

Secure Roaming with Identity Metasystems

Long Nguyen Hoang
Helsinki University of Technology
Finland

Pekka Laitinen
Nokia Research Center
Finland

N. Asokan
Nokia Research Center
Finland

silver@cc.hut.fi

pekka.laitinen@nokia.com

n.asokan@nokia.com

ABSTRACT

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 a 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 metasystem client 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. We use our implementation to demonstrate how credentials are stored in a trusted device in the form of a mobile phone but can be used on less trusted terminals in the form of PCs.

Categories and Subject Descriptors

K.6.5 [Management of Computing and Information Systems]: Security and Protection, Authentication.

General Terms

Management, Design, Security, Human Factors.

Keywords

Identity metasystem, Mobility, Roaming.

1. IDENTITY METASYSTEMS

As more and more transactions are carried out over the Internet, the need for easy-to-use and secure authentication mechanisms becomes increasingly evident. This has resulted in a number of identity systems intended to save the users from having to create, manage, and use various passwords for different service providers. Each of these systems uses different protocols and requires different types of user interaction. More recently, the notion of Identity Metasystem [1] has gained ground. The main purpose of 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

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

IDTrust '08, March 4-6, 2008 Gaithersburg, MD.

Copyright 2008 ACM 978-1-60558-066-1...\$5.00.

other while providing a consistent user experience regardless of which identity system is used.

Technically, identity metasystem introduces the concept of an “information card” modeled after a business card, license, badge, etc. In general, an information card is a digital representation of user identity. A card may be managed or self-issued. Managed cards are issued by trusted authorities known as identity providers and contain only metadata describing how to get security tokens from those identity providers. Secret credential needed when generating authentication token are managed by those identity providers. Self-issued cards, on the other hand are managed by users themselves with the help of the local identity system on their devices. Secret credential information needed to generate security tokens for self-issued cards should be persistently stored on user devices themselves (or be provided by the user every time). This leads to the issue of roaming. To illustrate the issue, let us consider the following example scenario:

Alice has a number of information cards on her home PC. She uses them to access her web-based emails as well as for other services. When she decides to go on a trip, she loads her information cards on her mobile phone before leaving. When Alice arrives at the station, she notices some kiosk PCs that she can use to access the Internet. Alice wants to check her mail. Since she is unsure whether the kiosks are trustworthy, she does not want to transfer all of her self-issued cards to the kiosk machine. Yet, she would prefer to use the kiosk for reading email because it has a convenient display and keyboard. Alice starts up the identity metasystem on the kiosk as well as on her phone. They discover each other and establish a secure Bluetooth [10] connection. With a single click on her phone, Alice allows her information cards to appear on the identity selector user interface (UI) on the kiosk. When Alice attempts to sign in to her e-mail account, her phone prompts her for authorization instead of asking her to username/password on the terminal. She confirms on her phone and her mails can then be read on the kiosk as normal. When she finishes her reading, she leaves the kiosk and the Bluetooth connection terminates. Her information cards, as a consequence, disappear from the kiosk’s identity selector UI. In this example, although Alice is willing to trust the kiosk as the means to read her e-mail while she is physically present at the kiosk, she does not want to reveal the keys that can be used by malicious software on the kiosk to access her mail after she has left.

The most well-known identity metasystems so far are Microsoft CardSpace [2][3][4][5] and the open-source Bandit project [6]. Both CardSpace and Bandit allow the possibility to *export* information cards from one device and import them into another but the export/import operation results in the transfer of the entire

information card, including the secret credentials that are part of self-issued cards. Currently these systems are unable to support the usage scenario described above. To support such scenarios, the identity metasytem functionality should be split up into parts so that some can stay on a trusted device like the mobile phone, while the others can be migrated to a less trusted terminal temporarily. In this paper, we show that there are different ways to partition the functionality of an identity metasytem thereby allowing two or more devices to co-operate in providing the identity metasytem client functionality. Therefore we argue that in order to realize such scenarios, the identity metasytem client should be structured as a collection of distributable components.

In this paper, we propose a distributed software architecture for identity metasytem clients. We also study different ways to distribute client functionality among two or more devices. We then describe how we have implemented our architecture using the Bandit client implementation where one part of the client is running on a PC and the other on a mobile phone.

The rest of this paper is structured as follows. In Section 2, we give a brief review of the identity metasytems architecture in general and its current realizations. In Section 3, we describe our distributed architecture and discuss different ways to partition the client functionality between two devices. In Section 4 we describe the technical details on how we have adapted the Bandit identity metasytem to our architecture to support secure roaming. Finally, in Section 6, we discuss possible extensions and conclude.

2. IDENTITY METASYSTEMS

2.1 General Concept

The identity metasytem model emphasizes the roles of three key agents: the **relying party** (RP), the **identity provider** (IdP) and the **identity selector system** (ISS). A RP is any system capable of authenticating users based on their information cards, it can be a program, a web server, or any other service. The user is required to prove their identity before accessing to the resource. The IdP, as in Kim Cameron’s vision [7], issues information cards to user and provides authentication service. Finally, the identity selector is the system that is responsible for handling messages between an RP and an IdP, and providing a secure mechanism for user to manage their information cards. Figure 1 expresses a simple authentication process in identity metasytem. Whenever a user wants to access a service (e.g., a web page), the RP offering the service first sends the security requirements on user authentication it expects to be satisfied (which is stated in RP’s policy). On the user’s terminal, the identity selector interface processes these policies and automatically displays appropriate information cards that satisfy the policies. The user can choose one of these cards. It should be noted that, conceptually, an information card contains nothing but metadata saying only where and how to retrieve the identity information. Once the user selects a card, the identity selector facilitates the authentication of the user to the identity provider (which requires an additional authentication mechanism to be specified in the information card. For example, this authentication may be based on a username/password or a digital signature and certificates) and sends a request for security token based on RP’s policies. The IdP verifies the request and issues a security token in return. To complete the authentication process, the security token is passed

to the RP through the identity selector system to prove the user’s identity to the service.

