

Managing Identity Management Systems

Haitham S. Al-Sinani

Thesis submitted to the University of London
for the degree of Doctor of Philosophy



2012

Managing Identity Management Systems

Department of Mathematics
Royal Holloway, University of London

To my wife, Nisnas!

(Haitham)

Declaration of Authorship

I, Haitham S. Al-Sinani, hereby declare that these doctoral studies were conducted under the supervision of Professor Chris J. Mitchell.

The work presented in this thesis is the result of original research carried out by myself, in collaboration with others, whilst enrolled in the Department of Mathematics as a candidate for the degree of Doctor of Philosophy. This work has not been submitted for any other degree or award in any other university or educational establishment.

Signed:

(Haitham S. Al-Sinani)

Date:

Abstract

Although many identity management systems have been proposed, intended to improve the security and usability of user authentication, major adoption problems remain. In this thesis we propose a range of novel schemes to address issues acting as barriers to adoption, namely the lack of interoperability between systems, simple adoption strategies, and user security within such systems.

To enable interoperability, a client-based model is proposed supporting interworking between identity management systems. Information Card systems (e.g. CardSpace) are enhanced to enable a user to obtain a security token from an identity provider not supporting Information Cards; such a token, after encapsulation at the client, can be processed by an Information Card-enabled relying party. The approach involves supporting interoperability at the client, while maximising transparency to identity providers, relying parties and identity selectors. Four specific schemes conforming to the model are described, each of which has been prototyped. These schemes enable interoperability between an Information Card-enabled relying party and an identity provider supporting one of Liberty, Shibboleth, OpenID, or OAuth.

To facilitate adoption, novel schemes are proposed that enable Information Card systems to support password management and single sign on. The schemes do not require any changes to websites, and provide a simple, intuitive user experience through use of the identity selector interface. They familiarise users with Information Card systems, thereby potentially facilitating their future adoption.

To improve user security, an enhancement to Information Card system user authentication is proposed. During user authentication, a one-time password is sent to the user's mobile device which is then entered into the computer by the user.

Finally, a universal identity management tool is proposed, designed to support a wide range of systems using a single user interface. It provides a consistent user experience, addresses a range of security issues (e.g. phishing), and provides greater user control during authentication.

Acknowledgments

First and foremost, I am extremely grateful to Allah Almighty for giving me the strength and motivation to do my PhD. I am enormously indebted to His Majesty, Qaboos bin Said, the Sultan of Oman; without his great and wise leadership I would not have possibly found the necessary funding to do my PhD.

I would like to express my deepest gratitude to Professor Chris J. Mitchell, my PhD supervisor and role model. I have greatly benefited from his guidance, kindness, patience, support and interest, and I wish to say a heartfelt thank you to him. Indeed, without his insightful ideas, invaluable comments and precious feedback, this thesis would never have become reality.

By the completion of this thesis, I am approaching the end of a 20-year long life while I have been officially enrolled as a student. During those years, I have gratefully received the support of many people; to them all I wish to say: thank you!

I am particularly grateful to my father, Mr. Said Al-Sinani, my father-in-law, Major General Said Al-Salmi, my mother, Mrs. Aysha Al-Saidi, and my mother-in-law, Mrs. Salamah Al-Subhi, and to all of my family members and friends, for their endless support, continuous endorsement and enlightening advice; to them all I wish to say a sincere thank you.

I am profoundly appreciative to my beloved wife, Aisha Al-Salmi (Nisnas), for her endless support, continued encouragement and great patience. To her, I wish to say a heartfelt thank you.

Finally, I owe sincere thanks to the Diwan of Royal Court of the Sultanate

of Oman for sponsoring me during my PhD studies. The Diwan's sponsorship is very greatly appreciated.

Contents

Abbreviations	xxi
1 Introduction	33
1.1 Introduction	33
1.2 Motivation	33
1.2.1 Lack of Interoperation	34
1.2.2 Low Levels of Adoption	34
1.2.3 Insufficient Levels of Security	34
1.2.4 Lack of Consistency	35
1.3 Contributions	35
1.4 Thesis Structure	37
1.5 Publications	40
I Background	43
2 Identity, Privacy and Security	47
2.1 Introduction	47
2.2 Identity	47
2.2.1 Definition	47
2.2.2 Properties	49
2.2.3 A Life Cycle	50
2.3 Privacy	52
2.3.1 Definition	52

CONTENTS

2.3.2	Related Concepts	52
2.3.3	Categories of Personal Data	54
2.3.4	Threats	57
2.3.5	Protection	59
2.4	Security Services and Mechanisms	64
2.4.1	Definitions	64
2.4.2	Security Mechanisms	66
2.4.3	User Authentication	72
2.4.4	Key Management Techniques	76
2.5	Protocols and Standards	78
2.5.1	HTML	78
2.5.2	Document Object Model (DOM)	81
2.5.3	HTTP	82
2.5.4	SSL/TLS	87
2.5.5	HTTPS	88
2.5.6	Web Service Protocols	89
2.5.7	SAML	91
3	Identity Management	93
3.1	Introduction	93
3.2	Definition	93
3.3	Need for Identity Management	95
3.4	Abstract Model	95
3.5	Supporting Infrastructure	96
3.5.1	IdP-RP Communications	97
3.5.2	Discovery	99
3.6	Single Sign on (SSO)	100
3.7	Properties of Identity Management Systems	100
3.7.1	Information Card Systems	101
3.7.2	Federated Systems	102

3.7.3	Communication-based Models	102
3.7.4	Other Properties	103
3.8	Cameron's Laws of Identity	104
4	Identity Management Systems	107
4.1	Introduction	107
4.2	Microsoft Passport	107
4.2.1	Introduction	107
4.2.2	Operation	108
4.2.3	Criticism and Consequences	109
4.3	CardSpace	110
4.3.1	Introduction	110
4.3.2	InfoCard Contents	111
4.3.3	Attribute Exchange	113
4.3.4	IdP Discovery	114
4.3.5	IdP-RP Negotiation	114
4.3.6	User Control and Consent	115
4.3.7	Supporting CardSpace	115
4.3.8	Security Policy	115
4.3.9	Operation	117
4.3.10	Token Processing	119
4.3.11	PPIDs and Digital Signatures	121
4.3.12	Proof of Ownership	123
4.3.13	IdPs and Auditing	125
4.3.14	Possible Limitations of CardSpace	127
4.4	Higgins	130
4.4.1	Introduction	130
4.4.2	InfoCards	130
4.4.3	Data Model	131
4.4.4	Architecture	131

4.4.5	Categories	133
4.4.6	Possible Limitations of Higgins	136
4.5	OpenID	136
4.5.1	Introduction	136
4.5.2	IdP Discovery	137
4.5.3	IdP-RP Negotiation	138
4.5.4	Identity Federation	138
4.5.5	User Control and Consent	139
4.5.6	Level of Assurance	139
4.5.7	Supporting OpenID	139
4.5.8	Operation	140
4.5.9	Attribute Exchange	142
4.5.10	Proof of Ownership	144
4.5.11	Possible Limitations of OpenID	144
4.6	OAuth	145
4.6.1	Introduction	145
4.6.2	User Control and Consent	146
4.6.3	Operation	147
4.6.4	Facebook Connect	151
4.6.5	Possible Limitations of OAuth	151
4.7	Liberty	151
4.7.1	Introduction	151
4.7.2	Supported Functionality	152
4.7.3	Attribute Exchange	153
4.7.4	User Control and Consent	154
4.7.5	IdP Discovery	154
4.7.6	Negotiation	155
4.7.7	SSO and Federation Profiles	155
4.7.8	Operation	157
4.7.9	Proof of Ownership	161

4.7.10 Possible Limitations of Liberty	161
4.8 Shibboleth	162
4.8.1 Introduction	162
4.8.2 Attribute Exchange	162
4.8.3 Architecture and IdP Discovery	163
4.8.4 Identity Federation	163
4.8.5 Shibboleth Profiles	163
4.8.6 Operation	164
4.8.7 Proof of Ownership	165
4.8.8 Possible Limitations of Shibboleth	166
4.9 Anonymous Credential Systems	167
4.9.1 Overview	167
4.9.2 U-Prove	169
4.9.3 IdeMix	176
4.10 Comparison	178
II Interoperability	181
5 A General Interoperation Model	185
5.1 Introduction	185
5.1.1 Overview	185
5.1.2 Motivation	186
5.1.3 Organisation	187
5.2 The Interoperation Model	187
5.2.1 System Entities	188
5.2.2 Overview of Operation	188
5.2.3 Requirements	189
5.2.4 Operation	191
5.3 Operational Issues	195
5.3.1 Triggering the Adaptor	195

5.3.2	Attribute Handling	196
5.3.3	Implementing the Adaptor as a Browser Extension	197
5.4	Advantages	197
5.4.1	Defeating Fake IdP Attacks	197
5.4.2	Client Interoperation	198
5.4.3	Consistency	198
5.4.4	Unintentional Leakage	198
5.5	Security Considerations	199
5.6	Potential Issues	200
5.7	Possible Extensions	201
5.7.1	Scope	201
5.7.2	U-Prove Tokens	201
5.8	Related Work	202
5.8.1	Specification and Open-source Development Projects	202
5.8.2	General-purpose Interoperation Models	204
5.8.3	Interoperation Between Specific Systems	205
5.8.4	Business Analysis of Interoperation Issues	206
5.9	Conclusions and Future Work	207
6	Interoperation Between an Information Card System and Liberty	209
6.1	Introduction	209
6.2	Interoperating with Liberty	210
6.2.1	System Entities	210
6.2.2	Requirements	210
6.2.3	Operation	211
6.2.4	Liberty Profiles	213
6.3	Discussion and Analysis	214
6.3.1	Applicability	214
6.3.2	Differences in Scope	216
6.3.3	Token Forwarding	217

6.3.4	Possible Extensions	219
6.4	Prototype Realisation	220
6.4.1	User Registration	220
6.4.2	Implementation Details	220
6.4.3	Prototype Operation	221
6.4.4	Potential Features and Issues	225
6.5	Related Work	226
6.6	Conclusions and Future Work	228
7	Enabling Interoperation Between Shibboleth and Information	
	Card Systems	231
7.1	Introduction	231
7.2	Interoperating with Shibboleth	232
7.2.1	Requirements	232
7.2.2	Operation	233
7.3	Implementation Issues	234
7.3.1	Token Storage and Forwarding	234
7.3.2	Attribute Handling	236
7.3.3	Possible Extensions	236
7.4	Prototype Realisation	237
7.4.1	Implementation Details	237
7.4.2	Prototype Operation	237
7.4.3	Potential Features and Issues	239
7.5	Related Work	239
7.6	Conclusions and Future Work	240
8	Client-based Interoperation Between OpenID and Information	
	Card Systems	241
8.1	Introduction	241
8.2	Interoperating with OpenID	242
8.2.1	Requirements	242

8.2.2	Operation	242
8.3	Discussion and Analysis	245
8.3.1	IDcard Contents	245
8.3.2	IdP User Authentication	247
8.3.3	Security Considerations	247
8.3.4	Attribute Mapping	249
8.4	Prototype Realisation	250
8.4.1	User Registration	250
8.4.2	Prototype Operation	250
8.4.3	Potential Features and Issues	254
8.5	Related Work	254
8.6	Conclusions and Future Work	255
9	Integrating OAuth with Information Card Systems	257
9.1	Introduction	257
9.2	Interoperating with OAuth	258
9.2.1	Requirements	258
9.2.2	Operation	259
9.3	Discussion and Analysis	261
9.3.1	Security Considerations	261
9.3.2	Attribute Mapping	263
9.4	Prototype Realisation	263
9.4.1	Registration	264
9.4.2	Prototype Operation	264
9.4.3	Potential Features and Issues	267
9.5	Conclusions and Future Work	267
III	Practicality and Security	269
10	Using an Information Card System as a Password Manager	273
10.1	Introduction	273

10.2 PassCard	275
10.2.1 Prerequisites	275
10.2.2 Operation	276
10.2.3 Discussion	279
10.3 Prototype Realisation	281
10.3.1 Registration	281
10.3.2 Operation	282
10.3.3 Discussion	288
10.4 PassCard Properties	292
10.4.1 Security	292
10.4.2 Usability	293
10.4.3 Limitations	293
10.5 Related Work	295
10.6 Conclusions and Future Work	296
11 Using an Information Card system as a Password-based SSO System	297
tem	297
11.1 Introduction	297
11.2 SingleSigner	298
11.2.1 Prerequisites	299
11.2.2 Operation	300
11.3 Implementation	303
11.3.1 Shared Properties	303
11.3.2 The URL Query Parameters Prototype	306
11.3.3 The Cookies Prototype	307
11.3.4 The Hidden Form Fields Prototype	308
11.4 Comparison	309
11.5 Discussions	310
11.5.1 Features	310
11.5.2 Limitations	310

11.5.3 Enhancements	311
11.6 Related Work	312
11.7 Conclusions and Future Work	313
12 Enhancing User Authentication in Information Card Systems	315
12.1 Introduction	315
12.2 The Scheme	317
12.2.1 System Entities	317
12.2.2 Operation	318
12.3 Discussion	320
12.3.1 Implementation Issues	320
12.3.2 Variants of the Scheme	320
12.3.3 Advantages	321
12.4 Security Analysis	322
12.4.1 Threats to the Mobile Device	322
12.4.2 Threats to the Supporting Infrastructure	322
12.4.3 Threats to the PC	322
12.5 Prototype Realisation	324
12.5.1 User Registration	325
12.5.2 Prototype Operation	325
12.5.3 Practical Issues	327
12.6 Related Work	327
12.7 Conclusions and Future Work	331
IV Universality	333
13 A Universal Client-based Identity Management Tool	337
13.1 Introduction	337
13.1.1 The Need for Authentication	337
13.1.2 Identity Management	338
13.1.3 A New Approach	339

13.1.4 CardSpace	340
13.1.5 Organisation	340
13.2 IDSpace	341
13.3 High-level Architecture	343
13.3.1 Context of Use	343
13.3.2 IDSpace Components	345
13.4 Supporting Functionality	350
13.4.1 Identity System Discovery	351
13.4.2 Identity System Selection	351
13.4.3 Card Selector Invocation	353
13.4.4 cCard Storage	353
13.4.5 cCard Format	354
13.4.6 cCard Contents	354
13.4.7 Process Isolation	355
13.4.8 Authentication Methods	355
13.5 IDSpace Operation	356
13.5.1 Initialisation	356
13.5.2 Protocol Flows	358
13.6 Mapping Specific Protocol Architectures onto IDSpace	364
13.6.1 IDSpace and OpenID	365
13.6.2 IDSpace and LEC	368
13.6.3 IDSpace and CardSpace	371
13.7 Implementation	371
13.8 Concluding Remarks	375
13.8.1 Relationship to the Prior Art	376
13.8.2 Novel Features	377
13.8.3 Future Work	377

CONTENTS

V Conclusions	379
14 Conclusions and Future Work	383
14.1 Summary and Conclusions	383
14.2 Possible Future Work	386
Bibliography	389

List of Figures

2.1	An Identity Life Cycle	51
2.2	CIA Triangle	65
2.3	A Simple Username-password HTML Form	81
2.4	Web Services Technologies	90
3.1	Identity Management Model	97
4.1	The CardSpace Identity Selector	112
4.2	CardSpace Operation (Using Managed Cards)	120
4.3	PPID and Signature Key Pair for Personal Cards	122
4.4	Higgins Architecture	132
4.5	An OpenID Login Form (Taken From <code>openid.net</code>)	140
4.6	OpenID Operation in <i>check_setup</i> Mode	143
4.7	Overview of OAuth Operation	148
4.8	A Liberty Circle of Trust	152
4.9	Operation of Liberty Browser-post and Artifact Profiles	159
4.10	Overview of LECP Operation	160
4.11	Overview of Shibboleth Operation	166
5.1	Interoperation Model Operation	189
6.1	Data Flows via Client Components	211
6.2	Exchanges Between the Principal Parties	215
6.3	A LibCard	221
7.1	Protocol Exchanges	235

LIST OF FIGURES

8.1 Protocol Exchanges	246
9.1 Protocol Exchanges	261
10.1 PassCard Operation in HTTP mode	279
10.2 PassCards	282
10.3 PassCard Co-operating with a CardSpace-enabled RP	285
10.4 PassCard Logo	286
10.5 Redirect URL (target URL → HS)	286
10.6 Redirect URL (HS → target URL)	287
11.1 SingleSigner Operation	304
11.2 SingleSigner Logo	306
12.1 Summary of the Protocol	319
12.2 Protocol Exchanges	319
13.1 IDSpace Context	343
13.2 IDSpace Components	346
13.3 The IDSpace Card Selector	373
13.4 Resized Screenshot of an XML-based, Encrypted cCard	373
13.5 An IDSpace PassCard	374
13.6 IDSpace IDcards	374
13.7 An IDSpace IDcard	375

List of Tables

2.1	HTTP Request and Response Messages	84
4.1	Auditing IdPs versus Non-auditing IdPs	127
4.2	General Comparison Between Identity Management Systems . . .	179
7.1	CardSpace-Shibboleth Attribute Mapping	236
8.1	CardSpace-OpenID Attribute Mapping	249
9.1	CardSpace-Facebook Connect Attribute Mapping	263

Abbreviations

ACM Association of Computing Machinery

AES Advanced Encryption Standard

API Application Programming Interface

APPEL A P3P Preference Exchange Language

AX Attribute eXchange

CA Certification Authority

CBC Cipher-block Chaining

CIA Confidentiality, Integrity, and Availability

CoT Circle of Trust

CRL Certificate Revocation List

DES Data Encryption Standard

DHKE Diffie-Hellman Key Exchange

DNS Domain Name System

DOM Document Object Model

EPAL Enterprise Privacy Authorization Language

GPS Global Positioning System

GUI Graphical User Interface

LIST OF TABLES

HTML HyperText Markup Language

HTTP Hypertext Transfer Protocol

HTTPS Hypertext Transfer Protocol Secure

I2P Invisible Internet Project

IBM International Business Machines (Corporation)

ICT Information and Communication Technology

IdP Identity Provider

IEEE Institute of Electrical and Electronics Engineers

IETF Internet Engineering Task Force

IP Internet Protocol

ISP Internet Service Provider

LEC Liberty-enabled Client

LECP Liberty-enabled Client and Proxy Profile

Liberty ID-FF Liberty Identity Federation Framework

Liberty ID-SIS Liberty Identity Service Interface Specifications

Liberty ID-WSF Liberty Identity Web Services Framework

LIP Local Identity Provider

LNCS Lecture Notes in Computer Science

LNEE Lecture Notes in Electrical Engineering

MAC Message Authentication Code

MITM Man-in-the-Middle

NIST National Institute of Standards and Technology

Nonce Number used once

OECD Organisation for Economic Co-operation and Development

OMB Office of Management and Budget

OP OpenID Provider

OS Operating System

OTP One-time Password

P3P Platform for Privacy Preferences

PII Personally Identifiable Information

PIN Personal Identification Number

PKI Public Key Infrastructure

RA Registration Authority

RP Relying Party

RSA Rivest-Shamir-Adleman

RSTRC Request Security Token Response Collection

RSTR Request Security Token Response

RST Request Security Token

SAML Security Assertion Markup Language

SIIP Self-issued Identity Provider

SIM Subscriber Identity Module

SIP Session Initiation Protocol

LIST OF TABLES

SMS Short Message Service

SP Service Provider

SREG Simple Registration OpenID Extension

SSL Secure Sockets Layer

SSO Single Sign On

STS Security Token Service

TGC Ticket Granting Cookie

TLS Transport Layer Security

TOR The Onion Router

TPM Trusted Platform Module

TTP Trusted Third Party

UA User Agent

UDDI Universal Description, Discovery and Integration

URI Uniform Resource Identifier

URL Uniform Resource Locator

W3C World Wide Web Consortium

WAYF Where Are You From

WSDL Web Services Description Language

WWW World Wide Web

XACML eXtensible Access Control Markup Language

XHTML eXtensible Hyper Text Markup Language

XML eXtensible Markup Language

XRDS eXtensible Resource Descriptor Sequence

XRI eXtensible Resource Identifier

Introduction

1.1 Introduction

This chapter provides an introduction to the rest of the thesis, and it is organised as follows. Section 1.2 outlines the research motivations of this thesis, and, in section 1.3, we summarise its main contributions. Section 1.4 describes the structure of the thesis, and, finally, section 1.5 concludes the chapter by listing relevant publications.

1.2 Motivation

In line with the continuing increase in the number of on-line services requiring authentication, there has been a proportional rise in the number of user-possessed digital identities needed for authentication purposes. This has contributed to the recent rapid growth in identity-oriented attacks, such as phishing, pharming, etc. One indication of this trend is that identity fraud rose by 13% in 2011, according to a specialised report on identity fraud released in 2012 by Javelin Strategy & Research¹.

In an attempt to mitigate such attacks, a relatively large number of identity management systems have recently been proposed. Unfortunately, although these systems have the potential to reduce the threat of identity attacks and improve user security, major problems, as described below, remain.

¹<http://www.businesswire.com/news/home/20120222005485/en/Identity-Fraud-Rose-13-Percent-2011-Javelin>

1.2.1 Lack of Interoperation

The vast majority of identity management systems are not interoperable. That is, a security token issued by an identity provider for one system cannot be used at a relying party supporting another system. This is likely to become a major usability issue in practice. Therefore, to make these systems available to the largest possible group of users, effective interoperability between identity management systems is needed.

1.2.2 Low Levels of Adoption

Despite the introduction of many identity management systems, the vast majority of websites still use username-password for authentication, and this is likely to continue for at least the next few years [110]. One major problem with those identity management systems that are based on Information Cards (see section 3.7.1), and with other similar systems providing more secure means of user authentication, is that the transition from username-password authentication is extremely difficult to achieve. Relying parties will not wish to do the work necessary to support an identity management system if very few users employ it; equally, users are hardly likely to use a sophisticated system if it is only supported by a tiny minority of websites.

1.2.3 Insufficient Levels of Security

Some identity management systems, including Information Card systems such as CardSpace, are susceptible to attacks arising from temporary unauthorised access to a user system, i.e. they offer inadequate levels of security for sensitive identity information stored in the client device.

In addition, many identity management systems are susceptible to fake identity provider attacks, in which a malicious relying party redirects a user browser to a false identity provider. The user then reveals to the fake identity provider secrets that are shared with a genuine identity provider. This

arises because, in the absence of a system-aware client agent, many identity management systems rely on browser redirects.

1.2.4 Lack of Consistency

The user experience of almost every identity management system is different, and this is likely to lead to user confusion and hence potentially give rise to security breaches. It is widely acknowledged that users fail to make good security decisions, even when confronted with relatively simple decisions [157]. The lack of consistency is likely to make the situation much worse, with users simply not understanding the complex privacy- and security-relevant decisions that they are being asked to make.

1.3 Contributions

In this thesis, we propose a number of novel schemes to address the problems identified in section 1.2, i.e. to enable interoperation between identity management systems, to increase the rate of their adoption, to improve user security (particularly user authentication), and to provide a consistent user experience. The main contributions of this thesis are as follows.

- To enable interoperation (see section 1.2.1), a novel model is proposed that supports interworking between Information Card-based identity management systems (including CardSpace and Higgins) and identity management systems not supporting Information Cards, such as OpenID, OAuth, Liberty, and Shibboleth. In this model, Information Card users are able to obtain a security token from an identity provider not supporting Information Cards, the contents of which can be processed by an Information Card-enabled relying party. The model we propose involves providing interoperation functionality at the client in a way that is as transparent as possible to identity providers, relying parties and identity selectors. Four specific schemes conforming

to this model are then described. These schemes enable interoperation between an Information Card-enabled relying party and an identity provider supporting one of Liberty, Shibboleth, OpenID, or OAuth.

- To enhance practicality and adoption (see section 1.2.2), two novel schemes are proposed that extend Information Card systems to enable them to support password management and password-based single sign on. These schemes do not require any changes to the servers of accessed web sites. The schemes provide a simple, intuitive user experience through their use of the identity selector interface. At the same time, they familiarise users with Information Card systems, thereby potentially facilitating future adoption of more secure means of authentication.
- To improve user security (see section 1.2.3), a novel scheme is proposed that enhances user authentication when using an Information Card system. During the process of user authentication on a personal computer (PC) using an Information Card system, a random and short-lived one-time password is sent to the user's mobile device; this must then be entered into the PC by the user when prompted.
- A novel, universal identity management tool is proposed, designed to support a wide range of identity management systems using a single user interface. The tool provides a consistent user experience, addresses a range of possible security issues (including phishing), and allows for greater user control during the authentication process (see section 1.2.4).

All the novel schemes proposed in this thesis have been successfully prototyped.

1.4 Thesis Structure

This thesis is divided into five parts organised as follows.

1. *Part I* provides background material for the rest of the thesis. It contains three chapters, as follows.
 - *Chapter 2* gives an introduction to, and definitions for, the concepts of identity, privacy, and security, as well as outlining associated protocols of importance in this thesis.
 - *Chapter 3* provides an introduction to identity management, covering related topics such as single sign on and Cameron's identity laws. The chapter also gives an abstract model for identity management, and considers a range of properties possessed by different types of identity management systems.
 - *Chapter 4* describes in detail those identity management systems of greatest relevance to this thesis, namely CardSpace, Higgins, OpenID, OAuth, Liberty, and Shibboleth. The chapter also gives an overview of certain other systems of background relevance, namely Microsoft Passport, U-Prove, and IdeMix.
2. *Part II* proposes a novel approach to supporting interoperation between a wide range of identity management systems. This part contains a total of five chapters. We first, in *chapter 5*, describe a general model for interoperation between an Information Card-based identity management system and almost any other existing identity management system. Using this model, Information Card users are able to obtain a security token from an identity provider not supporting Information Cards. After processing at the client, an enhanced token is produced that can be processed by an Information Card-enabled relying party. We then go on to describe four specific instantiations of this

model, enabling interoperation between an Information Card system and:

- Liberty (*chapter 6*);
- Shibboleth (*chapter 7*);
- OpenID (*chapter 8*); and
- OAuth (*chapter 9*).

3. *Part III* proposes novel schemes to enhance the practicality and security of identity management systems. This part contains three chapters, as follows.

- *Chapter 10* proposes a novel scheme that allows an Information Card system to be used as a password manager. Usernames and passwords are stored in personal cards, and these cards can be used to sign-on transparently to relevant websites. The scheme does not require any changes to login servers, default browser security settings, or to identity selectors; in particular, it does not require websites to support an Information Card system. The chapter describes how the scheme operates, and also gives details of a proof-of-concept prototype. Security and usability analyses are also provided.
- *Chapter 11* proposes a scheme that allows an Information Card system to be used as a password-based single sign on system. The chapter describes three related approaches to achieving single sign on using an Information Card system. In each case users are able to store credentials for a set of websites in a single personal card, and use it to seamlessly sign-on to all these websites. The approaches do not require any changes to login servers or to identity selectors and, in particular, they do not require websites to support Information Cards. The chapter also describes three proof-of-

concept prototypes and gives usability, security and performance analyses. Note that both chapters 10 and 11 are concerned with techniques intended to help improve the usability and security of password use, as well as potentially encouraging adoption of Information Card systems.

- *Chapter 12* proposes a scheme that uses a mobile device to enhance user authentication in Information Card systems. During the process of user authentication on a PC using an Information Card system, a random and short-lived one-time password is sent to the user's mobile device; this must then be entered into the PC by the user when prompted. The scheme does not require any changes to login servers, identity selectors, or to the mobile device itself. The chapter specifies how the scheme operates and gives details of a proof-of-concept prototype. Security and operational analyses are also provided.
4. *Part IV* proposes a universal identity management tool that can support a wide range of identity management systems using a single user interface. The tool is designed to enhance user privacy and to address a range of security issues, notably phishing attacks. This part consists of a single chapter, *chapter 13*, which describes this client-based identity tool. The goal of this tool is to simplify the use of a wide range of existing identity technologies, helping to encourage their use whilst imposing no additional burden on relying parties and identity providers. The chapter also describes examples of the operation of the scheme with certain existing identity management systems.
 5. *Part V* concludes the thesis by summarising the main contributions as well as highlighting possible areas for future work. This part of the thesis consists of a single chapter, *chapter 14*.

1.5 Publications

Publications describing some of the research results contained in this thesis are listed below (in chronological order of publication).

1. Haitham S. Al-Sinani, Waleed A. Alrodhan, and Chris J. Mitchell. CardSpace-Liberty integration for CardSpace users. In Ken Klingenstein and Carl M. Ellison, editors, *Proceedings of IDtrust '10 — the 9th Symposium on Identity and Trust on the Internet, April 13–15, 2010, Gaithersburg, Maryland*, pages 12–25. ACM, New York, 2010.
2. Haitham S. Al-Sinani and Chris J. Mitchell. Using CardSpace as a password manager. In Elisabeth de Leeuw, Simone Fischer-Hübner, and Lothar Fritsch, editors, *Proceedings of IFIP IDMAN '10 — the 2nd IFIP WG 11.6 Working Conference on Policies and Research in Identity Management, November 18–19, 2010, Oslo, Norway*, volume 343 of *IFIP Advances in Information and Communication Technology*, pages 18–30. Springer, Boston, 2010.
3. Haitham S. Al-Sinani and Chris J. Mitchell. CardSpace-Shibboleth integration for CardSpace users. In *ACNS '11 [industrial track proceedings], the 9th International Conference on Applied Cryptography and Network Security, June 7–10, 2011, Nerja, Malaga, Spain*, pages 49–66, 2011.
4. Haitham S. Al-Sinani and Chris J. Mitchell. Enhancing CardSpace authentication using a mobile device. In Yingjiu Li, editor, *Proceedings of DBSEC '11 — the 25th IFIP WG 11.3 Conference on Data and Applications Security and Privacy, July 11–13, 2011, Richmond, Virginia*, volume 6818 of *LNCS*, pages 201–216. Springer-Verlag, Berlin, 2011.
5. Haitham S. Al-Sinani and Chris J. Mitchell. *Method and apparatus for enabling authorised users to access computer resources*. UK patent application GB1115866.4, filed 14th September 2011.

6. Haitham S. Al-Sinani and Chris J. Mitchell. A universal client-based identity management tool. To appear in: *Proceedings of EuroPKI '11 — the 8th European Workshop on Public Key Infrastructures, Services and Applications, September 15–16, 2011, Leuven, Belgium*, LNCS. Springer-Verlag, Berlin, 2011.
7. Haitham S. Al-Sinani and Chris J. Mitchell. Client-based CardSpace-OpenID interoperation. In Erol Gelenbe, Ricardo Lent, and Georgia Sakellari, editors, *Proceedings of ISCIS '11 — the 26th International Symposium on Computer and Information Sciences, September 26–28, 2011, London, UK*, Lecture Notes in Electrical Engineering (LNEE), pages 387–393. Springer, London, 2011.
8. Haitham S. Al-Sinani and Chris J. Mitchell. Extending the scope of CardSpace. In Mehmet A. Orgun, Atilla Elçi, Oleg B. Makarevich, Sorin A. Huss, Josef Pieprzyk, Lyudmila K. Babenko, Alexander G. Chefranov, and Rajan Shankaran, editors, *Proceedings of SIN '11 — the 4th International Conference on Security of Information and Networks, November 14–19, 2011, Sydney, Australia*, pages 235–238. ACM, New York, 2011.
9. Haitham S. Al-Sinani. Integrating OAuth with Information Card systems. In Ajith Abraham, Daniel Zeng, Dharma Agrawal, Mohd Faizal Abdollah, Emilio Corchado, Valentina Casola, and Choo Yun Huoy, editors, *Proceedings of IAS '11 — the 7th International Conference on Information, Assurance, and Security, December 5–8, 2011, Malacca, Malaysia*, pages 198–203. IEEE, New York, 2011.
10. Haitham S. Al-Sinani and Chris J. Mitchell. Enabling interoperation between Shibboleth and Information Card systems. To appear in: *Security and Communication Networks*, 2012.

1. INTRODUCTION

11. Haitham S. Al-Sinani. Supporting interworking between OAuth and Information Card systems. To appear in: *Journal of Information Assurance and Security*, 7, 2012.

Part I

Background

Overview

Part I provides background material for the rest of the thesis. It contains three chapters, as follows.

1. *Chapter 2* gives an introduction to, and basic definitions for, the concepts of identity, privacy, and security, as well as outlining associated protocols of importance in this thesis.
2. *Chapter 3* provides an introduction to identity management, including coverage of single sign on and Cameron's identity laws. The chapter also gives an abstract model for identity management, and considers a range of properties possessed by different types of identity management systems.
3. *Chapter 4* provides a detailed description of a number of identity management systems of relevance to this thesis, namely CardSpace, Higgins, OpenID, OAuth, Liberty, and Shibboleth. The chapter also gives an overview of certain other systems of background relevance, namely Microsoft Passport, U-Prove, and IdeMix.

Identity, Privacy and Security

2.1 Introduction

This chapter provides an introduction to, and basic definitions for, the concepts of identity, privacy, and security, as well as outlining associated protocols of importance in this thesis.

The chapter is organised as follows. Section 2.2 describes the concept of identity, and, in section 2.3, we give an introduction to Internet privacy. Section 2.4 outlines several security services and mechanisms, as well as listing some key management techniques. Finally, section 2.5 concludes the chapter by summarising a number of protocols of relevance to this thesis.

2.2 Identity

2.2.1 Definition

The study of the notion of identity encompasses a wide range of disciplines, including sociology, psychology, philosophy, as well as computer science [32]. In the context of digital identity management, several definitions have been proposed for the term identity [19, 21, 30, 32, 36, 64, 117, 197, 200, 228, 229], including those given in published or draft standards [119, 120, 121, 132]. For example, in a draft of ITU-T X.1250, the term *identity* is defined as the ‘representation of an entity (or group of entities) in the form of one or more information elements which allow the entity(ies) to be uniquely recognised within a context to the extent that is necessary (for the relevant applications)’ [119]. Along similar lines, the recently published

ISO/IEC 24760-1 [132] defines the term as the ‘information used to represent an entity in an ICT (Information and Communication Technology) system. The purpose of the ICT system determines which of the attributes describing an entity are used for an identity. Within an ICT system an identity shall be the set of those attributes related to an entity which are relevant to the particular domain of application served by the ICT system. Depending on the specific requirements of this domain, this set of attributes related to the entity (the identity) may, but does not have to be, uniquely distinguishable from other identities in the ICT system’.

Note that ISO/IEC 24760-1 [132] deems any set of attributes describing a particular entity to be an identity for the associated entity; that is, in certain contexts, the identity information for different entities may be the same. However, in other standards, e.g. ITU-T X1252 [120], the explicit purpose of an identity is the capability of the identity information to sufficiently differentiate one entity from another (to the extent necessary in a specific context).

In this thesis, the term identity is used to refer to a digital representation of an entity in a given context. An *entity* is an item that has a recognisably distinct existence, and that can be uniquely identified in a specific domain, such as a person or a device. An identity consists of a set of attributes of an entity, where an *attribute* is defined as a characteristic or a property of an entity [132]. An entity could have more than one identity (see section 2.2.2). An *identifier* is a unique identity property that unambiguously distinguishes one entity from another in a given context. A *context*, or a *domain*, is an environment in which an entity can use a set of attributes for identification and other purposes [132]. For example, *PUAI003* (an identifier) is the username (an attribute) of Haitham Al-Sinani (an entity) in the RHUL IT system (a context or domain).

2.2.2 Properties

Certain properties associated with a human identity have been identified in the literature [19], including those specified in a document [187] published by the Organisation for Economic Co-operation and Development (OECD)¹. We next list a number of such properties.

- Identity is *social*. People are naturally social, and, in order to engage in social interactions, humans need something that persists and that can be used as a basis for recognition of individuals — an identity.
- Identity is *subjective*. Different people are likely to have differing experiences of interactions with an individual. It follows that one individual could be attributed distinct characteristics; i.e. different people are likely to construct disparate identities for the same person.
- Identity is *valuable*. If a history of an individual's previous actions is built, an exchange of identity information could create social capital and enable transactions which would not be possible without identity. Put differently, identity could lend predictability to afford a level of confidence during the decision-making process.
- Identity is *referential*. An identity is only a reference to a person; it is not a person in itself.
- Identity is *composite*. Whilst some identity information about a person arises voluntarily from the person him/herself, other identity data is developed by others without the person's involvement.
- Identity is *consequential*. Since a person's identity provides information about a person's past actions, the decision to exchange identity data has consequences. For example, whilst disclosure of identity information in a certain context could cause harm, failure to reveal identity information in another context could create a risk.

¹<http://www.oecd.org>

- Identity is *dynamic*. Identity information is not static and is always changing; an identity record might become inaccurate at any given moment. Examples of dynamic identity attributes include a person's age, address, level of education, etc.
- Identity is *contextual*. Individuals may have multiple identities, and they may choose to keep them utterly separate. For example, a person could be a professor at work and a husband at home.
- Identity is *equivocal*. An identification process is not foolproof; it is inherently error-prone. For example, a technical identification system could generate a number of false positives and false negatives.

2.2.3 A Life Cycle

A possible identity life cycle taken from Windley [229] is shown in Fig. 2.1 below; it includes five basic phases, as follows.

1. **Provision.** This represents the start of an identity's life cycle, and involves creating an identity record and populating it with attribute values. Provisioning can be third party-serviced (e.g. setting up a student account by a university administrator), or can be self-serviced (e.g. setting up an online account by populating an HTML form).

Note that the identity provisioning phase could involve initial entity authentication to establish the accuracy of the provided attributes. For example, setting up a bank account could require a presentation of a passport as well as a manual, face-to-face identification. During this phase, an identity could also be mapped to a certain level of assurance (see section 2.4.3.3).

2. **Propagation.** This depends on the nature of the system within which the identity is created, which may require the identity record to be propagated to other systems or sub-systems (e.g. in a simple system

propagation could mean storing the identity information in a file system or a local database). Propagation is therefore complementary to the provisioning phase and must also occur after the *maintenance* phase (see below).

3. **Usage.** Once provisioned and propagated, the identity can now be used (e.g. for authentication and authorisation purposes).
4. **Maintenance.** In this phase, an identity record is updated or altered, and, once changed, must be re-propagated. Common examples include password resets and address changes.
5. **De-provision.** This takes place at the end of the identity's life cycle, and refers to the complete deletion/removal of the identity record from its domain. For example, a university could de-provision a student's account from its IT system once the student has completed the degree programme.

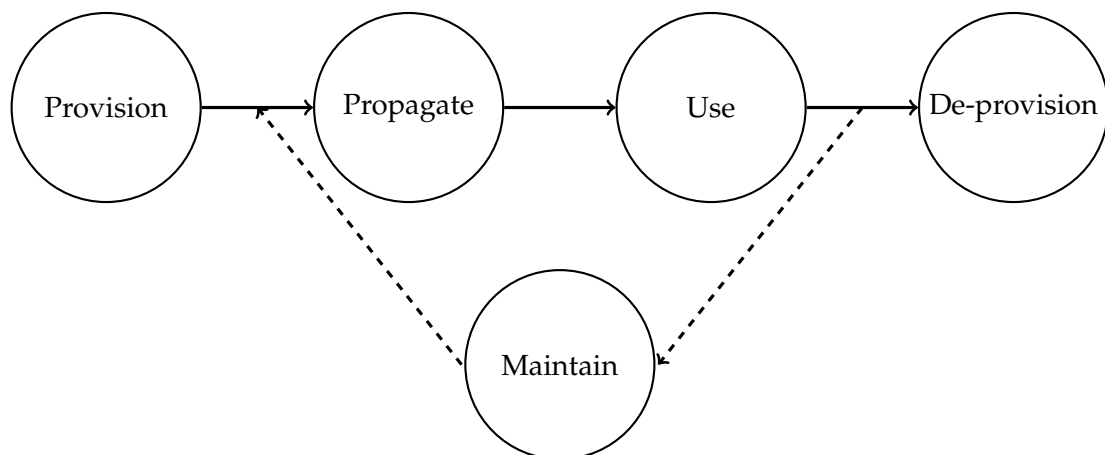


Figure 2.1: An Identity Life Cycle

2.3 Privacy

2.3.1 Definition

Privacy is defined by Windley [229] as the ‘protection of the attributes, preferences, and traits associated with an identity from being disseminated beyond the subject’s needs in any particular transaction’. More generally, the term privacy refers to different things in many contexts; however, in this thesis we focus on the *Internet privacy* of human entities.

Internet privacy involves the capability to control what information an individual discloses or withholds about themselves over the Internet, including determining who has access to such information, and for what purposes the information may or may not be used. With the advances in technology, Internet users are increasingly subject to privacy threats. For example, Internet users may become concerned if they discover that visited websites collect, store, and possibly share personally identifiable information (see section 2.3.2.1) about them. Internet privacy protection is therefore an issue of growing importance.

2.3.2 Related Concepts

2.3.2.1 Personally Identifiable Information (PII)

According to the U.S. Office of Management and Budget (OMB)² [222], and also as adopted by National Institute of Standards and Technology (NIST) [164], *PII* is defined as ‘any information about an individual maintained by an agency, including (1) any information that can be used to distinguish or trace an individual’s identity, such as name, social security number, date and place of birth, mother’s maiden name, or biometric records; and (2) any other information that is linked or linkable to an individual, such as medical, educational, financial, and employment information’. In the context of information security, the term PII refers to the information that can be used

²<http://www.whitehouse.gov/omb>

(or can be combined with other sources) to uniquely identify a person. OMB classes the following data items as PII: full name (if not common), national identification number, IP address (in some cases), vehicle registration plate number, driver's license number, face, fingerprints, handwriting, credit card number, digital identity, birthday, birthplace, and genetic information (see also section 2.3.3).

Note that multiple pieces of information, which are not sufficient on their own to uniquely identify an individual, might uniquely identify a person if combined. For example, it has been claimed that, in 1990, 87% of the US population could be uniquely identified by gender, ZIP code, and full date of birth [216].

2.3.2.2 Anonymity

Anonymity enables a subject to be indistinguishable within a set of subjects, known as the *anonymity set* (the set of all possible subjects) [197]. Anonymity is also defined as being 'a condition in identification whereby an entity can be recognised as distinct, without sufficient identity information to establish a link to a known identity' [132]. Anonymity therefore ensures that a user can use a resource without disclosing his or her identity.

If authentication and/or authorisation is required, anonymity typically involves the use of special anonymised credentials which can be cryptographically verified (see section 4.9). However, full anonymity on the Internet is not guaranteed, e.g. because IP addresses may be tracked thereby enabling the identification of the communicating device (albeit not necessarily the actual user). Nonetheless, tracking of IP addresses can be addressed using anonymising services, such as I2P³ (Invisible Internet Project) or TOR⁴ (The Onion Router); such anonymisers adopt a distributed technology approach, which could potentially grant a higher degree of anonymity than

³<http://www.i2p2.de/>

⁴<https://www.torproject.org/>

centralised anonymising services where a central point exists that knows and therefore could disclose a person's identity.

2.3.2.3 Pseudonymity

A person may wish to maintain a long-term relationship (or a reputation) with an entity, without necessarily revealing their PII. *Pseudonymity* can support such a situation, allowing the person to establish a unique identifier with the other entity. This identifier is called a *pseudonym*, which is defined as an 'identifier that contains the minimal identity information sufficient to allow a verifier to establish it as a link to a known identity' [132]. Such a pseudonym does not reflect the person's real identity [231]. Examples of pseudonyms include pen names, nicknames, etc. A pseudonym enables the other entity to link different messages from the same person and, thereby, maintain a long-term relationship. Pseudonyms can be either temporary or permanent. A pseudonym can have its value chosen by the person, or be assigned randomly [132]. The subject referred to by a pseudonym is called the *holder* of the pseudonym.

2.3.2.4 Unlinkability

True anonymity requires *unlinkability* [197], i.e. where an adversary's inspection of a message from, or action by, a pseudonym holder yields no new information about the holder's true identity. Unlinkability enables a user to make multiple uses of resources or services without others being able to link these uses together. Unlinkability necessitates that users and/or subjects cannot establish whether the same user caused certain specific operations in the system.

2.3.3 Categories of Personal Data

The P3P (Platform for Privacy Preferences — see section 2.3.5.3) specifications [227] define the following 16 categories of personal data which should

be protected when browsing websites.

1. *Physical contact information*, such as a telephone number of a physical address, allows a person to be contacted or located in the physical world.
2. *Online contact information*, such as an email address, allows a person to be contacted or located on the Internet. Such information is often independent of the specific computer used to access the network.
3. *Unique identifiers*, excluding financial and government-issued identifiers, are issued for the purposes of consistently identifying or recognising a person. They include identifiers issued by a website or service.
4. *Purchase information*, including information about payment methods, is generated as a result of the purchase of a product or service.
5. *Financial information* concerns an individual's finances, including account status and activity information. Examples include account balance, payment or overdraft history, and information about a person's purchases or use of financial instruments including credit or debit card information. Note that the information that is just derived from a discrete purchase by a person, as described in *purchase information*, does not fall under the definition of financial information.
6. *Computer information*, such as an IP address, domain name, browser type or operating system, relates to the computer that the individual is using to access the network.
7. *Navigation and click-stream data*, such as the pages visited and the period of time each page is visited, is passively generated by browsing a website.

8. *Interactive data*, such as queries to a search engine or logs of an account activity, is actively generated by explicit web interactions with a service provider.
9. *Demographic and socioeconomic data*, such as gender, age, or income, concerns the characteristics of an individual.
10. *Content*, such as the text of emails, bulletin board postings, or chat room communications, is the collection of words and expressions contained in the body of a communication.
11. *State management mechanisms*, e.g. HTTP cookies (see section 2.3.4.1), are techniques for maintaining a stateful session with a user, or automatically recognising users who have visited a particular site or accessed particular content previously.
12. *Political data*, including membership of, or affiliation with, groups such as religious organisations, trade unions, professional associations, political parties, etc.
13. *Health information* concerns a person's physical or mental health, sexual orientation, use of or inquiry into health care services or products, and purchase of health care services or products.
14. *Preference data*, such as a favourite colour or musical tastes, concerns a person's likes and dislikes.
15. *Location data*, such as position data provided by a Global Positioning System (GPS) receiver [90], can be used to identify a person's current physical location and track them as their location changes.
16. *Government-issued identifiers* are issued by a government for the purpose of consistently identifying a person.

Finally note that the P3P specifications also include a category of *other*, covering types of (personal) data not captured by the above definitions.

2.3.4 Threats

We next discuss a number of examples that can give rise to privacy threats when browsing the web.

2.3.4.1 Cookies

We describe three types of cookies, outlining how they can be used to pose a privacy threat.

1. *HTTP cookies* [153] are small amounts of textual data stored on a user's machine, initially set by, and only retrievable by, a specific web server. A web browser returns such a cookie to the originating web server on every request made to that server. Cookies can therefore link multiple HTTP requests together in an otherwise stateless HTTP protocol (see section 2.5.3). User state information can be stored in cookies; such information is typically used for user authentication, identification of a user session, user preferences, shopping cart contents, etc.

Two categories of cookie of particular importance in this thesis are as follows.

- *Session cookies*, also called *transient cookies*, are deleted when the user closes the web browser. Such cookies are stored in temporary memory and are not retained once the browser is closed.
- *Persistent cookies* can survive across a number of sessions, including after exiting the browser and/or after a machine reboot. Such cookies have an expiry date; if a cookie expires it is deleted.

However, cookies can be used for user-tracking and web profiling, in which a user's browsing activities are profiled/tracked. User profiling (or tracking) refers to the process of assembling and analysing multiple events, each attributable to a single originating entity. This could lead to the construction of a set of information relating to this individual,

e.g. certain patterns of activity or PII (see section 2.3.2.1). Such information could be used for a range of purposes, e.g. targeted advertising, some of which may not be in the user's interests.

If a cookie is used to store user credentials (e.g. a password), certain attacks (notably cross-site scripting [94]) can be used to fraudulently obtain such credentials.

Many modern web browsers allow users to delete or disable cookies. Although deletion/disabling of cookies might eliminate potential privacy threats, such an action could severely limit the functionality of many websites. Cookies remain a significant privacy concern.

2. *Flash cookies* operate in the same way as HTTP cookies, and are employed by *Adobe flash player* to store information in a user's computer. Flash cookies, or *local shared objects* [165], pose similar privacy risks to those arising from HTTP cookies. However, flash cookies are not as easily deleted or disabled. The option displayed in most modern browsers to block cookies does not affect flash cookies; browser extensions can, however, be used to block such cookies.
3. *Evercookies* is a term used to describe a cookie-like mechanism used by malicious software to store state on a user machine, and are intentionally designed to be difficult to delete. JavaScript [180] can be used to create cookies (so called *zombie cookies*) in a web browser that actively escape deletion by generating redundant copies of themselves in various forms and locations on the user platform, e.g. as flash local shared objects, or using various HTML version 5 storage mechanisms, window.name caching, etc.

2.3.4.2 Search Engines

Search engines are capable of tracking a user's searches. Personal data and patterns of activity can be deduced by analysing user searches, including

the search items used, the time of the search, and the URLs visited as a result of the search. Web profiles belonging to specific users can be built which could, for example, be sold to third parties, e.g. advertising agencies. Such actions, particularly when conducted in the absence of informed user consent, constitute a potential privacy threat.

