existing identity selectors (described in details in next section). However, compared to the high flexibility provided by the second approach, it has a limitation that it only supports a fixed use case in which user interacts with the identity selector on the terminal and uses the trusted device to confirm. Therefore, the second approach is selected for implementation in this thesis.

3.3 The Delegated-Card Approach

Technically, an information card is recognized as either a self-issued card or a managed card due to its <ic:Issuer> identifier. A simple way to make the ISS know where and how to get self-issued security tokens is to have special managed cards that have self-issued Issuer identifier and point to the security token service on the PTD. Those cards (called delegated cards) replace self-issued cards on the terminal. When user chooses a delegated
card on the terminal's ISS, the self-issued IdP on the PTD is asked for a self-issued security token. Other managed cards can be processed as usual. The structure of a delegated information card is listed in Appendix A.2.

This approach requires very little changes to existing identity selectors. The only change is with managed cards backed by self-issued cards. Whenever the user is asked to authenticate to the identity provider specified in the managed card, the identity selector should use our mechanism to get self-issued security token instead of directly querying the local card storage.

In principle this solution could work but current implementation of CardSpace does not accept it. The reason comes from the card installation process: cards with self-issued identifier are not allowed to be installed unless they are imported from a back-up store (via import/export function). Besides, Microsoft CardSpace API is so limited that it does not provide explicit APIs to enumerate cards’ info or to programmatically get self-issued security token from other sources than the local identity provider. The reason for this limitation may be to prevent Identity Laws especially rule User Control and Consent from being violated.

### 3.4 Identity Metasystem as Distributed Components

#### 3.4.1 System Architecture

Figure 3.1 depicts our own definition of identity metasystem architecture in terms of loosely-coupled identity components and interfaces between them.

**Relying Party, Identity Selector, Self-issued IdP, Remote IdP:** the roles of these four components have been clearly described in previous sections. It is noted that Relying Party and Remote IdP are considered external components in the whole system.

**IdP Authentication:** In order to obtain a security token from an IdP, the identity metasystem must first authenticate to that IdP. Authentication mechanisms can be based on self-issued card token or other existing authentication technologies such as username/password, Kerberos[36], or X509 certificate [18], which is explicitly stated in the IdP’s security policy. This component provides an abstract mechanism to authenticate a user to an identity provider.

**Trigger:** Trigger component is a "bridge" between identity metasystem -
aware applications and the identity selector system. Whenever the user accesses a service that supports information card technology, the Trigger activates the authentication process by first collecting the relying party’s policy, then starts the identity selector. At the end of the process, the trigger dispatches the security token returned from the identity selector to the relying party.

**Card Storage:** Responsibilities of Card Storage component include information card manipulation, card enumeration as well as filtering information cards that satisfy identity information required by relying party.

**Credential Storage:** Credential Storage component maintains user’s credentials including personal information, actual values of claims, certificates and other critical data. In practice, Credential Storage component and Card Storage component are usually combined. For example in self-issued cards, metadata and private user information are stored together in one XML document.

In our architecture, we have identified seven interfaces between components: (\(Ir, It, If, Ic, Ip, Is,\) and \(In\)). Table 3.1 lists operations exposed by each interface in details.
CHAPTER 3. SYSTEM DESIGN

- \textit{Ir}: This interface is used by Trigger component to collect RP’s certificates and policies.
- \textit{It}: Identity selector exposes this interface so that it can be activated together with parameters.
- \textit{If}: User credentials for authenticating to IdP such as password or Kerberos ticket are collected via this interface.
- \textit{Ic}: This interface exposes functionalities for card management including add/remove/enumerate cards.
- \textit{Ip}: Actual authentication to IdP is done here in which the IdP is asked for security token that satisfies claims stated in the RP’s policy.
- \textit{Is}: The Self-issued IdP component uses this interface to retrieve personal information for issuing security token including actual values of claim and keypair for each different RP-card pair.
- \textit{In}: This interface provides functionalities for import/export information cards from/to external devices.

In our distributed computing environment, identity components communicate via these well-defined interfaces. Every identity component is aware of other components and participates in typical processes to accomplish some requirements. For example, to populate the information card list on the user interface, Identity Selector component queries Card Storage component for each card.

Except for \textit{Ir} and \textit{Ip}, binding protocols for other interfaces are left unspecified, they can be RPC [35] calls or SOAP [15] requests or any custom-defined messages. More specifically, the binding protocol of \textit{Ip} is decided by the security policy of identity provider. If the IdP states in its policy that it requires username/password authentication, the identity selector must establish a secure connection (e.g. HTTPS connection) to the IdP in advance, capture user’s input and send those credentials via that channel using WS-Trust protocol. Alternatively, the authentication to the IdP can done using credentials collected from a smartcard without having to secure the underlying connection \textit{Ip}. There is no definitive protocol for exchanging information with self-issued IdP because it is located in the local machine.
<table>
<thead>
<tr>
<th>Interface</th>
<th>Operation</th>
<th>Input</th>
<th>Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ip</td>
<td>getSecurityToken</td>
<td>tokentype, relying party, credential&lt;sup&gt;1&lt;/sup&gt;, required claims, optional claims&lt;sup&gt;2&lt;/sup&gt;</td>
<td>display token, security token</td>
<td>Get security token from identity provider</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ic</td>
<td>getFirstCard</td>
<td>N/A</td>
<td>i-card&lt;sup&gt;3&lt;/sup&gt;</td>
<td>Get first card from store</td>
</tr>
<tr>
<td></td>
<td>getNextCard</td>
<td>N/A</td>
<td>i-card&lt;sup&gt;3&lt;/sup&gt;</td>
<td>Get next card from store</td>
</tr>
<tr>
<td></td>
<td>getCard</td>
<td>card ID</td>
<td>i-card&lt;sup&gt;3&lt;/sup&gt;</td>
<td>Get specific card from store</td>
</tr>
<tr>
<td></td>
<td>addCard</td>
<td>i-card&lt;sup&gt;3&lt;/sup&gt;</td>
<td>CardID</td>
<td>Add one card to store</td>
</tr>
<tr>
<td></td>
<td>editCard</td>
<td>Card ID,i-card</td>
<td>N/A</td>
<td>Modify one card</td>
</tr>
<tr>
<td></td>
<td>removeCard</td>
<td>Card ID</td>
<td>N/A</td>
<td>Delete one card from store</td>
</tr>
<tr>
<td>Ir</td>
<td>activateSelector</td>
<td>x-informationCard</td>
<td>security token</td>
<td>Browser extension HTML parsing</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>It</td>
<td>getToken</td>
<td>token type, relying party, required claims, optional claims</td>
<td>security token</td>
<td>Get security token from selector then pass it to relying party.</td>
</tr>
<tr>
<td></td>
<td>manageCards</td>
<td>N/A</td>
<td>N/A</td>
<td>Open selector’s user interface for card management</td>
</tr>
<tr>
<td>If</td>
<td>getCredential</td>
<td>N/A</td>
<td>credential</td>
<td>Obtain credential from user to authenticate to IDP</td>
</tr>
<tr>
<td>Interface</td>
<td>Operation</td>
<td>Input</td>
<td>Output</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>----------------------------</td>
<td>-------------------------------</td>
<td>------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Is</td>
<td>getUserPrivateData</td>
<td>card ID</td>
<td>private data</td>
<td>Manipulate claim values and other info such as key-pair, master key, PIN.</td>
</tr>
<tr>
<td></td>
<td>addUserPrivateData</td>
<td>card ID, private data</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td></td>
<td>editUserPrivateData</td>
<td>card ID, private data</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td></td>
<td>removeUserPrivateData</td>
<td>card ID, private data type</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>In</td>
<td>getCardContent</td>
<td>card ID</td>
<td>i-card</td>
<td>called when exporting cards</td>
</tr>
<tr>
<td></td>
<td>addUserPrivateData</td>
<td>card ID, private data</td>
<td>N/A</td>
<td>called when importing cards, adding personal cards</td>
</tr>
<tr>
<td></td>
<td>editUserPrivateData</td>
<td>card ID, private data</td>
<td>N/A</td>
<td>called when edit personal cards</td>
</tr>
<tr>
<td></td>
<td>removeUserPrivateData</td>
<td>card ID, private data type</td>
<td>N/A</td>
<td></td>
</tr>
</tbody>
</table>