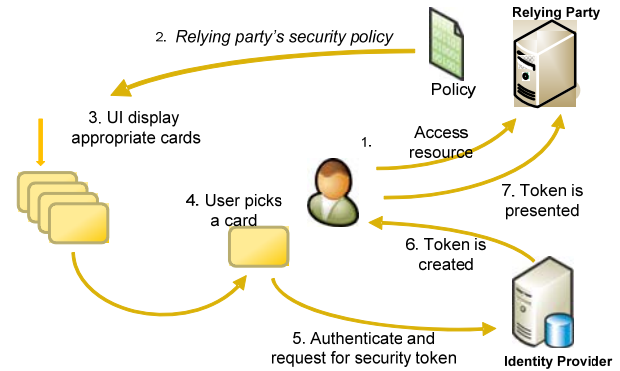


Figure 1: Identity metasytem authentication process

2.2 Microsoft CardSpace

Microsoft CardSpace [2][5], previously known as InfoCard, is the first commercial implementation of the identity metasytem concept and the one with the most widespread deployment since it is shipped as part of Windows Vista platform. Figure 2 depicts the system architecture of CardSpace. Services or applications on the platform trigger CardSpace system via an “*activator*” (*infocardapi*). The core part of the system, the CardSpace service handles all token and management requests, and invokes other services such as user interface or low-level storage. However, in the current version 1.0, secure roaming is not possible: in order to use the same information cards on multiple machines, the end user has to *export* his information cards to a file (with a *.crds* extension) and then *import* it into the CardSpace 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. If Alice in our example scenario in Section 1 uses CardSpace, she has to remember to remove all the cards including the secret data from the kiosk before she leaves. Moreover, even if she remembers to do this, she has no way of knowing if her cards and credentials had been copied before they were removed from the terminal.

2.3 Novell Bandit project

Bandit project [6] is an open-source, cross-platform implementation of the identity metasytem concept. Technically, Bandit consists of a set of loosely-coupled identity components that provide identity services while ensuring both interoperability and collaborations between agents. Figure 3 shows the high-level system architecture of Bandit. On the top is the management user interface, called *DigitalMe*. The main ISS module handles all requests. Bandit does not only specify and implement the identity selector as CardSpace does but also provides a common framework for easily integrating any digital identity management system to Bandit using Higgins framework [10]. Bandit supports the possibility of using multiple storage providers where information cards and other data used by the client can be stored persistently. Currently it allows the storage provider to be the local file system, or a Bluetooth-connected storage device, or an online database. As an open source project, Bandit constitutes an excellent platform for further development.

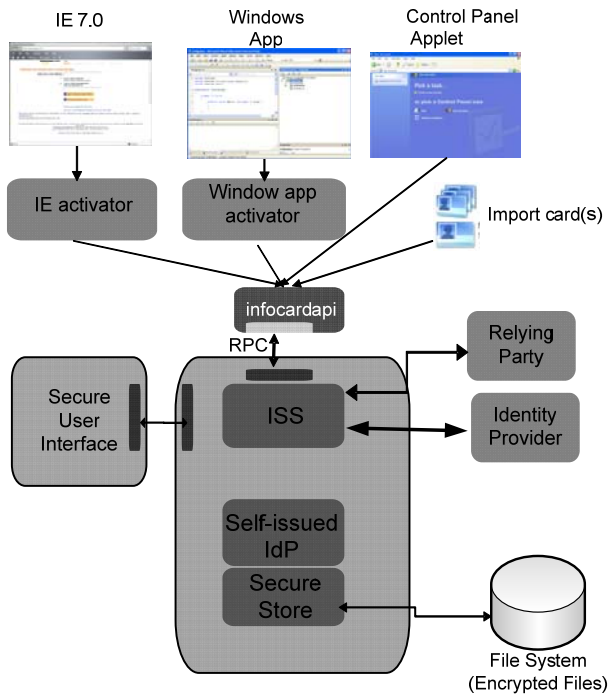


Figure 2: Microsoft CardSpace architecture [5]

3. ARCHITECTURE OF IDENTITY METASYSTEMS

3.1 General Architecture

There is no definitive identity metasytem architecture. In general, identity metasytems are expected to follow the principles set out by the “laws of identity” [7]. As we saw in Section 2, different realizations of the architecture have been implemented. Fortunately, those realizations have similar characteristics. Figure 4 depicts our own definition of identity metasytem architecture. The architecture consists of loosely-coupled identity components with interfaces between them:

Relying Party: the relying party can be considered a “remote” component in a whole identity metasytem. In general, a relying party is any system hosting some services which require authentication before being accessed. In the context of identity metasytem, a relying party first publishes its security policy (using [11][12]) then authenticates the user based on a security token-claim assertion.

Identity Selector: The role of the identity selector is to provide a consistent identity management tool for end users. This component also operates as a contextual connector between relying parties and identity providers such that the end user can choose which identity provider should be used to authenticate the end user to the particular relying party. Technically, the identity selector requests a security token from the identity provider and then passes that issued token to the relying party according to the information card chosen by the end user.