2.3.4.3 Data Logging

Operating systems (OSs) and applications can typically be configured to perform logging of usage, e.g. recording the times at which the user browses the Internet, the websites visited, etc. User privacy could be compromised if a third party obtains illegal access to such logs stored on a user platform. Such privacy risks can be mitigated, e.g. by encrypting the logs, disabling logging, or regularly deleting logs.

2.3.4.4 Internet Service Providers (ISPs)

ISPs provide Internet access to subscribers (users). All web data transmitted to and from user machines must pass via an ISP. Such an ISP therefore has the capability to monitor user activities on the Internet. ISPs are not expected to conduct such actions, not least for legal, ethical, business, and technical reasons; however, a degree of tracking and record retention may be required in some jurisdictions in order to enable law enforcement investigations.

2.3.5 Protection

We next discuss privacy protection.

2.3.5.1 Guidelines

Back in 1980, the OECD published guidelines on the protection of privacy and transborder flows of personal data, aimed to 'harmonise national privacy legislation and, while upholding such human rights, would at the

same time prevent interruptions in international flows of data' [186]. The OECD privacy guidelines have since been used to derive privacy laws governing the use of information systems now in place in a number of countries [21]. These guidelines contain eight basic principles for personal data protection, as listed below. A system that fails to fairly satisfy the requirements of any of these principles could be considered vulnerable to privacy violation.

1. **Collection Limitation Principle.** Collection of personal data should have limits, and should be obtained by lawful and fair means, and, where appropriate, with the knowledge or consent of the data subject.
2. **Data Quality Principle.** Personal data should be relevant to its use purposes, and, to the extent necessary for those purposes; such personal information should be accurate, complete and kept up-to-date.
3. **Purpose Specification Principle.** Purposes for (personal) data collection should be specified at or before the time of collection, and the subsequent use limited to the fulfilment of those purposes or such others as are not incompatible with those purposes and as are specified on each occasion of a change of purpose.
4. **Use Limitation Principle.** Personal data should not be revealed, made available or otherwise used for purposes other than those specified at the collection time, except with the consent of the data subject, or by the authority of law.
5. **Security Safeguards Principle.** Reasonable security safeguards should be employed to protect personal data against risks such as loss or unauthorised access, destruction, use, modification or disclosure of data.
6. **Openness Principle.** A general policy of openness about developments, practices and policies with respect to personal data should ex-

ist. Means should be readily available of establishing the existence and nature of personal data, and the main purposes of their use, as well as the identity and usual residence of the data controller.

7. Individual Participation Principle. A person should have the right to:

- a) obtain from a data controller, or otherwise, confirmation of whether or not the data controller has data relating to them;
- b) have communicated to them data relating to them:
 - within a reasonable time;
 - at a charge, if any, that is not excessive;
 - in a reasonable manner; and
 - in a form that is readily intelligible to them;
- c) be given reasons if a request made under subparagraphs (a) and (b) is denied, and to be able to challenge such denial; and
- d) challenge data relating to them, and, if the challenge is successful, to have the data erased, rectified, completed or amended.

8. Accountability Principle. Finally, this principle states that a data controller should be held accountable for complying with measures which give effect to the principles stated above.

2.3.5.2 Designing for Privacy

An identity management system conforming to ISO/IEC 24760-1 [132] shall obey all statutory and regulatory requirements for the protection of personal privacy. ISO/IEC 24760-1 requires that any sensitive information which an identity management system processes shall be clearly specified in the design phase. An ISO/IEC 24760-1-conformant identity management system should provide a range of privacy-related capabilities, including the following [132].

2. IDENTITY, PRIVACY AND SECURITY

- Mechanisms, including policies, processes, and technology, should be implemented to enable minimal disclosure.
- Entities that use identity information should be authenticated.
- The ability to link identities should be minimised.
- The use of identity information should be recorded and audited.
- Protection should be provided against inadvertently generating risks to privacy, e.g. those posed by poorly protecting identity information in logs and audit trails.
- Policies for selective disclosure should be implemented.
- Use of pseudonyms should be supported.
- Policies should be implemented to engage a human entity for explicit direction or consent, for activities related to sensitive personal data.

2.3.5.3 Privacy-protecting Technologies

A wide range of privacy-protecting languages and protocols exist. We next briefly review some of them.

- **P3P.** The Platform for Privacy Preferences (P3P)⁵ is a standard for communicating privacy practices and comparing them to user preferences. The P3P [227], developed by the World Wide Web Consortium (W3C)⁶, is designed to grant users more control over their personal information when browsing the web. It enables websites to express privacy policies in a standard format using the P3P Preference Exchange Language [155] (APPEL), which can be retrieved and interpreted by P3P-enabled web browsers.

⁵<http://www.w3.org/P3P>

⁶<http://www.w3.org>

A P3P-enabled website will have a set of policies, e.g. stating the intended uses of personal information that is gathered from site visitors. Using a P3P-enabled web browser, a P3P user can also define a set of policies, e.g. stating what personal information can be seen by the sites that they visit. During a site visit, the P3P functionality (expressed using XML — see section 2.5.6) compares what personal information the user is willing to release with what information the website wishes to obtain; if a mismatch occurs, P3P will inform the user and ask whether to proceed to the site and risk releasing personal information in ways at variance with the user's policy.

The Electronic Privacy Information Centre (EPIC)⁷ has criticised P3P, referring to the technology as a *Pretty Poor Policy*⁸. EPIC claims that P3P software is too complex and difficult for an average person to understand, and that many Internet users are likely to be unable to use the default P3P software or install additional P3P software. Furthermore, neither websites nor Internet users are obliged to use P3P.

- **XACML.** The eXtensible Access Control Markup Language (XACML) [88] is a standardised means of expressing privacy policies in an (XML-based) machine-readable language; such a language can be used by a software system to enforce privacy policies in IT systems.
- **EPAL.** The Enterprise Privacy Authorization Language (EPAL) [199] is very similar to XACML; however, it has not yet been standardised.
- **WS-Privacy.** Web Service Privacy (WS-Privacy) is a protocol under development intended for use in communicating privacy policies in web services. For example, it could be used to specify how privacy policy information can be embedded in the SOAP envelope (see section 2.5.6) of a web service message.

⁷<http://epic.org/>

⁸<http://www.epic.org/reports/prettypooprprivacy.html>

2.4 Security Services and Mechanisms

2.4.1 Definitions

The term *information security* is defined in ISO/IEC 27000 [133] and ISO/IEC 27001 [134] as the ‘preservation of confidentiality, integrity and availability of information; in addition, other properties such as authenticity, accountability, non-repudiation and reliability can also be involved’.

ISO/IEC 27000 [133] further states that information security ‘includes three main dimensions: confidentiality, availability and integrity. With the aim of ensuring sustained business success and continuity, and in minimising impacts, information security involves the application and management of appropriate security measures that involves consideration of a wide range of threats’.

According to the *Title 44 of the United States Code* [221], information security refers to the protection of information and information systems from unauthorised access, use, disclosure, disruption, modification, or destruction in order to provide: integrity (including non-repudiation and authenticity); confidentiality (including protecting personal privacy and proprietary information); and availability. Confidentiality, integrity, and availability are commonly abbreviated to *CIA* (see Fig. 2.2).

We next list a number of key concepts.

Confidentiality refers to the concealment of information or resources [36], such that they are only disclosed to authorised entities [134].

Integrity protects against improper modification or destruction of information. It refers to the trustworthiness of data or resources [36].

Availability ensures that systems or resources are accessible and usable upon demand by an authorised entity [134]. That is, it guarantees that systems work promptly and that service is not denied to authorised users [215].

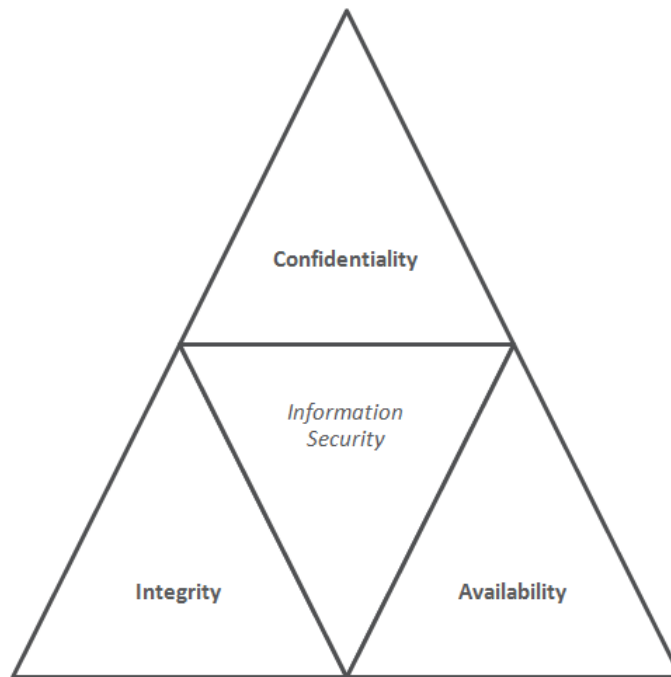


Figure 2.2: CIA Triangle

Non-repudiation involves the generation of evidence regarding the occurrence of a claimed event or action and its originating entities; it can be used to help to resolve disputes about the occurrence/non-occurrence of the event or action, and the involvement of entities in the event [133].

Accountability enables an entity to be held accountable for its actions and decisions [133].

Access control provides protection of system resources against unauthorised access [213]. A security policy (see below) regulates the use of system resources, including the use of a communication resource, and execution of a processing resource. Read and write operations on system resources can also be access-controlled.

Security policies are statements of what is, and what is not, allowed [36]. A security policy is a set of rules and practices which specify/regulate how a system or an organisation provides security services to protect sensitive and critical system resources [213]. A security policy is

enforced by security mechanisms (see section 2.4.2). Note that achieving overall/holistic security typically requires both technological security (e.g. application, OS, and network security) and physical security, as well as the deployment of appropriate security policies and procedures [76].

Several of the concepts introduced above (e.g. data confidentiality, data integrity, availability and non-repudiation) are sometimes referred to as *security services* [215]. Such services are intended to counter security attacks, and are provided using *security mechanisms*, which we next describe.

2.4.2 Security Mechanisms

A *security mechanism* is any process designed to detect, prevent, or recover from a security attack [215]. Security mechanisms are the means by which security services (which include authentication, access control, data confidentiality, data integrity, non-repudiation, and availability [215]) are provided [81]. We next outline the security mechanisms of relevance to this thesis.

2.4.2.1 Symmetric Cryptography

Introduction

When *symmetric cryptography* [167, 215] is used to protect a transmitted message, both the sender and recipient of the message must share the same secret key. We next describe two widely used classes of symmetric cryptographic mechanisms, namely symmetric encryption schemes and message authentication code algorithms.

Symmetric Encryption

When using a *symmetric encryption* scheme, a plaintext message is encrypted to generate ciphertext using a secret key. The same key must be

subsequently used to decrypt the ciphertext to produce the original plaintext. Two commonly used types of symmetric encryption algorithms are as follows.

1. *Block ciphers* take as input a block of plaintext (typically of 64 or 128 bits in length) and a key, and output a block of ciphertext. Such algorithms can be used in a variety of ways, called *modes of operation*, e.g. CBC (Cipher-block Chaining) and CTR (Counter), to encrypt messages of arbitrary length. Examples of such ciphers include AES (Advanced Encryption Standard) [179] and triple-DES [147], where DES stands for Data Encryption Standard.
2. *Stream ciphers* encrypt data one bit at a time. RC4 [148] is a widely used stream cipher.

Symmetric encryption algorithms are standardised in ISO/IEC 18033–3 [130] and ISO/IEC 18033–4 [131]. Symmetric encryption mechanisms can be used to preserve data confidentiality.

Message Authentication Codes

A *message authentication code* (MAC) [138] is a relatively short piece of data, used to authenticate a message. A MAC function/algorithm takes as input a secret key and a message (of arbitrary length), and produces a MAC (of a fixed length). Examples of MAC functions include CBC-MAC [137] and HMAC (Hashed-based MAC) [183]. MACs can be used to:

- protect data integrity, by allowing the verifier to detect changes in message contents; and
- provide the data origin authentication service, as a valid MAC could only have been generated by the party possessing the appropriate secret key.

Issues

One major practical issue with the deployment of symmetric cryptography is establishing/distributing the necessary secret keys in a secure manner.

2.4.2.2 Asymmetric Cryptography

Introduction

Asymmetric cryptographic schemes [167, 215] use keys in a somewhat different way to symmetric schemes. Instead of using secret keys (which must be shared by senders and recipients of protected messages), every participating user must have a related pair of keys, as follows:

1. a *public key*, which is publicly available and is used to encrypt, or verify a digital signature of, a message; and
2. a *private key*, which is kept secret and is used to decrypt, or digitally sign, a message.

It must be computationally hard to deduce the private key from the public key. We next outline two commonly used classes of asymmetric cryptographic mechanisms, namely asymmetric encryption schemes and digital signatures.

Asymmetric Encryption

In an *asymmetric encryption* scheme, a plaintext message is encrypted to generate ciphertext using a public key. The corresponding private key must subsequently be used to decrypt the ciphertext to produce the original plaintext. For example, suppose entity *B* possesses a public-private key pair. Entity *A* can encrypt a message intended for entity *B* using *B*'s public key. On receipt of the ciphertext, *B* can decrypt the ciphertext using its corresponding private key.

Examples of asymmetric encryption algorithms include ElGamal [89], and RSA (Rivest-Shamir-Adleman) [208]. Asymmetric encryption schemes are standardised in ISO/IEC 18033-2 [129]. Asymmetric encryption mechanisms can be used to preserve data confidentiality.

Digital Signatures

In a *digital signature* scheme, a message is signed using a private key (the signing key); the signature can be subsequently verified using the corresponding public key (the verification key). For example, suppose entity *A* possesses a public-private signature key pair. Entity *A* can sign a message using its private (signing) key. Entity *B* can then verify the digital signature using *A*'s corresponding public (verification) key.

Examples of digital signature algorithms include the ElGamal signature scheme [89] and the RSA signature scheme [208]. Digital signature techniques are standardised in ISO/IEC 9796 [135, 136], and ISO/IEC 14888 [126, 127, 128]. Digital signatures can be used to provide data origin authentication, data integrity, and non-repudiation services.

Issues

Establishing a secure binding between a public key and a user identifier is a major issue when using asymmetric cryptography. Several approaches have been designed to meet this requirement, notably public key infrastructures (see section 2.4.4.1), and webs of trust (e.g. as instantiated by Pretty Good Privacy (PGP) [101]). In practice, asymmetric cryptography is less efficient at processing large volumes of data than symmetric cryptography.

2.4.2.3 A Hybrid Approach

The two types of cryptography can be combined to take advantage of the efficiency of symmetric cryptography and the convenience of public-key cryptography (since a sender-receiver shared secret key is not required).

Using *hybrid encryption* (or *digital enveloping*) [114], data can be encrypted using a symmetric algorithm (for speed and size considerations [104]) employing a randomly selected key. The randomly chosen key must also be encrypted using the public key of the receiver. To decrypt, the receiver uses its private decryption key to recover the randomly chosen secret key, which can then be used to decrypt the data. Certain identity management systems, including CardSpace (see section 4.3), adopt this approach.

2.4.2.4 Hash Functions

According to Stallings [215], a '*hash function, H* , accepts a variable-length block of data, M , as input and produces a fixed-size hash value, $h = H(M)$ '. A cryptographic hash function must satisfy the following properties, as outlined by Stallings [215].

- *Preimage resistance* (one-way property). For any given hash value h , it is computationally infeasible to find a value y such that $H(y) = h$.
- *Second preimage resistance*. For any given block x , it is computationally infeasible to find $y \neq x$ with $H(y) = H(x)$.
- *Collision resistance*. It is computationally infeasible to find any pair (x, y) such that $H(x) = H(y)$.

Cryptographic hash functions are an integral part of the computation of most practical digital signature schemes; a hash function can also be used to construct a MAC function, as in the *HMAC* scheme [183]. Examples of hash functions include *Whirlpool* [124] and *SHA-1*, where SHA stands for Secure Hash Algorithm [84, 124, 182]. Cryptographic hash functions are standardised in ISO/IEC 10118 [122, 123, 124, 125].

2.4.2.5 Time-stamps

A *time-stamp* can be used to guarantee the freshness of a message. If used as part of a security mechanism, the integrity of a time-stamp must be pro-

tected, e.g. by using cryptographic means. Two commonly discussed types of time-stamps are given below.

1. *Clock-based time-stamps* consist of a time value, typically recorded from the clock of the message sender.
2. *Logical time-stamps* are bilaterally managed sequence numbers.

The use of clock-based time-stamps in protocol messages requires the communicating entities to be equipped with securely synchronised clocks. To cope with the fact that clocks can only ever be perfectly synchronised for an instant, and also with variable propagation delays, each entity must define a *time-acceptance-window*, so that a received message is only considered fresh if and only if its time-stamp falls within this window.

2.4.2.6 Nonces

RFC 2828 [213] defines a *nonce* as ‘a random or non-repeating value that is included in data exchanged by a protocol, usually for the purpose of guaranteeing liveness and thus detecting and protecting against replay attacks’.

A nonce (derived from *number used once*), which is typically randomly- or pseudorandomly-chosen, is used to guarantee freshness and prevent replay attacks on messages exchanged in a protocol. As introduced by Needham and Schroeder in the late 1970s, a nonce is generated by one party (*A* say) and sent to the other party (*B* say); *B* then includes it, or a function of it, in messages sent back to *A*. This then enables *A* to verify whether or not a received message is a response to a message it sent, and hence is fresh.

An alternative use of nonces is as follows. A sender includes a fresh nonce in each message it sends, and a receiver keeps a log of all received nonces. A replay attack is detected by the receiver if a message is received containing a previously used nonce. A nonce of this latter type may also be used in conjunction with a time-stamp (see section 2.4.2.5) to discard responses (messages) that are too old, thus limiting the period of time for

which received nonces must be kept; a number of identity management systems, including OpenID (see section 4.5), adopt such a technique.

2.4.3 User Authentication

2.4.3.1 Definition

ISO/IEC 27000 [133] states that *authentication* is the ‘provision of assurance that a claimed characteristic of an entity is correct’. RFC 2828 [213] defines authentication as ‘the process of verifying an identity claimed by or for a system entity’.

An authentication process consists of two steps [213, 215]:

1. an *identification step*, in which an identifier (e.g. a username) is presented to a security system; and
2. a *verification step*, in which authentication data (e.g. a password), which corroborates the entity-identifier binding, is presented.

Note that *user authentication* is distinct from *message authentication*, which allows a communicating entity to verify that the contents of a received message have not been modified and that the message source is authentic [215]. This thesis focuses on user authentication.

2.4.3.2 Authentication Methods

Techniques (or factors) [215] for authenticating a user’s identity can be divided into the following categories.

1. *Something the user knows*. Examples include a password, a personal identification number (PIN), or answers to a set of questions.
2. *Something the user possesses*. Examples include bank cards, smart cards, and one-time password (OTP) generating tokens.

3. *Something the user is or does*. Most examples of this approach are commonly associated with biometrics, and can be divided into two groups, as follows.

- a) *Static biometrics (something the user is)*. Examples include recognition by fingerprint, retina or iris pattern, and face.
- b) *Dynamic biometrics (something the user does)*. Examples include recognition by handwriting, manual signature, and voice patterns.

Techniques for user authentication can be used alone or in combination to enhance the security of the authentication process. If two factors are used, then this is referred to as *two-factor authentication* [17, 224]. A common example of two-factor authentication is the use of a bank card and a PIN. *Multi-factor authentication* is a generic term used to describe the case where two or more authentication factors are used.

Although the combination of authentication methods/factors can help to enhance security, it does not make the systems employing them absolutely secure. Indeed, the three categories listed above have been facetiously described as *something the user forgets*, *something the user loses* and *something the user ceases to be* [189].

2.4.3.3 Assurance Level

An *assurance level* is a defined level of trust associated with a credential [102]. Assurance levels can be specified in terms of the technology and/or processes used, and/or policy and practice statements governing the operational environment. In the context of user authentication, an assurance level identifies the degree of confidence:

- in the vetting process employed to establish the identity of the entity to whom a credential was issued; and

- that the entity using a credential is the same entity to which the credential was originally issued [37].

That is, an authenticating party's level of assurance in a user's claimed identity depends on the authentication mechanism and the initial registration process [64]. An authenticator could therefore choose to give the same user differing access privileges depending on its confidence in the identity provider's registration and authentication processes [21].

In principle, the required level of assurance [102] should correspond to the level of risks involved with the requested access to the protected resources [228]; that is, the higher the risk the greater the level of assurance should be.

NIST provides guidance [48] defining 4 assurance levels, ranging from *low* to *very high*. We next list these levels.

1. **Level 1: Low.** This level provides a relatively low level of confidence in the asserted identity. It does not require a proof of identity during the registration process, and does not require the use of cryptographic methods. Use of level 1 is appropriate when relatively minor negative consequences would result from a flawed authentication technique, and the authentication mechanism used provides minimal assurance. An example use case is where a claimant presents a self-registered user ID or PIN to create a customised web page or to access online materials, such as news or product documentation [32, 48, 102].
2. **Level 2: Moderate.** This level provides a moderate degree of confidence in the asserted identity. This level requires a more stringent identity proof than required by level 1, and the authentication mechanisms employed at level 2 must also be more secure; level 2 requires at least single-factor authentication (see section 2.4.3.2). Authentication failure here would result in *moderate* risks. An example use case is where a beneficiary updates an address of record through an insur-

ance provider's website. Such a website would require authentication to ensure that the address being updated is the correct beneficiary's address. If an incorrect address is provided, official documents (e.g. payment amounts, account status, etc.) might be sent to this address, leading to a moderate risk of unauthorised release of personally sensitive data [32, 48, 102].

3. **Level 3: High.** This level provides a high level of confidence in the asserted identity. It requires verification of identifying materials and information during the identity proof process. It mandates multi-factor authentication; authentication must be based on proof-of-possession of a key or OTP using a cryptographic protocol. Tokens can be implemented in software or hardware. Use of this level is appropriate when a substantial (or *high*) risk is associated with erroneous authentication. An example use case is where a patent attorney electronically submits confidential patent information to a patent authority; improper disclosure would give competitors a competitive advantage [32, 48, 102].

4. **Level 4: Very High.** This level provides a very high degree of confidence in the asserted identity, producing the highest practical level of authentication assurance based on proof-of-possession of a key using a cryptographic protocol. Level 4 is similar to level 3, except that only hardware-based cryptographic tokens are permitted. This level requires high levels of cryptographic assurance for all elements of credential and token management; all sensitive data transfers must be cryptographically authenticated using keys bound to the authentication process. An example scenario of level 4 transaction is the approval by an executive of a transfer of funds exceeding one million US dollars from an organisation's bank account [102].

2.4.4 Key Management Techniques

When using cryptography of any kind, key management is of great significance. Two widely discussed techniques for key management are described below.

2.4.4.1 Public Key Infrastructures

RFC 2828 [213] states that a public key infrastructure (PKI) is ‘the set of hardware, software, people, policies, and procedures needed to create, manage, store, distribute, and revoke digital certificates based on asymmetric cryptography’.

In a PKI [3], a trusted third party (TTP) known as a certification authority (CA) signs and publishes public-key certificates, i.e. signed data structures containing (amongst other things) a public key and an identifier. Such a certificate binds a user identity to a cryptographic public key. A widely used format for such certificates is defined in X.509 [139]. In addition to issuing certificates, a CA typically also regularly issues certificate revocation lists (CRLs), containing identifiers of certificates that are no longer valid. A CRL can be used by a verifier to check the revocation status of a certificate issued by the corresponding CA.

A CA might also support a variety of other administrative functions, although some such functions might be delegated to one or more registration authorities (RAs).

One particularly important such function is user registration. User registration requires that the identity of an applicant for a public-key certificate is verified before the CA issues a certificate for that user’s public key. The procedures for identity verification vary from one CA to another, and determine the level of assurance (see section 2.4.3.3) in the public key-applicant binding guaranteed by a certificate issued by that CA.

Extended validation (EV) certificates have been defined. To obtain such

a certificate, an applicant must go through additional validation steps (e.g. proving the physical location of the applicant) during the validation process. Some web browsers (e.g. Internet Explorer from version 7 onwards) show a green address bar if an SSL connection has been established using an EV certificate [34].

2.4.4.2 Diffie-Hellman Key Exchange

The Diffie-Hellman key exchange (DHKE) protocol [206] enables two entities to securely establish a shared secret key using an unsecured channel. The shared key can subsequently be used in symmetric cryptographic mechanisms (see section 2.4.2.1). The protocol objectives are accomplished even if all the messages exchanged during a protocol run are intercepted; however, the protocol is vulnerable to active attacks, including man-in-the-middle (MITM) attacks. The protocol does not require the communicating entities to have prior knowledge of each other. The security of the scheme rests on the difficulty of the discrete logarithm problem [215]. The protocol is used with some identity management systems, e.g. in OpenID; it is also employed in a number of commercial products.

We next describe how the DHKE protocol allows two users (referred to below as Alice and Bob) to securely establish a shared key over a non-secure channel. As a prerequisite for use of the protocol, Alice and Bob must agree on a large prime p and an element i of multiplicative order q modulo p , where q is a large prime dividing $p-1$ [215].

- Alice \rightarrow Bob: $A = i^a \bmod p$. Alice (independently) selects a secret random integer a , and sends Bob: $A = i^a \bmod p$.
- Bob \rightarrow Alice: $B = i^b \bmod p$. Bob (independently) selects a secret random integer b , and sends Alice: $B = i^b \bmod p$.
- Alice: computes the shared key as $B^a \bmod p$.

- Bob: computes the shared key as $A^b \bmod p$.

Note that if p and i are chosen appropriately then an adversary cannot determine the shared key even if all the exchanged messages are public.

2.5 Protocols and Standards

We conclude this chapter by introducing a number of technologies, protocols and standards of importance in this thesis.

2.5.1 HTML

2.5.1.1 Overview

HTML (HyperText Markup Language) is a markup language involving a predefined set of markup tags. Web pages are typically written in HTML, and transferred from servers to browsers using HTTP (see section 2.5.3). According to the HTML 4.01 specification [202], 'HTML gives authors the means to:

- publish online documents with headings, text, tables, lists, photos, etc.;
- retrieve online information via hypertext links, at the click of a button;
- design forms for conducting transactions with remote services, for use in searching for information, making reservations, ordering products, etc.; and
- include spread-sheets, video clips, sound clips, and other applications directly in their documents.'

A web browser (such as Internet Explorer or Firefox) uses HTML tags to interpret the content of an HTML document, and display it as a web page. HTML enables objects and images to be embedded in an HTML web page, and can be used to create interactive forms. Scripting languages, such as JavaScript [180], can be embedded in HTML documents; such scripts

affect the behaviour of HTML web pages when displayed by a browser. HTML [201, 202] is maintained by W3C. HTML 5 [87] is the fifth revision of the HTML standard; as of February 2012, HTML 5 is still under development.

2.5.1.2 HTML Forms

According to the HTML 4.01 specification [202], an HTML form is ‘a section of a document containing normal content, markup, special elements called controls (checkboxes, radio buttons, menus, etc.), and labels on those controls. Users generally *complete* a form by modifying its controls (entering text, selecting menu items, etc.), before submitting the form to an agent for processing (e.g., to a web server, mail server, etc.)’.

HTML forms are used to pass data, including user input, to a web server. The *input* element of an HTML form (see Listing 2.1) can be of type: text field, password, checkbox, radio button, submit button, etc.

The HTML 4.01 specification [202] states that the form element acts as a container for controls, and specifies a number of things including:

- the program that will handle the completed and submitted form (specified using the *action* attribute⁹);
- the character encoding that must be accepted by the server in order to handle this form (specified using the *accept-charset* attribute); and
- the method by which input data will be sent to the server (specified using the *method* attribute).

The method attribute is used to specify which HTTP method is used to submit the form data set (HTTP methods are described in section 2.5.3.4). Possible values for the method attribute are:

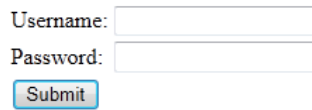
⁹Note that the processing program or agent must be able to parse name-value pairs in order to make use of them.

- *GET* (the default method), where the form data set is appended as a query string (see section 2.5.3.7) to the Uniform Resource Identifier (URI) specified by the action attribute, and this new URI is sent to the processing agent or program; and
- *POST*, where the form data set is included in the body of the form and sent to the processing agent or program.

Note that the GET method should not be used to transfer sensitive data (such as passwords) to a server, since the data will be displayed in the browser address bar and possibly also stored in server logs. The length of the data string that can be sent is also restricted when using the GET method (see also section 2.5.3.8). By contrast, the POST method allows for data strings of any length and type to be sent to a server, and the data is not displayed in the browser address bar. Listing 2.1 shows a piece of HTML which produces the username-password form shown in Fig. 2.3.

Listing 2.1: A Simple Username-password HTML Form (HTML Code)

```
<html>
<body>
<form action='http://www.website.com/ProcessingProgram.
  php' method='POST' accept-charset='ISO-8859-1'>
Username: <input type='text' name='username' /> <br />
Password: <input type='password' name='password' /> <br
  />
          <input type='submit' name='submit' value='
            Submit' />
</form>
</body>
</html>
```

A simple HTML form consisting of two text input fields. The first field is labeled 'Username:' and the second is labeled 'Password:'. Below the second field is a blue 'Submit' button.

Figure 2.3: A Simple Username-password HTML Form

2.5.2 Document Object Model (DOM)

The DOM is a W3C-standardised and language-independent means of deriving a hierarchical object model (referred to as the DOM from the structure of a document). An application programming interface (API) is also defined that allows scripting languages, such as JavaScript [180], to inspect and manipulate a parsed document within a browser [115]. Standards defined in the DOM include:

- *HTML DOM*, which is a standard model for HTML documents;
- *XML DOM*, which is a standard model for XML documents; and
- *core DOM*, which is a standard model for any structured document.

The DOM defines the objects and properties of all document elements and methods that can be used to access them. A client-side scripting language can use the DOM to read and modify the contents of a web page or completely alter its appearance. Additionally, a scripting language can interact with the browser's DOM event model. For example, a script can receive and react to user-originated events, such as mouse clicks and key presses, and can also create and dispatch events, such as simulating a mouse click on a button in an HTML document [76].

When using the DOM, a script can reference the document objects, such as forms, links, images, etc., via a hierarchical name that starts with the root document object. A hierarchical name might be the actual element name or it could be the sequential index of the traversed element. For example, the script could access an input element in an HTML form as either

document.formName.inputName or *document.forms[sequential index (e.g. 0)].elements[sequential index (e.g. 0)]*.

In many of the proof-of-concept prototypes described in chapters 6 to 13 of this thesis, the HTML DOM is used to allow a scripting language to inspect and manipulate HTML elements in a web page; the XML DOM is also employed to enable a script to parse and manipulate XML-based security tokens.

2.5.3 HTTP

Given the central role it plays in this thesis, we next give an introduction to HTTP (Hypertext Transfer Protocol). HTTP is the protocol which underlies the web, since it is used by web browsers to communicate with web servers, and by web servers to transfer HTML and/or other resources to browsers.

2.5.3.1 Definition

HTTP is ‘an application-level protocol for distributed, collaborative, hypermedia information systems’ [91]. According to RFC 2616 [91], one feature of HTTP is ‘the typing and negotiation of data representation, allowing systems to be built independently of the data being transferred’.

2.5.3.2 Roles (Client, Server)

HTTP is a request-response protocol, enabling a client to request and receive data from a server.

- The *client* is typically a program running on a user platform, used to send requests to servers and receive responses from them. It processes protocol messages on behalf of the user, and prompts the user to make decisions, provide secrets, etc. One particularly important example of a client, also known as a user agent, is a web browser, such as Internet Explorer, Firefox, etc.

- The *server* is a program, typically running on a remote machine, which can provide resources (such as HTML files), store content, or perform other functions. In this thesis we are specifically interested in web servers.

2.5.3.3 Requests and Responses

A client (web browser) submits an HTTP request message to a web server, which returns a response message to the client; such a response message contains information about the completion status of the request (i.e. whether or not it was successfully completed), and may also contain content requested by the client (see section 2.5.3.4 below). HTTP resources are identified and located on the Internet using URIs or, more specifically, URLs (Uniform Resource Locators) employing the HTTP or HTTPS schemes.

2.5.3.4 Messages

RFC 2616 [91] specifies that HTTP request and response messages consist of:

1. a start-line, which may be a request or a status line;
2. zero or more header fields, also known as *headers*, which are colon-separated, name-value pairs in clear-text string format;
3. an empty line, indicating the end of the header fields; and
4. an optional message-body (e.g. an HTTP response message may include HTML content).

The start-line and headers must end with a carriage return followed by a line feed (commonly written as CRLF).

Every HTTP request contains an HTTP *method*, used to indicate the operation that is being requested by the client. Examples of HTTP methods include:

2. IDENTITY, PRIVACY AND SECURITY

- *GET*, which should be used to retrieve data from the server (so that the client can display a representation of the retrieved data); and
- *POST*, which is typically used to submit data to the server to be processed (see also section 2.5.1.2).

Table 2.1 shows the structure of HTTP request and response messages.

Table 2.1: HTTP Request and Response Messages

	HTTP Request Message	HTTP Response Message
Start-line	Request line, e.g. <i>GET Thesis.html</i> , which requests a resource called <i>Thesis.html</i> from the server.	Status line, e.g. <i>HTTP/1.1 200 OK</i> , which indicates that the client's request has succeeded.
Headers	E.g. <i>Accept-Language: en</i> , which indicates the acceptable language(s) for the response (English in this case).	E.g. <i>Content-Type: text/html</i> , which indicates the content type.
Empty Line	CRLF	CRLF
Optional Message Body		E.g. HTML content
Example	<i>GET Thesis.html HTTP/1.1</i> <i>Accept-Language: en</i> <i>Host: www.website.com</i>	<i>HTTP/1.1 200 OK</i> <i>Date: Mon, 5 May 2012 22:38:34</i> <i>Content-Type: text/html</i> <i>Content-Length: 1354</i> <html> <body> <h1>PhD Thesis</h1> This thesis is </body> </html>

2.5.3.5 HTTP and State

HTTP is a stateless protocol, i.e. it does not require the server to retain information or status about a client between requests. This causes problems in

applications where session state is needed. Common methods for rectifying this problem to enable state to be included in HTTP include the following.

- *HTTP cookies* are used by a server to send state information to a client (browser); they are stored by the client, and returned by the client to the server on the next occasion that an HTTP request is sent to that server [29] (see also section 2.3.4.1).
- *Hidden (HTML) form variables* are HTML input tags of type *hidden* (e.g. `<input type='hidden' name='username' value='PUAI003' />`).
- *URL-encoded parameters* are described in section 2.5.3.7 below.

2.5.3.6 Redirects

An HTTP redirect is a response message containing a status code that instructs a UA (a web browser) to go to another location. An annotation is included in the message to describe the reason for the redirect. HTTP defines a number of status codes, including 300 (multiple choices), 301 (moved permanently), and 307 (temporary redirect). All these status codes require that the URL of the redirect target be given in the location header of the HTTP message response. Server-side or client-side scripts can redirect the UA from one site to another, either manually, e.g. via a clickable link or button, or automatically, e.g. using JavaScript. However, such redirects can pose security and privacy risks (see sections 3.5.1 and 3.7.3).

2.5.3.7 Query Strings

As mentioned in section 2.5.3.5, one means of passing data (including state) from a client to a server is to embed it in the URL. A query string, or a URL-encoded parameter, is the part of a URL that contains data to be passed to a web server. A question mark (?) is used to separate the query string from the rest of the URL [31]. A redirect URL may contain query strings, thereby

allowing a website in one domain to transfer arbitrary data to another site in a different domain.

An HTTP query string can be composed of a series of field-value pairs, separated by the ampersand (&) or the semicolon (;), e.g. `http://www.website.com/index.php?SessionID=UniqueSessionCode&Student=Haitham&Supervisor=Chris`. When a query string is employed to transfer a form data set using the GET method (see section 2.5.1.2), it contains a field-value pair for each field in the form (including the hidden fields). That is, form fields are included in the query string when the (GET) HTML form is submitted. In order for some characters, such as the space character, to be added to a URL, they must first be converted using URL encoding [31]. For example, the space character is encoded as the plus (+) sign.

An HTTP resource (identified by a URL) requested by client from a web server may be a plain file (such as an HTML file) or a program. If the resource is a:

- file, then any query string contained in the corresponding URL is ignored; and
- program, then the program may ignore part or all of the query string. However, the program can be coded to make use of query strings.

Note that data passed via query strings or HTML forms must be sanitised by the processing program or agent. This is to protect against a wide range of attacks, including malicious code or SQL injections [76].

2.5.3.8 Issues With Query Strings

Regardless of whether or not it contains a query string, the full URL including any query string is stored in the server log files. Query strings can therefore pose security and/or privacy issues, e.g. they can be used to track users in a manner similar to that posed by HTTP cookies (see section 2.3.4.1).

When a user requests a web page from a web server, a unique identifier can be appended as a query string¹⁰ to the URLs of all links the page contains. If the user follows one of these links, then the request to the server will include this unique identifier, thereby making it possible to establish that all these pages have been viewed by the same user, i.e. multiple requests may be linked and perhaps traced to the user responsible. Unlike HTTP cookies, which can be disabled, an end user cannot disable query strings, and hence query string-based tracking should work in most situations. Whereas query strings form a part of the URL, and are therefore included if the user saves or sends the URL to another user, cookies are not saved or stored with the URL. Nonetheless, as stated previously, cookies can survive across different browsing sessions (see section 2.3.4.1).

Possible limits on the length of a URL can also pose problems with the use of URL query strings. Some older servers may impose restrictions on the URL length; indeed the HTTP specification [91], whilst not specifying a maximum URL or path length, states that servers should be cautious about depending on URI lengths greater than 255 bytes, because they may not be properly supported by some older client or proxy implementations. Additionally, the HTML 3 specification [201] states that the attribute value, e.g. a URL set as the value of the *href* attribute in a hyperlink tag, is limited to 1024 characters¹¹; however, the HTML 4 specifications [202] omit this restriction¹². According to the Microsoft help and support centre, Microsoft Internet Explorer supports a maximum URL length of 2083 characters¹³.

2.5.4 SSL/TLS

The SSL (Secure Sockets Layer) protocol and its successor, TLS (Transport Layer Security), provide a secure communication channel above the IP layer

¹⁰Note that the addition of such a query string does not change the way the page is displayed to the user.

¹¹<http://www.w3.org/MarkUp/html3/HTMLandSGML.html>

¹²<http://www.w3.org/TR/html4/intro/sgmltut.html#h-3.2.2>

¹³<http://support.microsoft.com/kb/208427>

and below application layer protocols such as HTTP. SSL was produced by Netscape¹⁴, and was first published in 1995 [111]. A modified version of the proprietary SSL protocol was published in 1999 by the Internet Engineering Task Force (IETF) under the name TLS; TLS 1.0 [82] essentially functions as version 3.1 of SSL. A revised version of TLS, version 1.2, has recently been published [83]. SSL/TLS has been widely discussed in the literature [77, 214, 217].

SSL/TLS enables client-server applications to exchange data across a network in a way that provides reliable data confidentiality and integrity. To achieve this, SSL/TLS encrypts and MAC-protects segments of network connections above the transport layer, to provide confidentiality and integrity services.

2.5.5 HTTPS

HTTPS (Hypertext Transfer Protocol Secure) [207] is a combination of HTTP and SSL/TLS [215]. It provides an encrypted and integrity-protected client-server communication channel, building on an initial authentication process. This initial authentication can, in principle, be mutual, but in practice the server is typically authenticated to the browser (i.e. the client) but not vice versa. HTTPS is supported by modern browsers, including Internet Explorer, Firefox, Safari, and Chrome. Note that the trust inherent in HTTPS is based on the CA public keys that come pre-installed in the browser software, which are used to verify certificates provided by web servers.

If HTTPS is used, then the HTTP protocol elements are encrypted for transmission, including [215]:

1. the URL of the requested document;
2. the contents of the document;
3. the contents of HTML forms (as completed by a user);

¹⁴<http://netscape.aol.com/>

4. the cookies exchanged between a browser and a server; and
5. the contents of the HTTP header.

2.5.6 Web Service Protocols

Web service protocols are open, XML-based, specifications, which are often referred to collectively as *WS-** (since many of them have a name beginning with *WS-*). *WS-** protocols¹⁵ are designed to enable interoperable, cross-platform software communication. We outline a number of technologies and protocols associated with web service protocols, relevant to this thesis.

XML (eXtensible Markup Language) is a self-descriptive, platform-independent, markup language. It resembles a general-purpose version of HTML, except that whereas HTML tags serve to instruct web browsers how to render a web page, XML is designed to represent, transport and/or store a range of types of structured data. XML is platform-agnostic, and is readable by machines and also by humans, where size and complexity permits. XML is specified in a W3C recommendation [44], the most recent version of which was published in 2008.

SOAP is a standardised XML-based protocol [105], defining the format of web service messages and specifying how such messages should be exchanged between end points. A typical, simple SOAP¹⁶ message is enclosed inside an *envelope* tag, which contains a *header* and a *body* element [181].

WSDL (Web Services Description¹⁷ Language) [38, 69, 70] is an XML-based language which can be used to describe the functionality offered by a web service. Such a description specifies how a web service should

¹⁵http://en.wikipedia.org/wiki/List_of_Web_service_specifications

¹⁶SOAP was originally intended as an acronym for Simple Object Access Protocol; however, the current specification [105] explicitly drops this acronym.

¹⁷Note that the acronym has changed from version 1.1, where the D stood for Definition.

be called, what parameters it expects, and what data structures it returns [181].

UDDI (Universal Description, Discovery and Integration¹⁸) is an XML-based service enabling businesses worldwide to list themselves on the Internet. It also provides a mechanism to register and locate web service applications. Using UDDI, businesses can publish service listings online, thereby discovering each other. UDDI is designed to be interrogated by SOAP messages and to provide access to WSDL documents describing how to interact with the web services listed in a directory [181].

Fig. 2.4 (due to Malik and Tomlinson [162]) shows the relationships between web services technologies.

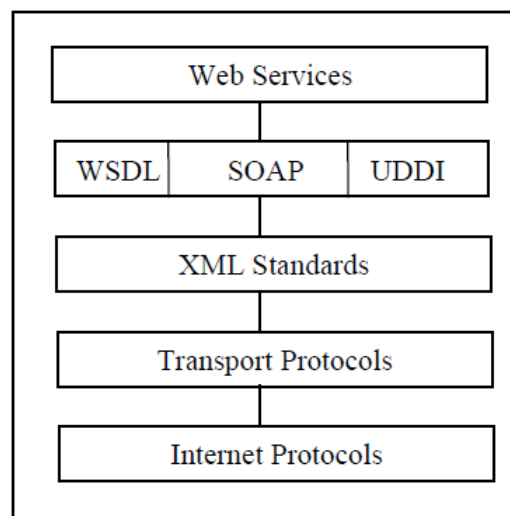


Figure 2.4: Web Services Technologies

We conclude this brief discussion of web services by listing the WS-* standards of greatest relevance to this thesis.

WS-Policy/WS-SecurityPolicy is used to describe security policies [27, 80].

Note that a website can alternatively describe its policy in HTML or XHTML (eXtensible Hyper Text Markup Language) [23].

¹⁸<http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm>

WS-MetadataExchange is used to fetch security policies and exchange service description metadata over the Internet [28]. Note that a website can transmit its security policy using HTTP/S.

WS-Trust is used to acquire security tokens (e.g. SAML tokens) from identity providers [24].

WS-Security is used to securely deliver security tokens to service providers [176]. Note that HTTP/S can also be used.

2.5.7 SAML

SAML (Security Assertion Markup Language) is an XML-based standard for exchanging identity-related information across the Internet. The SAML specifications cover four major elements.

SAML assertions are XML data structures containing cryptographically-protected identity information. An assertion may contain three types of statement:

1. an *authentication statement*, asserting that a user was authenticated at a particular time using a specific authentication method;
2. an *attribute statement*, asserting that a user is associated with certain attributes; and
3. an *authorisation decision statement*, asserting that a particular user is permitted to perform a certain action on a specific resource.

SAML protocols define data structures for sending SAML requests and returning assertions.

SAML bindings map SAML protocol messages onto standard communication protocols, e.g. HTTP.

SAML profiles describe how SAML assertions, protocols and bindings are combined together to support a particular use case.

Each of the elements listed above builds on the elements earlier in the list. For example, SAML assertions can be used without the protocol profiles and bindings, but not vice versa [21].

SAML 1.0 was first adopted as an OASIS standard in 2002; a minor revision, SAML 1.1 [161], was formally adopted in 2003. A major revision led to SAML 2.0 [59], which became a standard in 2005. The differences¹⁹ between version 1.1 and 2.0 are significant, and SAML assertions of the two types are incompatible.

SAML 2.0 incorporates features from SAML 1.1, Liberty ID-FF 1.2 (see section 4.7), Shibboleth 1.3 (see section 4.8), and other proprietary extensions. However, although ID-FF 1.2 was contributed to OASIS and used as input to the definition of SAML 2.0, there are a number of important differences²⁰ between SAML 2.0 and ID-FF 1.2; in particular, the two specifications, despite their common roots, are incompatible.

Finally, note that, although the specifications allow SAML to be used as a protocol to exchange identity data between entities, it has been widely used by identity management systems as a token format and not as a full-fledged identity management system [34]. In this thesis, unless otherwise stated, the term *SAML* refers to a token format.

¹⁹<https://spaces.internet2.edu/display/SHIB/SAMLDiffs>

²⁰<http://saml.xml.org/differences-between-saml-v2-0-and-liberty-id-ff-1-2> and <https://wiki.shibboleth.net/confluence/display/SHIB/SAMLLibertyDiffs>

Identity Management

3.1 Introduction

This chapter provides an introduction to identity management covering related topics. The chapter is organised as follows. Section 3.2 provides a definition of identity management, and, in section 3.3, we consider why it is needed. Section 3.4 gives an abstract model for identity management. Section 3.5 describes the supporting infrastructures, and section 3.6 addresses single sign on. In section 3.7, we consider a range of possible properties of identity management systems. Finally, section 3.8 gives Cameron's laws of identity.

3.2 Definition

In a recent draft of ITU-T recommendation Y.2720 [121] *identity management* is defined as 'a set of functions and capabilities (e.g. administration, management and maintenance, discovery, communication exchanges, correlation and binding, policy enforcement, authentication and assertions) used for:

- assurance of identity data (e.g. identifiers, credentials, attributes);
- assurance of the identity of an entity (e.g. users/subscribers, groups, user devices, organisations, network and service providers, network elements and objects, and virtual objects); and
- enabling business and security applications'.

We adopt this definition throughout this thesis.

Identity management encompasses the complete lifecycle of identity information from initial enrolment to archiving or deletion [132], and includes governance, policies, processes, data, technologies, and standards. Typical issues deemed to be part of identity management include:

- application(s) that implement an identity store;
- authentication of the identity of an entity;
- the provision of credentials and services to support authentication of an entity;
- establishment of the provenance of identity information;
- establishment of the link between identity information and an entity;
- maintenance of identity information;
- protection of the integrity of identity information; and
- mitigation of risks applying to identity information (e.g. theft or misuse) [132].

For an organisation, identity management forms a basis for the secure management of interactions with its employees and/or clients. For an individual, identity management can be used to help improve security and reduce the risk of privacy threats [132]. However, the implementation of identity management has the potential to raise serious privacy concerns for end users, because it involves the storage, processing, and transformation of identity information (including PII — see section 2.3.2.1). In addition, requirements for identity management and privacy may conflict; although protecting privacy requires that the amount of identity information held by a third party is minimised (see section 2.3.5), this may adversely affect the level of assurance regarding the accuracy of a claimed identity [21].

3.3 Need for Identity Management

As the number of on-line services requiring user authentication continues to grow, so does the number of (user-possessed) digital identities. Management of large numbers of such identities by a user gives rise to major usability and security issues. These issues appear likely to have contributed to the recent rapid growth in identity-oriented attacks, including phishing and pharming. To help address these issues, a range of identity management systems have been proposed. These systems are designed to simplify and increase the security of the management of user identities, thereby protecting against potential identity attacks.

Given the growing popularity of the web, this thesis focuses on web-based identity management systems. Such systems use the World Wide Web and WS-* protocols (see section 2.5.6) to communicate between the involved parties, and have been developed to help users manage their digital identities on the web [5, 21, 39, 145].

3.4 Abstract Model

The majority of proposed identity management systems share a number of technical features and have similar objectives. Architectures for identity management typically involve the following roles, as illustrated in Fig. 3.1.

1. The *user* wishes to access a resource offered by a service provider (see below). The user is also known as the *principal*. The user employs a *user agent* (UA), i.e. software (such as a web browser) used to send requests to identity providers and/or service providers (see below) and receive responses from them (see also section 2.5.3.2).
2. The *identity provider* (IdP)¹ issues an assertion token (i.e. a data structure containing statements about the user) to a user. The IdP is some-

¹The term *identity provider* is occasionally abbreviated to *IP* [34]; however, we do not adopt this abbreviation since IP is a widely-used acronym for the Internet Protocol [184].

times referred to as an *identity authority*.