Table 3.1: Interface operations

1 Credential being used to authenticate to identity provider
2 Optional claims which are selected to be send by end user. These claims are subset of optional claims stated in RP’s policy
3 i-card indicates filtered i-card document with only public metadata section[41]
4 i-card denotes full i-card document
3.4.2 Distribution of Identity Components

In practice the identity metasystem, which supports identity roaming, is expected to support different use cases in run time. For example, an end user may use his trusted device as a simple external storage device or a full-featured device including the user interface interaction to establish the connection to the terminal where service is consumed. The terminal, in return, adapts its functionality to different settings according to client profile set on the trusted device. We call these settings "configurations". In each configuration, there are two connected systems: one on a terminal and one on a trusted device; each system contains some of identity components (e.g. identity selector, credential storage) and each of those components is expected to cooperate seamlessly regardless of whether they are located in the same platform or distributed across different systems. The low-level connections between the identity components are assumed to be established beforehand (using implicit setting, configuration setting, or dynamic discovery) and secure enough. How identity components can communicate will be described in next section.

<table>
<thead>
<tr>
<th>Relying party</th>
<th>Conf.0</th>
<th>Conf.1</th>
<th>Conf.2</th>
<th>Conf.3</th>
<th>Conf.4</th>
<th>Conf.5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trigger</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>X</td>
<td>O</td>
</tr>
<tr>
<td>IdP Authentication</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Identity Selector</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Self-issued IdP</td>
<td>O</td>
<td>O</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Remote IdP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Card storage</td>
<td>O</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Credential storage</td>
<td>O</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

Table 3.2: Configurations

Table 5.1 summarizes six typical configurations that we have identified. X denotes identity component staying on the mobile device and O denotes identity component staying on the terminal. Configuration 0 is the default one that every identity metasystem supports. Except for configuration 5, the higher the configuration is numbered, the more secure it is and the more capabilities it requires from the mobile device. It depends on the amount of identity information that is encapsulated and processed in the client. To more complex tasks the device need to process, more resources it consume. For that reason, configuration 2 is considerably more secure than configuration
CHAPTER 3. SYSTEM DESIGN

1 because the interfaces between the device and the terminal are \( Ip \) and \( Ic \) whose data is not attractive to attackers. Consequently, the device in configuration 2 has the capability of generating/signing self-issued tokens that the device in configuration 1 does not have. Configuration 3 is more secure than configuration 2 because the terminal in configuration 3 does not even know which information cards belongs to the user. Configuration 4 is more secure than configuration 3 because the identity selector functionality is completely pushed to the trusted device.

**Configuration 0 - "all in terminal"**: This configuration is the simplest configuration in which all identity components are packed in the terminal side. This is also the starting point for every identity metasystem. Because user’s information is kept in one place, having the same identity profile on multiple machines securely is not possible. Configuration 0 is on the level of roaming that is currently supported by CardSpace, i.e., export/import functionality of cards. Although CardSpace allows user to export all his cards onto an external device and then import them from that device, CardSpace cannot directly use cards that are on an external device.

![Diagram of Configuration 0](image-url)

**Figure 3.2: Configuration 0 - all in terminal**
Configuration 1 - "external storage": In this configuration, the trusted device provides credential storage service and card storage service. It functions like an external database such as an USB stick or a smartcard. The Bandit project has similar architecture to this as they have card storage providers running on Bluetooth devices or remote servers. The advantage of this configuration is that the trusted device does very little processing. It is noted that communicating channels between the terminal and the device must be protected as the Self-issued IdP component in the terminal has direct access to the Credential Storage component running on the trusted device. However, the security of the transport protocol being used for Is in some cases can be sacrificed for better performance, for example USB connection. Although rule User Control and Consent of Identity Laws is relaxed to allow programmatic enumeration of information cards, the system is still secure as existing identity metasystems because every sensitive information is processed inside a trusted device and/or sent via secured channels.

Figure 3.3: Configuration 1 - mobile storage
Configuration 2 - "mobile identity provider": This configuration extends the functionality of the trusted device in configuration 1. The device can now operate as a Self-issued IdP. The Self-issued IdP component running on the trusted device is responsible for issuing security tokens when self-issued information cards are used. The Identity Selector on the terminal can also use managed information cards and request for security tokens from remote identity providers over the network as usual. This configuration is considerably more secure than configuration 1 since the interfaces between the terminal and the trusted device are \( Ip \) and \( Ic \) and the data transmitted over them is not that sensitive. Furthermore, user credentials always stay in the trusted device as the Credential Store component is only accessed locally by the Self-issued IdP component that is also residing in the same device. One tradeoff is the trusted device consumes more resources for issuing the security tokens (including parsing the request from the identity selector component, retrieving corresponding data, generating/signing/encrypting the security token before sending it back to the identity selector component).

![Diagram](image_url)
CHAPTER 3. SYSTEM DESIGN

Configuration 3 - "mobile identity selector": In this case, the trusted device supports built-in identity selector feature. The end user browses and uses services on the terminal but performs all the authentication related operations on his trusted device. Whenever he is asked to prove his identity, the Trigger component on the terminal activates the Identity Selector component on the trusted device so that he can choose one of his information cards. This gives such advantage and convenience to the end users because they perform all security related operations on their trusted devices (including the authentication related user interface If). This is also a benefit when the terminal itself does not have a proper display, e.g., auto ticket seller or electronic door access. It should be noted that the Identity Selector component needs to be implemented for each different platform (e.g. J2ME or Symbian) in order to display information cards in different types of device. The downside of this configuration is connectivity to the Internet is required when the trusted device needs to communicate with Managed IdPs.