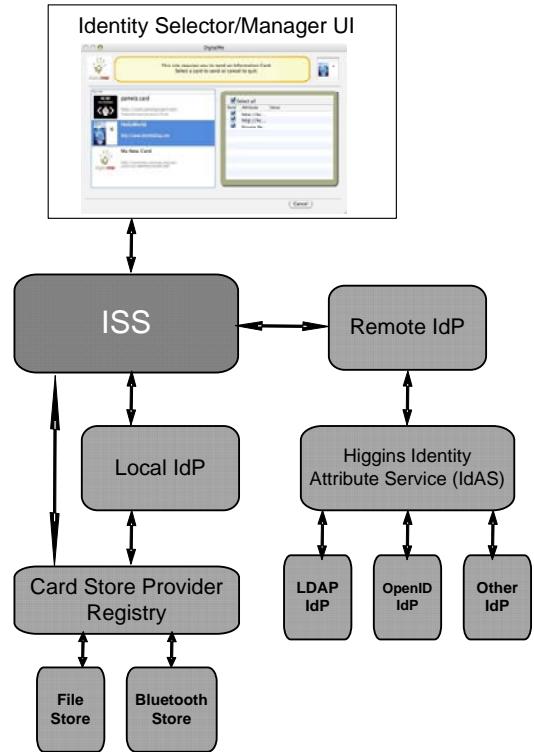


Figure 3: Novell Bandit architecture [6]

Self-issued IdP, Remote IdP: The identity provider is responsible for managing user’s digital identities in the form of information cards and issuing security tokens on demand. Remote IdP is maintained by a well-known, trusted organization while self-issued IdP is managed locally (in the operating system) by end users themselves. As the identity provider maintains all the user credentials, it must be secured. For example, the self-issued IdP can be protected using end user’s local trusted machine.

IdP Authentication: In order to obtain a security token from an IdP, the identity metasytem must first authenticate to the IdP that is specified in the chosen information card. Authentication mechanisms can be based on self-issued card token or other existing authentication technologies such as username/password, Kerberos [13], or Public Key Infrastructure (PKI)[14], depending on IdP’s security policy. In general, this component provides an abstract way to authenticate an end user to an identity provider.

Trigger: The trigger is a “bridge” between identity metasytem - aware applications and the identity selector system. Whenever an end user accesses a service that supports the identity metasytem, the trigger is used to activate the authentication process by first collecting the relying party’s policy, then activating the identity selector. At the end of the process, the trigger dispatches the security token returned from the identity selector to the relying party.

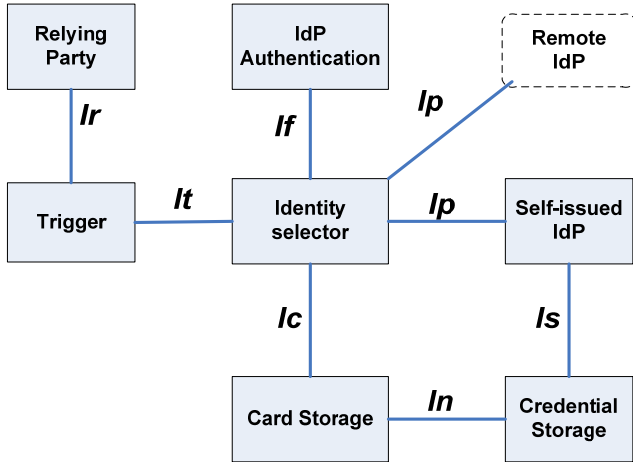


Figure 4: System components and interfaces

Card Storage: Responsibilities of the card storage include information card manipulation, card enumeration as well as filtering possible information cards based on what type identity information is required based on RPs' policies.

Credential Storage: The credential storage maintains user's credentials including personal information, certificates and other critical data. In practice, the credential storage and the card storage are typically combined. For example with self-issued cards, metadata and private user information are stored together in one XML document.

Every identity component is aware of other components. For example, component A may send a request to a specific component B (unicast) or a group of components (multicast) or all (broadcast). Here a middleware becomes useful because it provides a seamless mechanism for inter-component communication. We will discuss more about this in Section 4.

In distributed systems, components should communicate via well-defined interfaces. In Table A1, we list some common operations for each interface between the identity components in an identity metasystem. Figure A1 shows a simplified sequence diagram of identity component interaction for the scenario described in Section 1. The low-level connections between the identity components are assumed to be established beforehand (implicit setting, configuration setting, or dynamic discovery) and secure enough. If the transport layer is not secure enough then the security semantic needs to be improved in the application protocol layer between the distributed identity components which may be future topics. Interface **If** and **Ip** are special in that their binding protocols are decided on demand: security policy of identity provider decides the binding protocol of **Ip** and **If**. For example, if an IdP states in its policy that it requires username/password authentication, the identity selector must establish an SSL connection to the IdP, capture user's input from **If** and send those credentials through a secure channel over **Ip**. Alternatively, the authentication to the IdP can be done using a smart card (**If**) without having to use SSL connection. Binding protocols for other interfaces are unspecified, they can be RPC [15] calls or SOAP [16] requests or any custom-defined messages.

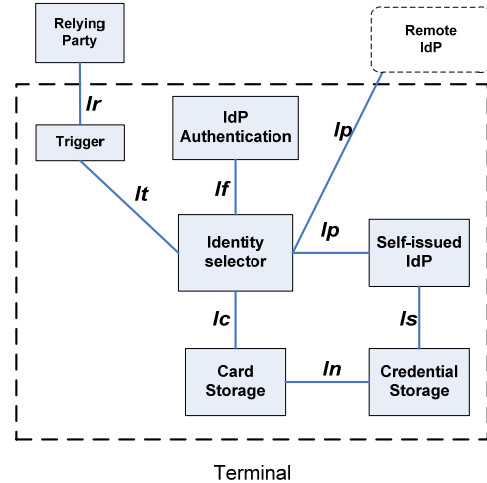


Figure 5: Configuration 0 – all in terminal

3.2 Distribution of Identity Components

An identity metasystem, which supports secure roaming, is expected to support various use cases in run time. For example, the 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, should adapt its functionality to different settings according to client profile set on the trusted device. This yields to a technical issue that identity metasystem components, not only within a system but also across distributed systems, should be able to seamlessly cooperate to complete a particular task. We call these use cases "configurations". In this section, we list out six typical configurations together with their strong points and drawbacks.