3. The *service provider* (SP) consumes an IdP-issued assertion token in order to make an authentication and/or authorisation decision. Since the SP relies on the correctness of the provided assertion token, it is also referred to as a *relying party* (RP), and we use this term throughout. The RP thus offloads the burden of user authentication to the IdP [152].

Web-based identity management systems typically employ the following sequence of interactions (see Fig. 3.1).

1. A user employs a UA to access an RP-protected resource. Before granting access, the RP indicates to the user that it requires an assertion token issued by an IdP trusted by the RP.
2. An IdP with which the user has a relationship is asked to supply an assertion token meeting the requirements of the RP.
3. If necessary², the IdP authenticates the user, and, if successful, issues an assertion token vouching for the user's identity.
4. The user presents the IdP-issued token to the RP, and the RP relies on the token to decide whether or not to grant access to the requested resource.

3.5 Supporting Infrastructure

We next consider certain properties of the infrastructure required to support the sequence of interactions identified immediately above.

²Unless re-authentication is explicitly requested by the RP, user authentication may be unnecessary if an authenticated IdP-user session already exists.

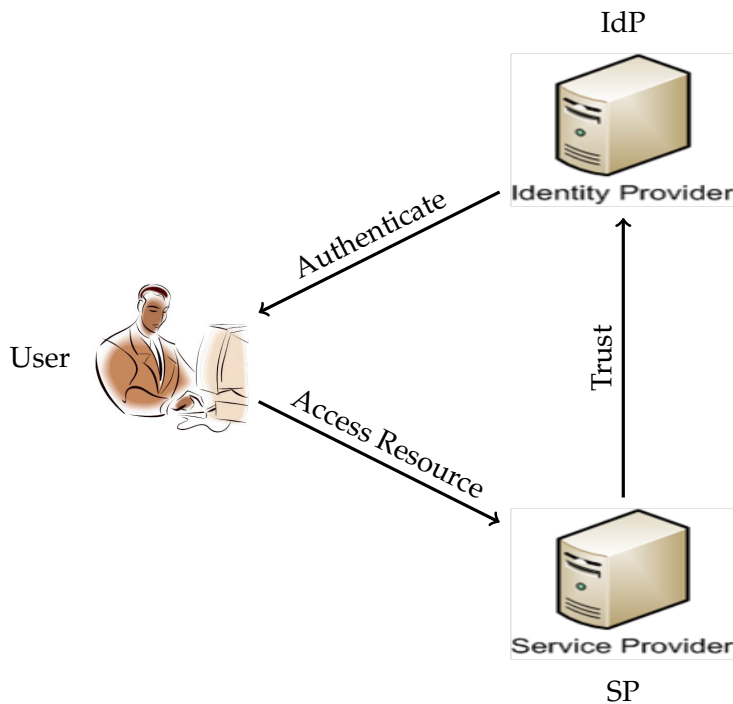


Figure 3.1: Identity Management Model

3.5.1 IdP-RP Communications

As stated previously, we assume that the involved parties (i.e. the RP, IdP and UA) intercommunicate using web protocols. An RP and IdP can intercommunicate in one of two main ways, as follows.

1. *Direct communication*, also referred to as *back-channel communication*, involves a direct communication channel between an RP and an IdP, i.e. communications do not pass via the user platform (or the UA). In some identity management systems an assertion token is passed from the IdP to the RP via such a back-channel; in such cases a temporary ID or a pseudonym is sometimes used to refer to the user instead of an identifiable user attribute, such as an email address. Such a technique helps to preserve user privacy.

If the back-channel between the IdP and RP is SSL/TLS-protected then (depending on the associated security policy) it may be unnecessary

for the IdP to independently encrypt the assertion token. This is because an SSL/TLS channel provides data integrity and confidentiality services (see sections 2.5.4 and 2.5.5).

2. *UA-managed communication* describes the case where the UA acts as a relay for communications between an RP and an IdP. There are various ways in which this could happen; we outline two of particular importance in identity management systems.

a) In the case where the UA is a web browser with no special functionality added to support identity management, HTTP redirection (see section 2.5.3.6) can be used. Such a UA is sometimes referred to as a *passive UA* or *passive client* [25, 26].

b) Some identity management systems require the user platform to be equipped with a special client component to support IdP-RP communications, thereby extending the functionality of an ordinary UA. Such a UA is sometimes referred to as an *active UA* or *active client* [25, 26].

The active client approach has a number of advantages, including the following.

- The client component could provide discovery services (see section 3.5.2 below).
- The client component could help to mitigate phishing attacks, to which the redirection approach is prone (see section 1.2.3).
- If an assertion token is passed from the IdP to the RP via the UA, then certain identity management systems enable the UA to provide *proof-of-rightful-possession* services, also known as *subject confirmation* or *proof-key* methods. These allow the UA (on behalf of the user) to prove its rightful possession of the assertion token to

the RP; this helps to mitigate a wide range of attacks including user-impersonation attacks.

- Some identity management systems use the client component to provide a consistent user experience, and/or to obtain user consent to the use of a particular IdP and/or to the transfer of an assertion token to the RP.

It is often necessary to explicitly protect the confidentiality and/or integrity of assertion tokens whilst in transit between the IdP and the RP via the UA. Integrity protection can be achieved either by computing a MAC using a shared secret key, or by the generation of a signature by the IdP using a dedicated IdP private pair. Similarly, confidentiality can be protected by encryption, e.g. using an asymmetric algorithm in conjunction with the public key of the RP. End-to-end protection of this type has value over and above the protection provided by SSL/TLS sessions, since it prevents manipulation or eavesdropping by a malicious user platform.

3.5.2 Discovery

The *discovery* process, also known as *discovery of identity source* [21, 118], is concerned with selecting and locating the IdP that is to be asked to provide an assertion token. There are two main approaches to providing a discovery service.

1. In a *server-based process*, the RP is responsible for performing discovery. Such an approach is typically used when there is no additional client component installed on the user platform, i.e. the passive client case. However, server-based discovery is susceptible to attacks in which a malicious RP redirects the UA to a fake IdP of its choosing; such a fake IdP could capture user credentials and/or fraudulently obtain sensitive user attributes (see section 1.2.3).

2. In a *client-based process*, which only applies in the active client case, a locally running client component is responsible for performing discovery. In this case, IdP addresses are typically stored by the client component. False IdP attacks are mitigated, since an RP can no longer redirect the user to an IdP of its choosing.

3.6 Single Sign on (SSO)

SSO [32, 78, 195, 228] involves a user authenticating only once to an IdP and thereby gaining access to multiple RPs without the need to sign-on separately at each of them. *Single sign off* is the reverse process, i.e. where a user signs-off only once and is then automatically signed-off from all accessed RPs. An identity management system that supports SSO typically also supports single sign off.

The notion of SSO is clearly attractive, not least from a user convenience perspective, particularly as the number of on-line services requiring authentication continues to grow. Furthermore, SSO helps to reduce the risk of exposure of passwords to malicious parties, including through key logging and shoulder surfing.

However, SSO also raises potential security and privacy concerns. Compromising the authentication process at the IdP enables the adversary to access all the participating RPs. In addition, SSO could result in a single point of failure; failing to authenticate to the IdP could automatically impede access to all the participating RPs. Use of SSO could also lead to privacy violations; user interactions on the web could be linked by an IdP to build a unique user profile.

3.7 Properties of Identity Management Systems

In this section we consider a range of properties that may be possessed by an identity management system.

3.7.1 Information Card Systems

*Information Card systems*³, also referred to as InfoCard- or iCard-based systems, are identity management systems that are based on the digital card metaphor. They must conform to the OASIS-standardised Information Card protocol [143]. Such systems are active client-based, where the active client uses a card-based user interface to enable users to manage and select IdPs.

Each InfoCard specifies a set of attribute (or claim⁴) types, the values of which can be obtained from the issuing IdP. This is analogous to real-world physical cards, where each card (e.g. a driving licence, credit card, passport, etc.) asserts a set of user attributes. Information Card systems also allow users to issue self-asserted claims (see section 4.3.1).

IdP discovery is performed on the user platform; if a user selects a card, the user is also implicitly selecting an IdP, and the card contains the URL of the IdP server. Information Card users can also review (and possibly modify) the contents of an IdP-supplied security token before it is released to an RP.

One widely discussed example of an Information Card-based system is CardSpace (see section 4.3). Other examples include Higgins (see section 4.4), OpenInfoCard⁵, and DigitalMe⁶, the last of which is supported by the Bandit Project⁷. This thesis concentrates on CardSpace, given its ubiquity as part of Windows Vista and Windows 7. However, many of the observations made in the remainder of this thesis regarding CardSpace also apply to other Information Card systems.

³<http://informationcard.net>

⁴Information Card systems are also occasionally referred to as claim-based systems [33].

⁵<http://code.google.com/p/openinfocard>

⁶<http://code.bandit-project.org/trac/wiki/DigitalMe>

⁷<http://bandit-project.org>

3.7.2 Federated Systems

In a *federated system*, RPs and IdPs group together to form a federation, also known as a *circle of trust* (CoT), bound together by contractual agreements and mutual trust.

The existence of a federation simplifies the discovery process, since the IdPs are defined by the CoT. Also, the trust relationships between parties are clear, since they are defined by the CoT. However, there may be only one IdP in a CoT, in which case the privacy problems inherent to centralised systems (see section 3.7.4 below) apply.

Examples of federated systems include Liberty and Shibboleth (introduced in sections 4.7 and 4.8, respectively).

3.7.3 Communication-based Models

As discussed in section 3.5.1, identity management systems can be classified according to how the RP communicates via the user platform with the IdP. There are two main ways in which this can be achieved, namely by using an HTTP redirect or involving an active client.

3.7.3.1 Redirect-based Identity Management Systems

In such a scheme, the UA is redirected by an RP to an IdP (and vice versa). In such a case the UA is essentially passive, and does not need to be aware of the identity management system in use (see also section 3.5.1). One major disadvantage of this approach is that a malicious RP can redirect the UA to a malicious IdP impersonating a genuine IdP, e.g. to fraudulently obtain user credentials. Examples of such systems include OpenID, Liberty (artifact and browser-post profiles), Shibboleth, and OAuth (e.g. as instantiated by Facebook Connect — see section 4.6.4).

3.7.3.2 Active Client-based Identity Management Systems

In schemes of this type the UA must incorporate an active client which acts as an intermediary between RPs and IdPs, and which must be aware of the identity management system in use. Typically all communications between an RP and an IdP occur via this active client, and there is usually no need for direct RP-IdP communication. Depending on the details of the system in use, the active client can prompt the user to select a digital identity, choose an IdP, review (and possibly modify) a security token created by the IdP, and approve a transaction. Phishing attacks are mitigated since an RP cannot redirect the UA to an IdP of its choosing. The active client can also aid in providing a consistent user experience, and its existence helps to give the user a greater degree of control (see also section 3.5.1). Examples of such systems include CardSpace and a Liberty-enabled client.

3.7.4 Other Properties

An isolated system, also sometimes referred to as *a silo system* [146], is a system in which an RP takes the role of the IdP and manages the entire identity life cycle, including issuing, consuming and deleting user identities. A user may be allowed to use a self-defined identifier (e.g. a username) and a self-issued credential (e.g. a password). This approach is simple to deploy and use, and the identity data are only ever released to one party (the RP). However, users must set up and remember an identifier-credential pair for every RP with which they have a relationship.

A centralised system [146] corresponds to the case where a single entity acts as the sole IdP for a set of RPs. The IdP manages the identity life cycle, including issuing, consuming and deleting security tokens for all its users. This can be regarded as a special case of the general model given in section 3.4 where there is a single IdP. A user only needs to manage a single identifier-credential pair since there is only one IdP. However, such

an approach poses a potentially serious privacy threat to users, since user attributes are managed and used by a single entity. This single IdP could abuse this knowledge, e.g. by profiling user activities. A single IdP also represents a single point of failure [117], since its unavailability would impede access to the participating RPs. Microsoft Passport (see section 4.2) was an example of such a system, although it never achieved much success in this form [56, 169].

3.8 Cameron's Laws of Identity

Cameron [56] has proposed a set of seven so called *laws of identity*, capturing what he asserts are fundamental requirements for a widely-acceptable identity architecture. These laws emerged from an open dialogue with identity management experts and the wider public. We present the laws (text in quotes) as stated by Cameron [56], along with a brief interpretation where necessary.

1. **User Control and Consent.** 'Technical identity systems must only reveal information identifying a user with the user's consent.' The user must therefore be made aware of what items of personal information are being requested, by whom, and for what purpose.
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.'
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.'
4. **Directed Identity.** 'A universal identity system must support both *omni-directional* identifiers for use by public entities and *unidirectional* identifiers for use by private entities, thus facilitating discovery while

preventing unnecessary release of correlation handles.' Correlation handles enable different user activities to be linked.

5. **Pluralism of Operators and Technologies.** 'A universal identity system must channel and enable the interworking of multiple identity technologies run by multiple identity providers.'
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.'
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 seems reasonable to believe that, by following the laws stated above, identity management systems can reach a reasonable level of usability, reliability, flexibility, and privacy. A number of these laws relate closely to the OECD principles for personal data protection (see section 2.3.5.1). As noted by Alrodhan [21], the requirements of:

- law 1 are covered by principles 6 and 7;
- law 2 are covered by principles 1, 2 and 4; and
- law 3 are covered by principles 1 and 3.

Identity Management Systems

4.1 Introduction

This chapter describes in detail those identity management systems of greatest relevance to this thesis, namely CardSpace, Higgins, OpenID, OAuth, Liberty, and Shibboleth. The chapter also gives an overview of certain other systems of background relevance, namely Microsoft Passport, U-Prove, and IdeMix.

The chapter is organised as follows. Sections 4.2–4.8 respectively contain descriptions of Passport, CardSpace, Higgins, OpenID, OAuth, Liberty and Shibboleth. Section 4.9 provides an overview of anonymous credential systems, as well as describing two examples, namely U-Prove and IdeMix. Finally, section 4.10 compares between these systems.

4.2 Microsoft Passport

The design of, and reaction to, Passport appear to have provided part of the motivation for the development of CardSpace and other similar systems.

4.2.1 Introduction

.NET Passport, introduced by Microsoft in 1999, is an SSO system that enables users to sign-on to multiple websites using a single set of credentials. A Passport user shares account credentials (typically an email address and password) with the Passport server, which could also store other user attributes such as first name, last name, date of birth, etc. [64]. Passport-

participating RPs must install special software to support Passport, and must also share a secret key with the Passport server. Passport requires the use of SSL/TLS between the user browser and the Passport server, and optionally between the browser and a participating RP [195].

4.2.2 Operation

Passport works in the following way.

1. A user employs a UA to visit a Passport-enabled RP, say `a.com`, and clicks on a special Passport sign on button.
2. The UA is redirected to the Passport site, which, if necessary (i.e. if a suitable cookie is not present on the user platform), authenticates the user.
3. If authentication is successful, the UA is redirected back to the RP. At the same time, four encrypted, time-limited, cookies are sent back to, and stored by, the UA (the browser). These cookies are a ticket granting cookie (TGC) containing a secret key, a participating-sites cookie, an authentication cookie, and a profile cookie. The last three cookies are encrypted using the secret key contained in the TGC; the TGC is itself encrypted under the secret key shared between the Passport service and the RP.
4. The RP (`a.com`) decrypts the received cookies, and, if satisfied, grants the user access.

Note that the process repeats if another Passport-enabled RP, say `b.com`, is visited; however, the Passport service no longer needs to authenticate the user as long as the cookies submitted by the browser to the Passport server remain valid. Every time the Passport service is used it updates these cookies, including the participating-sites cookie which contains a list of all Passport participating sites that the user has visited during this session. Finally,

if the user signs-out from Passport, then the user is also logged-out of all Passport-enabled RPs accessed during this session. The four cookies are deleted from the user's browser and the participating-sites cookie is used to notify the participating sites to close the user session.

4.2.3 Criticism and Consequences

Passport was initially supported by some well-known RPs, e.g. eBay and Visa; however, it was not a universal success and received much criticism [168]. It would appear that Passport was originally intended to become *the* IdP for the Internet. Because Microsoft alone provided the Passport service, the participating RPs needed to trust Microsoft to store the user credentials securely, and to authenticate the users correctly. Users would have also needed to be willing to accept Microsoft's involvement in web activities completely unrelated to Microsoft. However, although Passport failed to become the universal Internet IdP, it has been and continues to be successfully used with Microsoft-affiliated websites (see below). Other potentially undesirable properties of Passport include that it does not provide adequate user control, nor appropriate privacy protection [64]; indeed it could violate user privacy, since Passport enables Microsoft to track user activities over the Internet.

Microsoft has re-branded Passport as *Windows Live ID*, and it is now the main method for Microsoft-affiliated websites to authenticate users. Microsoft has published a number of white papers [56, 169], discussing the *failure* of Passport; this discussion appears to have influenced Microsoft's subsequent offerings in this area, including in particular CardSpace. It also appears to have provided the motivation for the development of the laws of identity (discussed in section 3.8).

4.3 CardSpace

4.3.1 Introduction

CardSpace [34, 168] is an identity management system that provides a secure and consistent way for users to control and manage personal data, to review data before sending it to a website, and to verify the identity of visited websites. It also enables websites to obtain personal information from users, e.g. to support user authentication and authorisation. CardSpace is InfoCard-based and involves an active client.

In CardSpace, digital identities are visually represented to users as InfoCards, implemented as XML files that list the types of claim made by one party about itself or another party. A claim is a piece of information about an individual or an entity that an IdP asserts about that individual or entity [143]. Both user identifiers (e.g. a username) and user attributes (e.g. birthday) are treated as claims. CardSpace is designed to reduce reliance on username-password authentication, and to provide a consistent user experience across the web to improve user understanding of the authentication process. CardSpace is also designed to reflect the seven identity laws proposed by Cameron (see section 3.8).

The concept of an InfoCard is inspired by real-world cards, such as driving licences and credit cards. A user can employ one InfoCard with multiple websites. Alternatively, just as different physical ID cards are used in distinct situations, separate InfoCards can be used at disparate websites, helping to enhance user privacy and security. If InfoCards are obtained from different IdPs, the credentials referred to by such cards are remotely stored in distinct locations, potentially improving reliability and security, as well as giving users flexibility in choosing points of trust.

There are two types of InfoCards: personal (self-issued) cards and managed cards. Personal cards are created by users themselves, and the claims

listed in such an InfoCard are asserted by the self-issued IdP (SIIP)¹ that co-exists with the CardSpace identity selector (see Fig. 4.1) on the user platform. Managed cards, on the other hand, are obtained from remote IdPs.

The InfoCards themselves do not contain any sensitive information; instead an InfoCard carries metadata that indicates the types of personal data that are associated with this identity, and from where assertions regarding this data can be obtained. The data referred to by a personal card is stored locally on the user platform, whereas the data referred to by a managed card is held remotely by the IdP that issued it [34, 63, 96, 142, 168, 169, 210].

By default, CardSpace is supported in Internet Explorer from version 7 onwards. Extensions to other browsers, such as Firefox² and Safari³, also exist. An updated version of CardSpace, known as *CardSpace 2.0 Beta 2*, was released, although Microsoft announced in early 2011 that it will not ship; instead Microsoft has released a technology preview of U-Prove (see section 4.9.2). Unless explicitly stated, in this thesis we refer throughout to the CardSpace version that is shipped by default as part of Windows Vista and Windows 7, that is available as a free download for XP and Server 2003, and which has been approved as an OASIS standard [143].

4.3.2 InfoCard Contents

The following data items are contained in an InfoCard [34].

1. *Card metadata*, which includes:
 - a) *card ID*, which can be used to index the card;
 - b) *card version*, used to determine whether a current card is newer or older than an existing card with the same card ID in the card store;

¹Although some CardSpace documents occasionally abbreviate the self-issued identity provider to *SIP*, we have used the abbreviation *SIIP* since *SIP* is widely used to refer to the Session Initiation Protocol [209].

²<https://addons.mozilla.org/en-US/firefox/addon/10292>

³<http://www.hccp.org/safari-plug-in.html>

4. IDENTITY MANAGEMENT SYSTEMS

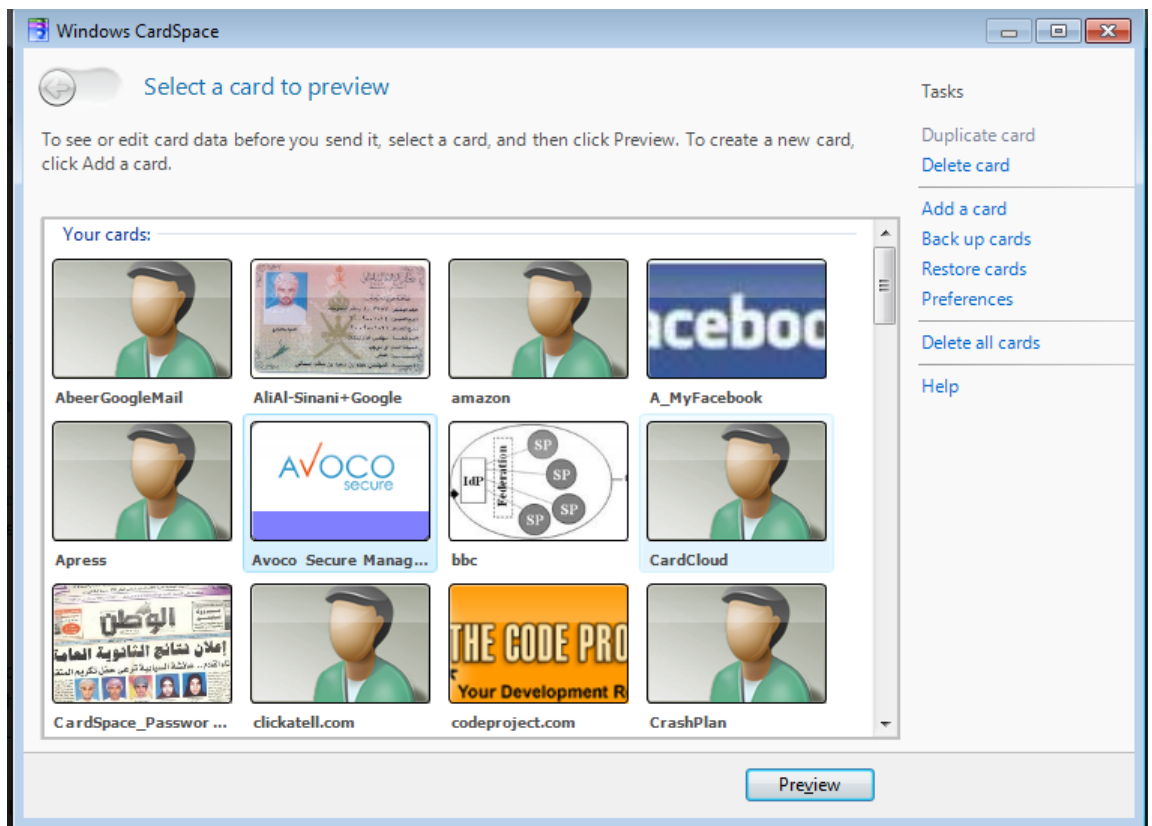


Figure 4.1: The CardSpace Identity Selector

- c) *issuance and expiry time*, timestamps specifying when the card was issued and when it will expire; and
- d) *display information*, an image and/or a name for the card.

2. *Issuer data*, which includes:

- a) *issuer name*, specifying the name of the IdP that issued the managed card. Since this is a name rather than a URI, a single IdP can have multiple URIs; this may be useful in certain scenarios, such as setting up a fallback IdP server (i.e. an STS)⁴ or deploying multiple servers in different geographical regions;
- b) *IdP location*, specifying the URIs of the IdP servers from which a security token can be obtained.

⁴The STS (Security Token Service) is responsible for security policy and token management within an IdP and, optionally, within an RP [142].

- c) *MEX address*, an HTTPS-based IdP URL used for metadata exchange and for IdP server authentication; and
 - d) *identity*, an X.509 certificate (see section 2.4.4.1) used to support authentication of the IdP to the user.
3. *Privacy policy*, a URL for a document specifying the IdP's privacy policy.
 4. *User authentication method(s)*, specifying the method(s) employed by the IdP to authenticate the user. CardSpace supports four authentication methods, namely username-password, a Kerberos version 5 service ticket, an X.509 version 3 certificate (either software-based or using a smart card), and a self-issued token.
 5. *A list of supported attribute types*, e.g. birthday, first name, and last name, the values of which are known by the IdP, and for which the IdP will be prepared to generate a security token.
 6. *A list of supported token type(s)* indicating which type(s) of security token, e.g. SAML assertions, the card issuer is capable of issuing.

4.3.3 Attribute Exchange

CardSpace supports the assertion of user attributes by an IdP to an RP, both when using third party IdPs and when using the SIIP. We consider the two cases separately below.

4.3.3.1 Personal Cards

The identity selector allows a user to create a personal card and populate its fields with self-asserted claims. To protect users from disclosing sensitive information, CardSpace restricts the contents of personal cards to non-sensitive data, such as that published in telephone directories. More specifically, personal cards can only contain claims of 14 predefined and editable

types, namely *first name, last name, email address, street, city, state, postal code, country/region, home phone, other phone, mobile phone, date of birth, gender, and web page*. Data inserted in personal cards is stored in encrypted form on the user platform.

4.3.3.2 Managed Cards

A managed card can specify any attribute type that is supported by the issuing IdP. The InfoCard will list the attribute types for which the IdP holds user-specific values.

4.3.4 IdP Discovery

When selecting an InfoCard (personal or managed), the user is also implicitly selecting an IdP. If a personal card is chosen then the selected IdP is the local SIIP, whereas if a managed card is chosen then the selected IdP is the remote IdP that issued it. The InfoCard XML encoding contains either the address of a remote IdP or an indication that it is a self-issued card.

4.3.5 IdP-RP Negotiation

Unlike Liberty or Shibboleth (described in sections 4.7 and 4.8, respectively), CardSpace does not require an explicit contractual agreement to be made or a federation process to be completed before it can be used. Instead it uses dynamic negotiation at run-time, much like OpenID (see section 4.5). An RP can specify its security requirements in a security policy that the CardSpace identity selector can retrieve when required. However, in practice this may mean that an implicit contractual agreement is necessary between an IdP and an RP before the system can be used.

If a remote IdP is being used and the RP is employing HTTPS, then the identity selector retrieves the RP certificate and sends it to the IdP. This enables the IdP to encrypt its security token using the RP's public key, thereby

providing end-to-end protection⁵. This contrasts with SAML profiles (and, analogously, Liberty and Shibboleth profiles) where metadata must be retrieved to allow the exchange of certificates.

4.3.6 User Control and Consent

The CardSpace identity selector allows the user to review an IdP-issued security token before it is released to the RP. A user can modify a personal card at will; however, a managed card cannot be modified by the selector, but can only be changed by the issuing IdP. When a remote IdP issues a security token, it also typically issues a *display token*, so that the identity selector can display it to the user before forwarding the real token to the RP; this is particularly useful if the security token is end-to-end encrypted to the requesting RP. We observe that in such a case the user has no guarantee that the display token actually matches the real token.

4.3.7 Supporting CardSpace

To enable support for CardSpace, an RP web page will typically include either an HTML or an XHTML tag. The HTML (object tag) syntax is used by an ActiveX object and appears to be more commonly adopted by websites. However, ActiveX objects could be disabled, e.g. if high browser security settings are chosen by the user. The XHTML is used by a binary behaviour object, and can thus still work even if ActiveX objects are disabled; however, not all websites support XHTML [34, 168].

4.3.8 Security Policy

An RP can specify its security policy using HTML or XHTML (see section 4.3.7). The defined policy syntax includes the following data items [143].

⁵Note that if a non-auditing IdP (see section 4.3.13) is used, then the RP certificate is not sent to the IdP.

1. **Issuer.** This optional parameter specifies the address of the IdP-STs from which a token is to be obtained. If it is omitted, then any IdP can be used. To specify the SIIP, the following URL is used: *http://schemas.xmlsoap.org/ws/2005/05/identity/issuer/self*.
2. **IssuerPolicy.** This optional parameter specifies the HTTPS-based URL of an endpoint from which the IdP security policy (WS-SecurityPolicy) can be retrieved using WS-MetadataExchange (see section 2.5.6).
3. **RequiredClaims.** This parameter specifies the type(s) of claim about the user that must be supplied to the requesting RP.
4. **OptionalClaims.** This optional parameter specifies the type(s) of claim that the RP would like to know about the user, but which are not mandatory.
5. **TokenType.** This optional parameter specifies the type(s) of token that the RP is willing to accept. If omitted, then the RP is willing to accept any token type.
6. **PrivacyURL.** This optional parameter specifies the URL from which the RP's privacy policy can be obtained.
7. **PrivacyVersion.** This optional parameter specifies the version number of the RP's privacy policy. If a *PrivacyURL* is specified, the value of the *PrivacyVersion* must be greater than 0. If the value changes from one visit to an RP to the next, the identity selector notifies the user, who can review the new privacy.

If CardSpace is used in the context of a web service, then the above policy data is supplied as WS-SecurityPolicy data using WS-MetadataExchange (see section 2.5.6) [143]. The RP security policy can also specify other requirements for the requested security token (e.g. the maximum token age).

4.3.9 Operation

We now describe CardSpace operation when using a personal or a managed card.

4.3.9.1 Personal Cards

When using personal cards, CardSpace adopts the following protocol. We first detail the personal card protocol for the case where the RP does not employ the optional STS, since this is the most important case in this thesis. The parties involved are a CardSpace-enabled RP and a CardSpace-enabled UA, e.g. a web browser capable of invoking the CardSpace identity selector (often referred to below as simply the *selector*), such as those shipped as part of Windows Vista and Windows 7.

1. UA → RP: HTTP/S Request. A user employs a UA to navigate to a CardSpace-enabled RP.
2. RP → UA: HTTP/S Response Including RP Policy. An RP login page is returned containing the CardSpace-enabling tags in which the RP security policy is embedded.
3. User → UA: Invoking the Selector. The UA offers the user the option to use CardSpace, e.g. via a button on the RP web page; selection of this option causes the UA to invoke the selector and pass it the RP policy. Note that if this is the first time that this RP has been contacted, the selector will display the identity of the RP, giving the user the option either to proceed or to abort the protocol.
4. Selector → InfoCards: Highlighting InfoCards. The selector, after evaluating the RP security policy, highlights the InfoCards that match the policy, and greys out the rest. InfoCards previously used with this particular RP are displayed in the upper half of the selector screen.

5. User → Selector: Selecting an InfoCard. The user chooses a personal card. Alternatively, the user could create and choose a new personal card. The user can also preview a card (with its associated claims) to review which claim values would be released if it was selected. Note that the selected InfoCard may contain several claims, but only the claims explicitly requested in the RP security policy will be passed to the requesting RP.
6. Selector ⇌ SIIP: RST-RSTR Exchange. The selector creates and sends a SAML-based Request Security Token (RST) to the SIIP, which responds with a SAML-based Request Security Token Response (RSTR).
7. Selector → UA: Passing RSTR. The RSTR is passed to the UA, which forwards it to the RP.
8. RP → UA: Grant/Deny Access. The RP processes the received token (see section 4.3.10), and, if satisfied, grants access to the user (with the appropriate privileges).

If the RP employs an STS, then the protocol flow is similar to that given above [142]. However, the security policy published by the RP website in step 2 would only specify which RP-STS should be contacted to obtain the RP's WS-SecurityPolicy information. The identity selector then contacts the specified RP-STS to retrieve the RP's WS-SecurityPolicy information using WS-MetadataExchange.

Once a security token is obtained from a SIIP in step 6 (or from a remote IdP — see section 4.3.9.2), the selector sends the security token to the RP-STS to authenticate the user; an RP site-specific authentication token is returned to the selector. This token is then passed to the browser in step 7, which posts it to the RP website using HTTPS/POST. Finally, in step 8 the RP website verifies the received token, completing CardSpace-based user authentication [142].

4.3.9.2 Managed Cards

The managed card operational protocol (see Fig. 4.2) is similar to that for personal cards, except that the remote IdP specified in the selected InfoCard is contacted instead of the SIIP.

The selector uses the standardised WS-* protocols (see section 2.5.6) to retrieve the IdP security policy. Depending on the IdP policy, the user may be requested by the selector to provide credentials for authentication to the selected IdP; these are then passed to the IdP. As stated previously, CardSpace supports four user authentication methods (see section 4.3.2).

If user authentication is successful, the selector retrieves a security token representing the selected digital identity from the STS of the remote IdP. The selector then passes the received token to the UA, optionally after first obtaining permission from the user. This may involve presenting the user with a display token prepared by the remote IdP, listing the claim values asserted in the real security token; the selector will only continue if the user is willing to release such values.

Finally, as in the personal card operational protocol, the UA forwards the token to the RP which processes it (see section 4.3.10) and, if satisfied, grants access to the user [142, 185].

4.3.10 Token Processing

The processing performed by the RP on the received token typically includes the following [34].

1. **Token Decryption.** If the received token is encrypted using the RP's public key⁶, then the RP uses the corresponding private key in order to retrieve the symmetric key used for token encryption, which the RP then uses to recover the original plaintext version of the token.

⁶For performance reasons, the token issuer typically uses a freshly-generated symmetric key to encrypt the token. It then uses the RP's public key to encrypt the symmetric key.

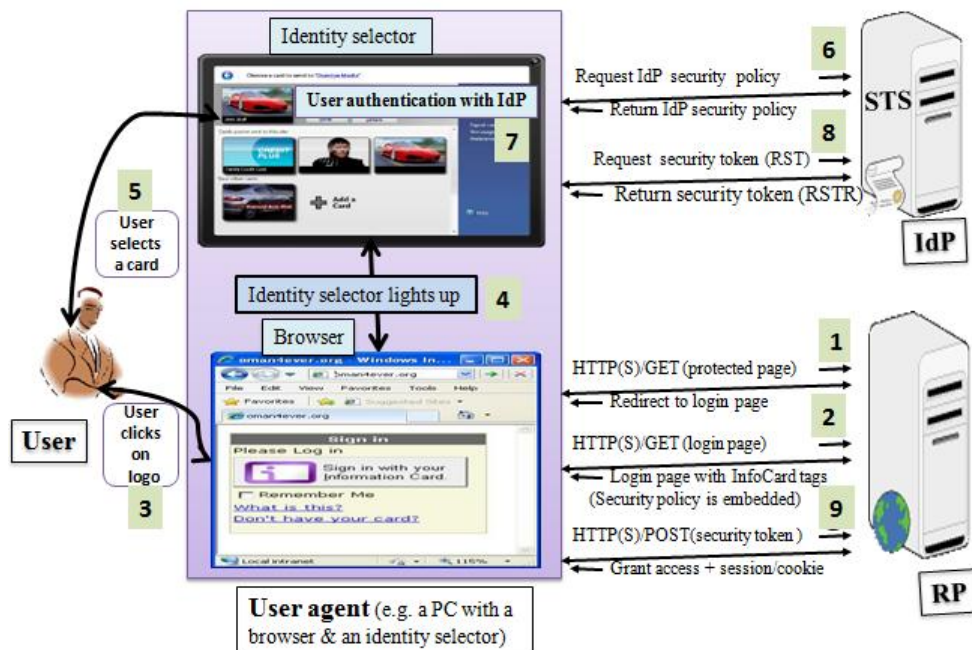


Figure 4.2: CardSpace Operation (Using Managed Cards)

2. **Token Integrity Check.** The RP uses the token issuer's public key to verify the digital signature on the token. This provides assurance that the token has not been tampered with and that its origin is as claimed. The RP can also verify the embedded proof key (see section 4.3.12) to verify the sender's rightful possession of the token.
3. **Token Validation.** This involves verifying time-stamps and the nonce in the token (see sections 2.4.2.5 and 2.4.2.6), and that the token was intended for the specified site. This provides protection against replay attacks.
4. **Retrieval of Claim Values.** The RP retrieves the IdP- or SIIP-asserted claims contained within the received token, and uses them for authentication and/or authorisation purposes. For example, the RP could use the PPID claim (see section 4.3.11 below) to associate the user with a particular account. In addition, the RP could use other claims to make an informed authorisation decision, e.g. selling wine based on a claim

stating that the buyer is old enough to drink (`OldEnoughToDrink = true`).

4.3.11 PPIDs and Digital Signatures

4.3.11.1 Overview

The private personal identifier (PPID) is a unique identifier that links a specific InfoCard to a particular RP [34, 35, 142]. It functions as an RP-specific *pseudonym* for a user [143]. In the case of personal cards, each such PPID has an associated signature key pair, the private key for which is held by the SIIP. CardSpace-enabled RPs can use the PPID, along with a digital signature, created using the private key, to authenticate a user.

Since the PPID is RP-specific, it does not function as a global user identifier, helping to enhance user privacy. In addition, compromising the PPID and key pair for one RP does not allow an adversary to impersonate the user at other RPs. This conforms to the *directed identity* law (see section 3.8), since the PPID cannot be used to correlate user accounts at different RPs.

We next describe how the PPID is generated in both personal and managed cards.

4.3.11.2 Personal Cards

When a user creates a new personal card, the identity selector generates an ID and a secret master key for this card and stores them with the card. The card ID is a globally unique identifier (GUID), and the master key is 32 bytes of random data. When a user employs a personal card at an RP for the first time, the selector generates a site-specific:

- PPID by combining the card ID with data taken from the RP certificate; and
- signature key pair by combining the card master key with data taken from the RP public-key certificate.

In both cases, the domain name or IP address of the RP is used if no RP certificate is available (see Fig. 4.3).

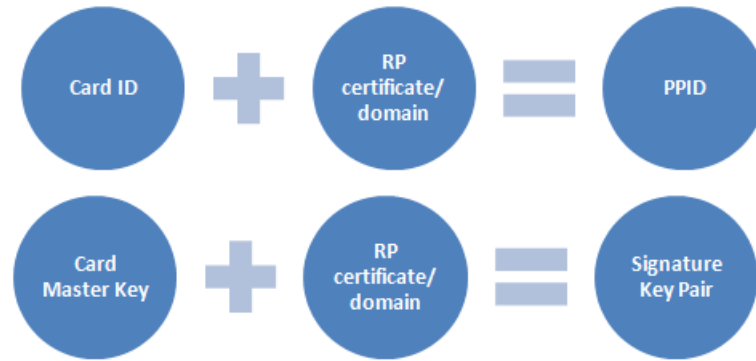


Figure 4.3: PPID and Signature Key Pair for Personal Cards

The selector only displays a shortened version of the PPID, referred to as a *user-friendly identifier* [143], to protect against social engineering attacks and to improve readability.

When a user first registers with an RP, the RP retrieves the PPID and the public key from the received security token and stores them. If a personal InfoCard is re-used at a site, the supplied token will contain the same PPID and public key as used previously, signed using the corresponding private key. The RP compares the received PPID and public key with its stored values, and verifies the digital signature. If all the checks succeed it has assurance that it is the same user.

The PPID could be used on its own as a shared secret to authenticate a user to an RP. However, it is recommended that the associated (public) signature verification key, as held by the RP, should also always be used to verify the signed authentication token to provide a more robust authentication method [34].

4.3.11.3 Managed Cards

A PPID for a managed card is generated rather differently. When creating a managed card, the IdP generates a card reference (i.e. the card ID) and

a card master key and stores them with the values of the claims about the user. When the user retrieves this card from the IdP, the IdP returns a representation of the card to the user containing the card ID, but not the claim values or the card master key. When the user installs the card, the identity selector generates a fresh salt value (a randomly generated n -bit value), and stores it with the card [35].

The PPID of a managed card is computed in one of two possible ways [35], depending on the *scope* of tokens to be provided to the corresponding RP. If the IdP of a managed card issues:

1. *scoped* tokens, i.e. tokens are intended for a specific RP, the IdP computes the PPID as a cryptographic hash of the card master key and the RP's certificate, possibly together with some additional entropy stored with the card; and
2. *unscoped* tokens, the identity selector must first send the IdP a PPID seed specific to the RP, computed as a hash of the card salt and the RP's certificate, and the IdP then uses this seed to compute the PPID using the card master key.

4.3.12 Proof of Ownership

A SAML token can be coupled with cryptographic evidence to demonstrate the sender's rightful possession of the token. A *proof key* is a key associated with a security token, and is used to generate such cryptographic evidence. A data string called the *proof-of-possession* is sent with the token to demonstrate the sender's knowledge of the proof key, thus establishing rightful possession of the token. This proof-of-possession includes either a digital signature or a MAC computed using the proof key [142, 178].

As implied above, a security token can be associated with the following two types of proof key.

1. Symmetric Proof Keys

If the RP requests a symmetric key token, a shared secret proof key is established between the selector and the CardSpace-enabled IdP [142], which is then revealed to the RP. This key is used to prove the subject's rightful possession of the security token. Whilst the use of such a key may offer speed and efficiency advantages [170], it involves revealing the identity of the RP to the IdP, which is not ideal from a privacy perspective.

2. Asymmetric Proof Keys

If the RP requests an asymmetric key token, the selector generates an ephemeral RSA key pair and sends the public key to the CardSpace-enabled IdP. The selector also sends a supporting signature to prove ownership of the corresponding private key [142]. If the IdP approves the public key, it forwards it to the RP in the security token. The private part of the RSA key pair is then used to prove the subject's rightful possession of the security token. Although the use of such a key may not be as efficient as the symmetric approach, it helps to protect user privacy since the identity of the RP does not need to be disclosed to the IdP. Note that, if no explicit key type is specified by the RP, then the selector should request an asymmetric proof key to maximise user privacy and security [143].

The default behaviour of the selector is different in the case of browser-based client interactions with an RP website, in which case *bearer* tokens (i.e. tokens without a proof key [143]) are requested. Because a web browser is only capable of submitting a token to a website passively over HTTP without any proof-of-possession, bearer tokens are used [170].

4.3.13 IdPs and Auditing

CardSpace-enabled IdPs may or may not possess an auditing capability, and their behaviour differs depending on their auditing capabilities [34]. We summarise some of the main properties of these two categories of IdP.

4.3.13.1 Auditing IdPs

We list seven security and privacy relevant properties possessed by auditing IdPs.

1. The RP address and certificate must be passed to the IdP.
2. Because it knows the RP address, the IdP could evaluate the trustworthiness of the RP on behalf of the user, and could, for example, refuse to issue a token for a known phishing site.
3. The IdP encrypts the token using the RP's public key, and the selector passes the token to the RP unmodified. That is, end-to-end encryption is applied.
4. The IdP can issue a token for a specific RP, preventing its re-use with other RPs.
5. The IdP is responsible for generating the display token. However, since the token is encrypted using the RP's public key, the selector cannot verify whether the contents in the display token actually match those of the real token.
6. The audit information retained by an IdP could be useful for law enforcement and for the provision of evidence in a legal case.
7. The IdP can track which RPs a user visits, which is a possible threat to user privacy.

4.3.13.2 Non-auditing IdPs

We list seven security and privacy relevant properties possessed by non-auditing IdPs, corresponding to the list previously given for auditing IdPs.

1. The RP address and certificate do not need to be passed to the IdP.
2. Because it does not know the RP address, the IdP cannot evaluate the trustworthiness of the RP on behalf of the user.
3. The IdP encrypts the token using a cryptographic key associated with the selector, which could then encrypt the token using the RP's public key (if the RP uses HTTPS). That is, token encryption is point-to-point rather than end-to-end.
4. The IdP cannot issue a token for a specific RP; as a result there is a risk of unauthorised reuse of a token with other RPs.
5. The IdP is responsible for generating the display token. We observe that, since the token is not encrypted end-to-end (see above), the selector, if specially programmed⁷, can verify whether the contents in the display token actually match those in the real token.
6. A non-auditing IdP is unlikely to have information that could be useful for law enforcement or for the provision of evidence in a legal case.
7. The IdP cannot track which RPs a user visits, which is potentially privacy-enhancing.

4.3.13.3 Comparison

Table 4.1 compares auditing and non-auditing CardSpace-enabled IdPs in terms of the points listed above.

⁷Note that the current CardSpace identity selector does not perform such a task.

Table 4.1: Auditing IdPs versus Non-auditing IdPs

	Auditing IdPs	Non-auditing IdPs
Must pass RP identity (address and certificate) to the IdP	Yes	No
Capable of verifying the trustworthiness of the RP	Yes	No
Type of encryption	End-to-end	Point-to-point
Type of security tokens	Scoped	Unscoped
The selector is capable of verifying that the contents of the display token match those in the real token	No	Yes, if specially programmed
Legal and law-enforcement cases	More useful	Less useful
Capable of user tracking	Yes	No

We conclude by observing that, while non-auditing IdPs offer potential privacy advantages over auditing IdPs, the converse applies with respect to security.

4.3.14 Possible Limitations of CardSpace

We next describe a number of possible security, privacy and usability shortcomings of CardSpace.

4.3.14.1 Unrestricted Access to the Selector

Anyone with access to a Windows user account can access the InfoCards for that account. This is a particular threat for personal cards, since the PII stored in them will also be available via the selector interface. CardSpace, by default, does not impose any additional protection on use of the selector.

CardSpace does allow individual InfoCards to be PIN-protected. Also, the entire Windows user account could be locked, e.g. using a password. Whilst the use of passwords and PINs for InfoCard protection can help, it

does not completely solve the problem, not least because one of the fundamental design goals of CardSpace is to reduce reliance on password authentication.

4.3.14.2 Lack of Support for Interoperability

CardSpace does not support interoperation with other identity management systems in the way defined in section 1.2.1 (see also section 1.3).

4.3.14.3 Submission of only one InfoCard to an RP

During the user authentication process, a CardSpace user can only select one InfoCard to support authentication to an RP. As pointed out by Chadwick et al. [65, 67], this is potentially limiting, since a real-world RP might request attributes that cannot solely be provided by a single IdP. For example, when a student (user) purchases a book from an online bookshop (RP), the RP might request both credit card data for payment and a student number to prove eligibility for a discount. It seems plausible that, in such a case, no single IdP will be able to assert both these attributes, since they are issued by very different types of entity (banks and educational institutions). CardSpace would thus be unusable, since it only allows use of one IdP. An *attribute-aggregation* technique [65, 67] could address this issue, whilst also helping to preserve user privacy.

4.3.14.4 Limited Support for User Authentication Methods

As stated in section 4.3.2, the CardSpace system currently only supports four authentication methods. This is potentially limiting since a CardSpace-enabled IdP can only perform user authentication using one of these four methods, whereas ideally an IdP should be allowed to perform user authentication using any method it deems appropriate.

This limitation arises because, when using a managed card, the user must provide the authentication credentials to the selector which in turns

delivers them to the managed card IdP. This is the result of a seemingly deliberate design choice made to enable a consistent user experience. The major disadvantage of this choice is that additional authentication methods can only be supported as and when the functionality is added to updates of CardSpace.

4.3.14.5 Limited Support for Roaming

CardSpace allows InfoCards to be exported to different user platforms to support roaming. However, as discussed by Hoang et al. [112], this has usability issues, since a user must import the InfoCards into every machine he or she uses; it also gives rise to security concerns if it is necessary to import InfoCards into an untrusted device. Ideally, it should be possible for a user to access their InfoCards at any time and using any platform. A number of approaches to addressing roaming have been proposed, including storing InfoCards in a trusted device, such as a mobile phone [112], and storing them in the cloud [66].

4.3.14.6 Lack of Support for Password Management

Despite the introduction of CardSpace (and other similar systems), the vast majority of websites still use username-password for authentication, and this is likely to continue for at least the next few years (see section 1.2.2). One major practical problem with CardSpace, and with other similar systems providing more secure means of user authentication, is that the transition from username-password is extremely difficult to achieve. RPs will not wish to do the work necessary to support CardSpace if very few users employ it; equally, users are hardly likely to use CardSpace if it is only supported by a tiny minority of websites.

4.3.14.7 A Platform-specific Selector

The (closed-source) CardSpace identity selector can only currently operate on relatively recent Windows operating systems, namely XP, Vista and 7. This could limit its adoption.

4.4 Higgins

4.4.1 Introduction

According to its website⁸, Higgins, launched in 2003, is ‘an open-source Internet identity framework designed to integrate identity, profile, and social relationship information across multiple sites, applications, and devices’. Like CardSpace, Higgins is InfoCard-based and involves an active client. Higgins supports and extends the functionality provided by the CardSpace system.

The Higgins project’s stated aims⁹ include to:

- provide a card-based consistent user experience for managing identity data;
- give users a greater degree of control over personal data stored on third party servers; and
- offer a single point of control for multiple identities, preferences and relationships.

4.4.2 InfoCards

In addition to personal and managed InfoCards, as supported by Microsoft CardSpace, Higgins introduces a third type of InfoCard, known as a *relationship card* (or *rCard*), which can be either self-issued or managed. An rCard includes a special claim type called a *resource-UDR* (where UDR stands for

⁸<http://www.eclipse.org/higgins/faq.php>

⁹<http://www.eclipse.org/higgins/higgins-charter.php>

Uniform Data Reference), the value of which must be an Internet address. If an RP policy requests an assertion for the resource-UDR claim, then the user must select an rCard. The RP can use the value of the resource-UDR claim to discover an entity holding the values of user attributes, so that the RP can directly retrieve them. This is potentially useful if the IdP-STs has delegated the storage of user attributes to a separate attribute service [21, 220].