![Diagram](image.png)

Figure 3.5: Configuration 3 - mobile identity selector
**Configuration 4 - "all in trusted device":** Next natural step is to have all the components in the trusted device. In this case, the user experience is the same as in configuration 0 except that now everything is done on the trusted device instead of the terminal. The difference between configuration 3 and 4 is that the end user both uses and authenticates to services in the trusted device, and this actually converges to the case where the whole ISS system is run in the trusted device. It is noted that the Trigger component must be understood by other identity-aware applications deployed in the trusted device so that the authentication process can be activated. For example, the web browser on the device calls a web plugin that activates the Trigger component. Although this configuration is the most secure in the context of mobile identity metasystem, it is not applicable for every device because of their limited capabilities (storage space, memory, processing power, etc.).

![Diagram](image-url)

**Figure 3.6:** Configuration 4 - all in trusted device
Configuration 5 - "external service operator": This is a special configuration which may be not possible with current technologies but it might be an interesting case in future. In this scenario, all of user’s essential data (personal and managed information cards, user credentials, and profile settings) are stored on a network server; the user device only operates as an access terminal for the end user. We assume that there is an external service operator providing personal identity management services via (mobile) network. For example, when an end user accesses a service using his trusted device, the device displays his information cards which have been loaded on demand from a remote server. In the case, if the end user loses his trusted device, he just goes to his personal identity management service to lock his profile usage for the lost device. The lost device poses no threat as there is no sensitive information stored in it.

![Diagram of Configuration 5 - external service operator](image-url)

Figure 3.7: Configuration 5 - external service operator
3.4.3 Middleware Architecture

In order to make distributed identity components be aware of each other and cooperate regardless of whether they are located in the same platform or distributed across different systems, a middleware is needed to ease the underlying communication. A source component does not need to know where the target is located and where or how to send a message to it. The message traversal is automatically handled by the underlying middleware. Figure 3.8 shows the high-level architecture of our proposed middleware. The middleware consists of an identity component container, connectivity endpoints and other modules. The main idea is based on Web Services model [6]. One identity component hosted by an identity system in side the component container is considered as a "service" of that system which then can be discovered by other identity components. Component communication makes use of SOAP messaging so that they can be implementation language and platform independent.

Connectivity Endpoint: Connectivity endpoint is used for transport-level messaging. It is also used for service broadcasting and discovery. Each endpoint is controlled by a Session Manager. To maximize the usability, various types of endpoints (Bluetooth, USB, IP, IrDA, etc.) are supported.

Session Manager: This module handles each user session. It maintains a list of connected target devices and their service profiles. A service profile contains information such as the unique component container identifier, expected configuration, and the transport media being used.

Repository Service: This module maintains a lookup table of references to identity components. The content of the lookup table is updated when two systems negotiate and configure their service profiles to work together or when a system disconnects. The purpose of the Repository Service is for dynamic communication in case we do not want to work with fixed, implicit configurations.

Component Container: This module wraps identity components that are running on the local system. Each Component Container is marked with a unique identifier so that one identity component can be identified uniquely among connected systems. For example one identity component can be addressed using naming convention as follows:

\[
\text{<Container identifier> - <Component name>}
\]

SOAP Engine: This module encapsulates requests from local identity components into SOAP requests [15] and parses SOAP responses sent from re-
mote identity components.

**Configuration Profile:** This module manages the service profile of the local system. It is useful when there is a need to store/retrieve persistent attributes of the local system (e.g. system identifier).

**Utility Module:** Some common functions for other modules (e.g. converter, encoder/decoder or file manipulation) as well as basic cryptography functions are provided in this module.

![Middleware High-level Architecture](image)

*Figure 3.8: Middleware High-level Architecture*

Typically, a user session goes through the following four phases.

**Phase 1 - Connection setup:** Initially, two systems must be able to locate each other and discover their offered services before exchanging messages. This phase depends on the discovery mode of user’s trusted device. The device should support two discovery modes. In active mode, the device broadcasts its profile and users can do pairing using the terminal. In listen mode, the terminal broadcast its service and users do pairing using the trusted device. There are discovery mechanisms that can be used by our
system (for example, Bluetooth Service Discovery Protocol [5] and Devices Profile for Web Services [10]).

**Phase 2 - Session initiation:** User confirms the connection setup (either on the trusted device or on the terminal, depending on the working mode), and optionally enters additional security code or PINs if required. After that, the Session Manager modules and the Repository Service modules on both systems are updated so that identity components can locate the correct target components in Phase 3. The terminal and the trusted device are now ready to exchange messages. Sequence diagrams of message flow in Phase 1 and Phase 2 are depicted in Figure 3.9 and Figure 3.10.

**Phase 3 - Message exchange:** Operation calls between the client and the server are synchronous requests/responses. The requesting component composes a request in the form of a XML structure in Appendix A.1, puts some parameters in, sends it to the middleware and waits for the response. The role of the middleware is to deliver the request to the right target component and return the response to the requesting component. Before sending out a request message, one identity component queries the Repository Service module for the target components and forms the request to be sent to the local Session Manager. The Session Manager determines how to reach the target component(s): if the target component belongs to the local Component Container, the message is forwarded to that container then to the target component. Otherwise, the Session Manager passes the request to its Endpoint to be transferred to the remote system hosting the target component. The Session Manager on the remote machine receives the request from its Endpoint and delivers it to its SOAP Engine. The request, after being processed by the SOAP Engine, is forwarded to Component Container then to the target identity component for final processing. The response message follows the same flow in which the source and target components switch their roles. Figure 3.11, Figure 3.12 depict simple flows of interaction between two identity components located in two devices and in the same device respectively.

**Phase 4 - Session termination:** In this final phase, user’s metadata information and any temporary data including session information, cache, and history data on the terminal are cleaned up. After the session termination the terminal should turn into its default configuration (i.e., configuration 0).

It should be noted that when identity components are distributed in different places, some of them require additional security because their parameters carrying user credential are sent over the network without protection. In some cases we can not rely much on the security provided by the transport
CHAPTER 3. SYSTEM DESIGN

layer. For example, if a Bluetooth connection is used without encryption, the attacker can sniff the conversation and capture user self-issued cards sent to the device when they are created on the terminal. The middleware certainly needs to provide a generic mechanism to protect the communication between components as optional security enhancement. The security provided by the middleware should be transparent to that of application messages and to that of transport layer.