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.

Configuration 1 - "external storage": In this configuration, the trusted device provides credential storage service and card storage service. It is more like an external database such as a USB stick or a smartcard. The advantage of this configuration is that the trusted device does very little processing. It is noted that communicating channels between the host and the device must be protected as the self-issued IdP in the terminal accesses directly to the credential storage component running on the trusted device. However, to gain better performance, we can rely on the security of the transport protocol being used in **Is** as long as it is secure enough.

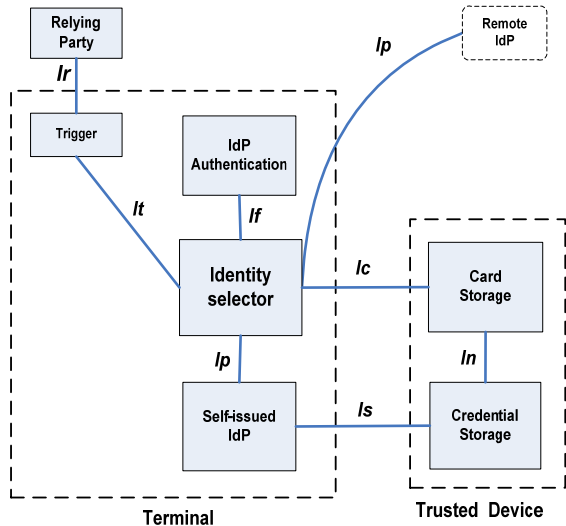


Figure 6: Configuration 1 – External storage

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 identity provider. 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's 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 trusted device. One tradeoff is the increase in the resource consumption in the trusted device when 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).

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 user interface on the trusted device so that he can choose one of his information cards. This gives 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.

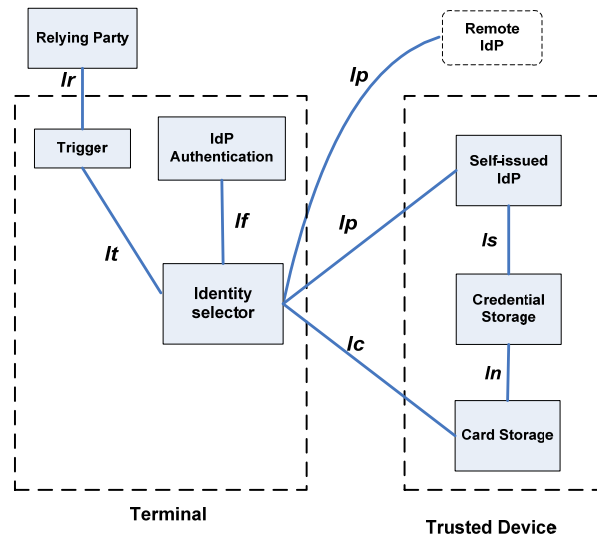


Figure 7: Configuration 2 – mobile identity provider

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 an identity metasystem is supported in the trusted device. 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.).

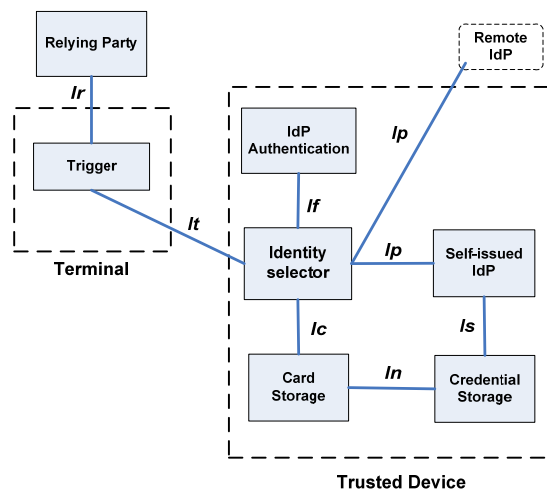


Figure 8: Configuration 3 – mobile identity selector

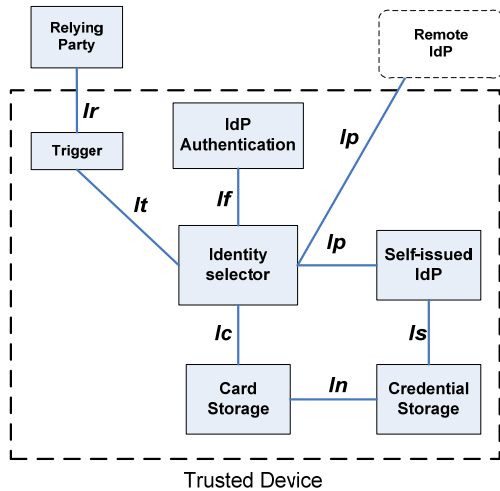


Figure 9: Configuration 4 – all in trusted device

Configuration 5 - "external service operator": This is a special configuration which is not possible with current technologies but we still describe it because 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 is accessing 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, logs in, and locks his profile usage for the lost device. The lost device poses no threat as there is no sensitive information stored on it.

4. IMPLEMENTATION

In the configurations listed in Section 3 there are two connected systems: one on a terminal and one on a trusted device; each system contains some of the identity components (e.g. identity selector, credential storage) and each of those components are expected to cooperate seamlessly regardless of whether they are located in the same platform or distributed across different systems. For example, authentication process in section 2.1 may be executed using any of the configurations listed in section 3.2 in which one identity component should be aware of others components. We have implemented the middleware for enabling seamless communication between the identity components. The result is that the main target **secure roaming of identity** becomes possible (with configurations 2, and 3).