4.4.3 Data Model

A major design goal of Higgins is to enable the integration of identity data from multiple, heterogeneous sources using a common *context data model*. Such heterogeneous sources include LDAP directories, SQL data bases, social networking sites, and email directories [21]. The context data model is intended to provide a foundation for integration, unification, and sharing of identity-related data¹⁰.

The Higgins context data model uses the concept of *contexts*, where personal data can be retrieved from a variety of sources, and then grouped together to form a context. Such a context can have a special set of rules associated with it, governing information exchange, reputation measurement, discovery, etc. For example, a context containing private medical data can be established by collecting relevant data from multiple providers, such as hospitals, clinics, pharmacies, practitioners, specialists, etc.; specific privacy provisions can then be applied to the newly created medical context [71].

4.4.4 Architecture

Fig. 4.4, due to Clippinger [71], provides an overview of the Higgins architecture. This figure uses the terms *context provider* and *digital subject*, which roughly map to *identity provider* and *digital identity*, respectively (see sections 2.2 and 3.4). In Higgins terminology, a digital subject only becomes a digital identity if a context provider (i.e. an authoritative TTP) vouches for

¹⁰http://wiki.eclipse.org/Context_Data_Model_1.0

the claim(s) made by the subject. Examples of possible context providers include credit card companies, banks, community associations, and government agencies. Clippinger [71] points out that the concept of *context* is ‘a critical notion to Higgins as all identifying, profile, and relationship information “claims” made about the identities and attributes of people, events, or things is contained in a context’ (see also section 4.4.3).

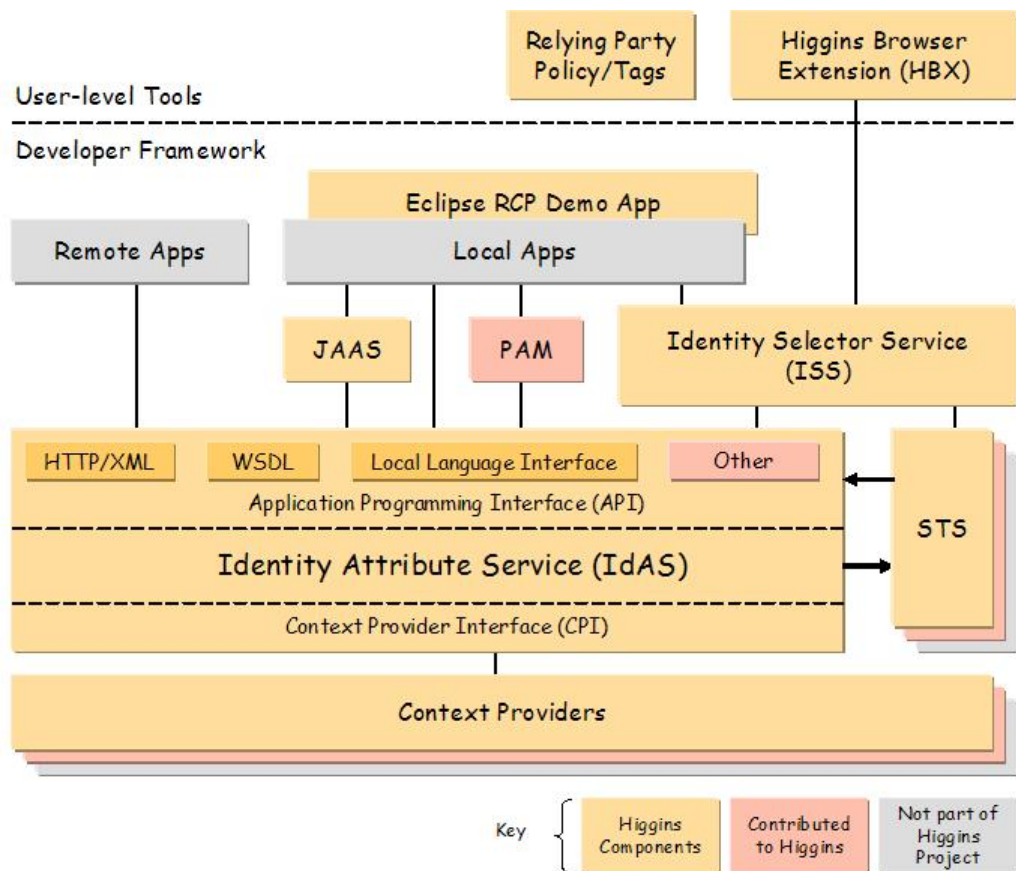


Figure 4.4: Higgins Architecture

The key components of the architecture are as follows.

Context Provider Interface This provides a common interface for context providers to use to access the Higgins framework.

Identity Attribute Service This provides a uniform, abstract data model (see section 4.4.3) that represents a set of contexts, including the digital subjects they contain.

Identity Selector Service This service, which operates in conjunction with a browser extension, performs a number of functions, including: negotiating between IdPs and RPs; displaying IdP and RP identities to the user; and obtaining claim values and releasing them to the browser, which submits them to the requesting RP. Note that the identity selector service is very similar to the CardSpace identity selector.

STS The STS (see section 4.3.2) is responsible for creating a security token (typically digitally signed) containing the identity attributes requested by an RP.

4.4.5 Categories

The components of a Higgins implementation can be divided into three main categories¹¹, namely active clients, personal data services, and identity services.

4.4.5.1 Active Clients

An active client is integrated with a web browser and runs on a PC or mobile device; Higgins calls it a *personal identity manager*. Active clients have been developed for a variety of platforms, including Windows, Mac OS, Linux, Android and iPhone, and browsers, including Internet Explorer, Firefox, Google Chrome and Safari. Two major versions have been released, namely active client 1.* and active client 2.*.

1. Active Client 1.*

This version only supports the OASIS-standardised Information Card protocol [143], and therefore functions very similarly to the CardSpace identity selector. Two versions have been released, namely selector 1.0 and selector 1.1.

¹¹http://wiki.eclipse.org/Exploring_Project

a) **Selector 1.0**

Three implementations of this version of the selector are as follows.

- The *GTK and cocoa selector 1.0*¹² is a stand-alone, cross-platform identity selector application, written in C++, that interacts directly with a user to manage and select InfoCards. It uses the GTK user interface (UI) toolkit on Linux and Windows, and the Cocoa UI toolkit on Mac OS. It offers a very similar functionality to the CardSpace identity selector.
- The *RCP selector 1.0*¹³ is a stand-alone, Eclipse RCP-based application, that is Windows- and Internet Explorer-specific. It is written in Java, and can be installed on an end user desktop machine. Since it implements the complete Higgins infrastructure and the Eclipse form-based identity selector, it gives the end user a secure and friendly way to manage and select InfoCards.
- The *Firefox-embedded selector 1.0* is a Firefox-specific and cross-platform selector that operates on Windows, Linux, and Mac OS. It is implemented as a Firefox add-on that relies on a hosted, persistent web service, which manages the user's InfoCards.

b) **Selector 1.1**

Three implementations of this version are as follows.

- The *AIR Selector 1.1*¹⁴ is an Adobe AIR-based identity selector that runs locally on Mac OS and Windows. It requires two ancillary components on the local machine, namely a Higgins selector switch and a Higgins browser extension. It also re-

¹²http://wiki.eclipse.org/GTK_and_Cocoa_Selector_1.0

¹³http://wiki.eclipse.org/RCP_Selector_1.0

¹⁴http://wiki.eclipse.org/AIR_Selector_1.1

quires a Higgins server that provides an InfoCard service.

- The *iPhone Selector 1.1*¹⁵ is a stand-alone iPhone application that can be launched both from the iPhone main menu and from an RP site. It uses a remote service for retrieving and managing the user's InfoCards.
- The *Android Selector 1.1*¹⁶ is a stand-alone Android application that can be launched automatically from an RP site.

2. Active Client 2.*

This version is currently under development. Earlier plans for this version include incorporating support for password management, Higgins rCards, as well other systems such as OpenID.

Higgins 2.0 is also intended to act as a client for the *personal data service* (PDS — see section 4.4.5.2 below); this could enable Higgins to provide a dashboard for personal information, including providing a means of managing who gets access to what personal data.

It is planned that version 2.* will support managed and personal InfoCards, as well as managed and personal Higgins rCards. It is also planned that version 2.* will be compatible with most modern web browsers, including Firefox version 3 onwards, Internet Explorer version 7 onwards, and Chrome. Supported platforms include Windows XP, Windows Vista, Windows 7, and Mac OS.

4.4.5.2 Personal Data Services

The notion of a *personal data service*¹⁷ is currently under development. Its main purpose is to provide the user with as much control as possible over their personal data, where such personal data is typically scattered across the Internet.

¹⁵http://wiki.eclipse.org/IPhone_Selector_1.1

¹⁶http://wiki.eclipse.org/Android_Selector_1.1

¹⁷http://wiki.eclipse.org/Personal_Data_Store_Overview

A personal data service would provide a user with a central point of control for personal information, including providing data management and data discovery, allowing the user to be discoverable by others.

In addition, it enables the user to share selected subsets of their identity attributes with other people and organisations that the user trusts.

Finally, a personal data service can store personal data locally, control access to remotely hosted personal data, synchronise data with other devices and computers, etc.

4.4.5.3 Identity Services

In order to test its products, Higgins provides code bases to enable experimental websites to act as IdPs and/or RPs.

4.4.6 Possible Limitations of Higgins

Higgins shares some of the limitations of CardSpace, in particular those described in sections 4.3.14.1–4.3.14.4.

4.5 OpenID

4.5.1 Introduction

OpenID¹⁸ [92, 203] is an open and decentralised identity management system that supports web SSO to multiple sites using a single digital identity. The term *OpenID* is also used to refer to a user ID (i.e. a username) used in the OpenID protocol.

In addition to user authentication, certain extensions to OpenID (see section 4.5.9) enable it to support attribute exchange between IdPs and RPs. OpenID is a passive client system, relying on HTTP redirects (see section 3.7).

¹⁸<http://openid.net/>

OpenID has been widely adopted, with more than one billion OpenIDs on the Internet and approximately nine million sites enabling OpenID consumer support¹⁹. OpenID-enabled IdPs²⁰ include Google, Yahoo, Microsoft, and MyOpenID. In the OpenID specifications, an OpenID IdP is referred to as *OpenID provider* (OP). However, for consistency we use the term *identity provider* (or IdP).

Two major versions have been released: OpenID 1.1²¹ [203], and OpenID 2.0 [92]; fortunately v2.0 is backward compatible with v1.1.

OpenID 2.0 (together with some OpenID 1.1 implementations) uses two types of identifier:

1. URLs, e.g. `http://Alice.OpenID-Provider.org`; and
2. XRIs²², e.g. `xri://Alice.OpenID-Provider.com`.

An OpenID user could adopt a self-owned URL, e.g. a personal page, or register a new URL at an IdP.

4.5.2 IdP Discovery

An RP can discover the IdP by first processing the OpenID identifier, provided by the user when use of OpenID is initiated, to identify an online document. The RP then requests this document and learns the IdP identity from it. This latter process can occur in two different ways, as follows.

- **HTML-based Discovery.** This mechanism is supported by the two major versions of OpenID. The RP uses the user-supplied OpenID identifier to request an HTML document containing an HTML link tag; this tag specifies the location of the OpenID-enabled IdP.

¹⁹<http://openid.net/2009/12/16/openid-2009-year-in-review/>

²⁰http://en.wikipedia.org/wiki/List_of_OpenID_providers

²¹In fact, OpenID 1.0 preceded OpenID 1.1; however, the latter is only a minor revision of the former.

²²An XRI, which stands for eXtensible Resource Identifier, is an Internet identifier [205].

- **XRDS-based Discovery.** This mechanism is supported by OpenID 2.0. In this approach, an XRDS document²³ [225] (obtained as described below) contains the necessary information to discover the required IdP. If the user's OpenID identifier is:
 - an *XRI*, then the RP retrieves the XRDS document identified by the user's supplied XRI;
 - a *URL*, then the RP uses the Yadis protocol [172] to retrieve the XRDS document; if this fails, then the RP reverts to HTML-based discovery.

Note that some OpenID implementations allow the user to tell the RP which IdP to use [32].

4.5.3 IdP-RP Negotiation

Unlike Liberty and Shibboleth (see sections 4.7 and 4.8), OpenID does not require pre-contractual agreements or a federation to be in place (see section 4.5.4 below); instead it uses dynamic negotiation at run-time (much like CardSpace). An OpenID RP can specify its (security) requirements in its authentication request. The RP can also dynamically negotiate with the IdP via a background channel to agree on algorithms to be used to exchange a secret key (see section 4.5.8).

4.5.4 Identity Federation

OpenID does not support identity federation as defined in the Liberty, Shibboleth or SAML specifications. However, if an OpenID user already has an RP-managed account, then the RP could locally link (or federate) this account with the user's (remote) IdP-managed identity, e.g. using the user's OpenID, i.e. the IdP-issued global identifier.

²³An XRDS (eXtensible Resource Descriptor Sequence) document is an XML-based file that can be used to help discover the location of the required IdP.

4.5.5 User Control and Consent

The IdP allows the user to review the contents of the token before it is released to the RP. However, the user cannot readily verify that the attribute values in the IdP-issued token are actually as stated by the IdP.

4.5.6 Level of Assurance

An optional OpenID extension [204] enables an RP and IdP to negotiate the level of assurance in the user authentication by the IdP (e.g. that it employs two- or multi-factor user authentication). An RP can either ask the IdP to specify what level it uses, or it can specify the minimum level it will accept.

The extension enables IdPs and RPs to define their own custom assurance levels, as well as supporting the NIST assurance levels (see section 2.4.3.3). An IdP can inform the RP of the NIST assurance level corresponding to the user authentication techniques and policies it employs when authenticating the end user. Note that, although Level 0 is not an assurance level defined by NIST, the OpenID documentation [204] states that it should be used to indicate that the user authentication does not satisfy the requirements of level 1 (the lowest NIST-defined level).

4.5.7 Supporting OpenID

An OpenID-enabled RP would typically display an OpenID login form (see Fig. 4.5) containing a text field allowing the user to enter an OpenID identifier. In order to enable UAs (including browser extensions) to automatically determine that this is an OpenID form, it is recommended that the *name* attribute of the form's field is set to:

1. *openid_url* in OpenID 1.1; or
2. *openid_identifier* in OpenID 2.0.



Figure 4.5: An OpenID Login Form (Taken From `openid.net`)

4.5.8 Operation

We next describe the operation of OpenID, covering the main differences between the two major versions. Before the protocol run, a user will typically have previously registered an OpenID identifier with an IdP. Fig. 4.6 gives an overview of the operation of the scheme.

1. UA \rightarrow RP: HTTP/S Request. A user navigates to an OpenID-enabled RP.
2. RP \rightarrow UA: HTTP/S Response. A login page is returned containing an OpenID login form.
3. User \rightarrow UA: OpenID Entry. The user enters an OpenID identifier into the OpenID form, and submits it.
4. RP: IdP Discovery. If necessary, the RP normalises the user-supplied OpenID identifier²⁴. The RP then performs IdP discovery (see section 4.5.2) to establish which IdP it needs to contact in order to verify the user identifier.
5. RP \rightleftharpoons IdP: Agreeing on MAC Key (Optional). The RP and IdP agree a shared secret MAC key (see section 2.4.2.1), typically established using the Diffie-Hellman Key Exchange protocol (see section 2.4.4.2). This shared key, referred to as an *associate handle* in the OpenID specifications, is used by the IdP and RP to MAC-protect and verify subsequent protocol messages, respectively, in a specified period of time. Note that

²⁴This typically involves transforming the identifier into a canonical URL.

this request-response process, known as the *association* mode, is transparent to the user (it takes place via a direct RP-IdP back-channel), and requires the two parties to be able to store the secret.

6. RP-IdP Communication. The RP and IdP can communicate in one of two modes:

- *checkid_immediate*, which involves direct RP-IdP communications without user interaction; and
- *checkid_setup*, where the user is interactively involved in RP-IdP communications.

Note that the *checkid_setup* mode is more commonly used; indeed, if *checkid_immediate* mode fails, the OpenID protocol typically reverts to *checkid_setup* mode.

If *checkid_immediate* mode is in use, the RP directly sends the IdP an OpenID authentication request, and the IdP replies directly with an OpenID authentication response; step 9 then takes place. However, when using *checkid_setup* mode, the RP redirects the UA to the IdP with an OpenID authentication request²⁵, and step 7 follows.

7. IdP \Rightarrow UA: User Authentication. If necessary, the IdP authenticates the user, using a method of its choice (outside the scope of OpenID). If successful, the IdP constructs an OpenID assertion token, including user attributes, a freshly-generated nonce²⁶, a current time-stamp, and a MAC computed on the token. If a shared key was agreed in step 5, the IdP uses it to generate the MAC; otherwise the IdP employs an internally-generated MAC key. The IdP requests user-permission to send the assertion token to the requesting RP.

²⁵OpenID requests and responses are typically sent embedded in URLs (alternatively they could be sent in HTML forms). OpenID messages are comprised of sets of name-value pairs.

²⁶Although mandatory in OpenID 2.0 (to prevent replay attacks), use of nonces is not mandatory in OpenID 1.1.

8. IdP \rightarrow UA \rightarrow RP: OpenID Token. The IdP redirects the UA back to the RP with a positive or negative OpenID authentication response, depending on whether or not the user granted permission in step 7.
9. RP \rightarrow UA: Grant/Deny Access. The RP verifies the MAC-protected OpenID authentication response, and, if satisfied, grants access. The verification process includes checking that the nonce has not been seen before, the time-stamp is sufficiently current, and the MAC is valid. If a shared secret was agreed previously (see step 5), the RP uses its copy to verify the MAC. RPs capable of storing shared secrets between sessions are called *stateful*.

However, *stateless* RPs cannot maintain secrets between sessions. If a secret was not agreed, the RP must make an extra request to the IdP to verify the MAC, typically via a TLS/SSL channel. This request-response process is known as the *check_authentication* mode. Note that, in order to prevent replay attacks, OpenID-enabled IdPs will only respond to the first verification request it receives for a specific nonce value [92].

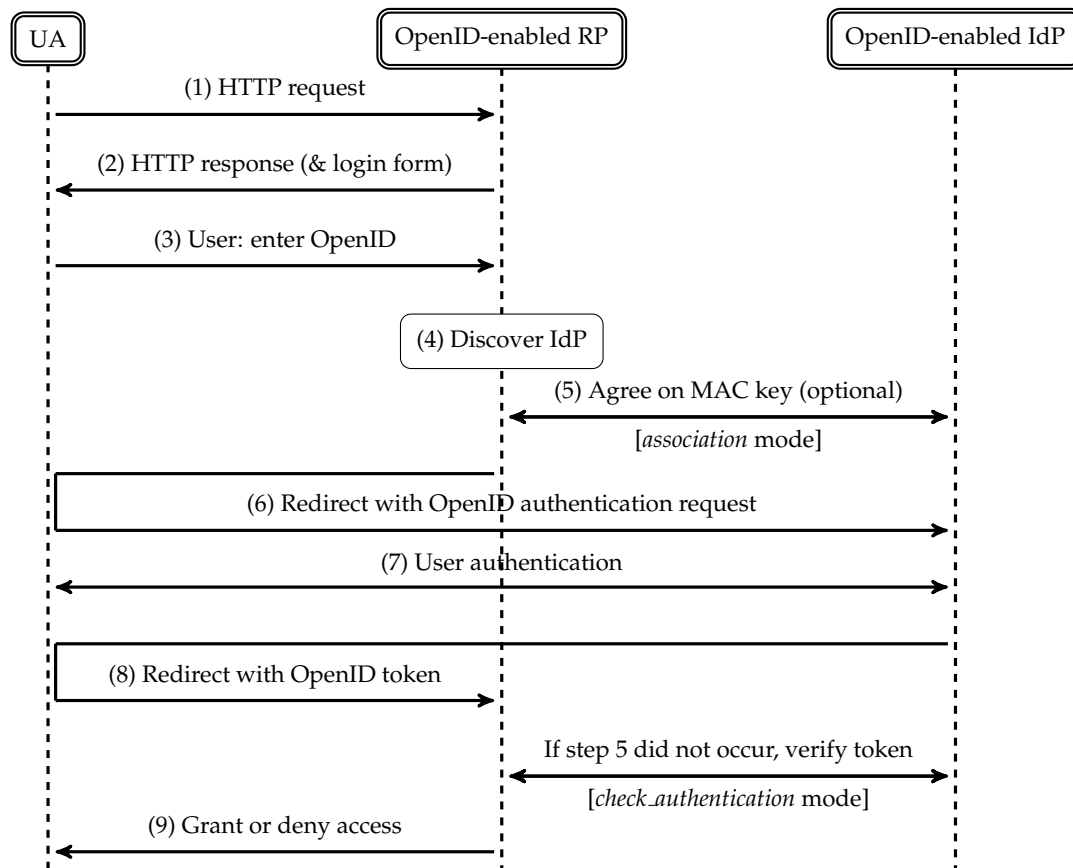
The use of SSL/TLS on the IdP-client and RP-client channels is strongly recommended. For additional security, the RP can add a freshly-generated nonce to its authentication request, which the IdP must include in the authentication response.

4.5.9 Attribute Exchange

OpenID supports attribute exchange via a range of ways; we describe two methods of particular importance to this thesis.

4.5.9.1 Simple Registration OpenID Extension (SREG)

SREG [116] is an extension to OpenID that allows a set of user attributes to be passed from an IdP to an RP. SREG supports the exchange of nine types

Figure 4.6: OpenID Operation in *check_setup* Mode

of attribute, namely *nickname*, *email*, *fullname*, *dob*, *gender*, *postcode*, *country*, *language* and *timezone*. The values of such attributes are held by an IdP, and are only sent to an RP with the approval of the user. As is the case with CardSpace and Shibboleth (see section 4.8), requesting attributes is optional, i.e. OpenID could be used purely for user authentication. Also as in CardSpace, an RP can mark requested attributes as mandatory or optional. Requests are issued as part of an OpenID authentication request in either *checkid_immediate* or *checkid_setup* mode. SREG operates with OpenID v1.1 and v2.0.

4.5.9.2 Attribute eXchange (AX)

AX [108] is an OpenID service extension that supports the exchange of user attributes. Unlike SREG, AX can be used to transfer arbitrary data between IdPs and RPs. AX defines the following operations:

- *fetch*, which enables an RP to retrieve user attributes from an IdP; and
- *store*, which allows an RP to save or update user attributes at an IdP.

AX supports a superset of the user attributes supported by SREG [32].

4.5.10 Proof of Ownership

At the time of writing, the OpenID specifications do not specify a means to prove ownership of an OpenID token [21, 113].

4.5.11 Possible Limitations of OpenID

We conclude by outlining possible limitations of OpenID.

4.5.11.1 Phishing

As discussed by many authors, including Lee et al. [158] and van Delf et al. [223], OpenID is vulnerable to phishing, since IdP discovery is performed by the RP, which redirects the user to an IdP; this means that a malicious RP could redirect a user to a fake IdP, which could then capture the user's credentials.

4.5.11.2 Threats to Privacy

Again as discussed by many authors, including Alrodhan [21], OpenID relies on a global identifier, which gives rise to significant privacy concerns. In OpenID, a user's identifier (i.e. his/her OpenID) is unique and global and must be released to requesting RPs. Malicious RPs could therefore collude

and use a user's OpenID identifier to trace user activities on the Internet, including discovering user preferences, interests and surfing behaviour, e.g. for targeted advertising. IdPs could also masquerade a user at will.

4.5.11.3 Other Limitations

OpenID shares some of the limitations of CardSpace, in particular those described in sections 4.3.14.2 and 4.3.14.6. Also like CardSpace, OpenID does not support attribute-aggregation (i.e. only one IdP can be queried in a single working session — see section 4.3.14.3).

Finally, as stated in section 4.5.10, currently OpenID does not support any proof-of-rightful possession methods, increasing the risk of an adversary illegitimately using a stolen OpenID token.

4.6 OAuth

4.6.1 Introduction

OAuth²⁷ (derived from Open Authorisation) is an emerging, open, identity management standard. The OAuth specifications describe a system designed to enable an end user to grant an Internet application controlled access to personal information (e.g. user attributes, photos, contact lists, etc.) stored at a third party site, without divulging long-term credentials such as passwords. In the language of access control systems, OAuth supports delegation of access rights to stored personal data. In the absence of a system like OAuth, applications must request user credentials in order to access user information held by a third party, which is clearly undesirable.

The four entities involved in the OAuth protocol are the:

1. *resource owner*, typically an end user or, more specifically, a UA (for consistency with the rest of the thesis, we use here the term *user* instead of *resource owner*);

²⁷<http://oauth.net/>

2. *client*, an application requesting access to user resources (i.e. the RP in our terminology);
3. *resource server*, a server hosting user resources; and
4. *authorisation server*, a server that issues an access token (see below) to a client after successfully authenticating the user and obtaining its authorisation (i.e. the IdP in our terminology).

Note that the latter two roles are typically performed by a single entity, which we refer to as an IdP for consistency with the rest of the thesis.

An *access token* is typically an opaque string that indicates permission to access specific information for a limited time period; such a token can be independently revoked. The access token must be kept confidential, and should be issued with the minimum necessary scope and lifetime.

At the time of writing, the OAuth specifications do not specify a method of communicating an authentication assurance level between IdPs and RPs (i.e. clients — see immediately above). OAuth supports the transfer of arbitrary data, enabling arbitrary user attributes to be provided to an RP by an IdP. OAuth does not support identity federation as defined in the Liberty, Shibboleth and SAML specifications.

4.6.2 User Control and Consent

The OAuth-enabled IdP (authorisation server) asks the user to authorise a request made by an identified third party application to access certain user data. The user is made aware of the private data being requested by the third party application, and must consent to its release. Given user authorisation, the OAuth IdP provides the third party application with an access token that it can use to gain controlled access to a defined set of data for a specified period of time. The user can revoke an access token at any time via the IdP.

However, a user cannot independently verify whether or not the data (user attributes) passed by an IdP to an RP actually match those that the user has authorised; the user must therefore rely on the IdP to act honestly. Of course, this is true more generally, since an IdP (authorisation server) could give an application access to any user data it chooses at any time, with or without user permission.

4.6.3 Operation

Two major (incompatible) versions of OAuth have been released: OAuth 1.0 [86] and 2.0 [107, 188]. We describe below the latest version, i.e. OAuth 2.0.

4.6.3.1 Overview of Operation

Fig. 4.7 [107] gives an overview of the operation of the OAuth protocol. It enables a client to request authorisation from the user for access to specific information held by a resource server, possibly via an intermediary authorisation server (which is the recommended option [107]). If necessary, the authorisation server first authenticates the user and, if successful, asks the user to authorise the client. If the user decides to grant the access request, an authorisation token (known as an *authorisation grant*) is sent to the client (four authorisation grant types are defined — see section 4.6.3.2). The client then requests an access token from the authorisation server, where the request includes the authorisation grant. The authorisation server authenticates the client and verifies the authorisation grant, and, if successful, issues an access token. Next, the client requests access to the private resource(s) from the resource server, presenting the access token. Finally, the resource server verifies the access token, and, if it is valid, meets the request.

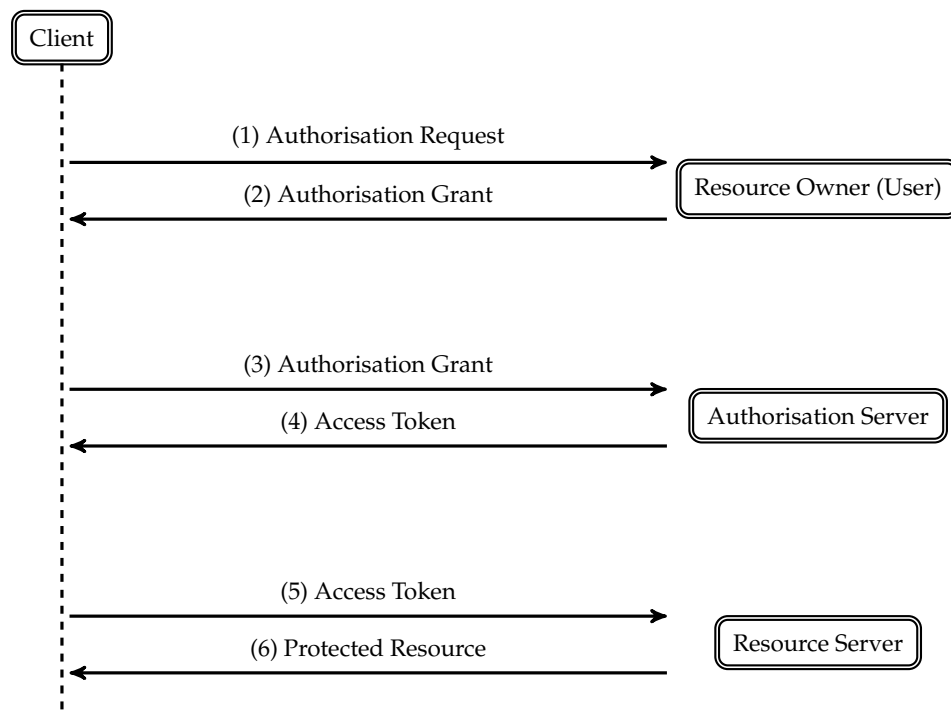


Figure 4.7: Overview of OAuth Operation

4.6.3.2 Authorisation Types

OAuth supports four authorisation types, i.e. means of supplying a client with an access code. These are known as: authorisation code, implicit, user password credentials, and client credentials, corresponding to four possible protocol flows. The two types of most relevance here are discussed below.

1. Authorisation Code Grant Type

An authorisation code is typically a short-lived random string. Such a value is supplied by an authorisation server to a client if a user authorises a request made by the client to access specific user resources.

2. Implicit Grant Type

An authorisation grant is said to be *implicit* if the access token is issued to the client as a direct result of user authorisation. This method requires fewer round trips to provide a client with an access token than the authorisation code type; hence, the implicit grant type improves

the responsiveness and efficiency of certain clients, including browser-hosted client applications.

4.6.3.3 Registration

Before use of the OAuth operational protocol, the client must register with the OAuth authorisation server. How this is achieved is beyond the scope of the OAuth specifications [107], but it could involve the use of an HTML registration form provided by the authorisation server. During registration, the authorisation server must collect certain data about the client, including the client type, its redirection URL, and any other server-required data, e.g. the client name. The authorisation server issues the registered client with a unique identifier and a secret used for client authentication when using the authorisation code grant type (see section 4.6.3.2 above).

4.6.3.4 Operation of Implicit Grant Type

In the remainder of this thesis we restrict our attention to the OAuth 2.0 protocol when using the *implicit* grant type. In this case the OAuth protocol operates as follows.

1. Client \rightarrow UA \rightarrow Authorisation Server: Access Request. The client redirects the user to the authorisation server, requesting access to private data. The redirect includes the client identifier, the scope of the requested access, an optional state parameter, and a redirection URL to which the authorisation server will redirect the user once access has been granted (or denied). The optional (but recommended) state parameter is set equal to an unguessable value [107]. This value is used by the client to match its initial redirection to the response from the authorisation server.
2. Authorisation Server \rightleftharpoons UA: Verification and Authorisation. The authorisation server validates the received access request (see step 1), en-

ensuring that all the required parameters are present and valid. The authorisation server verifies the client identity by comparing the redirection URL with the URL previously registered for this client. If the request is valid and the client identity is successfully verified, the authorisation server (if necessary) authenticates the user (via the UA) over a TLS-protected channel; the authentication method used is outside the scope of OAuth. It then asks the user whether the client's access request is authorised, where this query includes the client identifier and the scope of the requested access. If all the checks succeed, the protocol continues.

3. Authorisation Server \rightarrow UA \rightarrow Client: Response. The authorisation server redirects the UA back to the client using the redirection URL provided earlier (see step 1). The redirection URL includes the access token in a URL fragment. If the state parameter was present in step 1, the authorisation server must return it unmodified (in the URL) to the client to protect against attacks involving manipulated URLs and malicious redirections, including in particular cross-site request forgery attacks [94, 166]. The UA follows the redirection instructions by making a request to the client that excludes the access token-bearing fragment, although the UA retains the fragment information locally.
4. Client \rightleftharpoons UA. The client returns a web page containing an embedded script capable of accessing the full redirection URL, including the fragment retained by the UA (see step 3). The UA executes the embedded script, which extracts the access token from the URL fragment and passes it to the client over a secure channel.
5. Client \rightleftharpoons Resource Server. Finally, the client uses the access token to retrieve the required resource(s) from the resource server via a TLS-protected channel.

4.6.4 Facebook Connect

Facebook Connect²⁸ [171] implements the OAuth 2.0 standard, and uses it to provide an SSO service. Facebook Connect allows users to sign-on to applications (e.g. Facebook-affiliated websites) using their Facebook account, and also enables such applications to access Facebook-hosted user data, subject to user authorisation.

4.6.5 Possible Limitations of OAuth

OAuth shares a number of the limitations of OpenID, including those described in sections 4.5.11.1 and 4.5.11.3.

4.7 Liberty

4.7.1 Introduction

The Liberty Alliance is a large consortium, established in 2001 by approximately 30 organisations; it now has a global membership of more than 150²⁹. The Liberty Alliance Project³⁰ (or simply Liberty) builds open, standards-based specifications for federated identity, provides interoperability testing, and helps to prevent identity theft. Liberty also aims to establish best practices and business guidelines for identity federation.

According to its website, Liberty has been widely adopted, and more than one billion Liberty-enabled identities and devices exist³¹. The Liberty standards have been adopted by leading identity product vendors, including Sun³² and Ping Identity³³. As of mid 2009, the work of the Liberty Alliance project has been contributed to the Kantara Initiative³⁴.

²⁸<http://developers.facebook.com/docs/authentication/>

²⁹http://www.projectliberty.org/liberty/membership/current_members/

³⁰<http://www.projectliberty.org/>

³¹<http://www.projectliberty.org/liberty/adoption/>

³²<http://www.sun.com/software/products/identity/standards/liberty.xml>

³³<http://www.pingidentity.com>

³⁴<http://kantarainitiative.org/>

The Liberty specifications are divided into a number of frameworks, including: the identity federation framework (ID-FF) [226], the identity web services framework (ID-WSF) [219], and the identity service interface specifications (ID-SIS) [149]. In this thesis we focus on the ID-FF. Liberty ID-FF provides approaches for implementing federation and SSO, including supporting mechanisms such as session management and identity/account linkage.

4.7.2 Supported Functionality

The Liberty architecture [226] supports the following activities.

Identity Federation This process enables a link to be established between a user identity at an RP and an IdP, given user consent. At the time of federation, two user pseudonyms (see below) are created for the IdP-RP association, one for use by each party. *De-federation* is the reverse process.

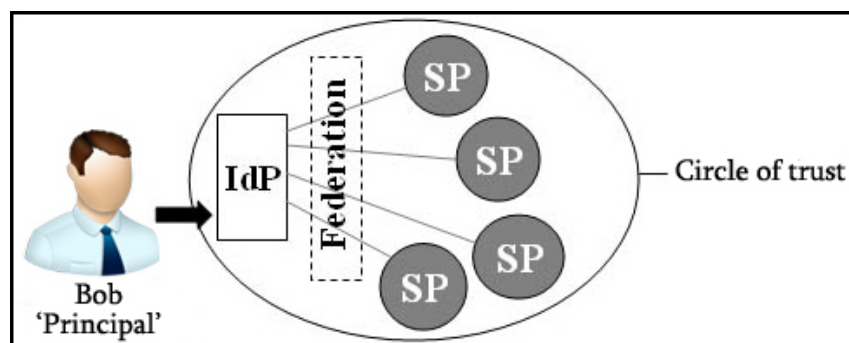


Figure 4.8: A Liberty Circle of Trust

Fig. 4.8 illustrates the Liberty notion of a circle of trust, i.e. a set of entities (users and RPs) with a shared trust in the identity management services provided by one or more IdPs (which also form part of the circle). A principal (or a user) can federate its various identities to a single identity issued by an IdP in a circle of trust, so that the user can access services provided by RPs belonging to the same circle of trust

by authenticating just once to the IdP (see also below). This relies on a pre-established relationship between the IdP and every RP in the circle of trust.

SSO This feature enables a Liberty user to log-in once to an IdP in a Liberty circle of trust and subsequently access protected resources at RPs in this circle without the need to log-in again. *Global log out* is the reverse process.

Pseudonyms Use of pseudonyms must be supported in implementations of Liberty. A pseudonym is an opaque (i.e. non-meaningful), unique handle (identifier) for a user, enabling the user's real identity to remain private, thereby potentially enhancing user privacy (see section 2.3.2.3). Pseudonyms can be temporary or persistent, and are included in SAML tokens exchanged between a Liberty IdP and RP.

Anonymity A Liberty RP can request a Liberty IdP to supply a temporary pseudonym that will help to preserve the anonymity of a user (see section 2.3.2.2). This identifier can be used by an RP to obtain information for or about the user, with the user's consent, without requiring the user to consent to a long-term relationship with the RP [226].

4.7.3 Attribute Exchange

Liberty ID-FF does not support the exchange of user attributes [22]; it only supports user authentication. However, what can be regarded as the *authentication* framework for Liberty, i.e. Liberty ID-WSF, does support attribute-sharing [21]. Liberty ID-WSF builds upon the ID-FF to support identity-based web services in a federated identity environment. For the purpose of this thesis, we are only concerned with Liberty ID-FF and we therefore do not describe the ID-WSF [219] any further.

4.7.4 User Control and Consent

The user is required to express consent to federation; however, the Liberty specifications do not specify a means for a user to review the contents of a Liberty token before it is released to an RP. However, this is unlikely to be a serious issue because it only contains an authentication assertion rather than user attributes.

4.7.5 IdP Discovery

The Liberty specifications do not mandate a means of IdP discovery; instead the specifications state that discovery is implementation-dependent and up to the RP, which may however choose to use the *Liberty identity provider introduction profile* [58].

If there is more than one IdP in a circle of trust, RPs will need a means of discovering which IdP a user is employing. This could involve using a common domain cookie (see below) or a WAYF (Where Are You From — see section 4.8.3) technique. Ideally, the IdP would write a cookie to the user platform that an RP could read. However, because of the cookie same domain policy (see section 2.3.4.1), an IdP in one DNS (Domain Name System) domain has no standardised way of writing a cookie that an RP in another DNS domain can read (see section 2.3.4.1). Liberty's solution to this problem is to set up a common domain for a circle of trust so that it is accessible to all members of the circle of trust. Entries within this DNS domain will point to IP addresses specific to each member of the circle of trust. For example, if the common domain is `CommonDomain.com`, the RP `rp.com` might be allocated the domain `rp.CommonDomain.com`, pointing to an IP address specified by `rp.com`; similarly the IdP `idp.com` might be allocated the domain `idp.CommonDomain.com`.

We next briefly describe the operation of the Liberty identity provider introduction profile [58].

1. The user signs-on to an IdP, which redirects the user browser to the IdP-specific instance of the common domain service.
2. The common domain service writes a cookie to the user platform indicating that the user is using this particular IdP.
3. If and when the user later navigates to an RP site, the RP redirects the user browser to its instance of the common domain service, receives the IdP cookie and reads the IdP list it contains, and redirects the browser back to the RP site with the IdP identifiers embedded in the URL, which are thereby available to the RP.
4. By analysing the URL, the RP learns the identities of one or more of the user-supported IdPs, and can thus engage in further Liberty protocol operations (see section 4.7.8 below). Note that the RP could prompt the user to select an IdP from the retrieved IdP list, e.g. organised in the order of most-recently used, or could simply display the most-recently used IdP.

4.7.6 Negotiation

As noted above, Liberty is designed around the notion of a circle of trust, where a circle of trust would typically involve several RPs and one or more IdPs. Liberty requires pre-existing contractual agreements to be established between the circle of trust members. User identities/accounts are also typically federated, with the consent of the user.

4.7.7 SSO and Federation Profiles

The Liberty ID-FF protocol specification [60] includes a specification of the SSO and federation protocol. The ID-FF bindings and profile specification [58] defines the notion of *profiles*³⁵, i.e. mappings of ID-FF protocol mes-

³⁵A Liberty ID-FF *profile* can also be defined as a combination of message content specifications and message transport mechanisms for a single type of client (i.e. a UA) [21, 58].

sages to particular communication protocols (e.g. HTTP — see section 2.5.3). The latter document also describes the common interactions and processing rules for these profiles.

The SSO and federation protocol has three associated profiles³⁶, which we next briefly describe.

4.7.7.1 Liberty Artifact Profile

This profile is designed for use in the case where the only user platform-supported communications channel available to the IdP and RP is by embedding data in the URL exchanged between them via HTTP redirection. Because of the limits on the amount of data that can be transferred in this way (see section 2.5.3.8), it might not be possible to transfer the SAML assertion itself via this channel; instead, it is transferred in two stages. Firstly, an identifier for the assertion, referred to as an *artifact* (a string with no semantic content), is transferred from the IdP to the RP by embedding it within the URL. Secondly, the RP uses the artifact to retrieve the full SAML assertion from the IdP using direct (back-channel) communication. To protect against replay and guessing attacks, the artifact is an opaque, pseudo-random nonce [226].

4.7.7.2 Liberty Browser-post Profile

In this profile, the entire SAML assertion is embedded in a POST-based HTML form (see section 2.5.1.2). As a result, the artifact and back-channel IdP-RP communications are not required. JavaScript-enabled browsers can perform an HTTP redirect between IdPs and RPs by using JavaScript to automatically send a form containing the authentication data.

³⁶While there are many ID-FF Liberty profiles, we are primarily concerned with the SSO and federation profiles.

4.7.7.3 Liberty-enabled Client and Proxy Profile (LECP)

Unlike the previous two profiles, the Liberty-enabled client (LEC) profile requires an enabling software component to be installed on the user platform. It supports interactions between Liberty-enabled clients (and/or proxies³⁷), RPs, and IdPs. A LEC is a UA that can directly communicate with IdPs; the means by which IdP discovery is accomplished is unspecified. The LEC profile involves sending Liberty messages in the body of HTTP requests and responses using HTTP/S POST, rather than relying upon HTTP redirects and encoding protocol parameters into URLs. As a result the LEC profile does not impose any restrictions on the size of the protocol messages. UA-IdP interactions are SOAP-based (see section 2.5.6), and the protocol messages include either a Liberty-enabled header (preferred) or an entry in the HTTP User-Agent header. The identifier `http://projectliberty.org/profiles/lecp` must be used when referencing this profile.

4.7.8 Operation

For clarity we divide our description of Liberty operation into two cases, as follows.

4.7.8.1 Liberty Artifact and Browser-post Profiles

We first describe the operation of the Liberty artifact and browser-post profiles, covering the main differences between them. As stated previously, both these profiles are federation-based, redirect-reliant, and passive client-dependent systems (see section 3.7). Fig. 4.9 gives an overview of their operation, which involves the following main steps.

1. UA → RP. The user navigates to a Liberty-protected RP page³⁸.

³⁷A Liberty-enabled proxy is an HTTP proxy [160] that emulates a Liberty client [58].

³⁸It is recommended that the request is made over an SSL/TLS secure channel.

2. RP: IdP Discovery. The RP discovers the IdP by some unspecified means.
3. RP \rightarrow UA \rightarrow IdP: Liberty Authentication Request. The RP generates a Liberty authentication request and redirects the UA to the discovered IdP.
4. IdP \rightleftharpoons UA: User Authentication. If necessary, the IdP authenticates the user by some means outside the scope of Liberty.
5. IdP \rightarrow UA \rightarrow RP: Liberty Authentication Response. The IdP generates a digitally-signed Liberty authentication response. If the browser-post profile is used, this response is included in an HTML form that is sent to the RP when the IdP redirects the UA to the RP; in this case the next step is skipped. If the artifact profile is used, the IdP generates an artifact (label) and embeds this in the URL before redirecting the UA to the RP.
6. RP \rightleftharpoons IdP (Artifact Profile Only). The RP uses the artifact received in the previous step to acquire the corresponding, signed authentication assertion from the IdP (using SOAP — see section 2.5.6). This communication takes place via a mutually-authenticated back-channel.
7. RP \rightarrow UA: Grant/Deny Access. The RP verifies the received response, and, if satisfied, grants access.

4.7.8.2 LECP

We next describe the operation of the LECP. As stated previously, this profile is federation-based and involves an active client (see section 3.7). Fig. 4.10 gives an overview of the operation of LECP, where *LibHeader* stands for *Liberty-enabled Header*.

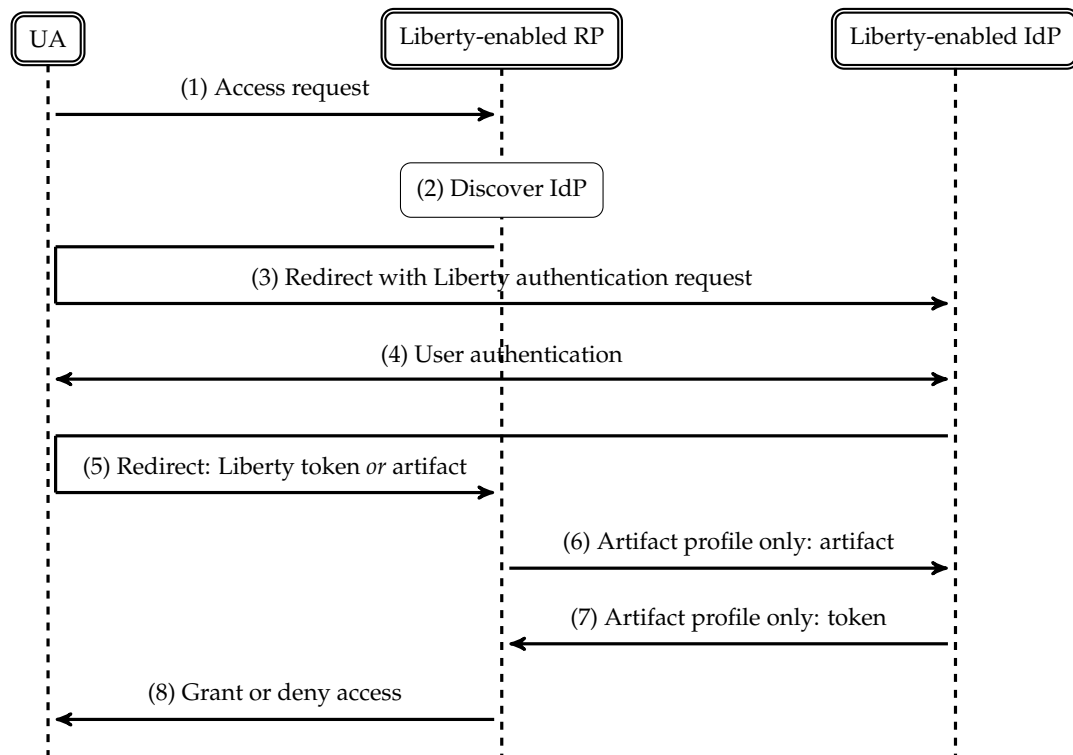


Figure 4.9: Operation of Liberty Browser-post and Artifact Profiles

1. LECP \rightarrow RP: HTTP Request. The user navigates to a Liberty-protected RP page, and either a Liberty-enabled header or a Liberty-enabled entry in the User-Agent header is included in the HTTP request.
2. RP \rightarrow LECP: HTTP Response + Liberty Authentication Request. The RP generates a Liberty authentication request and sends it to the LECP in the body of the HTTP response. Note that the RP can include a list of IdPs it recognises in the request.
3. LECP: IdP Discovery. The LECP discovers the required IdP by unspecified means. The IdP list provided by the RP in the previous step can be used by the LECP; for example, the LECP could compare the list of the user's IdPs against the list provided by the RP and then only display the intersection, so that the user can select an IdP.
4. LECP \rightarrow IdP: Liberty Authentication Request. The LECP issues an

HTTP POST containing a SOAP-based Liberty authentication request message intended for the appropriate IdP. This request must contain the same authentication request as received from the RP.

5. IdP \rightleftharpoons UA: User Authentication. If necessary, the IdP authenticates the user by some means outside the scope of Liberty.
6. IdP \rightarrow LECP: Liberty Authentication Response. The IdP generates a SOAP-based, digitally-signed authentication response and sends it to the LECP via an SSL/TLS channel.
7. LECP \rightarrow RP: Authentication Response. The LECP forwards the IdP-issued response to the RP, embedded in a (POST) HTML form, via an SSL/TLS channel.
8. RP \rightarrow UA: Grant/Deny Access. The RP verifies the received response, and, if satisfied, grants access.