There are two problems needed to be solved. First, the initial negotiation between the device and the terminal (Phase 1) must be secure. This issue can be solved using one of methods in [38]. This thesis assumes there is a method for securing connection between the device and the terminal. Second, messages being sent between the device and the terminal must be encrypted and filtered in application-message level. As shown in Figure 3.11, SOAP Engine being a gateway for all incoming messages can function like a XML firewall to allow only a certain set of configured operations from certain sources. For example in configuration 2, the trusted device allows only operations defined for \textit{Ip} and \textit{Ic} interfaces sent from the terminal. Session Manager can provide enhanced security by encrypting/decrypting messages with the session key which has been negotiated in the initial phase.
Figure 3.9: Connection set up in active mode
2. User switches to LISTEN mode

1. Broadcast service

3. Host Probe

4. Host Probe Match

5. Service Probe

6. Service Probe Match

7. List of available hosts

8. Selected host

Security handshaking

9. Request connect

10. Connect OK

11. Mobile profile

12. Profile accepted

13. Request confirmation

14. User confirmation

15. Update session info

16. Update session info

17. Handshake finish

18. Handshake OK

Figure 3.10: Connection set up in listen mode
Figure 3.11: Communication between two identity components in two devices

Figure 3.12: Communication between two identity components within one device
Chapter 4

Implementation

This chapter describes how configuration 2 (mobile identity provider) can be implemented. The system is a prototype of the model proposed in Chapter 3. Other configurations can be implemented based on this work.

4.1 Brief description of work flow

User session starts when the user uses his device to discover available terminals (Phase 1 in Section 3.4.3). The user must input some PIN numbers on the PTD then type the same on the terminal. Once the connection has been set up, the device is ready for requests from the terminal (Phase 3). When user goes to a web site (RP) that uses information-card technology, the Trigger component collects RP’s certificate, policies, and requested/optional claims. It then call the Identity Selector component with some parameters such as RP’s certificate file location, claims and expected output token file location. To display information cards, the user interface of Identity Selector asks the PTD for metadata of information cards through Ic interface. In particular, every time the user interface is updated or refreshed, the terminal asks for serial of cards until there is no card or an error is returned in the response from the device. When the user selects one of the cards on the user interface of the terminal, the Identity Selector examines the selected card’s metadata and composes a request containing the RP’s certificate, requested/optional claims and other parameters to sends to the corresponding IdP component.
CHAPTER 4. IMPLEMENTATION

Figure 4.1: Interaction between identity components
CHAPTER 4. IMPLEMENTATION

If the security token is expected to be issued by a managed IdP, the Managed IdP component on the terminal side processes the request as described in [28]. If the token is expected to be self-issued, the Identity Selector component sends a request to the Self-issued IdP component on the trusted device through our middleware. Upon receiving the request, the Self-issued IdP component first verifies the RP’s certificate and pops up a notification to the user asking for user confirmation. This is one way to prevent phishing attack because if the RP’s certificate is unknown to the device, it means that the RP is either a new service the user accesses for the first time or a phishing site. If the user agrees, the device generates a SAML token [30] with associated attribute values extracted from the Credential Storage on the trusted device through Is interface. A new keypair is generated and saved in the local store on the device if that RP is a new one. After that the security token is signed with the RP-specific key loaded from the local store and encrypted by the public key of the RP before sending it back to the terminal. The token is saved to a temporary file specified by the plugin above and that file is subsequently loaded by the plugin to send to the RP.

The user is also able to create self-issued cards or delete cards using user interface on the terminal. Those actions are transformed to requests to be sent to the device for further processing then the user interface is updated again. Figure 4.1 shows a simplified sequence diagram of identity component interaction.

4.2 Technical Issues

In this configuration, although the role of the trusted device in the authentication process is crucial, its functionality is quite simple. The trusted device provides functionality of Card Storage, Credential Storage and Self-issued IdP component. Most of complex processing including collecting certificate, user interface and service execution are done in the public terminal. User’s information cards (self-issued or managed) are stored as full i-card documents [41] on the trusted device side. For convenient design, Card Storage component and Credential Storage component are combined because the i-card format itself contains enough semantics to describe user credentials. The Self-issued IdP component just needs to a full i-card in order to generate a security token. This is why in our implementation there is no Credential Storage component and Card Storage component exposes two interface Is (to Identity Selector) and Ie (to Self-issued IdP) although in design Is interface should be provided by Credential Storage component.
CHAPTER 4. IMPLEMENTATION

Ic-operations are divided into two groups: one that involves card enumeration and one that involves card manipulation such as getting cards, adding cards, editing cards, removing cards. Card enumeration happens when displaying user’s information cards and it requires only card’s metadata such as name, card id, and picture. To make those operations work with metadata, some filtering rules are defined to be used by the Card Storage component on the device to expose only metadata part of information card content (denoted by i-card* in Table 3.1). It is noted that the separation of metadata part and user private data part of i-card format is one of our contributions to Higgins project because currently there is no clear separation between the two parts. The main idea is to give only non-sensitive metadata section of the i-card document to the terminal and let the terminal provide the user interface functions. When the trusted device connects to a public terminal, only the public portion of i-card structure is transferred to the terminal so that end user can visually use their information cards on the terminal’s identity selector. All "metacards" and other temporary data such as history, cache, session on the terminal will be deleted when the trusted device disconnects.

Ic-operations involving card manipulation require higher level of security for their parameters carrying user credential to be sent over the network. Therefore, it it important to mention that the terminal must be explicitly declared trusted by the end user in configurations where Ic is a cross-system interface (i.e., configuration 1, 2, 5). They either can only be executed from a trusted terminal like a home PC (or on the trusted device itself) or require a secure communication channel between the device and the terminal (e.g., USB cable or generic middleware message encryption).

Implementing Self-issued IdP component is relatively simple. The most complicated task is to generate RP-card-specific keypairs. A RP-card-specific key is used for signing self-issued security tokens. By checking the token’s signature, the RP can recognize from which user the token is issued. The keypair must be different for each RP-card combination to preserve user anonymity between RPs. Even the attack could get the token, he cannot use it for other RPs because he does not how to sign the token for those RPs. The keypair can be generated in any way as long as they are different for each RP-card combination. Since each self-issued card has a fixed secret part called MasterKey and RP identifier is different for different RPs, a RP-card-specific keypair can be derived from those two factors. Technically, the MasterKey is used as a seed for generating keypair. The algorithm for keypair generation follows X9.31 standard specified in [28]. However, we obtain that generating keypair is the heaviest processing. To improve the performance, the keypair is generated only once and stored in a repository, it can be loaded from the
store when needed. One way to store the generated keypairs is to put them in a new XML extension of the i-card format as private data. This is much more efficient than generating the same keypair for each request due to the complexity of the RSA keypair generation algorithm.