4.1 High Level Architecture

The main idea is based on the Web Services model [17]. One identity component hosted by an identity system is considered a "service" of that system, and it can be discovered by other identity components. Components/services communicate using SOAP messages so that they can be implementation language and platform independent. In this section we propose an extension to Novell's Bandit project which supports configurations specified in Section 3.2. In our work, we adapt the implementation of -

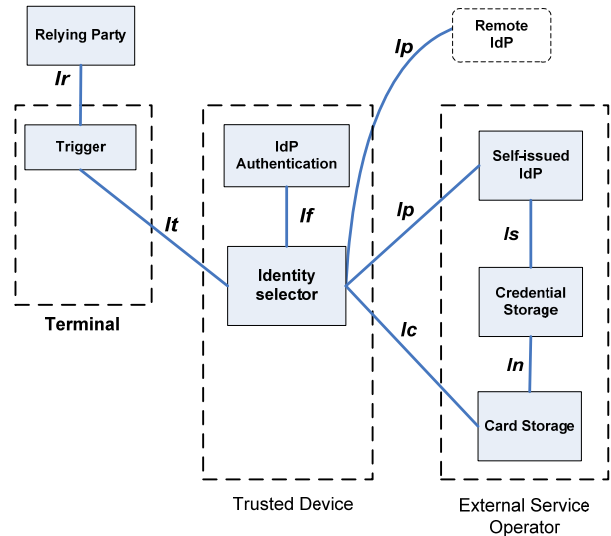


Figure 10: Configuration 5 – external service operator

identity components from the Bandit project to our middleware. Open-source identity components from other identity metasystem implementation can also fit into our middleware. Figure 11 shows the high-level architecture of our middleware. It is noted that so far only WS-Trust [18] is supported. Other methods, OpenID [19] for example, can be added to the system as an extension which can be considered in future study.

Connectivity Endpoint: Connectivity endpoint is used for transport-level messaging. It is also used for service broadcasting and discovery. Our middleware can support various types of endpoints (Bluetooth, USB, IP, IrDA, etc.). Each endpoint is controlled by a *Session Manager*.

Session Manager: This module handles the state of the each 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, the negotiated configuration, and the transport media being used. In addition, as a gateway for all incoming/outgoing messages, *Session Manager* can be a firewall, only allowing a certain set of configured operations from certain sources. For example in configuration 2, the trusted device allows only operations defined for *Ip* and *Ic* interfaces sent from the terminal.

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 the case where 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:

"<Container identifier> - <Component name>".

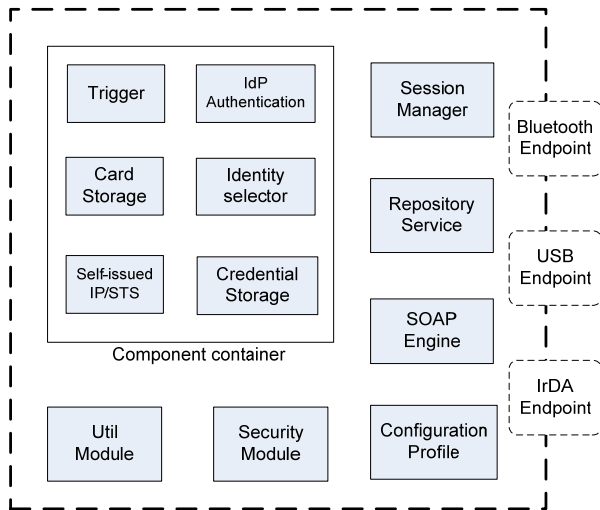


Figure 11: High-level system architecture

SOAP Engine: This module encapsulates requests from local identity components into SOAP requests [16] and parses SOAP responses sent from remote 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: The utility module provides some common functions for other modules (e.g. converter, encoder/decoder or file manipulation) as well as basic cryptography functions.

One user session goes through the following four phases:

Phase 1 - Connection setup: Before exchanging messages, the two systems must be able to locate each other and discover their offered services. This phase depends on the discovery mode of user's trusted device. 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. We assume there are discovery mechanisms that can be used by our system (for example, Devices Profile for Web Services [20]).

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 later. The terminal and the trusted device are now ready to exchange messages.

Phase 3 - Message exchange: 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 structure of the SOAP request is composed as follows:

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

The *Session Manager* determines how to reach the target component(s): if the target component belongs to local *Component Container*, the message is forwarded to that container which hosts the target component. Otherwise, the *Session Manager* passes the request to its *Endpoint* to be transferred to the remote system. The *Session Manager* on the remote machine receives the request from its *Endpoint* and delivers it to the *Component Container* hosting the target component. The container finally passes the request to the target identity component for processing. The response message follows the same path where the source and target have switched their places. The structure of the SOAP response is composed as follows:

```
<message>
  <response>
    <source>componentX</source>
    <target>componentY</target>
    <operation>OperationABC</operation>
    <status>error_code</status>
    <params>
      ParamsXYZ
    </params>
  </response>
</message>
```

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 mode (i.e., configuration 0).

4.2 Implementation Environment

We have implemented a prototype of our proposed extensions. In this section we describe the application flow of configuration 2 (mobile identity provider). On the terminal side, we use the identity components from Novell's Bandit project (implemented in C++ language) and wrap them in our middleware. On trusted device side, we decided to use J2ME [21] as a platform. This is both an advantage and a challenge since J2ME is widely supported on mobile phones but has a limited set of features. We have implemented the identity components for the J2ME environment from scratch, used open-source libraries such as kXML [22] for XML document processing and Bouncy Castle [23] for cryptographic operations.