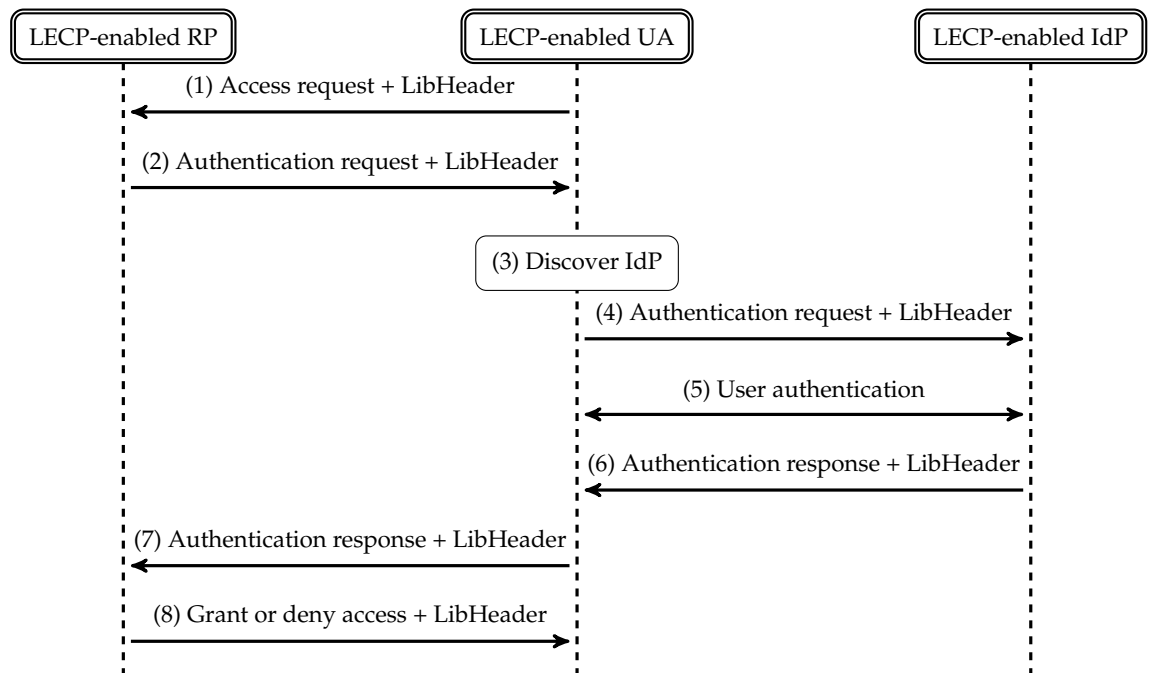


Figure 4.10: Overview of LECP Operation

4.7.9 Proof of Ownership

Liberty enables a user to prove to an RP that it owns the assertion generated by an IdP. The Liberty ID-FF supports SAML 2.0 assertions as a security token type. The SAML 2.0 specifications support three proof-of-possession methods (also referred to as *subject confirmation methods*): Holder-of-Key (HoK), Sender-Vouches, and bearer [59].

4.7.10 Possible Limitations of Liberty

Liberty shares a number of the limitations of OpenID, in particular some of those described in sections 4.5.11.1 and 4.5.11.3. We next briefly describe two possible limitations of Liberty that are of particular importance.

4.7.10.1 Phishing

As discussed by Alrodhan [21], in the Liberty ID-FF browser-post and artifact profiles, IdP discovery is performed by the RP, which redirects the user to an IdP; this means that a malicious RP might redirect a user to a fake IdP, which could then capture the user's credentials. However, such an attack is mitigated if the LEC profile is used, since in this case IdP discovery is performed at the user platform.

4.7.10.2 Threats to Privacy

In a Liberty circle of trust, an IdP is aware of all the RPs which the user tries to access. The IdP can thus track user activities, which could be used to create user profiles [21].

4.8 Shibboleth

4.8.1 Introduction

The Shibboleth³⁹ specifications [61, 62, 173, 211] define a set of possible interactions between an IdP and an RP, designed to support SSO and attribute exchange [62]. It is estimated that over 4 million university students, staff, and faculty are involved in Shibboleth federations⁴⁰. In August 2008, Shibboleth superseded Athens⁴¹ as the JISC⁴²-preferred federation identity management system for use by UK educational institutions⁴³. Shibboleth is a federation-based, redirect-reliant, and passive client-dependent system (see section 3.7).

Two major versions of Shibboleth have been released: Shibboleth 1.3, which builds on the SAML 1.1 specifications, and Shibboleth 2.0, which builds on the SAML 2.0 standards. Fortunately, v2.0 is backward compatible with v1.3. Whereas v1.3 supports SAML 1.1 assertions, v2.0 supports both SAML 1.1 and SAML 2.0 assertions⁴⁴. Shibboleth v2.0 offers a number of the features of SAML 2.0. For example, it incorporates support for passive and forced authentication requests. In addition, a v2.0-compliant RP can request a specific method of authentication from the IdP. Shibboleth v2.0 also supports an additional encryption capacity, and sets a default session lifetime of 30 minutes.

4.8.2 Attribute Exchange

Shibboleth uses the SAML attribute request protocol to support attribute sharing between IdPs and RPs. Use of this attribute exchange is, however, optional, since an RP may only require an authentication assertion. Approx-

³⁹<http://shibboleth.internet2.edu/>

⁴⁰[http://en.wikipedia.org/wiki/Shibboleth_\(Internet2\)](http://en.wikipedia.org/wiki/Shibboleth_(Internet2))

⁴¹<http://www.athens.ac.uk>

⁴²According to the JISC website (<http://www.jisc.ac.uk/>), the acronym *historically* stood for Joint Information Systems Committee.

⁴³<http://www.jisc.ac.uk/whatwedo/programmes/amtransition/iampsp.aspx>

⁴⁴<http://shibboleth.internet2.edu/shib-v2.0.html>

imately 40 attribute types have been defined as common identity attributes, including the six *highly recommended* attributes, namely *givenName*, *sn* (*surname*), *cn* (*common name*), *eduPersonScopedAffiliation*, *eduPersonuserName* and *eduPersonTargetedID* [211].

4.8.3 Architecture and IdP Discovery

The Shibboleth architecture includes the following primary components:

- an RP, which could consist of an assertion consumer service, an attribute requester, etc.;
- a WAYF service, an optional component supporting IdP discovery (alternatively, the role of this component can be taken by the RP); and
- an IdP, which can incorporate a range of functional components, such as an authentication authority, attribute authority, SSO service, etc. The authentication authority (or the authentication service) is an IdP component responsible for user authentication and issuing user assertions for use by RPs. It is also responsible for the generation of temporary user IDs in the form of pseudonyms.

4.8.4 Identity Federation

Shibboleth supports identity federation, as part of which IdPs and RPs exchange public-key certificates. SAML profiles, and by extension Shibboleth profiles, require agreements between system entities regarding identifiers, certificates, keys, etc. A metadata specification can be used to describe such information in a standardised way. During the federation process, a Shibboleth IdP and RP can use short-term random IDs to help preserve user privacy and maintain anonymity [21, 57].

4.8.5 Shibboleth Profiles

Shibboleth supports the following profiles.

4.8.5.1 Browser-post Profile

In this profile the SAML messages sent between the IdP and RP are embedded in (POST-based) HTML forms, which can be sent automatically by JavaScript-enabled browsers. This profile is similar to the Liberty ID-FF and SAML 2.0 SSO browser-post profiles (see section 4.7.7.2).

4.8.5.2 Artifact Profile

This profile involves embedding an artifact (i.e. an opaque reference) in a URL, that is sent from the IdP to the RP as a result of HTTP redirection. It also requires direct (back-channel) RP-IdP communication, by means of which the RP uses the artifact to retrieve the full SAML assertion from the IdP. This profile is similar to the Liberty ID-FF and SAML 2.0 SSO artifact profiles (see section 4.7.7.1).

4.8.6 Operation

We next describe the operation of Shibboleth, covering the main differences between the two profiles introduced above. Its operation is similar to that of the Liberty ID-FF/SAML 2.0 browser-post and artifact SSO profiles. Fig. 4.11 gives an overview of the operation of Shibboleth.

1. UA \rightarrow RP. The user navigates to a Shibboleth-protected page.
2. RP \rightarrow UA: Shibboleth Authentication Request. The RP generates an authentication request, and embeds it in a redirection of the UA to either a WAYF service or to an IdP. A WAYF service is typically used if the RP wishes to delegate the task of IdP discovery.
3. UA \rightleftharpoons WAYF: IdP Discovery (Optional). If a WAYF service is used, it interacts (via unspecified means) with the UA to allow the user to select an IdP. The WAYF service then redirects the UA to the user-selected IdP, forwarding the RP's authentication request. The WAYF

service can offer the user the option to store their choice of IdP for subsequent logins.

4. IdP \Rightarrow User: User Authentication. If necessary, the IdP authenticates the user by some means outside the scope of Shibboleth.
5. IdP \rightarrow UA \rightarrow RP: Shibboleth Token. The IdP generates a digitally-signed SAML assertion. Either the assertion itself (if the browser-post profile is used) or a SAML artifact (if the artifact profile is used) is sent via a redirection of the UA to the RP. Note that the SAML assertion may assert attributes in addition to asserting that the user has been authenticated. Note also that if the browser-post profile is used, the next step is skipped.
6. RP \Rightarrow IdP (Artifact Profile Only). The RP uses the artifact received in the previous step to issue an attribute query to the IdP, which directly responds with a SAML response message. This communication takes place via a mutually-authenticated back-channel.
7. RP \rightarrow UA: Grant/Deny Request. The RP verifies the SAML response, and, if satisfied, grants access.

4.8.7 Proof of Ownership

Much like Liberty, Shibboleth enables the user to prove to an RP that it owns the assertion generated by an IdP. As stated previously, Shibboleth 2.0 builds on SAML 2.0, which supports three proof-of-possession methods (see section 4.7.9). However, implementing these methods is not mandatory [21], i.e. the SAML assertion is not required to contain a proof-of-rightful-possession field if the RP does not mandate it.

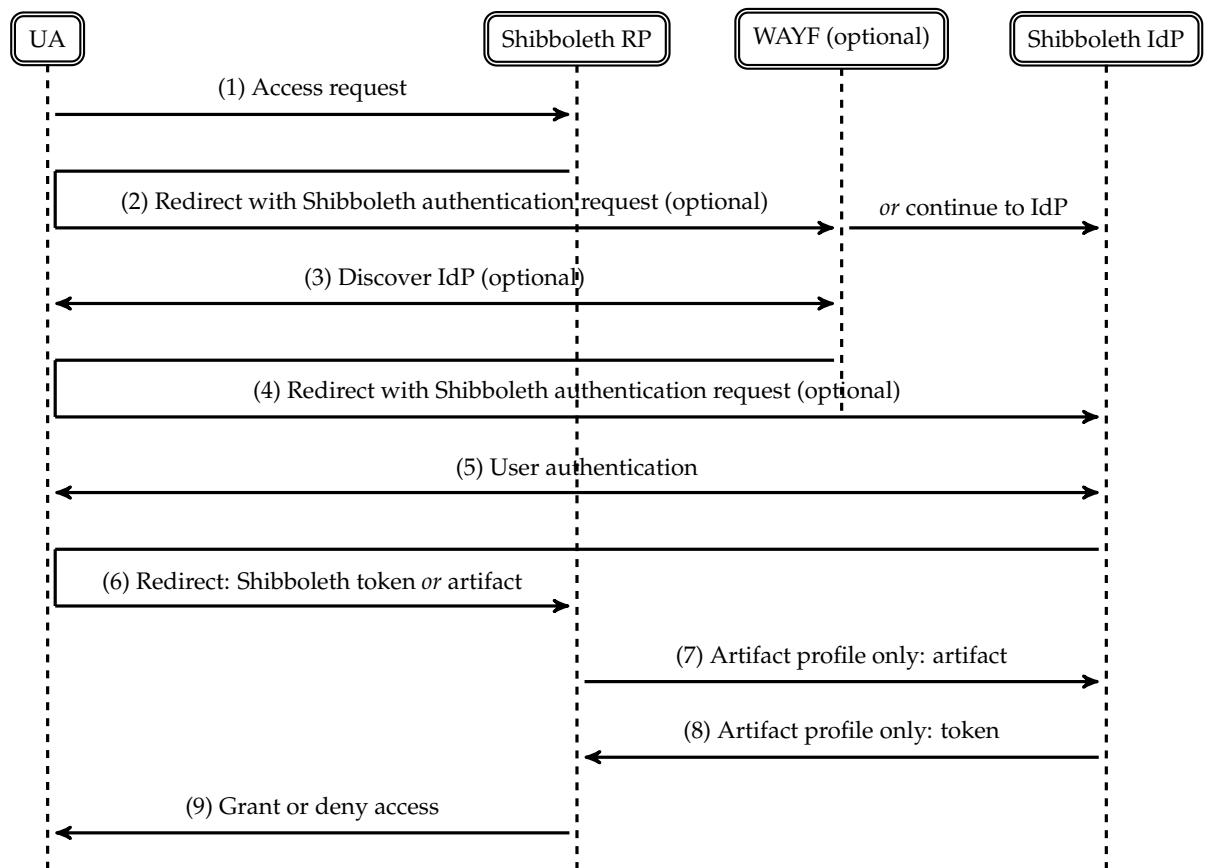


Figure 4.11: Overview of Shibboleth Operation

4.8.8 Possible Limitations of Shibboleth

Like Liberty, Shibboleth shares a number of the limitations of OpenID, in particular those described in sections 4.5.11.1 and 4.5.11.3. We next briefly describe three possible limitations of Shibboleth of particular significance.

4.8.8.1 Phishing

In Shibboleth, IdP discovery is not performed by the user platform; instead it is typically performed by the server. Therefore, a malicious RP or WAYF server could redirect a user to a fake IdP, which could then capture the user’s credentials.

4.8.8.2 Proof-of-possession

The use of proof-of-rightful-possession methods is optional in Shibboleth [62]. As a result, an IdP might not provide a user with the means to prove rightful possession of a security token to an RP, thereby increasing the risk of an adversary illegitimately using a stolen Shibboleth token.

4.8.8.3 Single Sign Off

As discussed by Alrodhan [21], currently the Shibboleth specifications do not support single sign off. Consequently, a Shibboleth user must sign-off from the IdP and from every RP to which the user has signed-on in order to terminate every logged-in session. This clearly adversely affects the usability and security of Shibboleth.

4.9 Anonymous Credential Systems

4.9.1 Overview

Anonymous credential systems [40, 41, 55] enable users to anonymously access RPs; this is achieved through the use of cryptographic constructs known as anonymous credentials [68, 75]. Such a system involves the following roles: an issuer (an IdP), a prover (a specially extended UA acting on behalf of the user), and a verifier (an RP). Issuers, UAs, and RPs interact with each other in two main ways, as follows.

1. In an *issuance protocol*, the IdP and the prover interact to produce a credential, held by the prover.
2. In a *proving (or presentation) protocol*, the prover proves properties of the user encoded within the credential to an RP, without necessarily disclosing the credential. Note that this protocol can typically be executed without real-time connectivity to the IdP. This contrasts with

other identity management systems, such as CardSpace in which use of a managed card requires online access to the issuing IdP [35].

Privacy features provided by anonymous credential systems include the unlinkability property (see section 2.3.2.4), which enables a user to visit an RP multiple times without the RP being able to link these visits [117]. In addition, a user is enabled to selectively disclose to an RP attributes contained in a given credential. An anonymous credential system may also offer additional functionalities, such as revocation of credentials [42, 50, 52, 177], anonymity revocation [51], and verifiable encryption of attributes under a trusted third party's encryption key [49, 54]. Examples of anonymous credential systems include U-Prove and IdeMix (described in sections 4.9.2 and 4.9.3, respectively).

Anonymous credential systems use a cryptographic technique known as zero-knowledge proofs [103], which enables an RP to be convinced that a user possesses a valid credential (issued by a trusted IdP), without disclosing the given credential. The zero-knowledge property ensures that no further information is revealed to the RP. However, note that in practice the use of an anonymous credential system does not necessarily guarantee full anonymity, e.g. since IP addresses can be tracked; such tracking might enable the communicating device to be identified, albeit not necessarily the actual user. As stated in section 2.3.2.2, tracking of IP addresses can be addressed using anonymising services, such as I2P or TOR.

The PRIME project⁴⁵ [53] used IdeMix for issuing and verifying credentials. The work of PRIME is being continued by PrimeLife⁴⁶, a follow-up project. We next describe two widely-discussed examples of anonymous credential systems, namely U-Prove and IdeMix.

⁴⁵<https://www.prime-project.eu/>

⁴⁶<http://www.primelife.eu/>

4.9.2 U-Prove

4.9.2.1 Overview

U-Prove [40, 41] is an anonymous credential system, that was originally developed by Credentica⁴⁷. Following the acquisition of Credentica by Microsoft, support for U-Prove has been incorporated in CardSpace 2.0 [194].

At the heart of the scheme is the notion of a *U-Prove token*, a cryptographically protected container of user attribute information of any type. Such a token is obtained by a prover as a result of executing the issuance protocol with an issuer (IdP), and is subsequently presented by the prover to a verifier (RP) as part of the presentation protocol. Although the issuer helps create the token, the final form of the token is known only to the prover, and thus the issuer cannot link subsequent uses of the token to the instance of the issuance protocol used to create the token. The prover can also use a U-Prove token non-interactively to sign arbitrary data. The roles of issuer, prover and verifier can be combined or split across multiple entities.

The U-Prove token is signed by the IdP using its private key; such a signature provides assurance of the token's origin and integrity. In addition, each U-Prove token includes a prover-generated public key that corresponds to a private key generated by, and known only to, the prover. The token private key can be used by the prover to digitally sign a message as part of the creation of what is known as a *presentation proof*, which is sent to the RP. Such a message may be provided by the RP or generated by the prover; if generated by the prover, then it must include a unique identifier of the target RP, such as its URL, as well as the presentation token creation time.

The prover's signature key pair is used to:

- demonstrate the prover's rightful possession of the U-Prove token, since the RP can establish that the presentation proof is signed by a

⁴⁷<http://www.credentica.com/>

party possessing the private key corresponding to the public key in the IdP-signed U-Prove token; and

- prevent replay attacks (including by legitimate RPs), since the presentation proof can only be generated by a party possessing the private key corresponding to the public key in the token.

A prover can generate arbitrarily many presentation proofs or signatures using the same U-Prove token. A U-Prove token can therefore be issued in long-lived form for use and reuse at any time and with any RP, until expiry or revocation. The U-Prove token, the presentation proof, and the message can be kept in an audit log for later verifications. The verification of a U-Prove token and a corresponding presentation proof only requires an authentic copy of the IdP parameters under which the U-Prove token was issued; such parameters, also known as the *issuer parameters* [43, 191], are generated by the IdP. The distribution and trust management of these parameters are outside the scope of the U-Prove specifications [190]; in practice, distribution of the issuer parameters could be achieved using an authenticated IdP-RP channel. They could also be distributed in a certificate signed by a trusted party [191].

A U-Prove token includes the following three fields.

1. The *token information field* contains IdP-encoded values which are always disclosed when presenting the U-Prove token to an RP. A typical use of this field is to encode token metadata, such as its validity period.
2. The *attribute fields* contain IdP-encoded values for the user attributes as asserted by the IdP; the prover can selectively hide or reveal the value of each field when using the U-Prove token.
3. The *prover information field* contains a prover-encoded value; such a value is invisible to the IdP, but is always revealed when presenting the

U-Prove token. A typical use of this field is for the prover-generated public key.

U-Prove tokens can also be used to establish a relationship with a particular RP. A universally unique token identifier can be computed from each U-Prove token. The IdP does not learn any information about this identifier during the issuance protocol; as a result, such an identifier cannot be used to correlate a presented U-Prove token to its issuance. In addition to identifying repeat visits, token identifiers can also be used for token revocation.

A U-Prove token can be optionally bound to a trusted device such as a smart card, mobile phone, or online server. If such a process, known as *device-binding* [45], is performed, the prover cannot use the token without the assistance of the associated device. The device can be used to protect multiple tokens issued by any number of IdPs, and can dynamically (i.e. at the time of use of the token) enforce policies on behalf of the IdPs, RPs, or other third parties. Such added protection can be achieved without the device needing to interact with the IdP and without compromising user privacy [192].

4.9.2.2 Distinctive Features of U-Prove Tokens

U-Prove tokens, which can be encoded in XML, are similar to SAML assertions or X.509 certificates; however, there are two major differences [193], as follows.

1. A U-Prove token is jointly generated by a prover and an IdP as a result of engaging in the interactive issuance protocol. Such a token contains no information identifiable by an IdP, apart from the certified claim values. The prover public key and the IdP signature for a token are randomised by the prover during the issuance protocol, and are hence not known by the IdP. The prover, and hence the user, is therefore

protected against tracking when using the U-Prove token, even if the issuer and the verifier collude.

2. When using a U-Prove token, the user can selectively conceal any of the encoded claims, without invalidating the IdP's signature. In particular, the user can hide all the claims, and hence only prove ownership of a certain U-Prove token, or reveal all of them. The latter case would be similar to presenting a signed SAML assertion or an X.509 certificate.

4.9.2.3 U-Prove-CardSpace Integration

We now describe the integration of U-Prove into CardSpace. Microsoft has specified [190] how U-Prove is integrated into the OASIS-standardised version of CardSpace (see section 4.3.1). This specification describes how U-Prove tokens can be generated and used within the CardSpace framework.

The role of a prover can be played by the CardSpace identity selector, in which case the managed InfoCard will be what is known as a *U-Prove InfoCard*. In such a card, the U-Prove token type, identified by the URL: `http://schemas.xmlsoap.org/ws/2010/03/u-prove/token`, must be included in the InfoCard's list of supported token types (see section 4.3.2).

A U-Prove InfoCard can be provisioned like any other managed card. When the card is used, the selector uses one or more U-Prove tokens obtained from the relevant IdP to create a presentation token, and sends it to an RP.

Since it is essentially a managed InfoCard, a U-Prove InfoCard can support any number of claims of any type. The PPID claim is treated differently; it is defined as the base-64 encoding of the token identifier of the U-Prove token used to generate the presentation proof [43]. Note that, to enhance user privacy, the PPID value is never disclosed to the IdP.

Given that claim values might change over time and that PPID values must remain persistent, two classes of U-Prove token are defined, namely:

- a *claim token*, which encodes the values of the claims (excluding the PPID claim) supported by a given InfoCard; and
- a *PPID token*, which only supports the PPID claim, and is typically reused in repeat visits to an RP.

U-Prove Issuance Protocol

The issuance protocol [43] involves four message exchanges between a selector and an IdP. The protocol is performed over WS-Trust [175] (see section 2.5.6). Multiple instances of the issuance protocol can be performed in one protocol run; each instance can be used to obtain a batch of U-Prove tokens of a particular type. Indeed, a prover typically obtains multiple U-Prove tokens signing the same set of attributes in one instance of the issuance protocol; multiple tokens are obtained to preserve unlinkability.

A summary of the protocol is given below.

1. Selector \rightarrow IdP: RST. The selector initiates the issuance process by sending the IdP a token request; such a request can include a list of requested claims, an InfoCard reference, and a request for a display token.
2. IdP \rightarrow Selector: RSTR + First Issuing Messages. The IdP authenticates the user (if necessary), validates the request, initiates the necessary instances of the issuance protocol, and returns the first token response containing the number and contents of the U-Prove tokens of both classes (i.e. claim or PPID), and the corresponding first message of each of the instances of the issuance protocol.
3. Selector \rightarrow IdP: RSTR + Second Issuing Messages. The selector validates the response, initiates the necessary instances of the issuance

protocol, and returns the second token response containing the second message of each instance of the issuance protocol.

4. IdP → Selector: Request Security Token Response Collection (RSTRC) + Third Issuing Messages. The IdP replies with the third and final token response containing the third message of each instance of the issuance protocol.

Finally, for each instance of the issuance protocol, the selector generates the U-Prove tokens using the exchanged cryptographic material.

While U-Prove uses cryptography to guarantee the authenticity and integrity of the exchanged attribute values, how the confidentiality of such values is protected is outside the scope of U-Prove [41]. The U-Prove protocols should be performed over a confidentiality-protected channel to protect the confidentiality of the claim values. Given that the exchange involves multiple messages, the IdP must maintain the cryptographic state of each issuance instance between message exchanges. This state must also be protected.

U-Prove Presentation

When using a U-Prove InfoCard, the selector generates a U-Prove presentation token and sends it to the RP. A U-Prove presentation token is a WS-Trust security token generated by the selector, which contains presentation proof(s) generated using corresponding U-Prove token(s), which can be:

1. a *claim* U-Prove token and the corresponding claim values, if at least one non-PPID claim is requested by the RP; and
2. a *PPID* U-Prove token if the PPID claim is requested.

Multiple long-lived tokens asserting the U-Prove InfoCard claim values may be obtained before card presentation, e.g. when the InfoCard is provisioned. IdPs can decide whether to issue long-lived tokens, on-demand

tokens, or both. Note that a U-Prove InfoCard can be presented to an RP without requiring real-time connectivity to the issuing IdP.

We next outline the steps which must be followed to generate a U-Prove presentation token [190]. The U-Prove presentation token is encoded in XML.

1. The selector chooses one or more U-Prove tokens associated with the U-Prove InfoCard.

If the PPID claim is requested, then a pair of claim and PPID U-Prove tokens already associated with the requesting RP is selected. If no such pair is available, then a fresh pair of tokens is selected and associated with this RP.

However, if the PPID claim is not requested (or not supported by the InfoCard), then a fresh claim U-Prove token is selected from amongst those not previously associated with an RP.

In both cases, if a U-Prove token of a specific class is not available, the selector must either obtain a fresh batch from the IdP using the issuance protocol or fail.

2. The selector must specify the presentation token creation time and must define the scope of the receiving RP.
3. The selector must generate a presentation proof for each selected U-Prove token, where each proof is encoded in a separate presentation XML element.
4. The selector can display to the user the disclosed claim values using a display token provided by the IdP. However, given that the PPID value is unknown to the IdP, the identity selector is responsible for providing a displayable PPID value. The selector can calculate the PPID value by computing the base-64 encoding of the token identifier of the PPID U-Prove token.

Finally, the selector sends the generated presentation token to the requesting RP, which must perform a range of verifications on the received token, as given in the U-prove specifications [190]. These include checking that the token creation time value is sufficiently close to the time the token was received, and that the scope value is within the RP's scope.

In addition, the RP must verify the token IdP and prover signatures (see section 4.9.2.1 above). The RP must also verify the token's freshness; it must ensure that all time-stamps are acceptable, and that the token has not been seen before. If all checks succeed, then the RP can extract and use the claim values.

4.9.3 IdeMix

IdeMix [55], an acronym for Identity Mixer, is an anonymous credential system. IdeMix's core protocols have been publicly specified [51].

4.9.3.1 System Entities

Analogously to U-Prove, the main entities involved in the IdeMix system are as follows.

1. An *issuer* (an IdP) issues credentials. The IdP must possess a public-private key pair.
2. A *verifier* (an RP) verifies credentials. Like the IdP, the RP must possess a public-private key pair.
3. A *user* (or a UA on behalf of the user) obtains a credential from an IdP and shows it to an RP. The user must possess a master secret, which is linked to all the pseudonyms and credentials issued to this user.

The user must register a pseudonym with an IdP; the IdP only knows the user by this pseudonym, and all credentials issued by the IdP are bound to this pseudonym. Pseudonym registration, credential issuance and credential verification involve interactive protocols between a user and an IdP,

or a user and an RP. A further system entity, known as a *de-anonymising organisation*, can, if certain conditions hold, revoke user anonymity; such an entity must possess an encryption-decryption key pair.

4.9.3.2 Operation

We now give an overview of the operation of IdeMix. We assume that the user has already contacted an IdP and registered a pseudonym with it.

1. A user, possessing a particular pseudonym (N , say), requests a credential containing certain attributes from an IdP.
2. The IdP determines whether the user possessing the pseudonym N is eligible to receive a credential with the requested attributes. If so, the IdP uses its private key to generate a signed credential containing the required attributes, and sends the credential to the UA. Note that the issued credential is bound to the pseudonym N .
3. The UA now uses a zero-knowledge proof to prove to an RP that it:
 - a) possesses a valid signature, generated by a certain IdP, on a statement containing specific attributes and the pseudonym N ; and
 - b) owns the master secret key related to the pseudonym N .

Note that no other information is revealed by the UA to the RP; in particular, the actual credential is not sent to the RP. As a result, multiple uses of the same IdeMix credential cannot be linked together.

The user can use a credential with any RP any number of times, without the credential uses becoming linkable to each other or to the pseudonym to which the credential was originally issued (however, one-show credentials are an exception⁴⁸). Like U-Prove, the unlinkability property is maintained, even if the IdP and RP collude. The IdeMix employs a special signature

⁴⁸One-show credentials are specially designed to be used only once; if used more than once, then user anonymity can be revoked.

scheme [51] which in addition to realising the system functionality, also allows for efficient implementation.

All user credentials are linked to the user's master secret; this implies that if a user decides to share a credential, the user's master secret must also be shared. This can help to discourage users from sharing credentials.

To enforce accountability, e.g. to identify users abusing the service, an optional anonymity revocation service can be supported. To enable this to happen, a UA can encrypt a credential using the public key of the de-anonymising organisation. This encryption is verifiable, i.e. so that an RP has guarantees that, if necessary, the de-anonymising organisation can decrypt the credential and reveal the user identity. Note that anonymity revocation requires user co-operation. The system allows the use of multiple de-anonymisers, so that a user can choose one that they trust. A de-anonymisation condition can also be included, specifying conditions under which user anonymity will be revoked by the de-anonymiser.

4.10 Comparison

We conclude by providing a general comparison between the identity management systems described in this chapter. The comparison is given in Table 4.2, where \checkmark refers to *supported* and \times to *not supported*.

Table 4.2: General Comparison Between Identity Management Systems

	Type	IdP discovery	Attribute exchange	Ownership proof	SAML tokens
CardSpace	client-based	selector-based	✓	✓	✓
Higgins	client-based	selector-based	✓	✓	✓
OpenID	redirect-based	server-based	✓	✗	✗
OAuth	redirect-based	server-based	✓	✗	✗
Liberty	redirect-based	server-based	✗	✓	✓
Liberty (LEC)	client-based	client-based	✗	✓	✓
Shibboleth	redirect-based	server-based (WAYF)	✓	✓	✓
Passport	redirect-based	server-based	✓	✗	✗
U-Prove	client-based	selector-based	✓	✓	✓
IdeMix	client-based	scenario-dependent	✓	✓	scenario-dependent

Part II

Interoperability

Overview

Part II of the thesis, which contains a total of five chapters, describes a novel approach to supporting interoperability between a wide range of identity management systems. First, in *chapter 5*, we describe a general model for interoperability between an Information Card-based identity management system and almost any other existing identity management system. Using this model, Information Card users are able to obtain a security token from an identity provider not supporting Information Cards. After processing at the client, an enhanced token is produced that can be processed by an Information Card-enabled relying party.

Four specific instantiations of this model are described, that enable interoperability between an Information Card system and:

- Liberty (*chapter 6*);
- Shibboleth (*chapter 7*);
- OpenID (*chapter 8*); and
- OAuth (*chapter 9*).

A General Interoperation Model

5.1 Introduction

5.1.1 Overview

As discussed in chapter 3, in order to simplify the management of identities and help to mitigate identity-oriented attacks, a number of identity management systems (e.g. CardSpace, Liberty, etc.) have been proposed. An IdP in such a system supplies a UA with a security token that can be consumed by an RP. However, in practice a fundamental problem arises because of the multiplicity of such systems. That is, while one RP might solely support an Information Card identity management system, another might only support a different type of system. Therefore, to make these systems available to the largest possible group of users, effective interoperability between identity management systems is needed.

In this chapter we propose a general, client-based approach to supporting interoperation between an Information Card-based identity management system (such as CardSpace or Higgins) and almost any other type of system (including systems such as OpenID, OAuth, Liberty and Shibboleth). Following this approach, Information Card users are able to obtain a security token from an arbitrary IdP, which, after processing at the client, can be processed by an Information Card-enabled RP. Interoperation is supported at the client in a way that is as transparent as possible to IdPs and RPs, and requires minimal changes to the existing user platform infrastructure. We specify the operation of our general model and also pro-

vide security and operational analyses. Subsequent chapters describe specific schemes that conform to this model; such schemes enable interoperation between an Information Card-enabled RP and one of a Liberty-enabled IdP (chapter 6), a Shibboleth-enabled IdP (chapter 7), an OpenID-enabled IdP (chapter 8), or an OAuth-enabled IdP (chapter 9).

The approach works with a variety of Information Card-based systems, including CardSpace and Higgins. For simplicity of presentation, in this chapter we describe its operation with CardSpace, a widely-discussed example of an Information Card-based system.

5.1.2 Motivation

We have chosen to consider interoperation between Information Card systems such as CardSpace and other systems (including Liberty, Shibboleth, OpenID, and OAuth), because of the wide adoption of these systems. For example, Liberty has gained the acceptance of many technology-leading organisations; indeed, more than one billion Liberty-enabled identities and devices exist (see section 4.7). Shibboleth has enjoyed widespread adoption, particularly in the educational sector (see section 4.8). In addition, OpenID has been widely adopted, with (as of 2009) more than one billion OpenIDs on the Internet and approximately nine million sites enabling OpenID consumer support (see section 4.5). Finally, OAuth 2.0 has had significant practical exposure, including deployment by Facebook (Facebook Connect) and Twitter.

Complementing this, the wide use of Windows, recent versions of which incorporate CardSpace, means that enabling interoperation of this type is likely to be of significance for large numbers of identity management users and RPs.

Another reason for supporting interoperation of this type is because of the similarity between the message flows in such systems. These systems also typically provide similar services, including user authentication and

exchange of user attributes.

As discussed in section 1.2.4, each identity management system offers somewhat a different user experience, and this is likely to lead to user confusion and hence potentially give rise to security breaches. Our approach could help to mitigate such problems by providing a consistent user experience using the concept of the card metaphor and identity selectors.

In addition, as stated in sections 1.2.3 and 3.5.2, many identity management systems are susceptible to fake IdP attacks, in which a malicious RP redirects a user browser to a false IdP. The user then reveals to the fake IdP secrets that are shared with a genuine IdP. This arises because, in the absence of a system-aware client agent, many identity management systems rely on browser redirects. Our approach also helps to mitigate attacks of this type.

5.1.3 Organisation

The remainder of the chapter is organised as follows. Section 5.2 describes the approach to supporting interoperation, and, in section 5.3, we give an analysis of its operation. Section 5.4 discusses possible advantages of the approach, and section 5.5 considers its security properties. Section 5.6 outlines a number of potential issues, and section 5.7 highlights possible extensions of the model. Section 5.8 reviews related work, and, finally, section 5.9 concludes the chapter.

5.2 The Interoperation Model

We now present our approach to interoperation. The model we propose makes use of personal cards to store information regarding identity relationships with specific remote (non-CardSpace) IdPs. Operation of CardSpace in the *normal* way using managed cards is not affected by the operation of schemes conforming to the general model.

5.2.1 System Entities

The entities involved in the model are:

- a CardSpace-enabled RP;
- a CardSpace-enabled UA, e.g. a suitable web browser such as Internet Explorer;
- a remote IdP that does not support CardSpace; and
- a software component installed on the user platform (referred to as the *adaptor*) providing the functionality described below. The adaptor could be implemented as a browser extension.

5.2.2 Overview of Operation

We now give a high-level overview of the interoperation model (a detailed, step-by-step description is given in section 5.2.4).

Following a user visit to a CardSpace RP, the adaptor pre-processes the RP-provided page, including determining whether the RP security policy specifies use of managed cards. The identity selector is activated, and the user selects a suitable (personal) InfoCard. The SIIP then generates an RSTR, which the selector attempts to send to the RP; this attempt is intercepted by the adaptor and the RSTR is temporarily stored at the user platform. The adaptor then redirects the UA to the remote IdP corresponding to the selected personal card, which, if it deems it necessary, authenticates the user. If authentication is successful, the IdP returns a security token to the UA. The adaptor now creates a new SAML token, referred to as the *encapsulating token*, incorporating some combination of the IdP-issued token, the SIIP-issued RSTR, and information extracted from these two tokens. The adaptor formats this encapsulating token to be as similar as possible to a standard CardSpace-complaint token. Finally, it forwards this encapsulating token to the RP.

A summary of the operation of the model is shown in Fig. 5.1. In this figure, steps 3, 6, 7, and 10 are performed by the adaptor, and this is indicated in the figure by the adaptor symbol.

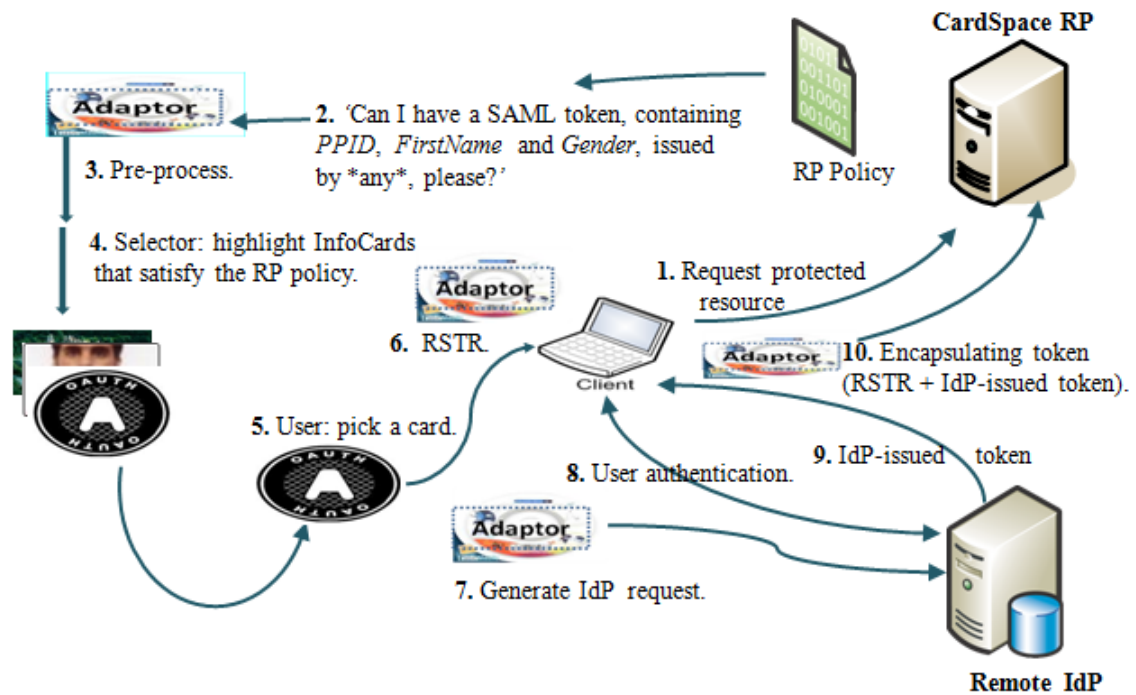


Figure 5.1: Interoperation Model Operation

5.2.3 Requirements

The following conditions must be met for use of the model.

1. The user must have an existing relationship with both an RP and an IdP (and so the IdP will have a means of authenticating the user). The RP must trust the IdP for the purposes of user authentication and for the provision of user attributes. The user must also trust the IdP.
2. The user must install the adaptor, which must be capable of performing a variety of tasks. In particular, it must be able to:
 - execute automatically;
 - scan RP-provided web pages;

5. A GENERAL INTEROPERATION MODEL

- modify web pages, if certain conditions hold;
 - intercept, inspect and modify messages exchanged via a browser between an identity selector and an RP, and between an IdP and an RP;
 - automatically forward security tokens to IdPs and/or to RPs;
 - engage in an interaction with an IdP, if necessary; and
 - provide a means for the user to enter data, if required.
3. The implementer of the model must decide which information needs to be stored in an IdP-specific personal card (as used to support the interoperation functionality). This information will necessarily be identity management system-specific. Given that CardSpace only permits a fixed set of fields to be stored in a personal card (see section 4.3.3.1), specific predefined fields must be chosen to be (mis)used for the storage of this information.
 4. The user, either prior to, or during, operation, must create a special personal card, referred to as an *InterCard* (Interoperation Card), which will represent the IdP. This personal card must contain information about the IdP, such as its URL, as well as a predefined sequence of characters (e.g. the word *interoperation*) used to trigger the integration software (see section 5.3.1).
 5. As well as being able to verify the CardSpace standard-compliant digital signature, the RP must also be able to verify the IdP signature embedded in the SAML token provided to the RP by the adaptor as part of the interoperation functionality.
 6. Where applicable, the RP must be willing to accept an encapsulating CardSpace-like SAML token (which may be unsigned) generated by the adaptor; this token will include both IdP-supplied user attributes

and a digitally-signed, SIIP-issued RSTR containing the RP-specific PPID for the InterCard.

7. Where applicable, the IdP must be prepared to provide SAML assertions for RPs for which a federation agreement does not exist for the user concerned¹.
8. The identity management system in use must not require direct, back-channel communication between the IdP and RP (see section 3.5.1). That is, all IdP-RP communications must pass via the user platform. This is because back-channel communication is inconsistent with Information Card-based systems which require all communications to pass via the identity selector on the user platform. Also, the adaptor is clearly not capable of intercepting back-channel communications.

5.2.4 Operation

We now specify the operation of our general model for interoperation. We specify its operation as a series of steps. Steps 1, 2, 4–7 and 13 of the model are the same as steps 1, 2, 3–6 and 8, respectively, of the CardSpace personal card protocol given in section 4.3.9.1, and hence are not described again here. Note that certain details of the steps given below will vary depending on the specific identity management system in use.

3. Adaptor → UA. The adaptor performs the following steps.
 - a) It scans the login page to detect whether the RP website supports CardSpace. If so, it proceeds; otherwise it terminates.
 - b) It examines the RP policy to check whether use of personal cards is acceptable. If so, it proceeds; otherwise it terminates, giving CardSpace the opportunity to operate normally.

¹It is thus not necessary for the user to federate their RP account with the IdP, which is likely to be relatively difficult to achieve.

- c) It temporarily keeps a local copy of any RP-requested claims². Note, however, that such a step is not necessary if the RP only requires user authentication, or if the identity management system in use only supports the assertion of user authentication and does not support the exchange of user attributes.
- d) It determines the communication protocol (HTTP or HTTPS) in use with the RP. Note that, in order to avoid making any changes to identity selectors, the model operates slightly differently depending on whether the RP uses HTTP or HTTPS. This is because, if HTTPS is used, the selector will encrypt the RSTR using the RP site's public key, and the adaptor does not have access to the corresponding private key. Hence, it will not know whether to trigger the interoperation functionality, and will be unable to obtain the IdP URL; such issues do not occur if HTTP is used, since the selector will not encrypt the RSTR.
- e) If necessary, and if HTTP is in use, it modifies the RP policy to include all the types of claim supported by the InterCard in the identity management system-specific implementation of the model. For example, if the particular implementation stores the URL of the IdP in the *web page* field of the InterCard, then it must ensure that the RP security policy includes the *web page* claim. Note that adding the claim types to the RP policy ensures that the token supplied by the SIIP contains the values of these claims, which can then be processed by the adaptor; otherwise these values would not be available to the adaptor.
- f) It embeds a function into the login page to intercept the RSTR that will later be returned by the identity selector.

²This is because the RP policy might be modified in step 3e, and the adaptor will use this stored copy in step 8 to determine which user attributes should be requested from the IdP in the authentication request.

8. Identity Selector \rightarrow Adaptor/UA \rightarrow IdP. Following selection by the user of a suitable InterCard, the RSTR created by the SIIP is intercepted by the adaptor (using the functionality added in step 3f), which prevents it from being sent to the RP. The RSTR, a SAML authentication response, is temporarily stored by the adaptor. If the RSTR contains the implementation-specific trigger sequence (see section 5.2.3), the adaptor continues; otherwise, it terminates. The adaptor then performs one of the following steps, depending on the communication protocol used on the UA-RP channel (see step 3d).

- If the RP uses HTTP, the adaptor uses the contents of the RSTR to construct an authentication request which it forwards to the appropriate IdP, having discovered its address from the RSTR.
- If the RP uses HTTPS, the adaptor first asks the user whether use of the interoperation model is required. If not, it terminates. If it is, the adaptor prompts the user to enter the URL of the IdP. The adaptor could also offer the user the option to store the supplied value for future interactions with this RP. Precisely as in the HTTP case, the adaptor then constructs an authentication request, and sends it to the identified IdP URL.

Depending on the identity management system in use, it may be possible to avoid the need for the user to enter the URL of the IdP. In some cases (e.g. in Shibboleth), the adaptor could use a WAYF-like component in order to perform IdP discovery. In other cases (e.g. in OpenID), the user-supplied identifier will contain a pointer to a web document, from which the required IdP URL can be retrieved. However, such an approach, although it maximises user transparency, is susceptible to fake IdP attacks, where the retrieved URL may point to a malicious IdP. It also involves extra round trips, which could affect system performance (see sec-

tion 8.3.1).

9. $\text{IdP} \rightleftharpoons \text{User}$. If necessary, the IdP authenticates the user³. Depending on the identity management system in use, the IdP may at this point ask the user to authorise the release of a token to the requesting RP.
10. $\text{IdP} \rightarrow \text{UA}$. The IdP issues a security response to the UA, which is read by the adaptor.
11. $\text{UA/Adaptor} \rightleftharpoons \text{IdP}$ (Optional). Depending on the identity management system in use, the adaptor may interact with the IdP. This might, for example, be to perform security checks on the IdP-issued token (as in OpenID), or to retrieve the user attributes requested by the RP (as in some implementations of OAuth).
12. $\text{UA/Adaptor} \rightarrow \text{RP}$. The adaptor now has the security response from the IdP (see the previous two steps), which we suppose contains some kind of security token (e.g. a SAML token) intended for processing by the RP. The adaptor also possesses the digitally-signed RSTR issued by the SIIP. The adaptor can now create a new SAML token, possibly incorporating both these tokens and/or information extracted from them. This encapsulating token can be formatted to resemble a standard CardSpace-complaint token, except of course for the fact that it contains another token (or tokens) embedded within it. The adaptor causes the UA to forward the newly created token to the RP, optionally after first obtaining permission from the user.

If the RP is capable of processing SAML 2.0 tokens, then it can use the SAML 2.0-supported holder-of-key method (see section 4.7.9), which can be symmetric or asymmetric, to express its proof-of-possession requirements. However, a symmetric proof key should only be used if the user is willing

³The authentication method used is typically outside the scope of the identity management system in use.

to disclose the identity of the RP to the IdP, and if the RP holds a valid certificate. For browser-based applications, or where no proof-of-possession is needed, or if the identity management system in use does not support proof-of-possession methods, the model allows the use of bearer tokens (see section 4.3.12).

Finally observe that the additional steps listed above can be integrated into the current CardSpace framework relatively easily, as the prototype implementations of specific schemes show (see chapters 6 to 9).

5.3 Operational Issues

We now consider implementation and applicability issues.

5.3.1 Triggering the Adaptor

The means by which the adaptor is triggered should be chosen carefully. The means specified in section 5.2.3 is to include a trigger sequence (e.g. the word *interoperation*) in a specific field of an InterCard. However, other approaches could be used, e.g. as follows.

- The adaptor could start whenever CardSpace is triggered. When a user selects an InterCard, the adaptor could offer the user two options via an HTML form: to continue to use CardSpace as usual, or to use the interoperation functionality. This approach gives a greater degree of user control, and hence implements Cameron's first identity law (see section 3.8). However, it is not particularly convenient, since it would always require users to make an explicit choice, although the effect could be reduced by storing the user preference.
- Alternatively, the adaptor could ask the user whether they wish to activate the interoperation model (e.g. via a JavaScript pop-up box). This has advantages and disadvantages similar to those of the first alter-

native. This is the approach adopted in the case where the RP uses HTTPS (see step 8 in section 5.2.4).

5.3.2 Attribute Handling

Different identity management systems use different sets of attribute types. For example, CardSpace personal cards support fourteen editable attributes (see section 4.3.3.1), whereas the OpenID SREG extension currently only supports nine attribute types (see section 4.5.9.1). Shibboleth and Facebook Connect support many more. Such differences in attribute types clearly cause a problem in creating an attribute request message for an IdP supporting one identity management system from a policy statement provided by an RP using another system. We outline below two possible approaches for dealing with this problem.

1. The RP could be restricted to requesting only attributes from the set supported by personal cards. The adaptor would need to convert the attributes requested by the RP into the form supported by the particular IdP, and include these converted attributes in the request message sent to the IdP. The token provided by the IdP will contain attributes in the IdP-supported format, and hence the RP will need to be able to process them. To assist in this process, the encapsulating token generated by the adaptor could include information to help the RP interpret the attributes.
2. Alternatively, the RP could be permitted to request any of the attribute types supported by the IdP. However, if the RP requests an attribute type not supported by personal cards, then clearly the selector will not highlight any of the personal cards.

In order to avoid this problem, the adaptor must modify the RP policy before it reaches the selector. In particular, as part of step 3, the adaptor must (after storing them) remove the attributes that are outside the set

supported by personal cards, and then insert them in the request sent to the IdP as part of step 8.

To support the broadest range of user attributes, the adaptor could be configured to support both of the approaches described above. Attribute mapping is, of course, unnecessary when operating with identity management systems that do not support the exchange of user attributes.

5.3.3 Implementing the Adaptor as a Browser Extension

If the integration adaptor is implemented as a browser extension, as is the case in the instantiations of the model described in subsequent chapters, then the RP must not employ an STS. Instead, the RP must express its security policy using HTML/XHTML, and interactions between the identity selector and the RP must be based on HTTP/S via a web browser (a simpler and probably more common scenario for selector-RP interactions⁴). This is because a JavaScript-based browser extension is by itself incapable of managing the necessary communications with an STS. Support for STS-enhanced RPs remains a possible topic for future work.