The last technical issue is that RP-card-specific keypair generation must be done either in trusted devices or trusted terminals (like home PC). Ideally, the keypair generation should be done by the IdP located in the trusted device. We observe that the processing duration in the trusted device is acceptably long for the first time and really short for the following uses (see Table 3.2). If the keypair pushed to the device comes from an untrusted terminal, it may not be safely used. An attacker is able to impersonate if he controls the malicious identity selector in the untrusted terminal and if he knows some claims of the user required by the RP. He can generate a token for himself on behalf of the user because he is the one who has given the keypair in the very beginning. The attack scenario is described as follows:

1. **RP → Untrusted terminal**: public $K_{RP}$, claims
   The attacker generates keypair $K_{T}$ on the untrusted terminal

2. **Untrusted Terminal → Device**: public $K_{RP}$, claims, keypair $K_{T}$

3. **Device → Untrusted Terminal**: $E_{K_{RP}}(S_{K_{T}}\{Token\})$

4. **Untrusted Terminal → RP**: $E_{K_{RP}}(S_{K_{T}}\{Token\})$

   The attacker has collected user claim values
   he can impersonate whenever he wants.

1. **RP → Untrusted terminal**: public $K_{RP}$, claims

2. **Untrusted Terminal → RP**: $E_{K_{RP}}(S_{K_{T}}\{Token\})$
4.3 Adapting Bandit Implementation

We realize that the Bandit code can be made use of with some modifications. Figure 4.2 shows Bandit’s high-level design that is mapped into the our model. The main module ISS loads complete set of available i-card documents [41] from the Store Provider through $Ic'$. To get a self-issued token, the selector passes an a whole i-card object together with parameters to the self-issued IdP component. This design causes security problems when being applied to distributed environment because the identity selector has access to all secret information and keeps the loaded i-card object in memory. To adapt Bandit implementation, Bandit’s components have been modified and wrapped in our architecture. Changes to Bandit identity components are explained in detail in Table 4.1. The Table tells the differences between the Bandit implementation and our implementation. Most of our implementation contribution is the middleware and implementation of identity components on phone. On trusted device side, the client application called DigitalMeJ2ME is run in J2ME platform [37]. Identity components of DigitalMeJ2ME (Card Storage and Self-issued IdP) have been implemented from scratch, using open-source libraries such as kXML [17] for XML document processing and Bouncy Castle[39] for cryptographic operations. The middleware described in Section 3.2.3 is implemented for both server side (in C++) and client side (in Java). Figure 4.3 maps the design in Figure 3.11 to sequence diagram. Specifications for operations in Configuration 2 written in VDM++ language [13] are provided in Appendix A.3.

![Figure 4.2: Bandit implementation](image-url)
<table>
<thead>
<tr>
<th>Interface</th>
<th>Operation</th>
<th>Bandit implementation</th>
<th>Our implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Input</td>
<td>Output</td>
</tr>
<tr>
<td>Ip</td>
<td><code>getSecurityToken</code></td>
<td>tokentype, i-card, relying party, credential required claims, optional claims</td>
<td>display token, security token</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>getNextCard</code></td>
<td>N/A</td>
<td>i-card</td>
</tr>
<tr>
<td></td>
<td><code>getFirstCard</code></td>
<td>N/A</td>
<td>i-card</td>
</tr>
<tr>
<td></td>
<td><code>getCard</code></td>
<td>card ID</td>
<td>i-card</td>
</tr>
<tr>
<td></td>
<td><code>addCard</code></td>
<td>i-card</td>
<td>card ID</td>
</tr>
<tr>
<td></td>
<td><code>editCard</code></td>
<td>card ID, i-card</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td><code>removeCard</code></td>
<td>card ID</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td><code>getToken</code></td>
<td>token type, relying party, required claims, optional claims</td>
<td>security token</td>
</tr>
<tr>
<td>It</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>manageCards</code></td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>If</td>
<td><code>getCredential</code></td>
<td>N/A</td>
<td>credential</td>
</tr>
</tbody>
</table>

Table 4.1: Comparison of Bandit implementation and our implementation
Figure 4.3: Sequence diagram of middleware message flow
4.4 Implementation Details

4.4.1 Use cases

![Use case diagram]

Figure 4.4: Use case diagram
<table>
<thead>
<tr>
<th>Use Case</th>
<th>Start DigitalMe</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actor</td>
<td>User</td>
</tr>
<tr>
<td>Preconditions</td>
<td>DigitalMe has not been started.</td>
</tr>
<tr>
<td>Description</td>
<td>Start DigitalMe server, check for connectivity and set to the default configuration 0. This may require administration right because the DigitalMe code is deployed as a daemon running in the local system of the terminal.</td>
</tr>
<tr>
<td>Exception</td>
<td></td>
</tr>
<tr>
<td>Postconditions</td>
<td>The system is started and ready to accept connection from any client</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Use Case</th>
<th>Start DigitalMeJ2ME</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actor</td>
<td>User</td>
</tr>
<tr>
<td>Preconditions</td>
<td>DigitalMeJ2ME application has not been started.</td>
</tr>
<tr>
<td>Description</td>
<td>User starts DigitalMeJ2ME application on the trusted device. Check for connectivity and set to the default configuration 2.</td>
</tr>
<tr>
<td>Exception</td>
<td>No supported connectivity.</td>
</tr>
<tr>
<td>Postconditions</td>
<td>DigitalMeJ2ME is started and ready to work</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Use Case</th>
<th>Discover available terminals</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actor</td>
<td>User</td>
</tr>
<tr>
<td>Preconditions</td>
<td>DigitalMeJ2ME has been started successfully.</td>
</tr>
<tr>
<td>Description</td>
<td>User uses the device to discover available terminals. Use the discovery mechanism of the device’s endpoint.</td>
</tr>
<tr>
<td>Exception</td>
<td></td>
</tr>
<tr>
<td>Postconditions</td>
<td></td>
</tr>
</tbody>
</table>
### Use Case: Connect to terminal

**Actor:** User

**Preconditions:** DigitalMeJ2ME client has been started successfully. DigitalMe server is ready to accept connection.

**Description:** User connects to a particular terminal. A secure connection is set up and a shared session key may be exchanged for message encryption. DigitalMe server switches to configuration 2. DigitalMeJ2ME shows connected screen.

**Exceptions:** If the connection is denied (i.e., server is busy) or connectivity is not supported or the connection pairing is not successful then a notification is sent to user through DigitalMeJ2ME.

**Postconditions:** DigitalMeJ2ME connects to DigitalMe server.

### Use Case: Disconnect

**Actor:** User

**Preconditions:** DigitalMeJ2ME is connected to DigitalMe.

**Description:** Either DigitalMe or DigitalMeJ2ME is disconnected. Connection and resources are freed. Finally, DigitalMe switches back to default configuration 0.

**Exceptions:**

**Postconditions:** DigitalMeJ2ME is in disconnected state. DigitalMe is in configuration 0.

### Use Case: Create new card

**Actor:** User

**Preconditions:** DigitalMeJ2ME is connected to DigitalMe.

**Description:** User creates a self-issued card using identity selector on DigitalMe server. An 'addcard' request with parameters is sent to Card Storage component on the phone. User must confirm on the phone then a new card is added. After that the card list on DigitalMe's identity selector is updated.

**Exceptions:** An error is notified through DigitalMe UI if user does not confirm on the device or the connection is broken.