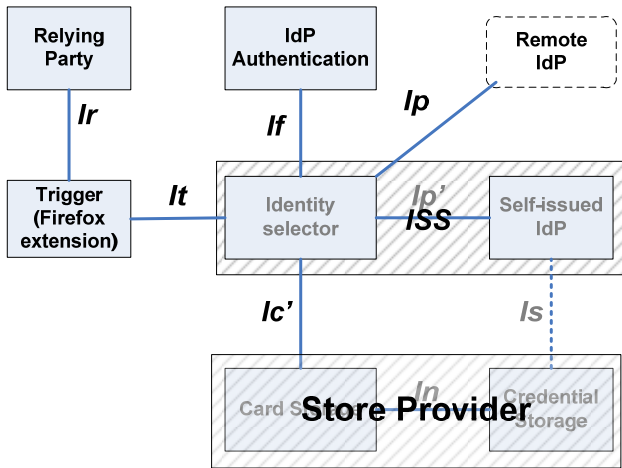


Figure 12: Bandit implementation

Let us start with the current implementation of Bandit project. Figure 12 shows Bandit's high-level design. The main module *ISS* loads complete set of available i-card documents [24] from the *Store Provider* through *Ic'*. To get a self-issued token, the selector passes an i-card object together with other parameters to the self-issued IdP module. This design causes security problems when we move to distributed environment because the identity selector has access to all secret information and keeps the loaded i-card object in memory. Besides, we have observed that the card enumeration is only needed in 3 cases. The first one is when populating and displaying user's information cards, which only needs card's metadata such as name, id, and picture. The second case is to import/export cards and this should be done directly from the storage component instead of selector component. The last case is to get the claim values and secret data in order to generate a self-issued security token. In this case, the self-issued IdP should have a direct and secure channel the credential store to access user's private data (*Is* in our design).

Therefore, we have made changes to Bandit's components and wrapped them in our architecture as described in 4.1. Table A2 compares the changes between the Bandit implementation and our current implementation.

In configuration 2, the user's information cards (self-issued or managed) are stored as i-card documents on the trusted device side. *Ic*-operations which involve card enumeration now only work with the metadata. We have defined some *filtering rules* to be used by Card Storage component to expose only metadata part of information card content (denoted by i-card* in Table A2). 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 use their information cards on the terminal identity selector. 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. All "metacards" and other temporary data such as history, cache, session on the terminal will be deleted when the trusted device disconnects.

For the *addCard*, *editCard* and *removeCard* operations, in the current implementation we assume that user can only add or edit cards from a trusted terminal like a home PC (or on the trusted device itself). These operations require that the terminal has been

explicitly declared trusted by the end user in configurations where *Ic* is a cross-system interface (configuration 2 for example).

When a relying party asks for a security token, Bandit's DigitalMe user interface on the terminal collects RP's certificate, policies, and requested claims. If the security token specified on RP's policy is expected to be from a managed identity provider, the authentication process is done normally as described in [3]. If the token is expected to be self-issued, the identity selector component constructs a request and sends it to the self-issued IdP component on the trusted device using our middleware. Once the self-issued IdP component receives the request, it generates a SAML token [25] with relevant attribute values extracted from the card store on the trusted device through *Is*. The security token is signed with a RP specific private key and encrypted by the public key of the RP before sending it back to the terminal and subsequently passed to the RP.

As described above, although the role of the trusted device in the authentication process is important, its functionality is quite simple. Most of complex processing including collecting certificate, user interface and service execution are done in the public terminal. The heaviest processing is to generate a specific RP-card key pair. Ideally, it should be done in the IdP in the trusted device. However, if the trusted device has limited computational power, RP-specific key pairs may be generated on a trusted terminal. Each RP specific key pair needs to be generated only once and transferred to the trusted device. One way to store the generated key pairs for later use is to put them in a new XML extension of the i-card format to store private data.

5. CONCLUSION

The notion of using a trusted device to secure transactions on a less trusted terminal is well-known [27][28][29]. However, these methods propose their own protocols/frameworks, making it difficult to interoperate with other identity systems. The distinguishing feature of our work is that we have shown how an existing identity metasystem can be extended to support the use of a trusted terminal. Our contribution 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. To summarize, we are able to move to the next level of current dimension of identity metasystem [30]. In our current work, we are planning to extend our implementation to support configuration 3 (mobile identity selector) and configuration 4 (all in mobile device) on various types of devices. We plan to support not only J2ME platform but also the Symbian operating system [26] because Symbian provides a strong platform for security processing plus a very good native user interface. In addition, we are developing mechanisms for securing inter-component communication which can be used instead of or in addition to any transport-level secure communication mechanisms. We also intend to study the usability and system performance aspects with the intention of improving the overall user experience. The final target is to give the end users better security and richer user experience.