5.4 Advantages

5.4.1 Defeating Fake IdP Attacks

The model mitigates the risk of a fake IdP attack, e.g. as resulting from a phishing attack. This is because the redirect to the IdP is initiated by the adaptor and not by the RP, i.e. the RP cannot redirect the user to an IdP of its choosing. By contrast, in identity management systems using HTTP redirects, (such as Liberty artifact and browser-post profiles, Shibboleth, OpenID, and OAuth), a malicious RP could redirect a user to a fake IdP, which might then be able to capture user credentials and/or sensitive at-

⁴<http://msdn.microsoft.com/en-us/library/aa480189.aspx>

tributes. This is a particular threat for static credentials, such as usernames and passwords.

5.4.2 Client Interoperation

IdPs and/or RPs may not be prepared to accept the burden of supporting two identity management systems simultaneously, at least unless there is a financial incentive to do so. Currently, major Internet players do not support interoperation between identity management systems. As a result, a client-side technique for supporting interoperation could be practically useful.

In addition, implementing the interoperation functionality on the client means that the performance of the server is not affected, since the integration overhead is handled by the client.

5.4.3 Consistency

A major problem faced by end users is that the user experience of almost every identity management system is different. It is widely acknowledged that users fail to make good security decisions, even when confronted with relatively simple decisions (see section 1.2.4). As a result, this lack of consistency is likely to make the situation worse, and users are likely to have difficulty understanding the complex privacy- and security-relevant decisions that they are being asked to make. The client-based approach to interoperation proposed here can help to address this issue by enabling the user to interact with a single selector interface, regardless of which identity management system is in use.

5.4.4 Unintentional Leakage

When using IdPs which provide assertions about user attributes, there is a danger that an end user could damage their privacy by unintentionally revealing attributes to an RP. In general, getting settings correct for systems handling PII is a non-trivial task [157]. The interoperation functional-

ity could help to improve user privacy by inspecting the IdP-issued security token and displaying a summary of its contents to the user before releasing it to the requesting RP. Note that such a function can only be provided if the token is not encrypted in such a way that only the RP can read it, e.g. as would be the case if it was encrypted using the RP's public key.

5.5 Security Considerations

As described in step 12 of section 5.2.4, the adaptor produces and sends to the RP a single SAML token (the encapsulating token) that combines information provided by the SIIP and the remote IdP. Specifically the encapsulating token contains a copy of the signed SAML token produced by the SIIP (i.e. the RSTR), and may also contain a copy of a signed or MAC-protected token generated by the IdP (depending on the specifics of the identity management system in use by this IdP). Thus the RP will potentially have two independently generated tokens containing user attributes. Where relevant, the RP can compare these two sets of attributes, potentially gaining greater confidence in their correctness as a result.

More generally, use of this general approach will benefit from the security features provided by CardSpace (see section 4.3).

Note that the interoperation model allows the user attributes to be stored remotely at the IdP; this has potential security advantages over storing the attributes locally on the user platform, as is currently the case with the personal card user attributes.

Even if the encapsulating token does not include a copy of an integrity-protected token generated by the IdP, in some circumstances the RP may still be able to gain additional assurance in the provided user attributes (and user authenticity). If the RP trusts that the correct adaptor is running unmodified on the user platform, then the RP will know that the user has been authenticated by a specific IdP (and possibly when and how), and that the

attributes have been provided by the IdP. In such a case, the RP would be provided with two-factor user authentication, based on selection of the correct InfoCard and user authentication at the IdP. This would offer a security advantage by comparison with the *native* CardSpace personal card protocol, which only provides a single-factor user authentication.

However, requiring the RP to trust that the correct software is running on the user platform is a significant trust assumption. We next consider two ways in which this assumption might be met.

- The interoperation software could be installed in a managed environment in which a user is only granted limited privileges insufficient to modify or replace the adaptor. However, in order for the RP to have assurance that the user platform is in such a controlled environment (and to avoid making extra changes to the RP server), the RP itself would probably need to belong to the managed environment.
- A more widely-applicable solution would be to make use of the functionality of the *trusted platform module* (TPM) [97, 104, 218], present on a large proportion of recently manufactured PCs. Using the remote attestation mechanism, an RP could be provided with guarantees about the software state of the user platform, including the presence of the expected integration software.

5.6 Potential Issues

If the web browser is compromised, then an adversary could steal the user token and use it to impersonate the user. Indeed, if we assume that the web browser is not a secure environment, it may be possible for a malicious plug-in or other malware to get access to sensitive information present in the (plaintext) RSTR⁵, the adaptor-generated SAML token (i.e. the encaps-

⁵If the RP does not use HTTPS, then the SIIP-issued RSTR will not be encrypted.

sulating token), or the IdP-issued security token. However, the same risks apply when manually entering credentials such as username and password into a browser [109].

The integration adaptor must scan every browser-rendered web page to detect whether it supports CardSpace, and this may affect system performance. However, informal tests on the prototypes (described in chapters 6 to 9) suggest that this is not a serious issue. In addition, the adaptor can be configured so that it only operates with certain RPs.

5.7 Possible Extensions

5.7.1 Scope

The model proposed here applies to users of Information Card-enabled RPs such as CardSpace. While the CardSpace identity selector can only retrieve security tokens from CardSpace-enabled IdPs, the model extends this capability to enable security tokens to be obtained from non-Information Card-enabled IdPs, using the identity selector and the adaptor.

However, interoperation between a CardSpace-enabled IdP and non-Information Card-enabled RPs (such as a Shibboleth-enabled RP) is not supported. Indeed, without technical co-operation from the bodies responsible for developing the specifications governing these systems, it appears likely to be difficult to implement bidirectional interoperation.

5.7.2 U-Prove Tokens

Given that CardSpace supports security tokens of any type, the model presented here can be extended to support U-Prove tokens (see section 4.9.2). In addition to supporting a SAML-based RSTR, the protocol specified in section 5.2.4 can also be configured to support a U-Prove-based RSTR, which is also XML-encoded. Indeed, the user experience would be precisely the same when using an InfoCard supporting U-Prove.

Note that, in CardSpace version 2.0, U-Prove tokens can only be chosen by selecting managed cards (as opposed to personal cards). This means that U-Prove tokens cannot be supported by the current version of the model, since it does not cover the case where the RP policy specifies use of a managed card (see section 5.2). However, we believe that the model could relatively easily be modified to support RP policies which request U-Prove tokens. The U-Prove token can be identified by the URL: `http://schemas.xmlsoap.org/ws/2010/03/uprove/token`.

5.8 Related Work

We review prior work on identity system interoperation under the following headings:

- specification and open-source development projects;
- general-purpose interoperation models;
- interoperation between specific systems; and
- business analysis of interoperation issues.

5.8.1 Specification and Open-source Development Projects

The Bandit⁶ and Concordia⁷ projects have developed open-source technologies in order to support interoperation between identity management systems. Whilst the model proposed here supports interoperation at the client, Concordia and Bandit appear to offer integration services at the server.

Project Concordia⁸ is a global initiative aimed at promoting harmonisation and interoperability between identity standards and protocols. Concor-

⁶<http://www.bandit-project.org>

⁷<http://kantarainitiative.org/confluence/display/concordia/Project+Concordia+Historical+Works>

⁸<http://kantarainitiative.org/confluence/display/concordia/Home>

dia⁹ has proposed a scheme supporting interoperation between Information Card-based identity management systems and federated identity management systems (specifically those built on SAML SSO profiles and those built on WS-Federation). This scheme enables users of one system to obtain security tokens from IdPs supporting the other system (see section 6.5).

The Bandit project, sponsored by Novell¹⁰, has as its objective the development of a common identity framework¹¹ and integration techniques to support interoperation between Information Card systems [21]. As the two projects had similar objectives, the Bandit team have joined forces with the Higgins project [168]. Higgins (see section 4.4) and Bandit have ensured that the (Bandit-supported) DigitalMe identity selector is interoperable with the Higgins identity selectors [21].

SWITCH¹² has developed specifications for an integration framework in order to support the exchange of WS-Trust messages in the Liberty ID-WSF [85]. This could potentially be useful in supporting interoperation between CardSpace and Liberty, and also for mapping between the necessary security tokens [21].

The Identity Commons working group¹³ aims to provide standardised attribute types to support interoperability. This is important since each identity management system has its own set of attribute types. For example, the family name attribute might be known as *last name* in one system and *surname* in another [210]. According to its website, the purpose¹⁴ of this working group is to ‘support, facilitate, and promote the creation of an open identity layer for the Internet, one that maximises control, convenience, and privacy for the individual while encouraging the development of healthy, interoperable communities’ [168].

⁹Concordia has joined the Kantara initiative (<http://kantarainitiative.org/confluence/display/concordia/Home>).

¹⁰<http://www.novell.com>

¹¹<http://www.bandit-project.org>

¹²<http://www.switch.ch/>

¹³<http://www.idcommons.org/>

¹⁴<http://www.idcommons.org/purpose-and-principles/>

5.8.2 General-purpose Interoperation Models

García and Oliva [98] seek to address identity management system interoperability at a pan-European level. They have analysed, and proposed changes to, existing interoperation schemes [99], as well as practically testing their viability. Problems encountered included trust issues, semantic interoperability (i.e. translation between representation formats), as well as identity delegation [100] and authorisation.

Bruegger et al. [46] outline their vision of global (electronic) identity interoperability, and discuss challenges to their vision together with steps they believe are necessary to tackle these challenges. They also consider the deployment of electronic ID cards in Europe, paying particular attention to interoperability issues for such cards. They have also proposed a scheme they call *TLS-Federation* [47]. One objective is to provide a regulatory and interoperable working framework for identity management, particularly at a pan-European level. It uses the TLS handshake protocol and TLS client authentication (the RP is assumed to configure the TLS server to request a client certificate for user authentication).

Koshutanski et al. [152] propose an identity management interoperation scheme using SAML 2.0 to act as a bridge between systems. The SAML protocol is used to request and receive authentication assertions. An abstract view is provided to a user of his or her identity information, such as identity certificates, username-passwords, public-private keys, etc. To avoid denial-of-service and to ensure availability, this user profile is replicated in encrypted form on trusted peers. However, the user must (at least initially) perform three authentication processes. Firstly, the user must authenticate to the digital ecosystem [174], e.g. using a username-password pair. Secondly, the user must enter a master password to decrypt the user profile retrieved from a trusted peer. Thirdly, the user must further authenticate to an RP-trusted IdP. This is clearly a non-trivial procedure, although the

three authentication processes might only need to occur together at the start of a session. Unlike the model described in this chapter, IdPs and RPs are required to support SAML. The scheme has the advantage of supporting user roaming at the cost of introducing trusted third parties.

Ates et al. [25, 26] propose a means of supporting interoperation between heterogeneous federation architectures, specifically between SAML 2.0 and WS-Federation 1.1. Unlike the model described here, which is based on locally running client software, a dedicated third party is responsible for implementing the interoperation functionality. The authors also argue that if IdPs and RPs are directly connected through trust links, then interoperation can be achieved by making an IdP or an RP responsible for the interoperability processes. This third party must implement the SAML and WS-Federation specifications, and it must also be trusted by IdPs and RPs. In addition, it must be able to translate between SAML and WS-Trust request and response messages. This third party could potentially become a single point of failure, and could also hinder performance if deployed on a wide scale. Since no prototype appears to have been developed [25, 26], issues which might arise during implementation and deployment have not been explored.

Jo et al. [141] propose a scheme to support interoperation between identity management systems, specifically CardSpace and OpenID, whilst also providing a degree of user anonymity. Unlike the model proposed here, a third party online server is needed. The authors claim that compatibility with username-password systems is of vital importance to an identity management system, and certain variants of their scheme are specifically designed to be compatible with such systems.

5.8.3 Interoperation Between Specific Systems

A CardSpace-Liberty interoperation scheme [22] has been proposed. This scheme has some properties in common with one of the specific schemes

conforming to the model described here (see chapter 6), notably that both schemes support interoperation at the client rather than at the server. However, there are a number of important differences, outlined in section 6.5.

Another scheme supporting interoperation between CardSpace and Liberty has been proposed by Jørstad et al. [144]. In this scheme, the IdP is responsible for supporting interoperation (see section 6.5).

In 2007, Internet2 announced¹⁵ plans to develop extensions to Shibboleth to support CardSpace. This included collaboration with Microsoft in order to add Information Card support to Shibboleth (see section 7.5).

Kim et al. [150] propose an OpenID authentication method using an identity selector. This scheme is designed to reduce phishing and hacking risks, and also to simplify user authentication (see section 8.5).

Microsoft and OpenID have announced plans¹⁶ to enable a level of interoperation. A stated aim of this effort is to reduce the risk of phishing in OpenID (see section 8.5).

5.8.4 Business Analysis of Interoperation Issues

Palfrey et al. [189] and Rundle et al. [210] analyse the interoperability issue, including assessing its potential benefits and drawbacks. They also discuss the role that the market and competition could play.

Building on a case study [189], Palfrey et al. conclude that currently there is no single, clear path to the type of interoperability that would lead to further innovation. However, they argue that a combination of industry efforts with a light-touch role for governments could potentially lead to greater levels of interoperability in the digital identity space. They also point out that interoperability could create new markets and lead to further innovations.

¹⁵<https://lists.internet2.edu/sympa/arc/i2-news/2007-05/msg00009.html>

¹⁶<http://www.guardian.co.uk/technology/blog/2007/feb/07/openidgetsab>

5.9 Conclusions and Future Work

In this chapter we have proposed a general, client-based model to support interoperation between an Information Card-based identity management system, such as CardSpace or Higgins, and almost any other type of system, including systems such as OpenID, OAuth, Liberty and Shibboleth. Following this model, Information Card users are able to obtain a security token from an arbitrary identity provider, which, after encapsulation at the client, can be processed by an Information Card-enabled relying party.

The model uses the identity selector interface and personal cards to give access to the interoperation functionality. Interoperation is supported at the client in a way that is as transparent as possible to identity providers and relying parties, and requires minimal changes to the existing user platform infrastructure. The model takes advantage of the similarity in message flows in identity management systems, and does not require technical co-operation from the bodies governing those systems.

Planned future work includes exploring the possibility of building a similar, client-based model to support interoperation between an Information Card-based identity provider and relying parties conforming to other identity management systems.

Interoperation Between an Information Card System and Liberty

6.1 Introduction

In this chapter we describe the first instantiation of the interoperation model given in chapter 5; it enables interoperation between an Information Card system and Liberty. Information Card users are able to obtain a security token from a Liberty-enabled IdP that is made usable by an Information Card-enabled RP. The approach works as long as the Liberty-enabled IdP supports either the browser-post or the Liberty-enabled client profile. In the latter case, the adaptor provides the Liberty-enabled functionality normally requiring a separate piece of client-installed software. Much of the material in this chapter has been published [8].

For simplicity of presentation, in this chapter we assume that the Information Card system is CardSpace, although an identical approach will work with other Information Card systems such as Higgins.

As discussed in chapter 5, the wide adoption of Liberty and the inclusion of CardSpace in recent versions of Windows means that enabling interoperation between the two systems could offer significant benefits. Another motivation for supporting CardSpace-Liberty integration is the similarity between the message flows of Liberty ID-FF and CardSpace. In addition, both schemes support SAML tokens.

The remainder of the chapter is organised as follows. Section 6.2 de-

tails the interoperation process. In section 6.3, an operational analysis is provided and, in section 6.4, we describe a prototype implementation. Section 6.5 highlights possible areas for related work, and, finally, section 6.6 concludes the chapter.

6.2 Interoperating with Liberty

We now describe how interoperation with Liberty is achieved.

6.2.1 System Entities

As in chapter 5, the entities involved are:

1. a CardSpace-enabled RP;
2. a CardSpace-enabled UA, e.g. a suitable web browser such as Internet Explorer;
3. a Liberty-enabled IdP; and
4. the integration software (the adaptor), which in this case we suppose takes the form of a browser extension installed on the user platform.

Fig. 6.1 gives an overview of the high-level interactions between the system components on the user platform. The components shown are: the adaptor (a browser extension/plugin), the UA (a browser), the identity selector, and the SIIP. The arrows indicate information flows.

6.2.2 Requirements

The scheme described here has the same operational requirements as those listed in section 5.2.3, where the IdP is a Liberty-enabled IdP and the Inter-Card we refer to below as a *LibCard*.

In addition, the CardSpace-enabled RP must support SAML 2.0 (see section 2.5.7), since, as stated in section 4.7.9, the Liberty-enabled IdP will gen-

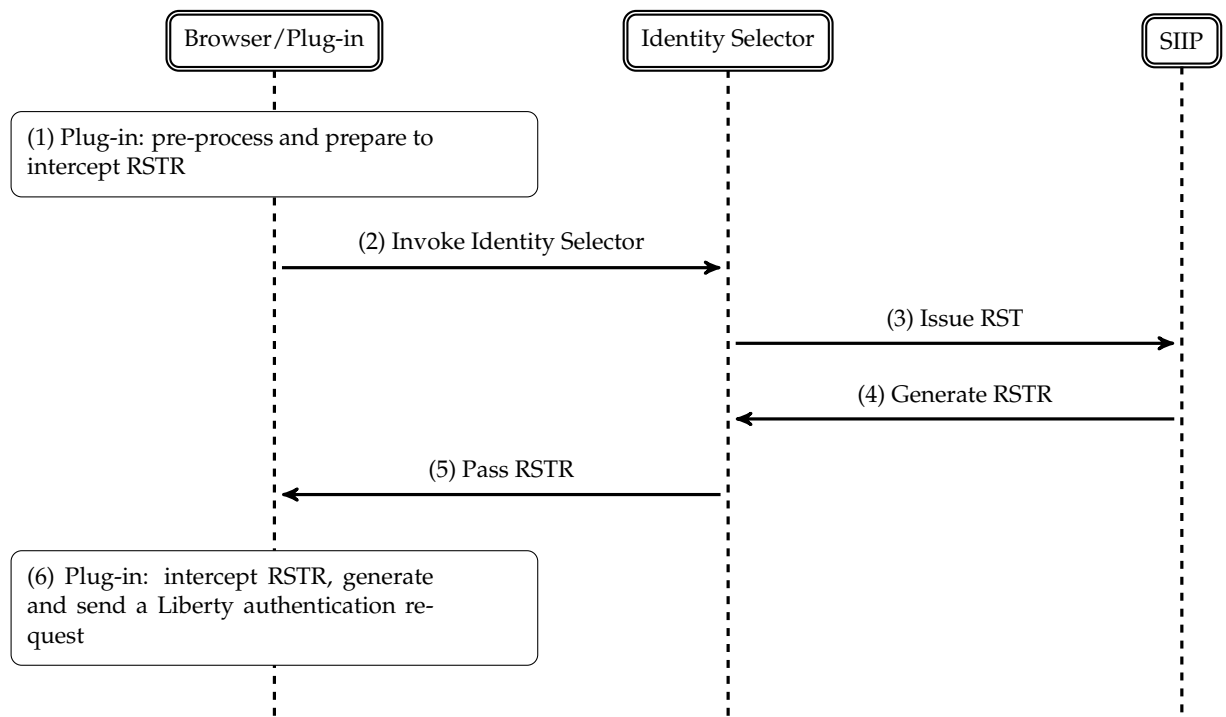


Figure 6.1: Data Flows via Client Components

erate a token in this format. The RP must also be prepared to accept SAML tokens in the format constructed by the adaptor.

Moreover, the Liberty-enabled IdP must support the Liberty browser-post profile or the Liberty-enabled client profile. If the IdP uses the browser-post profile then certain minor changes must be made to the way in which it submits a response back to the UA (see section 6.2.4 below).

6.2.3 Operation

Fig. 6.2 gives an overview of the operation of the scheme, with the step numbers shown. The sequence of steps is precisely as given in section 5.2.4. We specify below only those steps in which Liberty-specific operations are performed (observing that, as noted above, we use the term *LibCard* for the Liberty-specific InterCard). The sequence of steps given below applies for both the Liberty browser-post and Liberty-enabled client profiles.

3. In this case step 3c is null.

8. Identity Selector → Adaptor/UA → IdP. Following the selection by the user of a suitable LibCard, the RSTR created by the SIIP is intercepted by the adaptor, which temporarily stores it. The adaptor then performs one of the following steps, depending on whether or not the HTTP channel between the UA and the RP is SSL/TLS-protected.

- If the RP uses HTTP, the adaptor first examines the RSTR to discover whether or not it contains the LibCard-specific trigger sequence (e.g. the word *Liberty* — see section 5.2.3); if so, the adaptor proceeds; otherwise, it terminates. On proceeding, the adaptor uses the contents of the RSTR to construct a Liberty authentication request which it forwards to the appropriate Liberty IdP, having discovered its address (and the profile it employs) from the RSTR. The detailed format of the SAML authentication request will depend on the Liberty profile being used (see section 6.2.4 below).
- If the RP uses HTTPS, the adaptor first asks the user whether use of the integration scheme is required. If not, it terminates. If it is, the adaptor prompts the user to enter the URL of the IdP as well as the profile it employs. The adaptor could also offer the user the option to store the supplied values for future interactions with this RP. Precisely as in the HTTP case, the adaptor then constructs a Liberty authentication request, the form of which depends on the Liberty profile in use (see section 6.2.4 below), and sends it to the Liberty-enabled IdP.

10. IdP → UA. Following a successful user authentication in the previous step, the IdP sends a digitally-signed SAML token to the UA. The IdP response is Liberty profile-dependent (see section 6.2.4 below).

11. This step is null in this case.

12. Adaptor/UA → RP. The adaptor generates an unsigned SAML token that contains both the digitally-signed SIIP-issued RSTR as well as the digitally-signed (Liberty) IdP-issued token. The UA then forwards the token to the RP, optionally after first obtaining permission from the user.

6.2.4 Liberty Profiles

The detailed operation of steps 8, 10, and 12 depends on whether the Liberty browser-post profile or the Liberty-enabled client profile is in use. The URL: `http://projectliberty.org/profiles/brws-post` must be used when employing the browser-post profile, whereas the URL: `http://projectliberty.org/profiles/lecp` must be used when employing the LEC profile. In addition, when using the LEC profile, the authentication request must be submitted to the IdP as a SOAP request (see section 2.5.6) with a Liberty-enabled header; however, when using the browser-post profile, the authentication request to the IdP can be embedded in an HTML form containing a field called *LAREQ* carrying the `<lib:AuthnRequest>` protocol message [58, 60]. In order to support both profiles, the adaptor must therefore be capable of supporting both forms of communication with the IdP (see also section 6.3.1).

When using the LEC profile, in step 10 the IdP returns the authentication response to the client (which is responsible for forwarding it to the specified RP). In normal operation of the Liberty browser-post profile, however, the IdP sends the HTML form carrying the authentication response to the UA, and redirects the user via the UA to the specified RP. Such a procedure would deny the adaptor (i.e. the browser extension) the opportunity to intercept the communication and give the user the choice whether or not to allow the token to be sent to the RP (as is normally the case for CardSpace).

We therefore require a small modification to the way that the Liberty-enabled IdP operates. The IdP must be modified to redirect the UA to a

web page at the IdP server, rather than at the RP, thereby giving control to the adaptor. This could be achieved by requiring the IdP to set the action attribute¹ of the HTML form to an empty string or to hash (#)². In step 12, the browser extension resets the action attribute to the URL of the appropriate CardSpace RP, and, after obtaining user permission to release the authentication token to the given RP, automatically submits the HTML form, redirecting the UA to the RP website. This small change to the normal operation of the Liberty IdP helps to enhance user control (see also section 6.3.3), hence implementing Cameron's first identity law (see section 3.8).

Note that both the Liberty browser-post and LEC profiles require the RP URL to be specified as the value of the `<lib:AssertionConsumerServiceURL>` statement in the SAML authentication request [58]. To minimise the required changes to the operation of the IdP, the value of this field could be set to hash (#), implicitly instructing the IdP to include this value instead of the RP's URL in the action attribute of the HTML form sent back to the UA.

6.3 Discussion and Analysis

We now consider implementation and applicability issues of the scheme.

6.3.1 Applicability

The scheme described above supports both the Liberty browser-post and Liberty-enabled client profiles, introduced in section 4.7.7. However, the prototype described in section 6.4 only implements the Liberty browser-post profile. Adding support for the Liberty-enabled client profile is expected to be straightforward.

¹Observe that, in the standard Liberty browser-post profile case, the action attribute of the HTML form is set to the URL of the requesting RP, and the IdP redirects the UA to that RP.

²Note that whilst this has been shown to work successfully with Internet Explorer (versions 7 and 8), other browsers may not support an action attribute of an empty string or hash (#); hence setting the action attribute to a relative URL for the IdP login page may be required for such browsers.

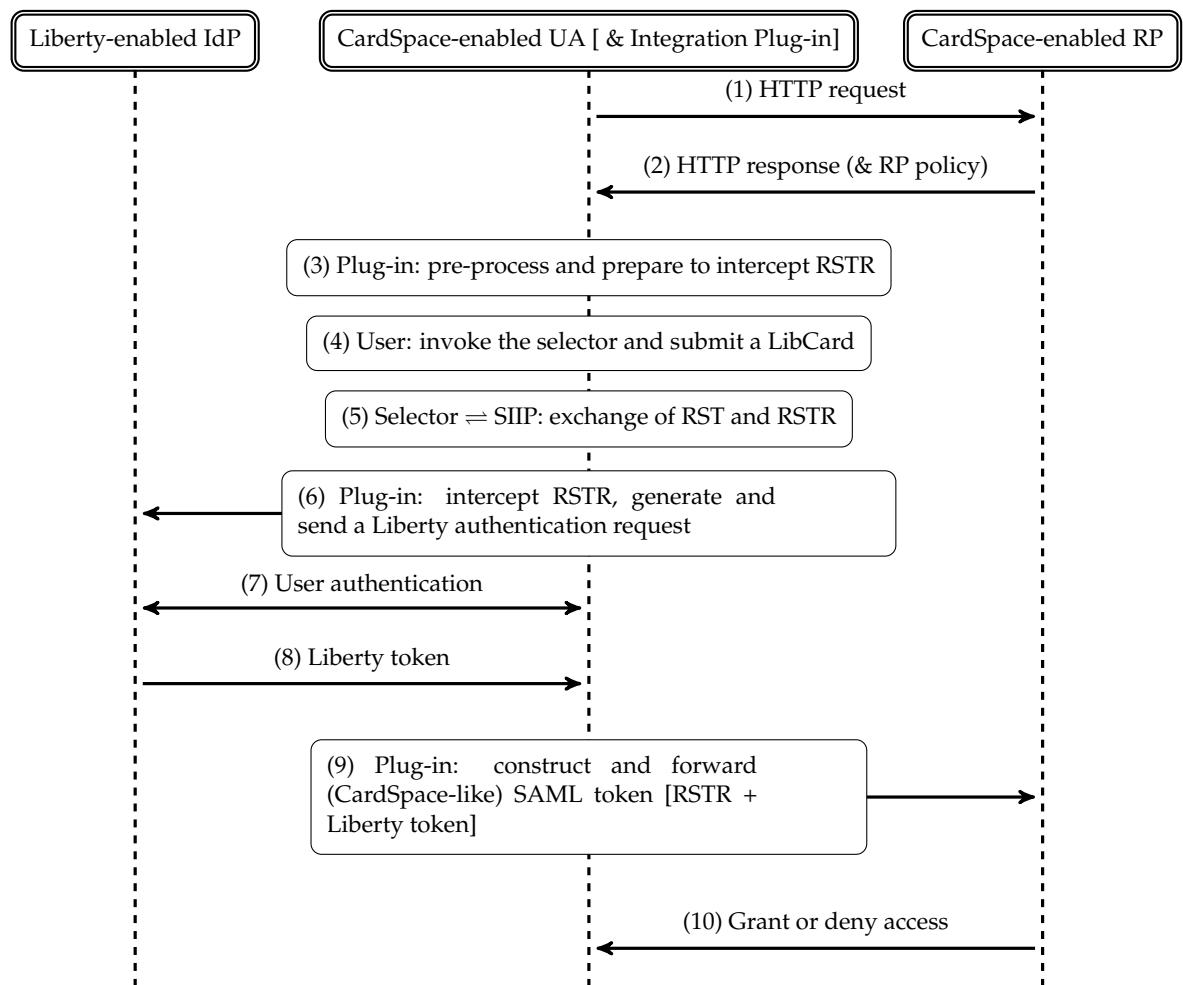


Figure 6.2: Exchanges Between the Principal Parties

Providing support for these two profiles is simplified by the many properties that the two profiles have in common. For example, both profiles support SAML. In addition, in both profiles the HTML form containing the authentication response must be sent to the UA using an HTTP POST; this form must contain the field *LARES* with value equal to the authentication response, as defined in the Liberty protocol schema [60]. Furthermore, in both profiles, the value of the *LARES* field must be encoded using a base-64 transformation [95].

6.3.2 Differences in Scope

There is a key difference between the Liberty ID-FF and CardSpace frameworks. CardSpace allows IdPs to assert a range of attributes about users, including simple authentication assertions, whereas Liberty ID-FF only supports authentication assertions (see section 4.7.3). In CardSpace, the user attributes are specified in a SAML attribute statement contained in a SAML request that can be processed by the local SIIP or the remote CardSpace-enabled IdP. However, an IdP conforming to Liberty ID-FF is only required to generate SAML authentication statements, which gives rise to an inter-operation problem.

Two possible solutions to this problem are as follows.

1. It could be assumed that the CardSpace-enabled RP is only concerned with user authentication, which seems likely to be a common case. In such a case a LibCard would contain the IdP URL and the trigger word, and a LibCard would only be used (in HTTP mode) if the RP security policy requests an assertion solely of the PPID attribute, i.e. via inclusion of `http://schemas.xmlsoap.org/ws/2005/05/identity/claims/privatepersonalidentifier` as an entry in the list of mandatory claims. In such a case, the browser extension can modify the RP policy to ensure it includes the fields used in LibCards. On selection of a LibCard, the browser extension (as in step 8 in section 6.2.3) intercepts the RSTR, and then creates and forwards a SAML authentication request to the user-selected IdP. While this is a straightforward task, it limits the applicability of the scheme.
2. Alternatively, it could be assumed that a CardSpace-enabled RP might be concerned with both user authentication and the assertion of user attributes, and that the RP policy permits assertions (for user attributes only) to be provided by the SIIP. In this case, along with requiring the PPID, the RP security policy would also specify the attributes re-

quired. This will cause the CardSpace identity selector to highlight the user-created LibCards that satisfy the requirements. To ensure that no additional changes are required at either the RP or the IdP, the browser extension could store attribute assertions created by the SIIP. The browser extension would then create a Liberty ID-FF conformant SAML authentication request, and forward it to the specified IdP. When the browser extension receives the response containing the authentication assertion from the IdP, it would add appropriate attribute assertion(s) from its local cache and then forward the entire SAML token to the RP. However, if the RP security policy dictates that security tokens must be wholly signed by the issuing IdP, then this solution would clearly fail.

The prototype implementation described in section 6.4 implements the first approach.

6.3.3 Token Forwarding

The means by which the security token is forwarded to the RP needs to be chosen carefully. In the discussion below, we refer to the numbered protocol steps given in section 6.2.3.

The responsibility for delivering the security token could be given to the Liberty IdP, as is normally the case when using the Liberty browser-post profile. In this case the RP address could be added to the SAML authentication request (as prepared in step 8) so that the IdP knows to which RP it must forward the token (again as is normally the case for the Liberty browser-post profile). Although this would avoid the need for changes to the normal operation of the Liberty IdP and potentially also help auditing, such an approach has privacy implications since the IdP would learn the identity of the RP.

As a result, as specified in step 12 of the scheme, the responsibility for

sending the security token to the RP is given to the adaptor. Thus a means is required for giving the adaptor (the browser extension) the address of the RP, so that it can forward the token. We next consider a number of possible ways in which the RP address might be made available.

- The RP address could be stored in the browser extension itself. Whilst this puts the user in control, it is not user-friendly, as it would require users to manually add the address of each RP into the code of the browser extension.
- After the security token has been returned from the Liberty-enabled IdP, the browser extension could ask the user to enter the RP address, e.g. using a JavaScript pop-up box or an HTML form. This has advantages and disadvantages similar to those of the previous alternative.
- The RP address could be stored using recently-developed techniques, such as *HTML v5 localStorage*³ [2]. The browser extension could store the RP URL in an HTML v5 *localStorage* object as part of step 3, so that the extension can retrieve the RP URL in step 12. However, at the time of writing, HTML v5 is not yet an official standard, and only recent versions of some modern browsers incorporate support for HTML v5; indeed no browser provides full support for HTML v5⁴.
- The browser extension could store the RP URL encrypted in a cookie⁵ as part of step 3, so that the browser extension can obtain the address in step 12. However, a simple implementation of this approach will not work, since the browser will be communicating with two different domains, the RP and the IdP domain, at the relevant times. In order to comply with the cookie's same-origin principle, the browser must be-

³http://www.w3schools.com/html5/html5_webstorage.asp

⁴http://www.w3schools.com/html5/html5_intro.asp

⁵Note that creation of, and access to, the cookie can be handled by the browser extension, transparently to RPs and IdPs.

lieve that it is communicating with the same domain when the cookie is set and when it is retrieved (see section 2.3.4.1).

This issue can be avoided in the following way. The browser extension encrypts and stores the RP address in a cookie in step 3, before the identity selector is invoked. As part of step 8, the browser extension retrieves the encrypted value from the cookie and sends it to the IdP as a hidden variable in an HTML form or as a query URL parameter (see section 2.5.3). At the same time the intercepted RSTR is also encrypted and sent to the IdP as a hidden form variable or as a query URL parameter. As part of step 10, the IdP returns the encrypted values of the RP address and the RSTR to the UA unchanged (again as a hidden form variable or as a URL parameter). In step 12, the browser extension retrieves and decrypts the enciphered values.

Note that the IdP is unable to read the RP address or the RSTR, since they are encrypted using a key known only to the browser extension, hence enhancing user privacy.

If the IdP, however, needs the RP address for auditing purposes (e.g. for legal reasons), or the IdP policy requires the disclosure of the RP identity (e.g. so it can encrypt the security token using the RP's public key), then the RP address could be sent to the IdP.

6.3.4 Possible Extensions

Although the instantiation of the general interoperation model we have given in this chapter is presented as Liberty-specific, we suspect that a very similar approach would also work with SAML-compliant IdPs; some minor changes, however, would be required. For example, the technical differences⁶ between Liberty ID-FF 1.2 and SAML 2.0 would need to be carefully

⁶<https://spaces.internet2.edu/display/SHIB/SAMLLibertyDiffs>

examined. However, SAML 2.0 has many similarities to Liberty ID-FF 1.2 (see section 2.5.7), and so a mapping seems likely to be possible.

Reconfiguring the scheme to interoperate with SAML-aware IdPs potentially significantly increases its applicability and practicality. For example, the exchange of identity attributes, which is not supported by the current scheme, would then be feasible. The reconfiguration of the scheme remains possible future work. Note that the extensions discussed in section 5.7 are also applicable here.

6.4 Prototype Realisation

This section provides technical details of a prototype implementation of the scheme which operates with IdPs conforming to the Liberty browser-post profile. Properties and possible limitations of the current prototype are also described.

6.4.1 User Registration

Prior to use, the user must have accounts with a CardSpace-enabled RP and a Liberty-enabled IdP. The user must also create a LibCard for the relevant Liberty IdP (or it could be created at the time of use). This involves invoking the selector and inserting the URL of the target Liberty IdP in the *web page* field⁷ and the trigger word *Liberty* in the *city* field. An example of a LibCard is shown in Fig. 6.3.

6.4.2 Implementation Details

The prototype is coded as a browser plug-in using JavaScript [180, 198], chosen to maximise portability. Indeed, JavaScript⁸ currently appears to be the most widely browser-supported and commonly used client-side scripting

⁷The *web page* field was chosen to contain the URL of the Liberty-enabled IdP since it seemed the logical choice; however, this is an implementation option.

⁸Throughout the description the term *JavaScript* is, for simplicity, used to refer to all variants of the language.

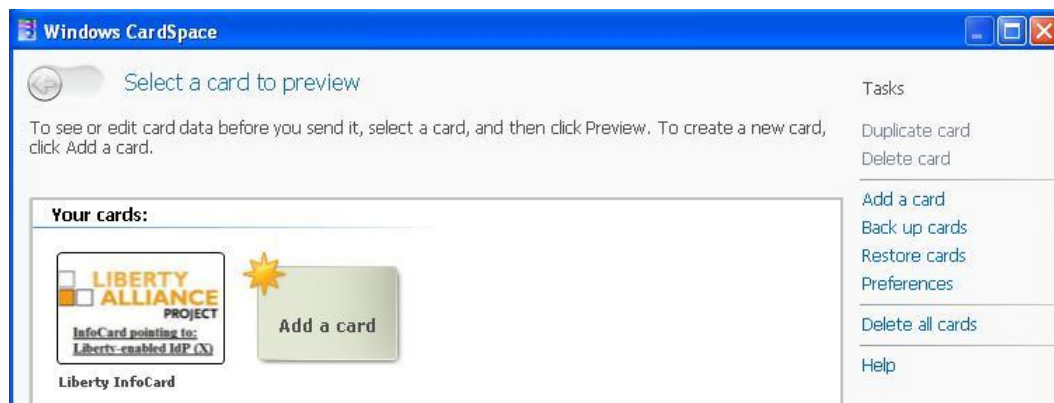


Figure 6.3: A LibCard

language [76]. Use of a browser-specific client-side scripting language, such as VBScript [151], was ruled out to maximise portability.

The JavaScript code is executed using a C#-driven *browser helper object* (BHO), a *dynamic-link library* (DLL) module designed as a plug-in for Internet Explorer. Once installed, the BHO attaches itself to Internet Explorer, thus gaining access to the current page's DOM (see section 2.5.2). The prototype can readily be enabled or disabled using the add-on manager in the Internet Explorer Tools menu.

The prototype implementation uses the DOM to inspect and manipulate HTML (see section 2.5.1) pages and XML (see section 2.5.6) documents.

The scheme operates with both the CardSpace and the Higgins⁹ identity selectors without any modification. Finally observe that the prototype plug-in does not require any changes to default Internet Explorer security settings, thereby avoiding potential vulnerabilities arising from such changes.

6.4.3 Prototype Operation

In this section we consider specific operational aspects of the prototype. We refer throughout to the numbered protocol steps given in section 6.2.3.

In step 3, before the HTML login page is displayed, the plug-in uses the DOM to perform the following processes.

⁹http://wiki.eclipse.org/GTK_Selector_1.1-Win

3.1 The plug-in scans the web page in the following way¹⁰.

- (a) It searches through the HTML elements of the web page to detect whether any HTML forms are present. If so, it searches each form, scanning through each of its child elements for an HTML object tag.
- (b) If an object tag is found, it retrieves and examines its type. If it is of type *application/x-informationCard* (which indicates website support for CardSpace), it continues; otherwise it aborts.
- (c) It searches through the *param* tags (child elements of the retrieved CardSpace object tag) for the *issuer* tag and examines its value; if it is `http://schemas.xmlsoap.org/ws/2005/05/identity/issuer/self`, indicating that the use of personal (self-issued) cards is acceptable, it continues¹¹; otherwise it terminates.
- (d) It also searches through the *param* tags for the *requiredClaims* tag, which lists the claims required by the RP security policy.
- (e) If the required claims include attributes other than the PPID claim, then the plug-in terminates, giving CardSpace the opportunity to operate normally. However, if only the PPID claim is requested, then the plug-in adds the *city* and *web page* claims to the *requiredClaims* tag, marking them as mandatory.

3.2 The plug-in adds a JavaScript function to the *head* section of the HTML page to intercept the XML-based authentication token (i.e. the RSTR) before it is sent back to the RP (such a token will be sent by the identity selector in step 8).

¹⁰The CardSpace documentation [142] specifies two HTML extension formats for invoking an identity selector from a web page, both of which include placing the CardSpace object tag inside an HTML form. This motivates the choice of the web page search method.

¹¹The plug-in also continues if the value of the issuer tag is set to *any*, *** or if the issuer tag is *absent*, since the use of personal cards is acceptable in these cases.

3.3 The plug-in obtains the action attribute of the CardSpace HTML form, encrypts it using AES in CBC mode¹² (see section 2.4.2.1) with a secret key known only to the plug-in, and then stores it in a cookie. This attribute specifies the URL of a web page at the CardSpace-enabled RP to which the authentication token must be forwarded for processing. If the obtained attribute is not a fully qualified domain name address, the JavaScript inherent properties, e.g. *document.location.protocol* and *document.location.host*, are used to help reconstruct the full URL.

3.4 The plug-in temporarily stores and then changes the current action attribute of the (CardSpace) HTML form to point to the newly created interception function.

3.5 The plug-in creates and appends an invisible HTML form to the HTML page to be used later for sending the SAML token request to the IdP.

In step 8 the plug-in uses the DOM to perform the following steps.

8.1 It intercepts the RSTR sent by the selector using the added function.

8.2 It parses the intercepted token. It then operates slightly differently depending on whether HTTP or HTTPS is in use. (The plug-in uses the JavaScript property *document.location.protocol* to determine whether HTTP or HTTPS is in use.)

- If HTTP is in use, the plug-in parses the RSTR and extracts the *city* and *web page* fields. If the *city* field contains the word *Liberty*, the plug-in proceeds; if not, it terminates. It also reads the *web page* field to discover the URL of the IdP.
- If HTTPS is in use, the plug-in asks the user whether use of the integration scheme is required, using a JavaScript pop-up box. If so, it proceeds; otherwise it terminates. On proceeding, it prompts

¹²Note that, ideally, an authenticated encryption mode should be used.

the user to enter the URL of the Liberty-enabled IdP. The plug-in offers the user the option to store the input value in a persistent cookie for future logins at this RP, using a plug-in-embedded checkbox.

The plug-in uses an XML parser built into the browser to read and manipulate the intercepted (XML-based) RSTR. The plug-in passes the token to the parser, which reads it and converts it into an XML DOM object that can be accessed and manipulated by JavaScript. The DOM views the XML token as a tree-structure, thereby enabling JavaScript to traverse the DOM tree to read (and possibly modify) the contents of the token elements. New elements can also be created where necessary.

- 8.3 It encrypts the RSTR using AES in CBC mode with a secret key known only to the plug-in.
- 8.4 It constructs a SAML authentication request, compatible with Liberty-conformant IdPs supporting the browser-post profile.
- 8.5 It writes the entire SAML request message as a hidden variable into the invisible HTML form created earlier.
- 8.6 It retrieves the encrypted RP URL from the appropriate cookie and inserts it together with the encrypted version of the RSTR into the invisible form as another hidden form variable.
- 8.7 It writes the URL of the Liberty-enabled IdP into the action attribute of the invisible form.
- 8.8 It auto-submits the HTML form (transparently to the user), using the JavaScript method *click()* on the *submit* tag.

In step 12, the plug-in operates as follows.

- 12.1 It recovers the encrypted string from the received IdP response and decrypts it using its internally stored secret key. The SIIP-issued RSTR and the RP URL are then recovered from the decrypted data.
- 12.2 It generates a SAML token containing a unique ID, fresh nonce and current time-stamp; the token is referred to here as the *encapsulating token*. The plug-in embeds the signed SIIP-issued RSTR retrieved in the previous step and the signed Liberty IdP-issued SAML token (after retrieving it from the appropriate HTML hidden variable) into the (unsigned) encapsulating token.
- 12.3 It displays a summary of the token contents to the user and requests consent to proceed. The summary indicates the types of attributes the encapsulating token is carrying, as well as the RP URL to which the token will be forwarded. The JavaScript *confirm()* pop-up box is used to achieve this.
- 12.4 If the user agrees to submission of the token, it:
- (a) inserts the RP URL (retrieved in step 12.1) into the action attribute of the HTML form carrying the encapsulating SAML token; and
 - (b) submits the token to the RP seamlessly using the JavaScript *click()* method.

The prototype has been successfully tested with experimental websites (acting as a Liberty-enabled IdP and a CardSpace-enabled RP) as well as with the current (unmodified) CardSpace and Higgins identity selectors.

6.4.4 Potential Features and Issues

The potential advantages and limitations described in the general interoperation model (see sections 5.4 and 5.6) are also applicable here. In particular, the scheme described here mitigates the risk of a fake IdP attack, e.g. as resulting from a phishing attack (see section 5.4.1).

Some older browsers (or browsers with scripting disabled) may not be able to run the integration plug-in, as it was built using JavaScript. However, most modern browsers support JavaScript (or ECMAScript), and hence building the prototype in JavaScript is not a major usability obstacle.

6.5 Related Work

The scheme described in this chapter has some similarities to a previous proposal for CardSpace-Liberty integration [22], referred to below as the *AM scheme*. Whilst both approaches concentrate on supporting integration at the client rather than at the server, there are a number of important differences. We next describe some of the most significant of these.

Instead of focusing on CardSpace users only, as is the case with the scheme described here, the AM scheme allows for interoperability in the case where the RP is Liberty-enabled and the IdP is CardSpace-enabled as well as vice versa. However, since, unlike the scheme described here, no prototype of the system has been described, issues which might arise during deployment have not been explored.

One important goal for any identity management system is ease of use. However, user interface issues, notably the operation of the integration software on the client platform, have not been explored for the AM scheme, whereas the scheme described in this chapter addresses this through a combination of a browser extension and the CardSpace identity selector interface. Also, again unlike the scheme described here, the relationship between the integration software and the web browser has not been specified for the AM scheme.

Yet again unlike the scheme described here, the means by which the integration software is triggered is also not clear for the AM scheme. For example, if the integration software is assumed to run at all times, then problems arise if the user wishes to use CardSpace or Liberty in the normal way.

The AM scheme only supports integration of CardSpace with one Liberty profile type, namely the Liberty-enabled client, and this is likely to restrict its applicability.

Finally, the AM scheme does not address how to handle the PPID, described in section 4.3.11, when supporting interoperation between RPs and Liberty-enabled IdPs. Additionally, it is not clear whether performing the task of IdP discovery is the responsibility of the RP, the integration software, or the user.

Another scheme supporting interoperation between CardSpace and Liberty has been proposed by Jørstad et al. [144]. In this scheme, the IdP is responsible for supporting interoperation. The IdP must therefore perform the potentially onerous task of maintaining two different identity management schemes. In addition, this scheme requires the user to possess a mobile phone supporting the Short Message Service (SMS). Moreover, the IdP must always perform the same user authentication technique, regardless of the identity management system the user is attempting to use. The IdP simply sends an SMS to the user, and, in order to be authenticated, the user must confirm receipt of the SMS. This confirmation is also an implicit user approval for the IdP to send a security token to the RP. By contrast, the scheme described in this chapter does not require use of a handheld device, and does not enforce a specific authentication method.

Concordia, a Kantara member¹³, proposed an interoperation scheme between CardSpace and SAML/WS-Federation. We next outline a variant that is similar to the scheme described in this chapter.

Following a user visit, a SAML-conformant RP (e.g. a Liberty-enabled RP [21]) generates a SAML authentication request¹⁴, and redirects the UA to a Concordia-specific entity, referred to below as the *interpreter*. The interpreter must act as a SAML-enabled IdP and as an Information Card-

¹³<http://kantarainitiative.org/confluence/display/concordia/Home>

¹⁴The SAML-enabled RP can specify a specific authentication method which must be employed when authenticating the user.

enabled RP. The SAML-enabled IdP, a subcomponent at the interpreter entity, converts the requirements of the SAML-enabled RP into the corresponding CardSpace-compatible claim type(s) in order to enable a suitable InfoCard to be selected. If a managed card is selected, the selector will send an RST to the Information Card-enabled IdP, which, given a successful user authentication, responds with an RSTR. The selector then sends the RSTR to the CardSpace-enabled RP, a subcomponent at the interpreter entity. The SAML-enabled IdP, a subcomponent at the interpreter entity, generates a SAML (authentication) response, extracted from the RSTR; this IdP then redirects the UA back to the SAML-enabled RP with the SAML response. If satisfied, the RP grants the user access, with the appropriate privileges.

By contrast, the scheme described here does not require use of a server-hosted interpreter entity, which must clearly be a TTP.

6.6 Conclusions and Future Work

In this chapter we have described a Liberty-based instantiation of the interoperation model given in chapter 5. CardSpace users (indeed, users of any Information Card system) are able to obtain a security token from a Liberty-enabled identity provider that satisfies the security requirements of a CardSpace-enabled relying party. The scheme uses a browser extension, and requires no major changes to servers. It uses the selector interface to enable interoperation between Liberty identity providers and CardSpace relying parties. The scheme extends the use of personal cards to support this process.

The scheme takes advantage of the similarity between the Liberty ID-FF and the CardSpace frameworks, and this should help to reduce the effort required for full system integration. Also, implementation of the scheme does not require technical co-operation between Microsoft and Liberty.

Possible future work includes exploring the possibility of building a sim-

ilar, client-based scheme to support interoperation between a CardSpace-enabled identity provider and a Liberty-enabled service provider.

Enabling Interoperation Between Shibboleth and Information Card Systems

7.1 Introduction

In this chapter we describe a second instantiation of the integration model given in chapter 5; it enables interoperation between an Information Card system and Shibboleth. Information Card users are able to obtain a security token from a Shibboleth-enabled IdP that is made usable by an Information Card-enabled RP. Much of the material in this chapter has been previously published [11, 16].

For simplicity of presentation, in this chapter we assume that the Information Card system is CardSpace, although an identical approach will work with other Information Card systems such as Higgins.