**Postconditions:** New card is added to storage.
### Use Case 4.1: Delete card

<table>
<thead>
<tr>
<th>Use Case</th>
<th>Delete card</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actor</td>
<td>User</td>
</tr>
<tr>
<td>Preconditions</td>
<td>DigitalMeJ2ME is connected to DigitalMe.</td>
</tr>
<tr>
<td>Description</td>
<td>User deletes a card on DigitalMe’s selector. A 'delete-card' request is sent to Card Storage component on the phone. User must confirm on the device when the card is deleted from Card Storage. After that the card list on DigitalMe’s identity selector is updated.</td>
</tr>
<tr>
<td>Exceptions</td>
<td>An error is notified through DigitalMe UI if user does not confirm on the device or the connection is broken.</td>
</tr>
<tr>
<td>Postconditions</td>
<td>The card is deleted from storage.</td>
</tr>
</tbody>
</table>

### Use Case 4.2: Edit card

<table>
<thead>
<tr>
<th>Use Case</th>
<th>Edit card</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actor</td>
<td>User</td>
</tr>
<tr>
<td>Preconditions</td>
<td>DigitalMeJ2ME is connected to DigitalMe.</td>
</tr>
<tr>
<td>Description</td>
<td>DigitalMe server acquires the selected card’s content from the phone, maps the content to the user interface and lets user edit it. When the editing is finished, user must confirm on the device then Card Storage on the phone is automatically updated. Card list on DigitalMe’s identity selector is updated also.</td>
</tr>
<tr>
<td>Exceptions</td>
<td>An error is notified through DigitalMe UI if user does not confirm on the device or the connection is broken.</td>
</tr>
<tr>
<td>Postconditions</td>
<td>The card is updated.</td>
</tr>
<tr>
<td><strong>Use Case</strong></td>
<td>Authenticate</td>
</tr>
<tr>
<td>-----------------</td>
<td>---------------</td>
</tr>
<tr>
<td><strong>Actor</strong></td>
<td>User</td>
</tr>
<tr>
<td><strong>Preconditions</strong></td>
<td>DigitalMeJ2ME is connected to DigitalMe.</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>Firefox plugin triggers the DigitalMe’ identity selector. User selects a card from the list on the terminal. If the token is expected to be from a managed IdP, use the Remote IdP component to get the token. Otherwise, ask the Self-issued IdP on the phone for the token. The Self-issued IdP component examines the request. If the RP is unknown (i.e. its certificate is unknown) then the user must be notified. If user accepts the new RP then a new keypair is generated and put to the store. DigitalMeJ2ME then asks for user final confirmation about issuing the token from the phone. If user accepts, the Self-issued IdP component loads the relevant self-issued card, generates a security token, signs it with respective keypair loaded from the store and encrypts it with RP’s public key. Finally, the token is sent to the terminal. The token is given to the Firefox plugin to send to the RP.</td>
</tr>
</tbody>
</table>
| **Exceptions**  | 1. If the RP is unknown and user does not accept.  
2. If user does not want to issue the token request by RP.  
3. If the connection is broken |
| **Postconditions** | User is authenticated to the RP. |
### 4.4.2 Class diagram

![Class Diagram](image_url)

**Figure 4.5:** Class diagram
4.4.3 Package diagram

![Package diagram](image)

Figure 4.6: Package diagram

4.5 Application

Figure 4.8a shows the initial screen on the phone after DigitalMeJ2ME application is started. There are two self-issued cards created for testing namely 'Personal Card' and 'Test Card'. Figure 4.8b depicts the discovery phase using Bluetooth. The user must first input some PIN numbers on the phone then type the same on the terminal like in Figure 4.9. Once the connection has been established, he can use his information cards on the identity selector on the untrusted terminal. It is noted that they are just icons and metadata which are unattractive to attackers. The user can create or delete any information cards on the screen and it will be automatically updated to the device. Whenever the user choose an information card to get a security token, the device display a notification screen (Figure 4.10) that shows brief information about the RP and asks for user confirmation. If he presses "Yes", the phone will generate an encrypted/signed token and send it to the RP via the identity selector on the terminal. When the user disconnects the phone, information cards on the terminal automatically disappear (Figure 4.11).
Figure 4.7: Deployment diagram

(a) DigitalMeJ2ME
(b) Discovering available terminals

Figure 4.8: DigitalMeJ2ME application
CHAPTER 4. IMPLEMENTATION

Figure 4.9: Pairing between terminal and phone

(a) Pairing on device

(b) Pairing on terminal

Figure 4.10: Issue token

(a) Choosing card on terminal

(b) Confirm on device
Figure 4.11: Device disconnected
Chapter 5

Discussion

5.1 System Evaluation

Evaluation is done during the design and after our system is implemented. There are three aspects to be considered:

- **Security**: This is the most fundamental criteria for every identity system. By using the design in Chapter 3 and leveraging the identity metasystem model, we have shown that our system does not compromise any user credentials. Besides, the use of personal trusted device as an additional authentication method does not only improve the security but also enhance the portability and usability.

- **Design**: This criteria is used to evaluate the efficiency of the design. As said in Chapter 4, our server application based on Bandit project can be run on multiple platforms such as Linux and MacOS and compatible with other agents (identity providers, relying parties) on the Internet, while the client application makes use of J2ME environment to be able to run on different models of mobile phones. The whole system has been tested with real devices and the results were promising.

- **Usability**: Our system supports different scenarios depending on the user context. Users now can have their identities/information cards loaded on their phone and safely use them everywhere even in untrusted machines without worrying their credentials being collected. In addition, the user is able to use identity selector interface either on the terminal or on the device.
5.1.1 Known Limitations

Due to the restricted resource of the mobile phone and strict requirement of J2ME environment, several limitations are noted. This is both an advantage and a challenge since J2ME is widely supported on mobile phones but has a limited set of features. However, it is noted that these limitations can be seen as sacrifices made for better performance. The limitations are:

- **Certificate chain verification on mobile device**: Each mobile device has a fixed set of root certificate installed when it is produced by the manufacturer. It leads to a problem that if the RP’s certificate is considerably invalid if it is issued by a root CA which is not installed in the mobile phone. Installing root CA certificates to mobile phones manually is rather complicated for end users. Besides, certificate chain whose depth is more than 2 is hard to be verified on the phone due to limited connectivity and limited computational power.

- **Inter-process communication**: One of the limitations of J2ME environment is that it doesn’t allow communication between midlets and native applications. Therefore, J2ME can only be applied to configuration 2 and configuration 3 where only one midlet is needed to be running at a time. Configuration 4, which requires a built-in identity selector activated when an identity-aware application is invoked, is incapable for J2ME. This can be solved when we implement the client application in other platforms such as Symbian OS.

5.1.2 Performance Analysis

The whole system has been deployed and tested in real life. The server code, DigitalMe, is deployed in some kiosk PCs running OpenSuse Linux 10.3 with GTK/GNOME environment and a Firefox plugin. The client code, DigitalMeJ2ME, is run as a MIDP application on mobile phone Nokia N92 and E51, but in theory it should work in all mobile devices supporting J2ME MIDP 2.0. The connection being used between the phone and the terminal is Bluetooth.