6. REFERENCES

- [1] Microsoft organization. Microsoft's Vision for an Identity Metasystem. Microsoft Whitepaper, May 2005. <http://msdn2.microsoft.com/en-us/library/ms996422.aspx>http://zoo.cs.yale.edu/classes/cs457/tsui_digital_identity_management.doc
- [2] David Chappell. Introducing Windows CardSpace. Windows Vista Technical Articles, April 2006 <http://msdn2.microsoft.com/en-us/library/aa480189.aspx>
- [3] Arun Nanda, Microsoft Corporation. Identity Selector Interoperability Profile V1.0 April, 2007 <http://www.microsoft.com/downloads/details.aspx?FamilyID=B94817FC-3991-4DD0-8E85-B73E626F6764&displaylang=en>
- [4] Michael B. Jones. The Identity Metasystem: A User-Centric, Inclusive Web Authentication Solution. W3C Workshop on Transparency and Usability of Web Authentication. New York City, March 2006
- [5] CodeIdol.com. InfoCard Architecture and Security <http://codeidol.com/csharp/indigo/InfoCard/InfoCard-Architecture-and-Security/>
- [6] Novell corp. Bandit project http://www.bandit-project.org/index.php/Welcome_to_Bandit
- [7] Kim Cameron. The Laws of Identity. Microsoft Whitepaper, May 2005. <http://www.identityblog.com/stories/2004/12/09/thelaws.html>
- [8] Microsoft organization. Windows Data Protection. Microsoft MSDN, 2001. <http://msdn2.microsoft.com/en-us/library/ms995355.aspx>
- [9] Higgins Project. Higgins home page <http://www.eclipse.org/higgins/>
- [10] The Official Bluetooth® Technology Info Site <http://www.bluetooth.com/bluetooth/>
- [11] Web Services MetadataExchange. August 2006. <http://schemas.xmlsoap.org/ws/2004/09/mex/>
- [12] Web Services Policy Framework. March 2006. <http://schemas.xmlsoap.org/ws/2004/09/policy/>
- [13] Massachusetts Institute of Technology. Kerberos: The Network Authentication Protocol <http://web.mit.edu/Kerberos/>
- [14] Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 3280. April 2002. <http://tools.ietf.org/html/rfc3280>
- [15] Sun Microsystem. Remote Procedure Call. Request for comments 1831. August 1995. <http://tools.ietf.org/html/rfc1831>
- [16] W3C Working Group. Simple Object Access Protocol (SOAP). <http://www.w3.org/TR/soap/>
- [17] W3C Working Group. Web Services Architecture. February 2004 <http://www.w3.org/TR/ws-arch/>
- [18] OASIS. WS-Trust Version 1.3. March 2007 <http://docs.oasis-open.org/ws-sx/ws-trust/200512/ws-trust-1.3-os.html>
- [19] OpenID community. OpenID Authentication 2.0 - Draft 11. 2007. http://www.openid.net/specs/openid-authentication-2_0-11.html
- [20] Microsoft organization. Devices Profile for Web Services February 2006 <http://schemas.xmlsoap.org/ws/2006/02/devprof/>
- [21] Sun Microsystem. Java 2 Platform, Micro Edition (J2ME). Java ME homepage <http://java.sun.com/javame/index.jsp>
- [22] kXML A small XML pull parser <http://kxml.sourceforge.net/kxml2/>
- [23] The Legion of the Bouncy Castle. Bouncy Castle Java cryptography API <http://www.bouncycastle.org/java.html>
- [24] Information card format <http://en.wikipedia.org/wiki/I-Card>
- [25] OASIS. SAML Token Profile Version 1.1. February 2006 <http://docs.oasis-open.org/wss/oasis-wss-SAMLTOKENProfile-1.1>
- [26] Symbian Ltd. Symbian OS 2003 <http://www.symbian.com>
- [27] B. Parno, C. Kuo, and A. Perrig., Phoolproof phishing. In Proceedings of Financial Cryptography and Data Security 2006, Lecture Notes in Computer Science 4107, Springer.
- [28] M. Mannan, P.C. van Oorschot, Using a Personal Device to Strengthen Password Authentication from an Untrusted Computer. In Proceedings of Financial Cryptography and Data Security 2007.
- [29] D. Balfanz, E. Felten, Hand-held computers can be better smart cards. In Proceedings, 8th conference on USENIX Security Symposium - Volume 8, 1999.
- [30] Axel Nennker. The CardSpace dimensions. Published on Ignisvulpis's blog June 2007. <http://ignisvulpis.blogspot.com/>