As discussed in chapter 5, the wide adoption of Shibboleth (notably by educational institutions) and the inclusion of CardSpace in recent versions of Windows means that enabling interoperation between the two systems could offer significant benefits. CardSpace-Shibboleth interoperation is also attractive since both schemes support:

- user authentication;
- the exchange of user attributes; and

- SAML tokens.

The remainder of the chapter is organised as follows. Section 7.2 details the interoperation process, and, in section 7.3, we discuss implementation issues. In section 7.4 we describe a prototype realisation, and section 7.5 highlights possible areas for related work. Finally, section 7.6 concludes the chapter.

7.2 Interoperating with Shibboleth

We now describe how interoperation with Shibboleth is achieved. As in chapter 5, the entities involved are: a CardSpace-enabled RP, a CardSpace-enabled UA (e.g. a suitable web browser), a Shibboleth-enabled IdP, and the integration software (the adaptor), which in this case we suppose takes the form of a browser extension installed on the user platform.

7.2.1 Requirements

The scheme described in this chapter has the same operational requirements as those listed in section 5.2.3 (excluding requirement 4), where the IdP is a Shibboleth-enabled IdP.

In addition, in order to enable IdP-discovery, the adaptor must be able to operate a WAYF-like component and should offer the user the option to store their choice of IdP for future logins.

The RP must be prepared to accept SAML tokens in the format constructed by the adaptor. Furthermore, the IdP-issued security token must conform to SAML 1.1 (since CardSpace supports this format); however, note that this requirement is trivially easy to meet since, as stated in section 4.8.1, an IdP conforming to Shibboleth 1.3 normally generates tokens in SAML 1.1 format, and an IdP conforming to Shibboleth 2.0 can generate tokens conforming to either SAML 1.1 or SAML 2.0.

7.2.2 Operation

Fig. 7.1 gives an overview of the operation of the scheme, with the step numbers shown. The sequence of steps is precisely as given in section 5.2.4. We specify below only those steps where Shibboleth-specific operations are performed.

3. In this case steps 3d and 3e are null.

8. Selector \rightarrow Adaptor/UA \rightarrow IdP. Unlike in the standard case, the SIIP-issued RSTR is intercepted by the adaptor, which temporarily stores it. The adaptor then performs the following steps.
 - a) It asks the user whether use of Shibboleth-based authentication is required. If so, it proceeds; otherwise it terminates, allowing CardSpace to operate normally. The adaptor could offer the user the option to store their answer for subsequent logins at this RP.
 - b) It displays a WAYF-like component to allow the user to select an appropriate IdP. The adaptor could offer the user the option to store their selection for subsequent logins at this RP.
 - c) It constructs a SAML authentication request, and forwards it to the user-selected IdP. Note that this request will also indicate the RP-requested user attributes (if any) which are to be asserted by the IdP. The adaptor will know what they are since they were stored by it earlier.

10. IdP \rightarrow UA. Following a successful user authentication in the previous step, the IdP generates and returns to the UA a digitally-signed SAML token, containing an authentication statement and, possibly, an attribute statement.

11. This step is null.

12. Adaptor/UA \rightarrow RP. The adaptor generates an unsigned SAML token that contains both the digitally-signed SIIP-issued RSTR as well as the digitally-signed (Shibboleth) IdP-issued token. The UA then forwards the adaptor-generated SAML token to the RP, optionally after first obtaining permission from the user.
13. RP \rightarrow UA. The RP verifies the received SAML token (including verifying the RSTR signature, PPID, the Shibboleth signature, nonces, timestamps, etc.), and, if satisfied, grants access.

If an attribute assertion was requested by the CardSpace-enabled RP, then the RP could, in step 13, compare the (locally) SIIP-asserted attributes with the (remotely) Shibboleth-asserted attributes. Such a procedure potentially gives the RP added guarantees about the validity of these attributes.

7.3 Implementation Issues

We now consider implementation issues.

7.3.1 Token Storage and Forwarding

The means by which the security token is forwarded to the CardSpace-enabled RP and how/where the RSTR token is stored should be chosen carefully. We propose the use of similar techniques to those used for the CardSpace-Liberty scheme, described in section 6.3.3.

In the scheme described here, the adaptor stores the RP address as well as the RSTR in encrypted form in a cookie (or cookies) as part of step 3, so that the adaptor is able to retrieve them in step 12.

As part of step 8, the adaptor retrieves the encrypted value from the cookie and sends it to the IdP as a hidden variable in an HTML form or as a query URL parameter. At the same time, the adaptor also encrypts the intercepted RSTR and sends it to the IdP as a hidden form variable. As part of step 10, the IdP must then return the RP address and the RSTR unchanged.

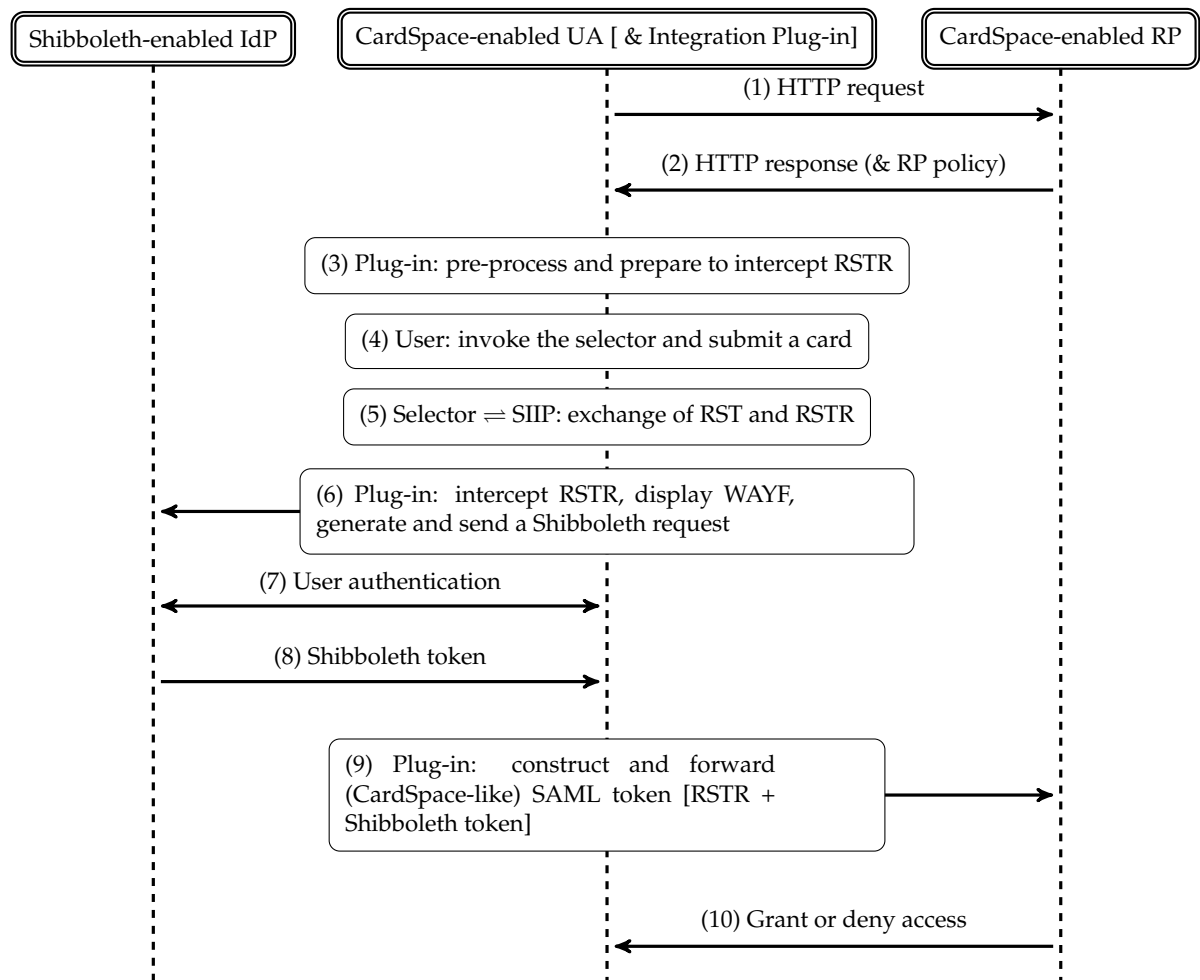


Figure 7.1: Protocol Exchanges

As part of step 12, the adaptor retrieves and decrypts the (enciphered) RP address and RSTR values.

Like the CardSpace-Liberty interoperation scheme (see section 6.3.3), the IdP is unable to read the RP address or the RSTR, since they are encrypted using a key known only to the browser extension, hence enhancing user privacy. If the IdP, however, needs the RP address for auditing purposes (e.g. for legal reasons), or the IdP policy requires the disclosure of the RP identity (e.g. so that it can encrypt the security token using the RP's public key), then the RP address could be sent to the IdP.

Finally, we observe that the Shibboleth specification allows the RP to use a hidden form variable called *RelayState* to maintain state in an RP-UA-

IdP session. A Shibboleth-compliant RP could insert data into this variable and the IdP must return this data intact in the same hidden form variable. We propose to use this RelayState variable in our scheme to contain the encrypted versions of the RP address and the RSTR.

7.3.2 Attribute Handling

CardSpace and Shibboleth use two different sets of attribute types¹; this clearly causes a problem in creating a SAML (attribute) request message for a Shibboleth-enabled IdP from a policy statement provided by a CardSpace-conformant RP. We adopt the two approaches discussed in section 5.3.2 to deal with the problem. For the first approach, an example mapping is given in Table 7.1.

Table 7.1: CardSpace-Shibboleth Attribute Mapping

CardSpace personal cards	Shibboleth
givenname	givenName
surname	sn
givenname + surname	cn
emailaddress	mail

7.3.3 Possible Extensions

Although the integration scheme is presented as Shibboleth-specific, we suspect that a modified version of the scheme could also be applied to other SAML-compliant IdPs. Given that SAML 2.0 builds on SAML 1.1, Liberty ID-FF 1.2 and Shibboleth 1.3, a mapping seems likely to be possible. Reconfiguring the integration scheme to interoperate with any SAML-aware IdP could potentially significantly increase its applicability (see also section 6.3.4); such a reconfiguration remains possible future work. Note that the possible extensions discussed in section 5.7 are also applicable here.

¹As stated in section 4.3.3.1, CardSpace personal cards currently only support fourteen editable attributes, whereas Shibboleth supports many more.

7.4 Prototype Realisation

We next give details of a prototype implementation of the scheme which operates with the Shibboleth browser-post profile.

7.4.1 Implementation Details

As in the CardSpace-Liberty prototype described in section 6.4, the prototype described here is coded as a JavaScript browser plug-in, executed using a C#-driven BHO (see section 6.4.2). It can readily be enabled or disabled using the add-on manager in the Internet Explorer Tools menu. The integration plug-in does not require any changes to default Internet Explorer security settings, thereby avoiding potential vulnerabilities arising from such changes. The scheme operates with both the CardSpace and the Higgins identity selectors without any modification (see section 6.4.2).

7.4.2 Prototype Operation

We now consider operational aspects of the prototype. Prior to use, the user must have accounts with a CardSpace-enabled RP and a Shibboleth-enabled IdP. We refer throughout to the numbered protocol steps given in section 7.2.2.

The implementation of step 3 is precisely the same as that of step 3 of the CardSpace-Liberty integration prototype described in section 6.4.3, except that step 3.1 (e) is excluded, and step 3.1 (d) is replaced with the following.

- 3.1 (d) The plug-in retrieves the *requiredClaims* and *optionalClaims* tags from the *param* tags. It obtains and temporarily stores in a cookie the mandatory and optional claim types listed in these tags.

In step 8 the plug-in uses the DOM (see section 2.5.2) to perform the following steps.

- 8.1 It intercepts the RSTR sent by the selector.

- 8.2 Using a JavaScript pop-up box, it asks the user whether use of the interoperation scheme is required. If so, it proceeds; otherwise it terminates, giving CardSpace the opportunity to operate normally. On proceeding, the plug-in offers to store the user's answer using a plug-in-embedded checkbox; if checked, the plug-in stores the user answer in a persistent cookie (see section 2.3.4.1).
- 8.3 It encrypts the RSTR using AES in CBC mode (see section 2.4.2.1) with a secret key known only to the plug-in.
- 8.4 It prompts the user to select an IdP using a WAYF-like component, implemented as a plug-in-embedded HTML form containing a drop-down list.
- 8.5 It offers to store the user's choice of IdP using a plug-in-embedded checkbox; if checked, the plug-in stores the user selection in a persistent cookie.
- 8.6 It constructs a SAML 1.1 request which conforms to Shibboleth standards. Note that this request will also indicate the RP-requested user attributes (if any) that are to be asserted by the IdP. The plug-in will know what they are since they were stored by it earlier.

The plug-in performs a mapping between the CardSpace-enabled RP-requested user attributes and the Shibboleth-supported attributes as in Table 7.1. The mapping is performed using JavaScript regular expressions, specifically using the *match* method with its global (g) and case-insensitive (i) parameters.

- 8.7 It writes the entire (base 64-encoded) SAML request message as a hidden variable (SAMLRequest) into the invisible HTML form created earlier.

8.8 It retrieves the encrypted URL of the RP from the appropriate cookie and inserts it, together with the encrypted version of the RSTR, into the invisible form as the hidden form variable RelayState.

8.9 It writes the URL of the IdP into the action attribute of the form.

8.10 It auto-submits the HTML form (transparently to the user), using the JavaScript method *click()* on the submit tag, thus redirecting the user to the IdP.

The implementation of step 12 is precisely the same as that of step 12 of the CardSpace-Liberty integration prototype described in section 6.4.3.

The prototype has been successfully tested with experimental websites acting as the Shibboleth-enabled IdP and the CardSpace-enabled RP, as well as with the current (unmodified) CardSpace and Higgins identity selectors.

7.4.3 Potential Features and Issues

The potential advantages and limitations described in the integration model chapter (see sections 5.4 and 5.6) are also applicable here, as are the issues and features listed in section 6.4.4.

7.5 Related Work

In 2007, Internet2 announced² plans to develop extensions to Shibboleth to support CardSpace. This included collaboration with Microsoft in order to add Information Card support to Shibboleth. However, unlike the scheme described in this chapter, such work does not seem to be based on a browser extension running on the user platform. Instead, it appears³ that the inter-operation functionality is performed by Shibboleth IdPs and RPs, which is likely to require significant changes to the servers.

²<https://lists.internet2.edu/sympa/arc/i2-news/2007-05/msg00009.html>

³<https://lists.internet2.edu/sympa/arc/shibboleth-dev/2007-05/msg00021.html>

7.6 Conclusions and Future Work

In this chapter we have described a Shibboleth-based instantiation of the interoperation model given in chapter 5. CardSpace users (indeed, users of any Information Card system) are able to obtain a security token from a Shibboleth-enabled identity provider that can be processed by a CardSpace-enabled relying party. The scheme uses a browser extension, requires no major changes to identity providers and relying parties, and does not require any changes to the deployed CardSpace identity selector.

The interoperation scheme takes advantage of the similarity between the Shibboleth and the CardSpace frameworks, and this should help to reduce the effort required for full system integration. Enabling interoperation between CardSpace and Shibboleth may be attractive since both schemes support user authentication as well as the exchange of user attributes. In addition, both schemes support SAML tokens. Moreover, implementation of the scheme does not require technical co-operation between Microsoft and Internet2.

Future work may explore the possibility of building a similar, client-based scheme to support interoperation between a CardSpace-enabled identity provider and a Shibboleth-enabled service provider.

Client-based Interoperation Between OpenID and Information Card Systems

8.1 Introduction

In this chapter we describe a third instantiation of the interoperation model given in chapter 5; it enables interoperation between an Information Card system and OpenID. Information Card users are able to obtain a security token from an OpenID-enabled IdP, the contents of which can be processed by an Information Card-enabled RP. The scheme uses a browser extension, is transparent to OpenID IdPs and to identity selectors, and only requires minor changes to the operation of an Information Card-enabled RP. Much of the material in this chapter has been published [12].

As in chapters 6 and 7, for simplicity of presentation we assume that the Information Card system is CardSpace, although an identical approach will work with other Information Card systems such as Higgins.

As discussed in chapter 5, the wide adoption of OpenID (see section 4.5.1) and the inclusion of CardSpace in recent versions of Windows means that enabling interoperation between the two systems could offer significant benefits. CardSpace-OpenID interoperation is also attractive since both schemes support user authentication as well as the exchange of user attributes.

The remainder of the chapter is organised as follows. Section 8.2 details the interoperation process. In section 8.3 an operational analysis is provided

and, in section 8.4, we describe a prototype implementation. Section 8.5 reviews related work, and, finally, section 8.6 concludes the chapter.

8.2 Interoperating with OpenID

We now describe how interoperation with OpenID is achieved. As in chapter 5, the entities involved are: a CardSpace-enabled UA (e.g. a suitable web browser such as Internet Explorer), a CardSpace-enabled RP, an OpenID-enabled IdP, and the integration software (the adaptor), which in this case we suppose takes the form of a browser extension installed on the user platform.

8.2.1 Requirements

The scheme described here has the same operational requirements as those listed in section 5.2.3 (excluding requirements 5 and 7), where the IdP is an OpenID-enabled IdP and the InterCard we refer to below as an *IDcard*.

8.2.2 Operation

Fig. 8.1 gives an overview of the operation of the scheme. The sequence of steps is precisely as given in section 5.2.4. We specify below only those steps where OpenID-specific operations are performed (observing that, as noted above, we use the term *IDcard* for the OpenID-specific InterCard).

8. Selector → Adaptor/UA → IdP. Following the user selection of a suitable *IDcard*, the SIIP-issued RSTR is intercepted by the adaptor, which temporarily stores the RSTR. The adaptor then performs one of the following steps, depending on whether or not SSL/TLS is used to protect the UA-RP channel.
 - If the RP uses HTTP, the adaptor uses the contents of the RSTR to construct an OpenID authentication request, which it forwards to the appropriate IdP, having discovered its address from the RSTR.

Note that this request will indicate which RP-requested user attributes (if any) are to be asserted by the IdP. The adaptor will know what they are since they were stored by it earlier.

- If the RP uses HTTPS, the adaptor performs the following tasks.
 - a) It asks the user to discover whether the use of the integration scheme is required. If it is, it proceeds; otherwise it terminates. The adaptor could also offer the user the option to store their answer for future interactions with this RP.
 - b) It asks the user to enter their OpenID identifier. The adaptor can then use this identifier to perform IdP discovery (see section 4.5.2). If this fails (e.g. because of network failure, unpaid hosting costs, etc.), the adaptor can ask the user to enter the URL of the IdP (other possibilities exist — see section 8.3.1). The adaptor could also offer the user the option to store their selection for subsequent logins at this RP.
 - c) It constructs an OpenID authentication request (precisely as in the HTTP case), which it forwards to the discovered IdP.

The following details of step 8 apply regardless of whether HTTP or HTTPS is in use.

- The format of the IdP authentication request is dependent on the version of OpenID being used (see the discussion below), which the adaptor will know from the IdP discovery process.
- The OpenID authentication request includes a designated return-page, so that the IdP will know to which URL the authentication response must be returned. This return-page is computed by the adaptor, which sets the URL of the visited RP page to be the value of the designated return-page.

- The more commonly used `checkid_setup` mode is adopted; the `checkid_immediate` mode is not supported as it requires direct, back-channel RP-IdP communication without any user interaction.
9. IdP \rightleftharpoons User: User Authentication + OpenID Token Generation. If necessary, the IdP authenticates the user. If successful, the IdP requests permission to send the MAC-protected OpenID assertion token (see step 7 of the OpenID protocol given in section 4.5.8) to the designated RP return-page (see step 8 above).
 10. IdP \rightarrow UA \rightarrow RP: OpenID Token. The IdP redirects the UA back to the RP return-page with a positive or negative OpenID authentication response, depending on whether or not the user granted permission in step 9. The RP will receive the IdP-issued token unchanged (embedded in the URL); however, the RP will ignore it because a CardSpace-enabled RP will not be equipped with the means to process such a token. Note that this should not change the appearance of the RP page as displayed by the UA (see sections 2.5.3.7 and 2.5.3.8).
 11. Adaptor \rightleftharpoons IdP: Token Verification. The adaptor verifies the MAC-protected OpenID authentication response by interacting with the IdP using the `check_authentication` mode via a TLS/SSL channel. If the verification succeeds, it moves to the next step (step 12); if it fails, the adaptor informs the user and terminates.
 12. Adaptor/UA \rightarrow RP: Token Forwarding (conditional — see previous step). The adaptor constructs a CardSpace-compatible SAML token, and forwards it to the RP. This encapsulating SAML token includes the IdP-provided user attributes and the digitally-signed SIIP-issued RSTR which contains the PPID (see also sections 8.3.2 and 8.3.3).

13. RP → UA: Grant/Deny Access. The RP verifies the SAML token (including verifying the RSTR signature, PPID, nonce, time-stamps, etc.), and, if satisfied, grants access.

The detailed operation of steps 8 and 11 is dependent on the OpenID version in use. The authentication request name-space field (`openid.ns`) must be set to `http://specs.openid.net/auth/2.0` for OpenID 2.0, and one of `absent`, `http://openid.net/signon/1.1`, or `http://openid.net/signon/1.0` for OpenID 1.1. Similarly, the field `openid.ns.sreg` = `http://openid.net/extensions/sreg/1.1` [116] is added to the authentication request when requesting identity attributes using the SREG extension in OpenID 2.0.

8.3 Discussion and Analysis

8.3.1 IDcard Contents

A native OpenID authentication request to an OpenID-enabled IdP typically includes the user-supplied OpenID identifier, an RP return-page to which the IdP must send the authentication response, and a list of requested attributes. The RP must, of course, also know the IdP address. In the protocol described in section 8.2.2 the user's OpenID identifier and the URL of the IdP are specified in the IDcard¹ in the case where the RP uses HTTP². The following alternative approaches avoid the need to store this data in the IDcard.

- The adaptor (implemented as a browser extension) could prompt the user to enter the OpenID identifier that they wish to use, after they have submitted an IDcard, e.g. as part of step 8 in section 8.2.2. This approach could be inconvenient, since the user would have to enter

¹The RP return-page is computed by the browser extension itself.

²As specified in section 8.2.2, if the RP uses HTTPS, then the browser extension uses the user's OpenID identifier to perform IdP discovery; if this fails, the extension prompts the user to enter the URL of the IdP.

8. CLIENT-BASED INTEROPERATION BETWEEN OPENID AND INFORMATION CARD SYSTEMS

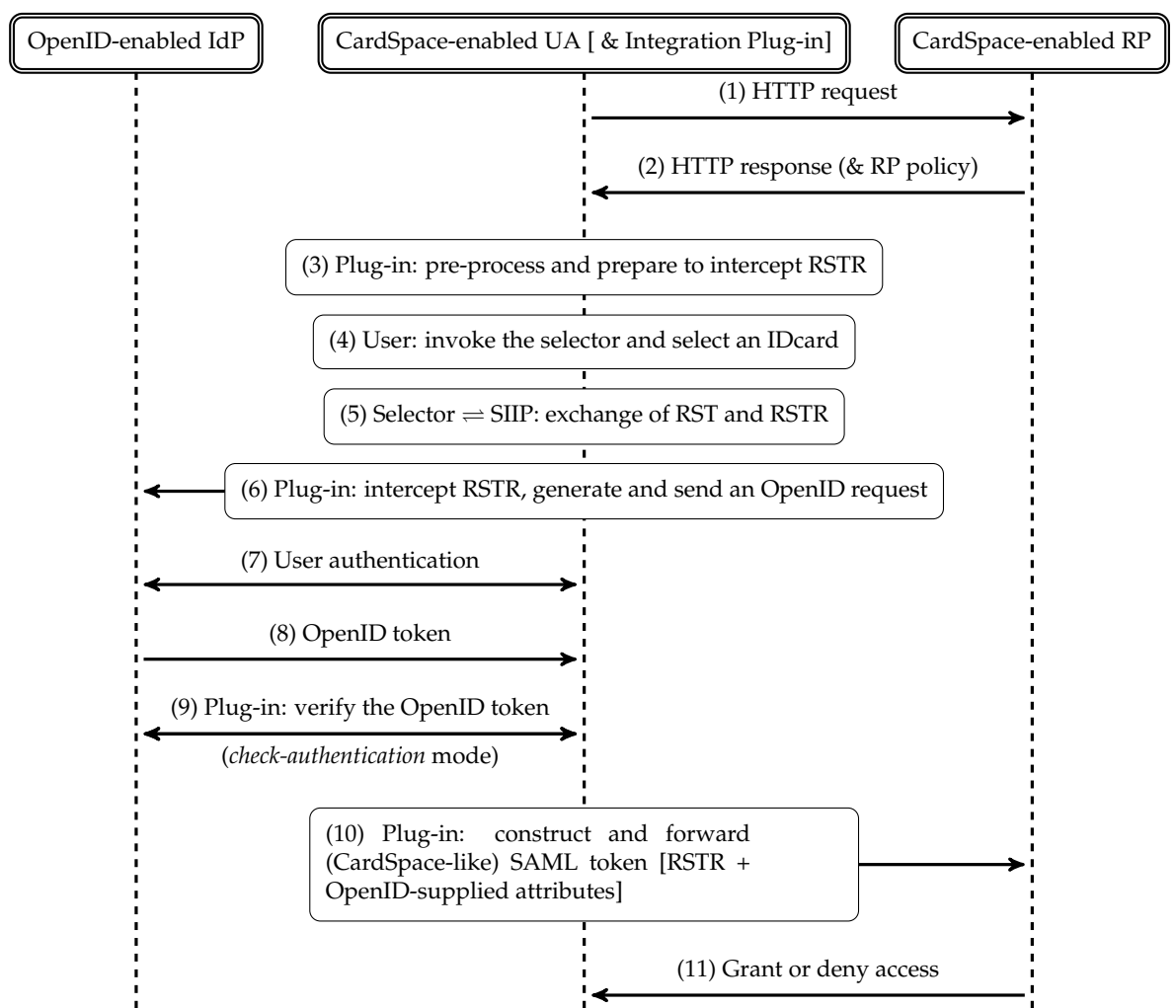


Figure 8.1: Protocol Exchanges

the identifier every time; however, the impact could be significantly reduced if the user entry is stored for subsequent logins.

- The browser extension could maintain a list of the service URLs of widely used IdPs, enabling it to deduce which IdP it needs to contact from the user's OpenID identifier. This would potentially maximise user transparency, but could give rise to storage issues and operational problems, e.g. in the case where an IdP is not in the list. The latter issue could be addressed by prompting the user to enter the URL of an unknown IdP, which the browser extension could then add to its internal list for future use.

- The browser extension could discover the IdP from the user-supplied OpenID identifier, e.g. by fetching an HTML document from the URL associated with this identifier. However, such an approach is vulnerable to phishing attacks and requires extra round trips.

8.3.2 IdP User Authentication

The SAML token created by the browser extension in step 12 of section 8.2.2 could be extended to contain an additional field to indicate that the user has been authenticated by a specified IdP, as well as when and how. Of course, the RP would need to be modified to be able to process such an extra field, although this is likely to be relatively straightforward.

This authentication statement could also include the original token generated by the IdP. Since this is a MAC-protected token, verifying it would give the RP added guarantees about the user authenticity. If such an approach is implemented, then the extension must skip the token verification process and send the token unchanged³ to the RP, since an OpenID-enabled IdP will only verify a token once (see section 4.5.8).

8.3.3 Security Considerations

The unsigned encapsulating token generated by the browser extension in step 12 of section 8.2.2 includes the IdP-supplied user attributes, the signed SIIP-issued RSTR containing the PPID, and (optionally) the MAC-protected IdP-issued token. The RP compares the SIIP-asserted PPID (and the public key) in the encapsulating token with its stored values and verifies the digital signature (see section 4.3.11). The RP can thus authenticate the user, link the user to his/her account, and consume the IdP-supplied attributes, e.g. for authorisation purposes.

³The OpenID token should be sent in an authentication statement contained within the SAML token, to allow RPs to choose whether or not to process it.

If the RSTR also contains self-issued attributes, the RP could compare them with the IdP-provided attributes; such a procedure could give the RP added guarantees about the accuracy of these attributes.

In addition, an RP could optionally also verify the MAC in the IdP-issued token, which can be embedded unchanged in the encapsulating token. However, for the RP to be able to verify the MAC, the extension must skip the verification process (see section 8.3.2) and the RP must initiate on-line interaction with the IdP using the `check_authentication` mode.

Unless it has compromised the user platform, a malicious entity will be unable to create an acceptable encapsulating token to masquerade as a legitimate party since it will not have access to three key token components:

- the PPID;
- the SIIP-signed RSTR, which is only issued if the appropriate InfoCard is selected on the correct user platform; and
- the MAC-protected OpenID token, which is only issued if the genuine user has been authenticated by the OpenID-enabled IdP. As stated previously, this token can be sent to the RP unchanged.

In addition, nonces and time-stamps are used to prevent replay attacks, and RPs can also employ IP address validation. As stated in section 4.5, the use of SSL/TLS is strongly recommended when using OpenID.

Note that, in protocol step 4, the selector identifies the RP to the user and indicates whether or not they have visited that particular RP before; if the user is visiting this RP for the first time, CardSpace requests the user's permission to proceed⁴ (see section 4.3.9.1). This helps to increase security, since the user and the RP are both identified to each other.

The scheme also strengthens OpenID against fake IdP attacks, e.g. as resulting from a phishing attack (see section 5.4.1). This is because the redirect

⁴Note that this gives a security advantage by comparison with *native* OpenID, which does not identify the RP to the user.

to the IdP is initiated by the browser extension and not by the RP; i.e. the RP cannot redirect the user to an IdP of its choosing. By contrast, in normal operation of OpenID a malicious RP could redirect a user to a fake IdP, which might capture the user credentials (see section 4.5.11.1).

Finally note that the scheme allows the user attributes to be remotely stored at the IdP; this has potential security advantages over storing the attributes locally on the user platform, as is currently the case with CardSpace SIIP-issued attributes.

8.3.4 Attribute Mapping

As stated in section 4.3.3.1, CardSpace personal cards support fourteen editable attributes, whereas the OpenID SREG extension only supports nine attribute types. The prototype described in section 8.4 uses the mapping in Table 8.1 to convert between attribute types.

Table 8.1: CardSpace-OpenID Attribute Mapping

CardSpace personal cards	OpenID SREG extension
givenname	nickname
surname	fullname
emailaddress	email
dateofbirth	dob
gender	gender
postalcode	postcode
country	country

The OpenID SREG extension also supports *language* and *timezone* attributes, which have no corresponding attribute types in CardSpace personal cards.

The protocol specified in section 8.2.2 could also be used to support the transfer of arbitrary data between IdPs and RPs using the (OpenID) AX extension (see section 4.5.9.2); however, this has not yet been prototyped.

8.4 Prototype Realisation

We next give details of a prototype implementation of the scheme. The description applies to both OpenID 1.1 and OpenID 2.0. The prototype uses the OpenID `checkid_setup` mode, operating with the SREG extension.

As in the CardSpace-Liberty and CardSpace-Shibboleth prototypes (described in sections 6.4 and 7.4, respectively), the prototype described here is coded as a JavaScript browser plug-in, executed using a C#-driven BHO (see section 6.4.2). It can readily be enabled or disabled using the add-on manager in the Internet Explorer Tools menu. The integration plug-in does not require any changes to default Internet Explorer security settings, thereby avoiding potential vulnerabilities arising from such changes. The scheme operates with both the CardSpace and the Higgins identity selectors without any modification (see section 6.4.2).

8.4.1 User Registration

Prior to use, the user must have accounts with a CardSpace-enabled RP and an OpenID-enabled IdP. The user must also create an IDcard for the relevant IdP. This involves invoking the selector and inserting the user's OpenID identifier at the target IdP in the *web page* field, the URL of the IdP in the *street* field, and the trigger word *OpenID1.1* or *OpenID2.0* in the *city* field. For ease of identification, the user can give the personal card a meaningful name, e.g. of the target IdP site. The user can also upload an image for the card, e.g. containing the logo of the intended IdP or just of OpenID. When a user wishes to use a particular IdP, the user simply chooses the corresponding IDcard.

8.4.2 Prototype Operation

In this section we consider specific operational aspects of the prototype. We refer throughout to the numbered protocol steps given in section 8.2.2.

The implementation of step 3 is precisely the same as that of step 3 of the CardSpace-Shibboleth integration prototype described in section 7.4.2, except that steps 3.1 (e) and 3.3 are replaced as specified below, and step 3.5 is skipped.

- 3.1 (e) If necessary, and after keeping track of the original policy settings, the plug-in modifies the RP policy so that the *city*, *street* and *web page* claim types are specified in the *requiredClaims* tag.
- 3.3 The plug-in obtains the action attribute of the CardSpace HTML form and stores it in a cookie. This attribute specifies the URL of a web page at the CardSpace RP to which the security token must be forwarded for processing. If the obtained attribute is not a fully qualified domain name address, the JavaScript inherent properties, i.e. *document.location.protocol* and/or *document.location.host*, are used to help reconstruct the full URL.

In step 8 the plug-in uses the DOM (see section 2.5.2) to perform the following steps.

- 8.1 It intercepts the RSTR sent by the selector.
- 8.2 It uses the JavaScript property *document.location.protocol* to determine whether HTTP or HTTPS is in use. As stated previously, it operates slightly differently in these two cases.
 - If HTTP is being used, the plug-in parses the RSTR and extracts the *city*, *web page* and *street* fields. If the *city* field contains the word *OpenID1.1* or *OpenID2.0*, the plug-in proceeds; if not, it terminates. It then reads the *web page* field to discover the user's OpenID identifier, and obtains the URL of the IdP from the *street* field.
 - If HTTPS is being used, the plug-in uses a JavaScript pop-up box to ask the user whether use of the integration scheme is required.

If so, it proceeds; otherwise it terminates. On proceeding, the plug-in uses a JavaScript pop-up box to prompt the user to enter their OpenID identifier, the URL of the IdP, and the version of OpenID to be used (i.e. 1.1 or 2.0). The plug-in uses a plug-in-embedded checkbox to offer the user the option to store these values in a persistent cookie (see section 2.3.4.1) for future logins at this RP.

8.3 It constructs an OpenID authentication request, compatible with the OpenID version in use (see step 8.2 above). The plug-in defaults to creating an OpenID 1.1-compatible authentication request if no version is specified. This involves generating a nonce and time-stamp, and also determining the required and optional attribute types to be sent to the IdP. The plug-in retrieves all the CardSpace-supported claims it stored earlier. It then maps between their types and the SREG-supported attribute types, using Table 8.1. As in the CardSpace-Shibboleth prototype (see section 7.4.2), the mapping is performed using JavaScript regular expressions.

As stated in section 8.2.2, the plug-in uses the OpenID `checkid_setup` mode, and skips the optional initiation phase in which the RP and IdP exchange a shared secret. It also sets the return page (to which the IdP sends the authentication response) to equal the currently-visited RP page.

8.4 It redirects the user to the IdP along with the OpenID authentication request, using the JavaScript inherent property `window.location`.

In step 11 the plug-in performs the following steps.

11.1 It parses the (OpenID) IdP-issued authentication response that is embedded in the URL.

11.2 It verifies the authentication response, including verifying that the return URL (`openid.return_to`) is the same as the current page, checking the nonce and time-stamp, and requesting the IdP to verify the IdP-provided MAC on the authentication assertion.

The plug-in uses the OpenID `check_authentication` mode, so the MAC verification is performed by the IdP in the following way; it issues an HTTPS request to the IdP with exact copies of all fields from the authentication response except for the `openid.mode` field whose value the plug-in changes from `id_res` to `check_authentication`. The IdP responds with a boolean value (true or false). If all of the checks succeed, the plug-in continues to the next step; otherwise it terminates, informing the user that the process can no longer continue.

In step 12 the plug-in performs the following steps.

12.1 It constructs a CardSpace-compatible encapsulating token, inserting the user attributes received from the IdP into the token. It also embeds the signed SIIP-issued RSTR into this token.

12.2 It creates and appends an invisible HTML form, (with the method attribute set to POST), to the current page.

12.3 It writes the encapsulating token as a hidden variable into the HTML form, with the name attribute of this variable set to the name of the CardSpace object tag.

12.4 It writes the end-point URL of the RP into the action attribute of the invisible form.

12.5 Finally, it auto-submits the HTML form to the RP (transparently to the user), using the JavaScript inherent method `submit`.

The prototype has been successfully tested with the *MyOpenID*⁵ IdP and with an experimental implementation of a CardSpace-enabled RP, as well as with the current (unmodified) CardSpace and Higgins identity selectors.

8.4.3 Potential Features and Issues

The potential advantages and limitations described in the interoperation model chapter (see sections 5.4 and 5.6) are also applicable here, as are the issues and features listed in section 6.4.4.

8.5 Related Work

Kim et al. [150] have proposed an OpenID authentication method using an identity selector. This scheme is designed to reduce phishing and hacking risks, and also simplify user authentication by automatically performing the OpenID-based login process without the need to manually input the OpenID URL. The scheme uses a specially modified identity selector to enable OpenID authentication, unlike the scheme proposed here which uses an unmodified selector.

Microsoft and OpenID have announced plans⁶ to enable a level of interoperation. A stated aim of this effort is to reduce the risk of phishing in OpenID by enabling an OpenID user to employ CardSpace when authenticating to an IdP. The scheme proposed here inherently provides a level of protection against phishing since the redirect step to the IdP is initiated by the adaptor (and not by the RP), and also supports the use of CardSpace to authenticate to IdPs.

⁵<https://www.myopenid.com/>

⁶<http://www.guardian.co.uk/technology/blog/2007/feb/07/openidgetsab>

8.6 Conclusions and Future Work

In this chapter we have described an OpenID-based instantiation of the interoperation model given in chapter 5. CardSpace users (indeed, users of any Information Card system) are able to obtain a security token from an OpenID identity provider, which, after encapsulation at the client, can be processed by a CardSpace-enabled relying party. The scheme is transparent to OpenID providers and identity selectors, uses a browser extension, and requires only minor changes to a relying party. It uses the identity selector interface and personal cards to enable interoperation.

The integration scheme takes advantage of the similarity between the OpenID and the CardSpace frameworks, and this should help to reduce the effort required for full system integration. Also, implementation of the scheme does not require technical co-operation between Microsoft and the OpenID Foundation.

Possible future work includes exploring the possibility of building a similar, client-based scheme to support interoperability between a CardSpace-enabled identity provider and an OpenID-enabled relying party.

Integrating OAuth with Information Card Systems

9.1 Introduction

In this chapter we describe a fourth and final instantiation of the interoperation model given in chapter 5; it enables interoperation between an Information Card system and OAuth. Information Card users are able to obtain a security token from an OAuth-enabled system, which, after encapsulation on the client platform, can be processed by an Information Card-enabled RP. The scheme uses a browser extension, is transparent to OAuth providers and identity selectors, and only requires minor changes to the operation of an Information Card-enabled RP. Much of the material in this chapter has been published [6, 7].

As in chapters 6 to 8, for simplicity of presentation we assume that the Information Card system is CardSpace, although an identical approach will work with other Information Card systems such as Higgins.

As discussed in chapter 5, we consider CardSpace-OAuth interoperation because of OAuth's fast-growing adoption by widely used Internet service providers, such as Facebook and Twitter. Complementing this, the wide use of Windows, recent versions of which incorporate CardSpace, means that enabling interoperation between the two systems is likely to be of significance for large numbers of identity management users and RPs. CardSpace-OAuth interoperation is also attractive since both the schemes support the

exchange of user attributes.

The remainder of the chapter is organised as follows. Section 9.2 details the interoperation process. In section 9.3 an operational analysis is provided and, in section 9.4, we describe a prototype implementation. Finally, section 9.5 concludes the chapter.

9.2 Interoperating with OAuth

We now describe how interoperation with OAuth is achieved. As in chapter 5, the entities involved are:

- a CardSpace-enabled RP;
- a CardSpace-enabled UA (e.g. a suitable web browser);
- the integration software (the adaptor), which as in previous chapters we suppose takes the form of a browser extension installed on the user platform; and
- an OAuth resource and authorisation server — for simplicity, we assume that the roles of both the resource and the authorisation servers are performed by a single entity, which we refer to throughout as the *OAuth-enabled IdP*.

The adaptor performs the functions of an OAuth client. It obtains an access token from the authorisation server and uses this token to obtain user attributes from the resource server (see section 4.6.1).

9.2.1 Requirements

The scheme described here has the same operational requirements as those listed in section 5.2.3 (excluding requirements 5 and 7), where the IdP is an OAuth-enabled IdP and the InterCard is referred to below as an *OAuthCard*.

In addition, the user must register the RP with the IdP. This involves the user interacting (via the UA) with an HTML registration page hosted by

the IdP, and using this page to send the IdP the RP's name, its URL, and (optionally) its locale. The IdP then issues an identifier for this RP, to be used by the adaptor to identify the RP to the IdP.

9.2.2 Operation

Fig. 9.1 gives an overview of the operation of the scheme. The sequence of steps is precisely as given in section 5.2.4. We specify below only those steps where OAuth-specific operations are performed (observing that, as noted above, we use the term *OAuthCard* for the OAuth-specific InterCard).

8. Selector → Adaptor/UA → IdP. After the user has selected a suitable OAuthCard, the SIIP-issued RSTR is intercepted by the adaptor, which temporarily stores the RSTR. The adaptor then performs one of the following steps, depending on whether or not SSL/TLS is used to protect the UA-RP channel.

- If the RP uses HTTP, the adaptor uses the contents of the RSTR to construct an OAuth request, which it then forwards to the appropriate IdP, having discovered the IdP's URL from the RSTR.
- If the RP uses HTTPS, the adaptor first asks the user whether use of the integration scheme is required. If not, it terminates. If it is, the adaptor prompts the user to enter the URL of the IdP and the IdP-specific identifier for the RP. The adaptor could also offer the user the option to store the supplied values for future interactions with this RP. Precisely as in the HTTP case, the adaptor then constructs an OAuth request and sends it to the URL of the IdP.

Note that, in both cases, the implicit grant type (see section 4.6.3.2) is adopted. Again in both cases the OAuth request includes: the *redirect_uri* parameter, to which the IdP must later redirect the UA; the

scope parameter, showing the scope requested¹; and the *state* parameter.

9. IdP \rightleftharpoons User. This step is the same as step 2 of the OAuth 2.0 protocol (implicit grant type) given in section 4.6.3.4.
10. IdP \rightarrow Adaptor/UA. The IdP redirects the UA back to the RP URL provided by the adaptor, including the access token (embedded in the URL fragment) and the state parameter. The adaptor checks that the value in the state parameter is the same as it generated in step 8, is sufficiently current, and that a response to this parameter has not already been received. The adaptor then adds the received value to a list of acknowledged requests.
11. IdP \rightleftharpoons Adaptor/UA. The adaptor uses the contents of the access token received in the previous step to construct a request to the IdP for the values of the user attributes requested by the RP. The request is sent to the IdP via a TLS-protected channel. The IdP validates the request, including verifying the accuracy of the provided access token, and, if successful, meets the request.
12. Adaptor/UA \rightarrow RP. On receipt of the user attribute values from the IdP, the adaptor uses them to construct a CardSpace-like encapsulating token and submits it to the RP. This token includes the IdP-supplied user attributes and the digitally-signed, SIIP-issued RSTR containing the PPID (see also section 9.3.1).
13. RP \rightarrow UA. The RP verifies the received encapsulating token (including verifying the RSTR signature, PPID, nonces, time-stamps, and any other *standard* CardSpace checks), and, if satisfied, grants access.

¹This scope parameter specifies the RP-requested user attribute types (if any), the values for which are to be provided by the IdP. The adaptor will know what they are since they were stored by it earlier. The scope parameter helps the IdP determine the scope of the access request; the IdP requires the user to authorise the release of the requested attribute values.

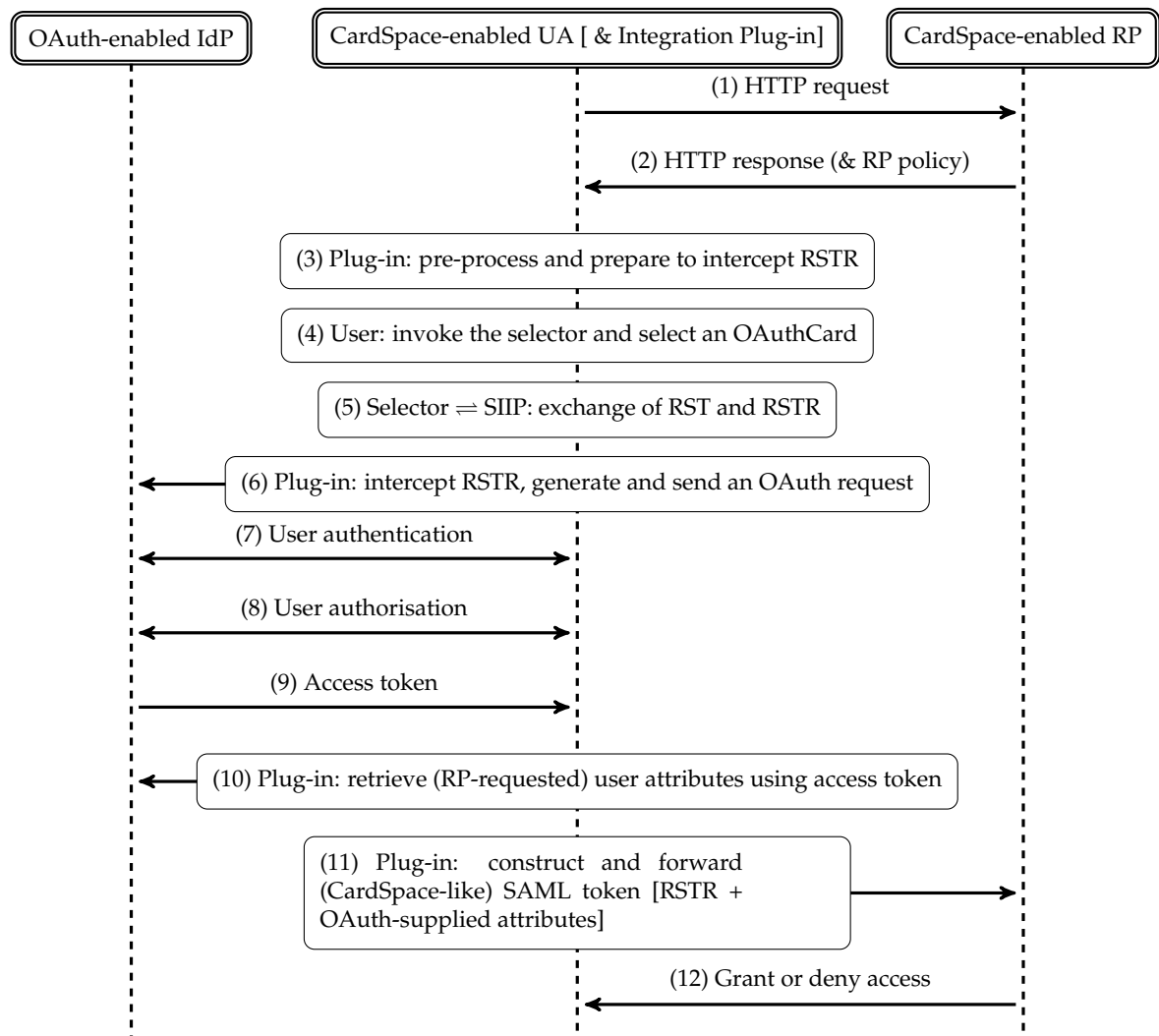


Figure 9.1: Protocol Exchanges

9.3 Discussion and Analysis

9.3.1 Security Considerations

The unsigned encapsulating token generated by the browser extension in step 12 of section 9.2.2 includes the PPID, the user attributes as provided by the IdP, and the digitally-signed, SIIP-issued, RSTR. Just as in standard use of CardSpace, the RP can compare the SIIP-asserted PPID and the public key in the encapsulating token with its stored values and can also verify the digital signature (see section 4.3.11).

In addition, if the RSTR contains self-issued attribute values, the RP can compare them with the IdP-provided attributes; such a procedure potentially gives the RP greater confidence in the accuracy of these attributes.

Unless it has compromised the user platform, a malicious entity will be unable to fabricate an encapsulating token to masquerade as a legitimate party since it will not have access to the PPID and the private key necessary to sign the RSTR, both of which are only available if the appropriate InfoCard is selected on the correct user platform.

Note that, in protocol step 4, the selector identifies the RP to the user and indicates whether or not they have visited that particular RP before; if the user is visiting this RP for the first time, CardSpace requests the user's permission to proceed (see section 4.3.9.1). This helps to increase security since the user and the RP are both identified to each other.

The scheme allows the user attributes to be stored remotely at the IdP; this has potential security advantages over storing the attributes locally on the user platform, as is currently the case with CardSpace SIIP-issued attributes.

The adaptor-generated SAML token could be extended to contain an additional field to indicate that the user has been authenticated by a specified IdP, as well as when and how. Of course, the RP would need to be modified to be able to process such an extra field, although this is likely to be relatively straightforward.