Table 5.1 lists the maximum memory usage and average duration (including time for sending/receiving messages over the network) for each operation call on the mobile device. Actual time for each operation call can be measured with real devices but maximum memory usage can not be monitored because of limitation to J2ME environment. Instead, the memory monitoring has been done with the phone emulator provided by Sun Wireless Toolkit.


### Table 5.1: Performance evaluation of DigitalMeJ2ME in configuration 2

<table>
<thead>
<tr>
<th>Operation</th>
<th>Maximum memory usage (KB)</th>
<th>Average duration(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add card</td>
<td>538</td>
<td>3</td>
</tr>
<tr>
<td>Delete card</td>
<td>140</td>
<td>0.5</td>
</tr>
<tr>
<td>Get card</td>
<td>202</td>
<td>0.5</td>
</tr>
<tr>
<td>Get security token (first time visiting RP)</td>
<td>970</td>
<td>21</td>
</tr>
<tr>
<td>Get security token (with loaded keypair)</td>
<td>503</td>
<td>3</td>
</tr>
</tbody>
</table>

The most heavy task is to get security token for a first-visited RP (including generating the RP-card-specific keypair) taking around 20-21 seconds, which is acceptable. If the keypair has been generated, the time for issuing security token significantly reduces to 2-3 seconds. Other tasks are handled less than one second. It is similar for the maximum memory usage, get security token for a first-visited RP consumes resource most.

#### 5.2 Potentials of Identity Metasystem

The identity metasystem is still a new concept. It shows the potentials of solving such security problems concerning the integration of individual identity systems. In practice, there are other cases which can be applied but they require some fundamental issues to be solved.

Let us take the buying ticket from the local cinema for example. In this example, the cinema is both relying party and identity provider for itself. An user first needs to register with the cinema’s IdP then he is given an information card that is loaded to his mobile phone. Whenever he wants to go to the cinema, he goes to its web site to purchase tickets. At the entrance, he just uses his phone to show the ticket before going in to enjoy his movies.

In the context of identity metasystem, the electronic tickets are just tokens collected from the cinema’s IdP. The first problem is how to visualize those tokens as they are also part of user identity. Tokens certainly are not information cards because they may only be valid for one or a couple of times. When going to the cinema, people show their tickets (tokens) instead of cards. One
solution for that is the cinema may deploy its client on the user’s device but it leads to a result of different clients for each different cinema running on user device. It’d be more convenient if the tokens can also be visualized and used inside the identity selector on user device but in order to that, we need to a way to describe the metadata of tokens.

When the first problem could be solved, we then face the second problem of how identity metasystem works offline where connectivity is limited. Typically, the issued token may be securely stored on user device for long time. Then they are sent directly to the RP to prove user identity without necessarily having to contact the remote IdP again. Before being used, they may be transferred from one device to another like we do with real tickets. This problem evolves the handling the lifecycle of credentials, according to associated policies and user’s preferences, locally on user’s device clients. The problem is also referred as delegation credentials, i.e. tokens issued by one authority is delegated to the local IdP for offline transactions.

The root cause of two problems above is that current design of identity metasystem is relationship-oriented, not credential-oriented. An identity system is called relationship-oriented if it relies on the relationships between users and online identity providers that create short-term credentials during transactions. An credential-oriented identity system, in contrast, advocates offline identity providers and long-term credentials at user’s client. Each design has its own advantages and drawbacks. Relationship-oriented identity systems guarantee freshness and up-to-date attributes of the identity tokens issued but they need to be online during transactions while credential-oriented identity systems can be offline during transactions but the validity of issued tokens is a hard problem. There exists no efficient solution to this problem, yet.

Another problem is with end user interaction. There are cases in real life where a person is an identity provider to other persons. For example, Alice wants to use the facilities in Bob’s house for a day, she can use her phone to ask for access rights from Bob. If Bob agrees, Alice is given a temporary token valid in one day so that she can use Bob’s services. In this case, Bob is an identity provider and his house is a relying party. Technically, this is a micro version of identity metasystem in the Internet and it can be solved if each person is equipped with a personal-added device that runs a unique portable security token service (P-STS). A P-STS is different from local self-issued STS is that it has a unique (omnidirectional) identifier. P-STSes can be plugged into any system which has connectivity or run on their own then they can communicate with each other on behalf of their owners.
However, it leads to another technical issue: the identifier of P-STS needs to be unique for each person. The question is how that identifier is specified. Device-specific identifiers do not fix the case because of the many-to-many relationship between people and the devices they use. Furthermore, in case one of user devices is lost, there must be extra security mechanisms to merge the lost identity information to a new P-STS. If we manage to have a unique identifier for each individual person then claim-based identity systems would become useless.

The final issue discussed in this section is about the claim-based characteristic of state-of-the-art identity system. OpenID tries to enable identity roaming by providing unique identifiers that can be accessed anywhere on the Internet but one may say that OpenID, in some ways, sees user as web resource. CardSpace takes another solution by issuing information cards and let users themselves manage their identities but it is limited to certain security contexts. Some vendors may let users put personal information to their devices and use them as the means to authenticate to services. However, that approach may be criticized user as client device due to the reason that each device with a private key provisioned on it represents its owner’s identity directly. They all give a feeling that something critical is missing. What we need is a broader and more philosophical solution that considers user as human being. Computer applications and appliances exist to server us and respect our wishes around sharing and privacy even when we cannot or do not want to be online. We believe this be the answer for all problems mentioned above. The role of personal devices would become both the gateway between the virtual world and the real world and the means for users to control their identities; therefore, users are not heavily bounded to their devices, i.e. identity roaming would become more flexible.

Due to limited space and the scope of this thesis, we only give some of our thoughts on how to realize that vision. Higgins has done great when providing a common data model for personal identity information including relationship and social interactions. By adding timeline, events, a trust model, or semantic of contexts to that model, we can have a virtual world that reflects any user behavior and identity information in the real world. In that virtual world, end user interaction would become interaction between digital identities. Solving the problem of delegation of user credentials from one digital identity to another one would become as simple as stating a fact or "event" which can be verified by when needed. To protect digital identity privacy (e.g. from lost devices), one way is to have chronological versions of digital identities. As digital identities are just bits of information, they can be encrypted or decrypted or manipulated in any way we want. When
one digital identity is changed to a newer version, its older versions would become inaccessible.

Obviously it needs a lot of research studies to make this real including personal identifier, reachability of identity provider, information sharing and so on. However, we believe the final achievement would be much better than what we have with state-of-the-art identity systems.
Chapter 6

Conclusions and Future Work

In this thesis our research in identity metasystem metadata and in particular in the area of identity roaming is presented. The main contribution of this thesis is two fold: (1) specifying the architecture for identity metasystem implementation that makes it easy to distribute identity components; and (2) using (1) to implement a two-device configuration which enables digital identity roaming across security domains. The distinguishing feature of this research study is that it shows how an existing identity metasystem can be extended to support the use of an trusted device.