APPENDIX

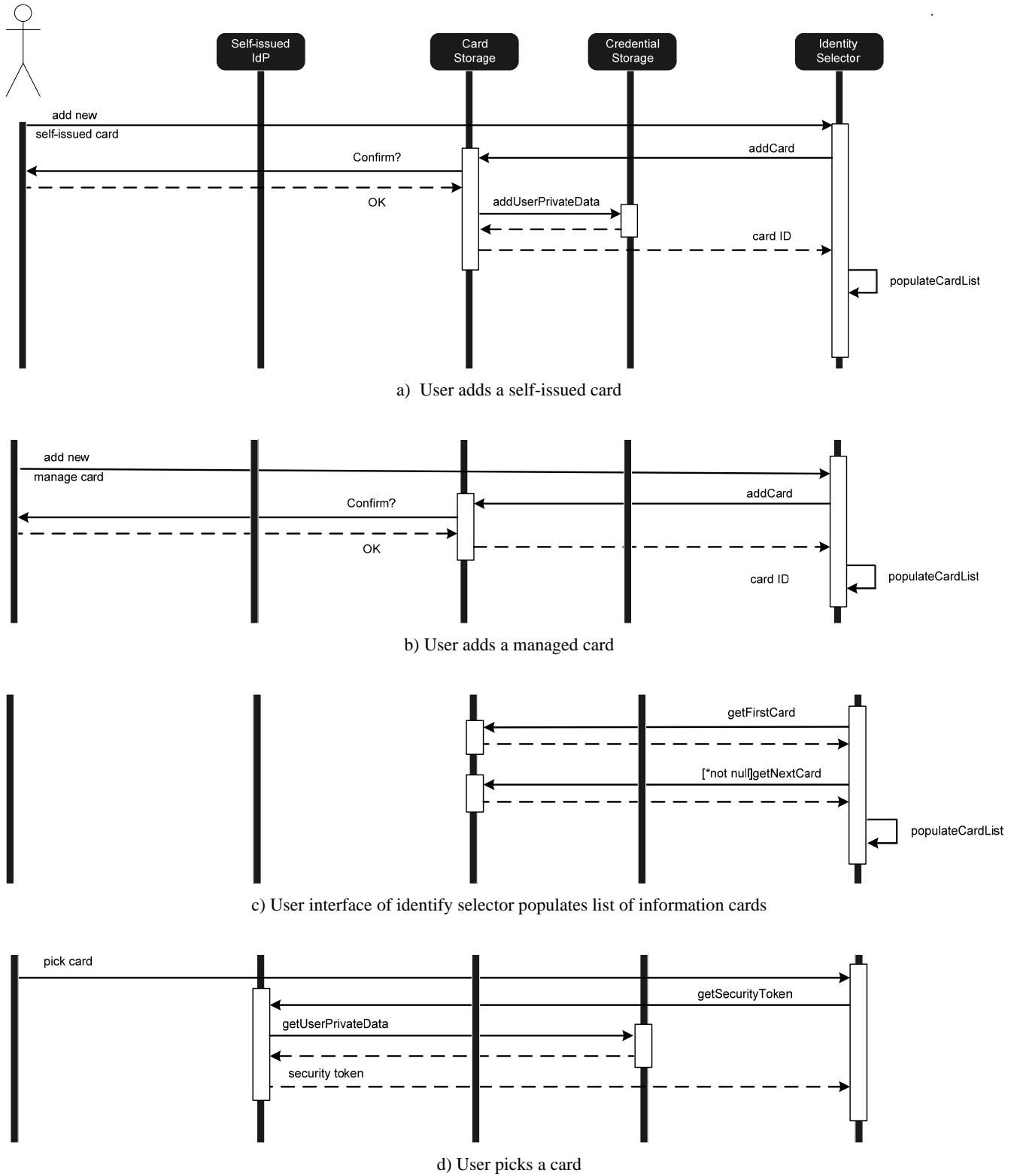


Figure A1: Interaction between identity components

Table A1: Interface operations

<i>Interface</i>	<i>Operation</i>	<i>Input</i>	<i>Output</i>	<i>Description</i>
<i>Ip</i>	<i>getSecurityToken</i>	token type , relying party, credential ¹ required claims optional claims ²	display token, security token	Get security token from identity provider
<i>Ic</i>	<i>getFirstCard</i>	N/A	i-card* ³	Get first card from store
	<i>getNextCard</i>	N/A	i-card* ³	Get next card from store
	<i>getCard</i>	N/A	i-card* ³	Get specific card from store
	<i>addCard</i>	i-card ⁴	card ID	Add one card to store
	<i>editCard</i>	card ID, i-card ⁴	N/A	Modify one card
	<i>removeCard</i>	card ID	N/A	Delete one card from store
<i>Ir</i>	<i>activateSelector</i>	x-informationCard	security token	Browser extension HTML parsing
<i>It</i>	<i>getToken</i>	token type , relying party, required claims optional claims	security token	Get security token from selector then pass it to relying party.
	<i>manageCards</i>	N/A	N/A	Open selector's user interface for card management
<i>If</i>	<i>getCredential</i>	N/A	credential	Obtain credential from user to authenticate to IdP
<i>Is</i>	<i>getUserPrivateData</i>	card ID	private data	Manipulate claim values and other info such as key-pair, master key, PIN.
	<i>addUserPrivateData</i>	card ID, private data	N/A	
	<i>editUserPrivateData</i>	card ID, private data	N/A	
	<i>removeUserPrivateData</i>	card ID, private data type	N/A	
<i>In</i>	<i>getCardContent</i>	card ID	i-card ⁴	called when exporting cards
	<i>addUserPrivateData</i>	card ID, private data	N/A	called when importing cards, adding personal cards
	<i>editUserPrivateData</i>	card ID, private data	N/A	called when edit personal cards
	<i>removeUserPrivateData</i>	card ID, private data type	N/A	

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* denotes filtered i-card document with only public metadata section

⁴ i-card denotes full i-card document

Table A2: Comparison of Bandit implementation and our implementation

Interface	Operation	Bandit implementation		Our implementation	
		Input	Output	Input	Output
Ip	<i>getSecurityToken</i>	token type , i-card, relying party, credential required claims optional claims	display token, security token	token type , <i>i-card*</i> , relying party, credential required claims optional claims	display token, security token
Ic	<i>getFirstCard</i>	N/A	i-card	N/A	<i>i-card*</i>
	<i>getNextCard</i>	N/A	i-card	N/A	<i>i-card*</i>
	<i>getCard</i>	card ID	i-card	card ID	<i>i-card*</i>
	<i>addCard</i>	i-card	card ID	i-card	card ID
	<i>editCard</i>	card ID, i-card	N/A	card ID, i-card	N/A
	<i>removeCard</i>	card ID	N/A	card ID	N/A
It	<i>getSecurityToken</i>	token type , relying party, required claims optional claims	security token	token type , relying party, required claims optional claims	security token
	<i>manageCards</i>	N/A	N/A	N/A	N/A
If	<i>getCredential</i>	N/A	credential	N/A	credential
Is	<i>getUserPrivateData</i>				
	<i>addUserPrivateData</i>				
	<i>editUserPrivateData</i>				
	<i>removeUserPrivateData</i>				
In	<i>getCardContent</i>				
	<i>addUserPrivateData</i>				
	<i>editUserPrivateData</i>				
	<i>removeUserPrivateData</i>				



Require access control for cross device boundaries.



Haven't been implemented yet. In current implementation, **Is** and **In**-operations are replaced by *getCard*, *addCard* with full i-card document