In order to realize that, a major part of the work was dedicated to the analysis of distributing identity components between devices. Also architecture and implementation framework have been provided. All of them have been tested in real systems and the results were quite promising. To summarize, we are able to move to the next level of current dimension of identity metasystem [29].

For future work, it’s reasonable to plan to extend implementation in Chapter 4 to support configuration 3 (mobile identity selector) and configuration 4 (all in mobile device). As explained in Section 5.1.1, Symbian operating system is the chosen platform for future implementation because it provides a strong platform for security processing plus a very good native user interface. Recently, Nokia has published On-board Credential (ObC) [14] framework. According to literature study in Chapter 3, the ObC subsystem can provide functionalities of Credential Storage component and Self-issued IdP component. Another possible extension is that the client system is deployed in compact devices such as USB sticks. User plugs an USB stick and selects information cards on the terminal. When the USB stick is plugged out, all cards on the terminal disappear. For average users, it is more convenient than Blue-
tooth with discovery/pairing mechanisms. Security handshaking or transport layer encryption can be skipped because the USB physical connection itself is trustworthy enough. However, this extension requires hardware-specific driver programming and firmware programming.

The vision of user-centric identity management has attracted many research studies in different areas for the simple fact that information security is not merely a technical problem solved just by applying one or two secure frameworks. The lesson learned is that when developing concepts and models it is important to keep in mind that human factor is an important part of the whole system. The final target is to give the end users better security and richer user experience.
Bibliography


BIBLIOGRAPHY


Appendices

A.1 Middleware Request/Response Format

```xml
<message>
  <request>
    <source>componentX</source>
    <target>componentY</target>
    <operation>OperationABC</operation>
    <params>ParamsXYZ</params>
  </request>
</message>

<message>
  <response>
    <source>componentX</source>
    <target>componentY</target>
    <operation>OperationABC</operation>
    <status>error_code</status>
    <params>ParamsXYZ</params>
  </response>
</message>
```
A.2 Delegated Card Format

<ic:InformationCard ...>
  <ic:InformationCardReference>
    <ic:CardId>
      xs:anyURI
    </ic:CardId>
    <ic:CardVersion>
      xs:unsignedInt
    </ic:CardVersion>
  </ic:InformationCardReference>
  <ic:CardName>
    xs:string
  </ic:CardName>
  <ic:CardImage>
    xs:base64Binary
  </ic:CardImage>
  <ic:Issuer>
    http://schemas.xmlsoap.org/ws/2005/05/identity/issuer/self
  </ic:Issuer>
  <ic:TimeIssued> xs:dateTime </ic:TimeIssued>
  <ic:TokenServiceList>
    <ic:TokenService>
      <usa:EndpointReference>
        https://localhost:1234/PTD/mex
      </usa:EndpointReference>
      <ic:UserCredential> <ic:UserCredential>
    </ic:TokenService>
  </ic:TokenServiceList>
  <ic:SupportedTokenTypeList>
    ...
  </ic:SupportedTokenTypeList>
  <ic:SupportedClaimTypeList>
    ...
  </ic:SupportedClaimTypeList>
  <ic:RequireAppliesTo> ... </ic:RequireAppliesTo>
  <ic:PrivacyNotice> ... </ic:PrivacyNotice>
</ic:InformationCard>
A.3 Operation Specifications in VDM++

class Object

types
public
  ClaimType :: char*;
public
  Claim :: char*
end Object

class Card is subclass of Object

instance variables
   public cardId : N;
   public content : ClaimType \rightarrow Claim;
end Card

class CardStorage

instance variables
    store : N \rightarrow Card;
    currentIndex : N := 0;
    inv 0 \leq currentIndex \land currentIndex \leq 99

functions
    getMaxAvailable (s : N-set) m : N
post if card s = 0
    then m = 0
    else m \in s \land \forall s' \in s \setminus \{ m \} \cdot m > s'

operations
public
    getCard : N \rightarrow Card
    getCard (id) \triangleq
    return (store (id))
    pre id \in \dom store ;

public
    addCard : Card \rightarrow N
    addCard (c) \triangleq
    (  dcl newid : N := getMaxAvailable (\dom store) + 1;
         store := store \uplus \{newid \mapsto c\};
        return newid
    );
deleteCard : \mathbb{N} \to ()
deleterCard (id) \triangleq
\quad store := \{id\} \triangleleft store;

public

getFirstCard : () \to Card
getchFirstCard () \triangleq
\begin{array}{l}
\quad current\text{Index} := 1;
\quad \text{while} (\text{currentIndex} \notin \text{dom store})
\quad \quad \text{do currentIndex} := \text{currentIndex} + 1;
\quad \text{return (getCard (currentIndex))}
\end{array}
\quad \text{pre dom store} \neq \{\} ;

public

getNextCard : () \to Card
getchNextCard () \triangleq
\begin{array}{l}
\quad \text{while} (\text{currentIndex} \notin \text{dom store})
\quad \quad \text{do currentIndex} := \text{currentIndex} + 1;
\quad \text{return (getCard (currentIndex))}
\end{array}
\quad \text{pre currentIndex} < \text{getMaxAvailable (dom store)} ;

public

getCardByReferenceId (id : \mathbb{N}) c : Card
getchCardByReferenceId () \triangleq
\begin{array}{l}
\quad c \in \text{rng store} \wedge c.\text{cardId} = id
\end{array}

end CardStorage

class SelfIdP is subclass of Object
instance variables
\begin{array}{l}
certstore : \text{token} \xmapsto{m} \text{token};
\quad cs : \text{CardStorage};
\end{array}

operations
public

getToken : \text{token} \times \mathbb{N} \times (\text{ClaimType-set}) \to \text{token}
getchToken (RPcert, referenceCardId, requiredclms) \triangleq
\begin{array}{l}
\quad \text{if} (RPcert \notin (\text{dom certstore}))
\quad \quad \text{then certstore} := \text{certstore} \uplus \{RPcert \mapsto \text{generateNewKeyPair} ()\};
\quad \quad \text{dcl} c : \text{Card} := cs.\text{getCardByReferenceId} (referenceCardId),
\quad \quad t : \text{Claim-set};
\quad \quad \text{for all} p \in \text{requiredclms}
\end{array}
do t := t \cup \{c.content(p)\};
return signAndEncrypt(RPcert, certstore(RPcert), t)

);

signAndEncrypt : token \times token \times (Claim-set) \to token
signAndEncrypt (RPcert, keypair, clms) \triangleq
is not yet specified;
generateNewKeyPair : () \to token
generateNewKeyPair () \triangleq
is not yet specified

end SelfIdP
A.4 Application Package

Actual implementation including source code can be found in the included CD. Instructions of how to compile and run are also provided in the CD.