As discussed in section 5.5, even if the encapsulating token does not include a copy of an integrity-protected token generated by the IdP, the RP may still be able to gain additional assurance in the user attribute values in the encapsulating token and in the authenticity of the user.

9.3.2 Attribute Mapping

CardSpace and OAuth (e.g. as instantiated by Facebook Connect) use two different sets of attribute types², which gives rise to a compatibility issue. We adopt the two approaches discussed in section 5.3.2 to deal with this problem. An example attribute type mapping, as required to support the first approach, is given in Table 9.1.

Table 9.1: CardSpace-Facebook Connect Attribute Mapping

CardSpace (Personal Cards)	OAuth (Facebook Connect)
givenname	first_name
surname	last_name
emailaddress	email
dateofbirth	birthday
gender	gender
country	locale
city	location
web page	website

9.4 Prototype Realisation

We next give details of a prototype implementation of the scheme. The description applies to Facebook Connect, an implementation of OAuth 2.0. The prototype uses Facebook Connect's client-side flow³ (i.e. the implicit grant type).

As in the previous prototypes, the prototype described here is coded as a JavaScript browser plug-in, executed using a C#-driven BHO (see section 6.4.2). It can readily be enabled or disabled using the add-on manager in the Internet Explorer Tools menu. The plug-in does not require any changes to default Internet Explorer security settings, thereby avoiding potential vulnerabilities arising from such changes. The scheme operates with

²As stated in section 4.3.3.1, CardSpace personal cards only support fourteen editable attributes, whereas Facebook Connect supports many more.

³<http://developers.facebook.com/docs/authentication/>

both the CardSpace and the Higgins identity selectors without any modification (see section 6.4.2).

9.4.1 Registration

Prior to use, the user must have accounts with a CardSpace-enabled RP and with Facebook. The user must register the RP with Facebook⁴ (see section 9.2.1). The user must also create an OAuthCard in which the IdP-specific RP identifier is inserted in the *first name* field, and the trigger word *OAuth* in the *last name* field. The Facebook URL is contained in the source code of the browser extension, and thus does not need to be included in the OAuthCard. For ease of identification, the user can give the OAuthCard a meaningful name and an image.

9.4.2 Prototype Operation

We next consider specific operational aspects of the prototype. We refer throughout to the numbered protocol steps given in section 9.2.2.

The implementation of step 3 is precisely the same as that of step 3 of the CardSpace-OpenID integration prototype described in section 8.4.2, except that step 3.1 (e) there is replaced by the following.

- 3.1 (e) If necessary, and after keeping track of the original policy settings, the plug-in modifies the RP policy so that the *first name* and *last name* claim types are specified in the *requiredClaims* tag.

In step 8 the plug-in uses the DOM (see section 2.5.2) to perform the following steps.

- 8.1 It intercepts the RSTR sent by the selector using the added function.
- 8.2 It uses the JavaScript property *document.location.protocol* to discover if HTTP or HTTPS is in use. As stated previously, it operates slightly differently in these two cases (see step 8).

⁴<https://developers.facebook.com/setup/>

- If HTTP is in use, the plug-in parses the RSTR to obtain the *first name* and *last name* fields. If the *last name* field contains the word *OAuth*, the plug-in continues; if not, it terminates. The identifier of the RP is recovered from the *first name* field.
- If HTTPS is in use, the plug-in uses a JavaScript pop-up box to ask the user whether use of the integration scheme is required. If so, it proceeds; otherwise it terminates. On proceeding, it prompts the user to enter the identifier of the RP, as issued by Facebook. The plug-in offers the user the option to store the input value in a persistent cookie (see section 2.3.4.1) for future logins at this RP, using a plug-in-embedded checkbox.

- 8.3 It constructs an OAuth request, compatible with Facebook Connect. This involves generating a nonce and time-stamp (used to build the state parameter), and also determining the required and optional attribute types to be requested from Facebook. The plug-in retrieves all the CardSpace-supported claim types it stored earlier. It then maps between them and the Facebook-supported attribute types, using Table 9.1. As in the CardSpace-Shibboleth prototype (see section 7.4.2), the mapping is performed using JavaScript regular expressions. The plug-in sets the value of the `redirect_uri` parameter (to which Facebook will send the response) to the URL of the currently-visited RP page. In addition, it sets the value of the `response_type` parameter to *token*, signifying use of the implicit grant type.
- 8.4 It encrypts the RSTR and the value of the state parameter using AES in CBC mode, with a secret key known only to the plug-in. The encrypted values are temporarily stored in a cookie.
- 8.5 It redirects the user to Facebook along with the OAuth request, using the JavaScript inherent property *window.location*.

In step 10 the plug-in performs the following steps.

10.1 It parses the Facebook-issued response, embedded in the URL.

10.2 It verifies the response, as specified in step 10.

In step 11, the plug-in uses the provided access token to request and retrieve the RP-requested user attribute values from Facebook open graph⁵ using a TLS-protected channel. The Facebook open graph⁶ holds structured attributes about the user.

In step 12 the plug-in performs the following steps.

12.1 It generates an encapsulating token, which includes the user attribute values received from Facebook. It also embeds the signed SIIP-issued RSTR within the token, after retrieving the RSTR from the appropriate cookie and decrypting it.

12.2 It creates and appends an invisible HTML form (with the method attribute set to POST) to the current page.

12.3 It inserts the entire encapsulating token into the form as a hidden variable, with the name attribute of this variable set to the name of the CardSpace object tag.

12.4 It inserts the URL of the RP into the action attribute of the invisible form.

12.5 Finally, it auto-submits the HTML form (transparently to the user), using the JavaScript inherent method *submit*.

The prototype has been successfully tested with Facebook and an experimental implementation of a CardSpace-enabled RP, as well as with the current (unmodified) CardSpace and Higgins identity selectors.

⁵<http://developers.facebook.com/docs/reference/api/user/>

⁶http://en.wikipedia.org/wiki/Social_graph#Open_Graph

9.4.3 Potential Features and Issues

The potential advantages and limitations described in sections 5.4 and 5.6 are applicable here, as are the issues and features listed in section 6.4.4.

9.5 Conclusions and Future Work

In this chapter we have described an OAuth-based instantiation of the interoperation model given in chapter 5. CardSpace users (indeed, users of any Information Card system) are able to obtain a security token from an OAuth provider, the contents of which can be processed by a CardSpace-enabled relying party. The scheme is transparent to OAuth providers and to identity selectors, uses a browser extension, and requires only minor changes to a CardSpace-enabled relying party. It uses the identity selector interface and CardSpace personal cards to enable interoperation between OAuth providers and CardSpace relying parties.

Planned future work includes investigating the possibility of extending the CardSpace identity selector to simultaneously support security tokens from a variety of identity providers, such as OpenID, Liberty, Shibboleth, as well as CardSpace remote and self-issued identity providers.

Part III

Practicality and Security

Overview

Part III of this thesis describes three novel schemes designed to enhance the practicality and security of identity management systems. It contains three chapters, as follows.

1. *Chapter 10* describes a scheme that allows an Information Card system to be used as a password manager.
2. *Chapter 11* is concerned with a scheme that allows an Information Card system to be used as a password-based single sign on system.
3. *Chapter 12* specifies a scheme that uses a mobile device to enhance user authentication in an Information Card system.

Using an Information Card System as a Password Manager

10.1 Introduction

The most widely used means of user authentication remains the use of passwords, despite their well-known shortcomings. Moreover, as the number of on-line services requiring authentication continues to grow, users increasingly re-use passwords, write them down in insecure ways, and/or employ passwords which can readily be guessed. The result is an ever-increasing risk of exposure of passwords to malicious parties. Passwords could also be stolen [72, 110] through key logging, phishing, sniffing, shoulder surfing, etc.

An approach that enables the use of site-unique strong passwords whilst also maintaining user security and privacy would thus be highly beneficial. Password managers of various types have been proposed to meet this need. A password manager stores usernames and passwords and makes them available when required. Typically, users are not required to remember any passwords apart from a single master password which can be used to lock/un-lock the password manager [79]. Password managers can be particularly helpful when a relatively large number of passwords are required to access multiple on-line services. Password managers can be seen as potential alternatives to SSO systems such as Windows Live ID, formally known as Passport (see section 4.2), OpenID (see section 4.5), and Liberty

(see section 4.7).

Despite the introduction of Information Card-based systems (such as CardSpace and Higgins), the vast majority of websites still use username and password for authentication, and this is likely to continue for at least the next few years (see sections 1.2.2 and 4.3.14.6). One major problem with Information Card systems, and with other similar systems providing more secure means of user authentication, is that the transition from username-password authentication is extremely difficult to achieve. RPs will not wish to do the work necessary to support Information Card systems if very few users employ them; equally, users are hardly likely to use an Information Card system if it is only supported by a tiny minority of websites. In this chapter we propose *PassCard*, a novel scheme designed to help overcome this barrier to change by allowing an evolutionary deployment of an Information Card system, initially as a password manager and subsequently, once users are familiar with its operation, as a more sophisticated means of user authentication.

PassCard operates with a variety of Information Card-based systems, including CardSpace and Higgins (see sections 4.3 and 4.4, respectively). However, for simplicity of presentation, in this chapter we describe its operation with CardSpace.

The goal is to develop a simple and intuitive approach to password management, transparent to identity selectors and RPs. PassCard is designed to operate with existing websites without any modification, and, in particular, RPs are not required to support CardSpace. Usernames and passwords are stored in personal cards, and these cards can be used to sign-on transparently to corresponding websites. Much of the material in this chapter has been published [10, 14].

The remainder of the chapter is organised as follows. In section 10.2, we introduce PassCard. We describe a prototype implementation in section 10.3, and, in section 10.4, we outline a number of PassCard features

and limitations. Section 10.5 reviews related work, and, finally, section 10.6 concludes the chapter.

10.2 PassCard

We now present PassCard. The parties involved are:

- an RP, i.e. a website that the user is currently visiting;
- a CardSpace-enabled UA (e.g. a web browser capable of invoking the identity selector, such as Internet Explorer);
- a browser extension implementing the protocol described in section 10.2.2; and
- a user-selectable HTTP server (HS), which is used in a passive manner (indeed, the HS can be any HTTP-based website). The HS is involved solely to enable PassCard to support HTTPS-enhanced websites. It is necessary because, if the visited website uses HTTPS, then the SIIP-issued RSTR, which contains the username-password pair for the visited site, will be encrypted using the public key of the visited site; therefore, the RSTR will not be available to the PassCard browser extension. The introduction of the HS addresses this issue, as we describe below.

10.2.1 Prerequisites

The scheme has the following operational requirements.

- Either prior to, or during, use of the scheme, the user must create a special personal card for each RP with which the scheme is to be used, referred to here as a *PassCard*. A PassCard contains the RP-specific username and password in specified card fields, the choice of which is implementation-specific. Basic protection against phishing can be provided if the URL of the target website is also included in a field of the

PassCard. However, this is optional, as users may wish to use a single PassCard with multiple websites sharing the same user credentials.

- The user must install the PassCard browser extension, which, amongst other things (see section 5.2.3), must be able to add a clickable PassCard icon to the RP web page (see Fig. 10.4). This enables the user to invoke the selector and to subsequently select (or create) a PassCard.

10.2.2 Operation

We now specify the operation of PassCard, which differs depending on whether the RP uses HTTP or HTTPS. We therefore divide the description into two cases. Steps 1–3b are the same for both cases, and are described first.

1. UA → RP: HTTP Request. The user employs a UA to visit an RP login page.
2. RP → UA: HTTP Response. The login page is returned.
3. Browser Extension → UA. The extension performs the following processes.
 - a) It scans the RP page for a form containing a username field, a password field, and a submit button. If all of these are found, it continues; otherwise it terminates.
 - b) It determines the protocol (i.e. HTTP or HTTPS) in use with the RP.

Execution continues as described in sections 10.2.2.1 and 10.2.2.2, depending on whether the RP is using HTTP or HTTPS.

10.2.2.1 HTTP-based PassCard

Steps 3c–8 (below) apply for RPs using HTTP. Fig. 10.1 gives an overview of the operation of the scheme in this case.

3. Browser Extension → UA. Following step 3b in section 10.2.2, the browser extension continues as follows.
 - c) It adds CardSpace-enabling tags to the login page, setting the embedded security policy to require a token asserting claims of the types in which the user credentials are stored.
 - d) It adds a function to the login page to intercept the RSTR that will be returned by the selector.
 - e) It embeds a PassCard icon in the page, causing it to appear above the submit button (see Fig. 10.4).
4. User → UA. The user clicks on the added icon and the selector lights up.
5. The user selects (or creates) and submits a PassCard. The selector creates and sends an RST to the SIIP, which responds with an RSTR.
6. Selector → UA. The selector passes the RSTR to the UA.
7. Browser Extension → UA. The browser extension performs the following tasks.
 - a) It intercepts and parses the RSTR.
 - b) If the token contains the URL of the target site, the extension compares it with the URL of the visited site, and only proceeds if they match.
 - c) It extracts the username and password from the specified fields of the RSTR.
 - d) It auto-populates and submits the login form.
8. RP → UA. The RP verifies the provided values and, if satisfied, grants access.

10.2.2.2 HTTPS-based PassCard

Steps 3c–8b (below) apply for HTTPS-enabled RPs.

3. Browser Extension → UA. Following step 3b in section 10.2.2, the browser extension continues as follows.
 - c) It obtains the web page's URL, referred to throughout as the *target URL*.
 - d) It causes the PassCard icon to appear above the submit field, in such a way that clicking it results in an HTTP redirect to the HS.
4. User → UA. If the user clicks the icon, the browser is redirected to the HS, and the target URL is also transmitted as a URL query parameter.
5. Browser Extension → UA. While interacting with the HS page, the browser extension:
 - a) adds an invisible password login form to the returned HS page, if it does not already have one, at which point steps 3c–3e of section 10.2.2.1 are executed;
 - b) recovers and temporarily stores the target URL; and
 - c) transparently invokes the selector, at which point steps 5–7c of section 10.2.2.1 are executed.
7. Browser Extension → UA. The browser extension continues as follows.
 - d) It encrypts the username and password values with a secret key known only to the browser extension (see section 10.2.3.3).
 - e) It transparently redirects the user to the target URL, and the encrypted username-password values are transmitted as URL query parameters.
8. Browser Extension → UA. While interacting with the target URL site, the extension:

- a) recovers and decrypts the username and password values; and
- b) auto-populates and submits the login form, after which step 8 of section 10.2.2.1 is executed.

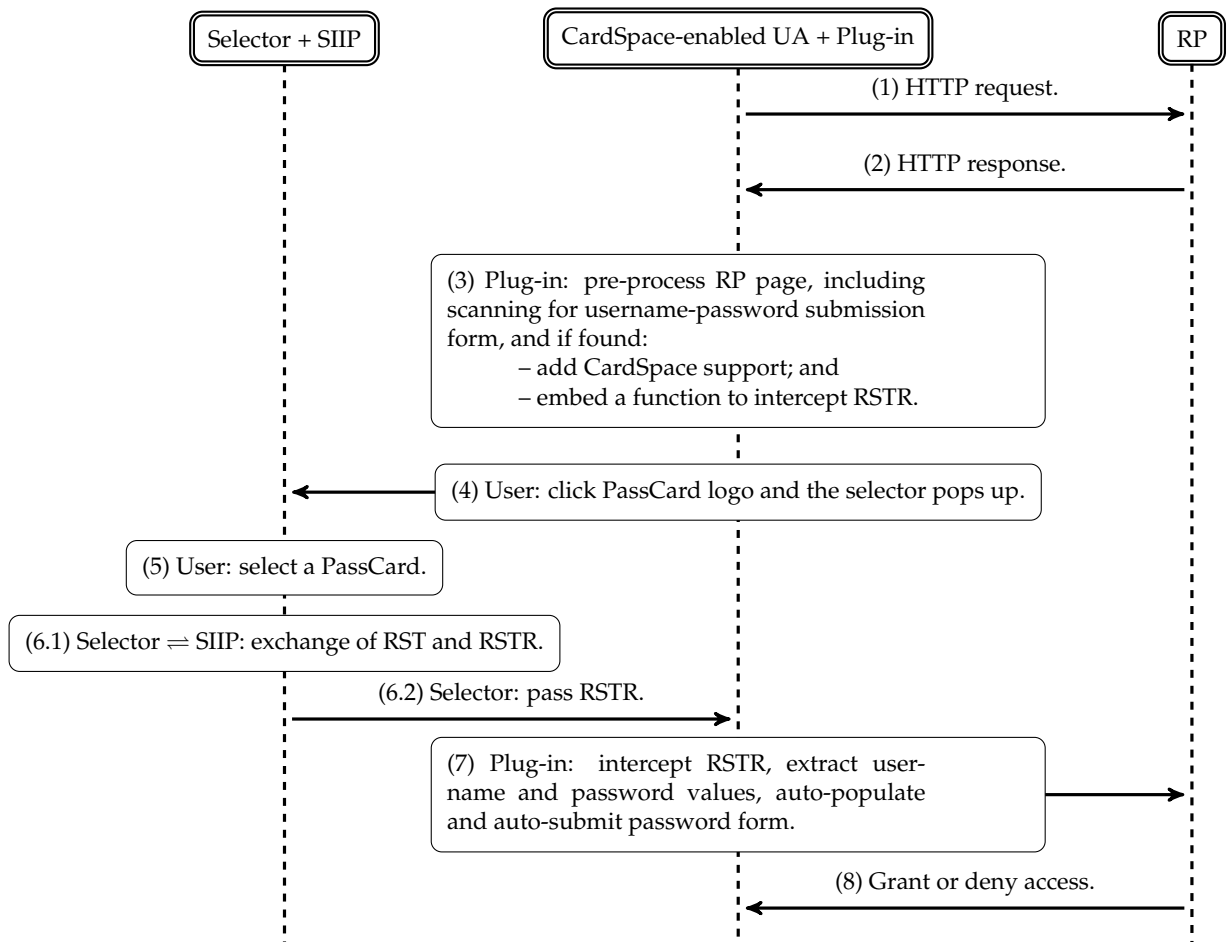


Figure 10.1: PassCard Operation in HTTP mode

10.2.3 Discussion

10.2.3.1 User Experience

The PassCard user experience when operating in HTTP mode is precisely the same as with conventional password-based authentication except that, instead of manually entering and submitting a username and password, the PassCard user selects and submits a PassCard. The user experience in HTTPS mode is similar to that of HTTP mode, except that users (depending

on their Internet speed, machine speed, etc.) may or may not experience a redirect from the target HTTPS site to the HS and vice versa, resulting in the temporary display of the HS web page. Note that, in both PassCard modes, users are not required to click the PassCard icon more than once or to remember any passwords.

10.2.3.2 CardSpace-enabled RPs

Regardless of whether or not an RP already supports CardSpace, the plug-in will always add the PassCard icon to the RP web page, as long as it detects username-password prompts on the page. This means that, if an RP supports CardSpace and simultaneously supports username-password authentication, as is the case for the *myOpenID* website¹ (sampled on 23/04/2012), the browser extension will still insert the PassCard icon above the submit button of the password-based login form. Informal tests on the prototype implementation suggest that this will not disrupt normal operation of CardSpace.

The RP page will thus display both the CardSpace and the PassCard logos (see Fig. 10.3). In such a case, users will have (at least) the following three login options:

1. to populate the username and password fields and submit the login form manually;
2. to use PassCard to auto-populate and auto-submit the login form; or
3. to click the CardSpace logo to use CardSpace-based authentication.

10.2.3.3 Use of Cryptography

The encryption of the username and password in step 7d of section 10.2.2.2 is not necessary to prevent channel eavesdropping, because an SST/TLS channel is already established between the browser and the target HTTPS

¹<https://www.myopenid.com/>

site. However, if the username and password are sent in plaintext as part of the URL (as is the case in the PassCard prototype described in section 10.3 below), then they will be vulnerable to shoulder-surfing attacks since they will be shown in the browser address bar (and possibly also in the browser status bar). The prototype implementation uses a simple symmetric encryption scheme for username-password encryption to minimise the overhead.

10.3 Prototype Realisation

We next describe a prototype implementation of the PassCard scheme. The prototype is coded as a browser plug-in in JavaScript, executed using a C#-driven BHO (see section 6.4.2). PassCard can be enabled or disabled using the add-on manager in the Internet Explorer Tools menu.

As is the case with the integration prototypes (chapters 6 to 9), the PassCard prototype operates with both the CardSpace and the Higgins identity selectors without any modification. It has been disseminated as an open-source research project².

10.3.1 Registration

Prior to, or during, use of PassCard, the user must create a PassCard, inserting their username in the *first name* field and password in the *last name* field. The user also has the option to insert the URL of the target website in the *web page* field³. For ease of identification, the user can give the PassCard a meaningful name, e.g. of the corresponding website. The user can also upload an image for the PassCard, e.g. containing the icon of the intended site. Example PassCards are shown in Fig. 10.2.

²<http://iescripts.org/view-scripts-808p1.htm> and/or <http://sourceforge.net/projects/passcard/>

³The *web page* field was chosen to contain the URL of the target website since it seemed the logical choice; however, like the use of the *first name* and *last name* fields, this is an implementation option.



Figure 10.2: PassCards

10.3.2 Operation

The prototype implements the protocol steps specified in section 10.2.2.

- In step 3a, the plug-in processes the RP web page in the following way.
 1. It scans the web page for a form tag.
 2. If a form tag is found, it searches the form for three input tags referring to username, password, and submit, using the following procedure:
 - a) it searches for an input tag of type *text*;
 - b) if found, it searches for another input tag of type *password*; and
 - c) if found, it searches for another input tag of type *submit*. If no input tag of type *submit* is found, the plug-in searches for an input tag of type *image* and, if unsuccessful, searches for an event-based input tag of type *button*.
 3. If all the three fields are detected, then the plug-in highlights the username and password fields in green for ease of identification. A potential advantage of this step is that if the wrong fields are highlighted, then the user will know that the PassCard scheme should not be used.

The above process involves the following detailed processing.

- Highlighting does not take place unless a username field, a password field and a submit button have all been detected in a single form, as a web page could potentially contain more than one input tag of type text, such as those used for searching.
 - To differentiate between registration and login pages, the plug-in terminates if it detects more than one password field between the username and submit fields. Whereas it appears common for a login page to only have a single password field before the submit button, registration pages typically have two password fields (before the submit button): the first for the user to enter their password, and the second to confirm their password. Examples include the registration and login pages hosted by major websites such as Google, Yahoo, Microsoft Research, and SpringerLink⁴.
 - When searching for the form submission button, if no submitting input tag is found then the plug-in searches for an image tag. This is because, instead of a submit button, some websites display a clickable image⁵ with similar functionality.
 - Whereas it appears common for a username field to be immediately followed by a password field, a submit button may not always immediately follow a password field. For example, some major sites (including Google, Yahoo, Facebook, and SpringerLink) add a *Stay signed in* or *Remember me* check box between the password field and the submit button⁶. The plug-in addresses this issue by skipping all tags between the password field and the submit button, including those of type *checkbox*.
- In step 3b, the plug-in uses *document.location.protocol*, a JavaScript inherent property, to discover whether HTTP or HTTPS is in use.

⁴Websites most recently checked on 23/04/2012.

⁵This includes an image tag embedded in a hyperlink (anchor) tag, an image tag on its own, an image tag embedded inside a button tag, or an event-based button tag.

⁶Websites most recently checked on 23/04/2012.

10.3.2.1 HTTP-specific Implementation Details

We refer to the step numbers given in section 10.2.2.1.

- Following step 3b in section 10.2.2, the plug-in performs the following processes using the login page provided by the RP.
 - 3.c) It adds an HTML object tag that allows the user to invoke the selector. Within the object tag, it sets the param tags to indicate that the RP security policy requires PassCards to contain two fields: the *first name* and the *last name* fields, (or three fields if protection against phishing is required, in which case the third field would be the *web page* field). Alternatively, the security policy could be configured so that the *web page* field is optional.
 - 3.d) It adds a function to the head section of the RP login page to intercept the (XML-based) RSTR returned by the selector.
 - 3.e) It inserts the PassCard logo, causing it to appear just before the login button in the RP page, as illustrated in Fig. 10.4. The logo is associated with an *on-click* event, so that, if clicked, the selector is invoked (after calling the added function). To cater for users with sight difficulties or web browsers configured not to display images, a text field can replace the logo. This text is also displayed when the mouse is held over the PassCard logo, indicating that PassCard can be used to sign-on (see Fig. 10.3).

- In step 7, the plug-in performs the following steps.
 - 7.a) It intercepts the RSTR using the added function.
 - 7.b) It parses the intercepted token and extracts the values of the *first name* and *last name* fields.

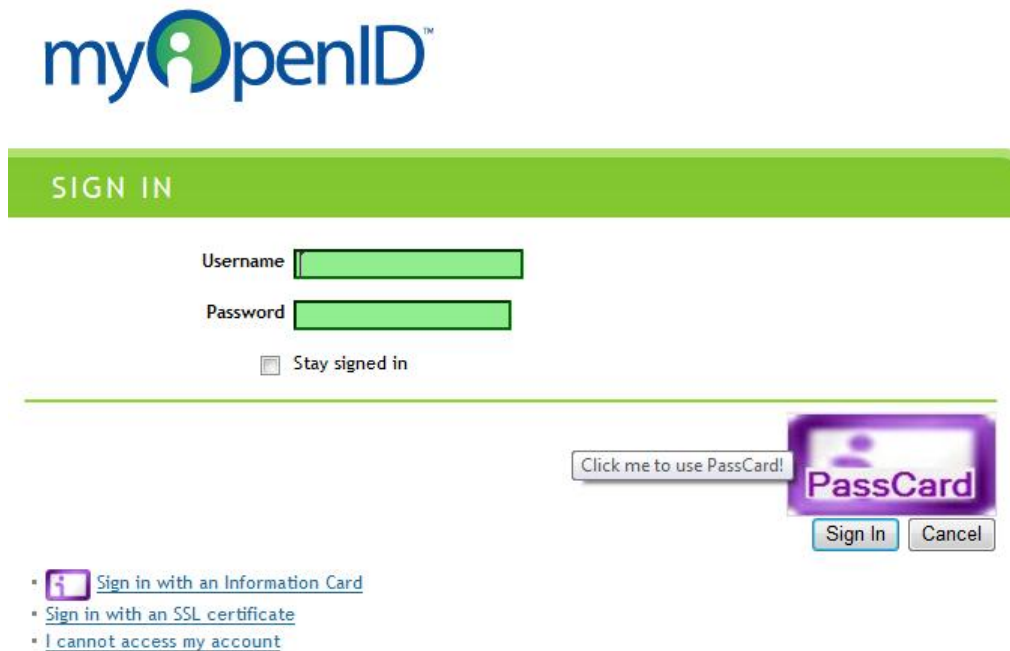


Figure 10.3: PassCard Co-operating with a CardSpace-enabled RP

- 7.c) It checks whether the HTTPS-mode-signalling cookie is set (see section 10.3.3.1); if so, it moves to step 7d of section 10.2.2.2. If not, it continues.
- 7.d) If a URL is present in the *web page* field of the RSTR, it compares it with the URL of the visited website, and only proceeds if they match.
- 7.e) It automatically fills in the username and password fields with the *first name* and *last name* values, respectively.
- 7.f) It auto-submits the login form using the JavaScript *click()* method.

10.3.2.2 HTTPS-specific Implementation Details

We refer to the step numbers given in section 10.2.2.2.

- In steps 3c and 3d, the plug-in:



Figure 10.4: PassCard Logo

1. obtains the HTTPS site's (full) URL, i.e. the target URL (see section 10.2.2.2), using the JavaScript property *document.location.href*, removing any query parameters that may be attached to the URL; and
 2. creates an HTML hyperlink (*a*) tag, and sets its *href* attribute to point to the HS. It also embeds an HTML image (*img*) tag, pointing to the PassCard logo, inside the hyperlink tag. In addition, it creates a title and alternative text, so that if the mouse is held over the PassCard logo, or if the image is not displayed, text is displayed indicating that PassCard can be used to sign-on.
- In step 4, if the PassCard logo is clicked, the plug-in redirects the user to the HS. The target URL and the PassCard namespace (e.g. used to differentiate different PassCard versions) are sent as query parameters, as in the example shown in Fig. 10.5.

```
http://www.oman4ever.org/adminlogin.jsp?PassCard.targetURL=https://www.google.com/accounts/Login&PassCard.ns=http://www.oman4ever.org/PassCard
```

Figure 10.5: Redirect URL (target URL → HS)

- In step 5, the plug-in processes the HS web page in the following way.

5.1 If no password-based login form is served by the HS web page, it creates a *div* section and inserts an HTML form. The form includes username (input tag of type text), password (input tag of type password) and submit (input tag of type submit) elements. The div section is then embedded in the HS page.

To keep the changes to the HS page transparent to the user, the browser plug-in hides this form by setting the style of the div section, which contains the added form, to *visibility: hidden*. Note that embedding a form in the HS page enables the re-use of code that was previously developed to handle HTTP mode.

5.2 It parses the URL query parameters to obtain the target URL, and stores it in a cookie.

5.3 It creates the HTTPS-mode-signalling cookie (see section 10.3.3.1).

5.4 It auto-invokes the selector using the JavaScript *click()* method.

- In steps 7d and 7e, the plug-in:
 1. encrypts the username and password values using AES in CBC mode (see section 2.4.2.1) with a secret key known only to the plug-in; and
 2. redirects the user to the target URL, using the JavaScript property *window.location*. The encrypted username and password values and the PassCard namespace are sent as query parameters, as in the example shown in Fig. 10.6.

```
https://www.google.com/accounts/Login?PassCard.identifier=950a0d3  
ae1ee2c074753c0b51820c9a95f5e9fece7c5534dff25666001350ee428d62d  
472f551e2bbe37b6443283e3b48&PassCard.verifier=9b9a2b5514c9643a2c  
b6a2614530145d3dab76b12f292adcaadc73910921c408677ba99e960c248f  
d6de3a1feced4c2a&PassCard.ns=http://www.oman4ever.org/PassCard
```

Figure 10.6: Redirect URL (HS → target URL)

- In step 8, the plug-in processes the target URL web page in the following way.
 - 8.1 It parses the URL query parameters to obtain the encrypted values of the username and password, and decrypts them using its internally stored secret key.
 - 8.2 It locates the username, password, and submit fields.
 - 8.3 It auto-populates the username and password fields with the decrypted username and password values.
 - 8.4 It creates a cookie with a three-second lifetime to ensure that it does not re-submit the username-password values within a three-second interval (see section 10.3.3.3).
 - 8.5 It auto-submits the login form using the JavaScript click method.

10.3.3 Discussion

We next outline certain issues that arose during prototype development.

10.3.3.1 HTTP/HTTPS Modes

Since the same program code is used to handle both the HTTP and HTTPS cases (to keep development efforts to a minimum and maximise code efficiency), a cookie is used to signal to the browser extension that HTTPS mode has been activated. This instructs the browser extension to perform step 7d of section 10.2.2.2 instead of step 7d of section 10.2.2.1. More specifically, the HTTPS-mode-signalling cookie is created between steps 5b and 5c of section 10.2.2.2, and this cookie is checked between steps 7c and 7d of section 10.2.2.1. If it is set, HTTPS mode is activated; otherwise HTTP mode continues.

10.3.3.2 HTTP Server

We now consider the role of the HS in the PassCard system. The HS can be any HTTP site, and is not actively involved in the protocol except to serve a web page when requested. It simply acts as a convenient way to avoid the problems arising when an RP uses HTTPS. The choice of HS is not particularly security-sensitive in that it does not learn any sensitive data; it simply learns that a client at a particular IP address is using PassCard. The choice of HS is a configuration option in the prototype, so if the default site is regarded as privacy-threatening then a user can change it to any trusted address, e.g. that of a personal web page. Alternatively, a user could use a local web server. Setting up a local web server is typically straightforward, e.g. by installing XAMPP⁷.

The use of a personal web page has other advantages, including minimising the load on the default HS and avoiding problems arising if the default HS is unavailable. Use of a personal page is also likely to avoid any possible user concerns about the involvement of a third party server.

The HS could also be implemented as a proxy server. This would maximise user transparency in the sense that users would not experience a redirect from the target HTTPS site to another HS and vice versa. However, such a change would deny users the opportunity to use their own site. In addition, the proxy site will need to be trusted by the user, since username-password values will need to be given to the proxy server so that it can deliver them transparently to the target HTTPS site. Furthermore, if an HTTP proxy [160] is used, the user must force the browser to use the proxy. This is potentially inconvenient and may not always be possible; for example the user may not have sufficient privileges to make the necessary change to the browser settings, e.g. when using a device in an Internet café [93]. Finally, PassCard would stop working if the proxy site was unavailable for any rea-

⁷<http://www.apachefriends.org/en/xampp.html>

son.

10.3.3.3 Failed Authentication Attempts

When some websites, e.g. that of the RHUL's network authentication page⁸, have incorrect credentials submitted to them, they do not change the URL (e.g. by redirecting the user to an error page). Instead they inform the user of the failure of their authentication attempt on the same login page, maintaining the same URL in the browser's address bar.

When the PassCard plug-in is executing in HTTPS mode, such an event will cause it to run again, because the page has been refreshed but the URL is unchanged. When it runs, the plug-in will see the same URL with the same query parameters (as were redirected/sent by the PassCard plug-in when running on the HS's page), and will thus attempt to automatically sign-on the user again with the same (wrong) credentials, because it believes the user has just been redirected from the HS. The website will deny access once again, and the same process will repeat indefinitely, resulting in either the user being locked out after a certain number of failed authentication attempts or the browser becoming stuck in an (infinite) loop.

To address this issue, the plug-in creates a short-lived cookie just before attempting to automatically sign-on the user for the first time, to ensure that it does not attempt to log-in the user again before a certain period of time (e.g. 3 seconds) has elapsed. If the plug-in discovers that the user has been denied access following an attempt to automatically sign-on, it informs the user and terminates. It also attempts to close the visited page, after first obtaining permission from the user. This termination is based on the assumption that the user has either inserted the wrong credentials in the current PassCard or has simply selected the wrong PassCard.

Other possible means of addressing this particular issue were tested, but were found not to work. For example, it is tempting to try to use the HTTP

⁸<https://nac.rhul.ac.uk/authentication/>, most recently checked on 23/04/2012.

referrer⁹ field, which identifies the web page that the user was visiting before they arrived at the current page. Indeed, JavaScript provides its own built-in property for HTTP referrer, namely *document.referrer*. Thus, it would appear that the PassCard plug-in could use this to sign-on the user only if the user was redirected from the HS; if not, the plug-in could simply terminate on the assumption that the user has already attempted to sign-on but has been denied access. However, informal tests suggest that this technique does not work for the PassCard plug-in, probably because the redirect from the HS to the target site is initiated by the plug-in itself (using the *window.location* property) and not by the HS server. In any case, the use of the HTTP referrer field should be avoided because if the HS and the target URL fall under the same domain, the HTTP referrer field would prevent the plug-in from automatically signing in the user.

10.3.3.4 CardSpace-enabled RPs

We next discuss two approaches for dealing with RPs which are CardSpace-enabled, and which provide alternatives to the approach described in section 10.2.3.2.

1. After identifying that an RP supports username-password authentication, the browser extension could be configured to detect whether the RP already supports CardSpace; if so, the browser extension would deactivate PassCard. However, since the RP offers passwords as a login option, offering PassCard may help password users and does not prevent users employing CardSpace-based authentication.
2. Instead of automatically de-activating PassCard if the RP already supports CardSpace, as suggested above, the plug-in could ask the user (e.g. via a JavaScript pop-up box) whether to activate PassCard. Although this would provide a greater degree of user control, repeated

⁹<http://tools.ietf.org/html/rfc2616#section-14.36>

user prompting could become very intrusive. Nevertheless, this effect could be mitigated if the user's answer was stored for use in future interactions with this RP.

10.4 PassCard Properties

We now consider certain features and limitations of PassCard.

10.4.1 Security

PassCard uses the functionality of the CardSpace identity selector, and is supported by its built-in security features. For example, the selector runs in a separate private desktop session, mitigating the risk of other applications, e.g. malware, from interacting or interfering with it. In addition, all values inserted in the fields of a PassCard are stored in encrypted form on the user platform.

The selector identifies the RP to the user and indicates whether or not they have visited that particular RP before; if the user is visiting this RP for the first time, CardSpace requests the user's permission to proceed. This enhances security by comparison with conventional password-based authentication, where an HTTP-based RP is not identified to the user.

As with any local password manager, PassCard (when running in HTTP mode) avoids the need for trusted third parties. In addition, the automatic form-filling feature reduces exposure to shoulder-surfing attacks and also helps to thwart key loggers.

Depending on how it is used, PassCard can help to reduce the threat of phishing attacks involving impersonation of legitimate websites. This is achieved by comparing the URL included in a PassCard (if present) with that of the visited website. PassCard also supports the use of strong per-site passwords, since users no longer need to memorise or write down passwords.

Finally, note that the PassCard browser extension does not require any changes to default browser security settings, thereby avoiding potential vulnerabilities resulting from making such changes.

10.4.2 Usability

PassCard provides a simple, intuitive user experience through its use of the selector interface. At the same time, it familiarises users with CardSpace, thereby potentially facilitating future adoption of more secure means of authentication. Unlike other password managers which represent credentials in text form, PassCard credentials are stored in PassCards which can be equipped with a readily recognisable image, e.g. an RP logo.

PassCard operates transparently to external parties, and hence does not require any changes to RPs, identity selectors or to default browser security settings. PassCard is also flexible, since users can choose whether or not to use it simply by electing to click the PassCard icon (or not).

Finally, by making use of CardSpace features, PassCard supports a degree of roaming. A user can transfer PassCards from one PC to another using the CardSpace backup facilities. Indeed, if the CardSpace backup file, which holds data in encrypted form, is stored on a portable storage medium, e.g. a USB drive, then full mobility is provided, as well as robustness in the form of protection against loss of credential data.

10.4.3 Limitations

Perhaps the most obvious limitation of PassCard is that anyone with access to a Windows user account can access the PassCards and use the stored credentials. This is a fundamental limitation of CardSpace which, by default, does not impose any additional password protection on the use of the selector (see section 4.3.14.1). To address this issue, we observe that CardSpace allows individual InfoCards to be PIN-protected, which should be considered for PassCards stored on machines accessible to other users. In addition,

it may be possible to cause CardSpace to run under User Account Control¹⁰ (UAC), so that running CardSpace causes Windows to prompt the user for an administrator password. This possibility remains an issue for future research.

The browser plug-in must scan every browser-rendered web page to detect whether it supports username-password authentication, and this may affect system performance. However, informal tests on the PassCard prototype suggest that this is not a serious issue. In addition, the browser extension can be configured so that it only operates with certain websites, thereby reducing any performance impact.

If the web browser is compromised, then an adversary could steal the RSTR (and thus the embedded username-password pair). Nonetheless, as stated in section 5.6, the same risks apply when manually entering credentials (e.g. username and password) into a browser.

Like OpenID (see section 4.5), we use URL query parameters in the PassCard prototype to exchange data between the HS site and the target HTTPS site. This could give rise to issues arising from URL size limitations (see section 2.5.3.8). However, modern web browsers can support URLs of considerable length. For example, as stated in section 2.5.3.8, Internet Explorer supports a maximum URL length of 2083 characters, which is much larger than the length of a typical PassCard-generated URL (approximately 300 characters), as informal prototype testing suggests. Therefore, URL size limitations are not likely to be a major usability barrier.

Finally, some older browsers may not be able to run PassCard, as it was built using JavaScript. However, as stated in section 6.4.4, most modern browsers support JavaScript, and so this seems unlikely to be a major usability obstacle.

¹⁰[http://technet.microsoft.com/en-us/library/cc709691\(ws.10\).aspx#BKMK_S1](http://technet.microsoft.com/en-us/library/cc709691(ws.10).aspx#BKMK_S1)

10.5 Related Work

Password managers, which store passwords in a (secure) location either on the user PC or remotely, are widely available. They typically store passwords in encrypted form and, unlike PassCard, require users to use a single master password to access the password store. Some are also capable of masking passwords, and others, much like PassCard, provide automatic password entry. Examples of password managers include open-source schemes such as Password Safe¹¹, KeePass¹², Qubliette¹³, Password Gorilla¹⁴, and PINs¹⁵ as well as commercial products such as RoboForm¹⁶, Any Password¹⁷, and Turbopasswords¹⁸.

Perhaps the most distinctive feature of PassCard is its dependence on CardSpace, whereas most of the other password managers are independent applications. PassCard can therefore benefit from the CardSpace security features, which may give users greater confidence in its use. Most importantly, it is hoped that its introduction, with immediate practical benefits to the end user, will help encourage the adoption of more sophisticated identity management schemes like CardSpace. Such schemes offer the potential for a step forward in the practice of user authentication and authorisation, with potential benefits for all legitimate parties operating via the Internet. Indeed, without simple paths to adoption for schemes like CardSpace, there is a danger that it and all the other identity initiatives will fail.

¹¹<http://passwordsafe.sourceforge.net/>

¹²<http://KeePass.info/>

¹³<http://tranglos.com/free/oubliette.html>

¹⁴<http://fpx.de/fp/Software/Gorilla/>

¹⁵<http://mirekw.com/winfreeware/pins.html>

¹⁶<http://roboform.com/>

¹⁷<http://anypassword.com/>

¹⁸<http://chapura.com/passwordmanager.php>

10.6 Conclusions and Future Work

In this chapter we have proposed a novel scheme that enables CardSpace (or indeed a wide variety of Information Card systems) to be used as a password manager. Users store their usernames and passwords in personal cards, and use such cards to transparently sign-on to corresponding websites. PassCard is based on a browser extension, and requires no changes to login servers; in particular, it does not require websites to support an Information Card-based system, such as CardSpace or Higgins. Neither does PassCard require any changes to the current CardSpace identity selector, or to default browser security settings.

PassCard uses the identity selector interface to seamlessly authenticate users to websites. It extends the use of personal cards to allow for transparent password management. Such an approach could help to extend the applicability of Information Card systems, such as CardSpace.

Planned future work includes building a scheme that enables the use of PassCard in smart phones, such as Apple's iPhone, Samsung's Galaxy, or HTC desire. A further possible topic for future work would be to investigate the possibility of building a portable version of PassCard to support users who do not have installation privileges or are forced to use untrusted machines, e.g. when travelling.

Using an Information Card System as a Password-based SSO System

11.1 Introduction

In this chapter we propose *SingleSigner*, a simple scheme that allows an Information Card system (such as CardSpace and Higgins) to be used as a password-based SSO system. It is intended to improve the usability and security of password use as well as potentially encouraging adoption of Information Card systems. We describe three alternative approaches to implementing the SingleSigner functionality, all of which take advantage of the identity selector interface to offer SSO functionality. The goal is to develop a visual approach to SSO that is transparent to both the identity selectors and the RPs. The techniques we discuss work with existing (unmodified) web servers, and, in particular, RPs are not required to support CardSpace or Higgins.

SingleSigner is an extension to the PassCard scheme presented in the previous chapter. The main novel feature of SingleSigner is the storage and subsequent use of multiple sets of credentials in a single InfoCard, allowing the provision of SSO functionality. An example use-case involves a user storing the login credentials of their favourite (or most frequently-visited) websites, e.g. a university portal, G-mail/Hotmail, Facebook, YouTube and Twitter, in a single personal card; selection of such a card automatically logs-in the user to all the relevant sites.

Like PassCard, SingleSigner operates with a variety of Information Card systems, including CardSpace and Higgins (see sections 4.3 and 4.4, respectively). However, for simplicity of presentation, in this chapter we only describe its operation with CardSpace.

The remainder of the chapter is organised as follows. Section 11.2 describes SingleSigner. In section 11.3 we present three prototype implementations using different approaches to implementing the SSO functionality, and in section 11.4 we compare them. Section 11.5 considers potential features and issues as well as possible enhancements. Section 11.6 highlights possible areas for related work, and, finally, section 11.7 concludes the chapter.

11.2 SingleSigner

We now describe SingleSigner. The idea behind SingleSigner is to store a set of user credentials in a special personal card, which, if selected, will transparently and automatically sign-on the user to a pre-defined set of websites. The parties involved are a set of RPs, a CardSpace-enabled UA (e.g. a web browser such as Internet Explorer), and a browser extension installed on the user platform implementing the protocol described in section 11.2.2.

Whenever a user visits a website requiring username-password authentication, the SingleSigner functionality can be invoked by clicking on a special icon added to the site's web page by SingleSigner. This causes the selector to run, at which point the user must select a special personal card containing the credentials for the visited site (encoded in a SingleSigner-specific format). The user will be automatically logged-on to the visited site and also to all the other sites whose credentials are stored in the selected card.

The version of the system described in section 11.2.2 and the prototypes

described in section 11.3 only work if the visited site does not use HTTPS¹. However, this limitation can be removed by following the approach employed in PassCard as described in section 10.2.2.2, i.e. via the introduction of an HS (see sections 10.2 and 10.3.3.2).

11.2.1 Prerequisites

The scheme has the following operational requirements.

- Either prior to or during use of the scheme, the user must create a special personal card, referred to as an SSOcard, containing the (URL, username, password) triples for the websites supported by this card. These triples must be stored using a specific encoding in pre-defined card fields². For ease of identification, the user can give the SSOcard a meaningful name, e.g. some representation of the corresponding websites. The user can also upload an image for the SSOcard, e.g. containing the logos of the sites whose credentials it contains.
- A special browser extension must be installed on the user platform. This must be able to implement the protocol described in section 11.2.2, including reading and modifying browser-rendered web pages, reading RSTR messages, and adding a special icon to RP web pages to enable the user to invoke the scheme.

As in PassCard (see sections 10.2.3.2 and 10.3.3.4), the browser extension, as long as it detects username-password prompts on the RP login page, will always add the special icon; this is regardless of whether or not the RP already supports CardSpace. Informal tests on the prototype implementations suggest that this will not disrupt normal operation of CardSpace.

¹Note that only the visited site must not use HTTPS; other sites included on the same card can use either HTTPS or HTTP (see section 11.5.2).

²The credential sets could alternatively be stored in a single card field, separated using a special character. However, this could make using the scheme more difficult.

11.2.2 Operation

SingleSigner operates as described below; a summary of the operation of the scheme is shown in Fig. 11.1. Steps 1, 2, 3a, 3c and 4 of the SingleSigner scheme are the same as steps 1, 2, 3a, 3d and 4, respectively, of the (HTTP-based) PassCard scheme given in sections 10.2.2 and 10.2.2.1, and hence are not described again here.

3. Browser extension → UA: Pre-processing. The browser extension performs the following processes using the login page provided by the RP.
 - b) It adds CardSpace-enabling tags to the login page, including embedding a security policy. The embedded policy must request all the card fields used by the implementation of SingleSigner, where the fields must be marked as optional. If all fields were marked as mandatory then only those cards containing data in every SSO-card field would be highlighted by the selector.
 - d) It causes a special icon to appear above the submit button, in such a way that clicking it invokes the selector.
5. User → Selector: Card Selection. The user selects and submits an SSO-card. Alternatively, the user could create and choose a new SSOcard. The selector creates and sends an RST to the SIIP, which responds with an RSTR.
6. Selector → UA: RSTR. The selector passes the RSTR to the UA.
7. Browser Extension [Intercepts] RSTR. The browser extension performs the following tasks.
 - a) It intercepts and parses the RSTR.

- b) It extracts the URL for the visited site together with the username and password associated with this URL from one of the specified fields.
 - c) It auto-populates and auto-submits the login form using the extracted username and password.
 - d) The website server verifies the credentials it receives, and, if satisfied, grants access.
8. Browser Extension [Performs] SSO. The browser extension repeats steps 7b–7d for every other website included in the user-selected SSO-card, invoking a new browser window for each site³. Note that the detailed operation of this step will vary depending on the method being used (see below).

There are a variety of ways in which the user credentials could be sent to a website in step 8. We next discuss three possible approaches to achieving this, namely: URL query parameters, cookies, and hidden form variables (see sections 2.5.3.7, 2.3.4.1, and 2.5.3.5, respectively). Note that the choice of approach only affects step 8. The three approaches are compared in section 11.4.

11.2.2.1 URL Query Parameters

For each site listed in the SSOcard, the browser extension creates a URL containing the site's address and the user credentials for this site, as taken from the SSOcard. The browser extension then invokes a browser window for each site, redirecting each window to the corresponding site URL. Finally, the browser extension reads the credentials from the URL, and auto-populates and submits the login form.

³A new browser window is invoked in order to maintain the established authenticated session with each of the websites. Following a successful authentication process, most websites typically create a short-lived cookie (a session cookie — see section 2.3.4.1) which will be deleted if the browser window is closed or if a certain period of inactivity elapses.

11. USING AN INFORMATION CARD SYSTEM AS A PASSWORD-BASED SSO SYSTEM

11.2.2.2 Cookies

From a user perspective this approach is similar to the URL query parameters approach, except that the user must append a *flag* word to each credential triple when the SSOcard is created; this word must be manually removed once the SSOcard has been used.

The browser extension first examines the RSTR to detect if the flag word is present at the end of each set of credentials; if so, it runs in exactly the same way as the URL query parameters protocol, except that, before the browser extension automatically populates and submits the login form, it sets a persistent cookie (see section 2.3.4.1) in order to store the username and password values for future logins. A cookie is thus created for each website whose credentials are stored in the SSOcard. If the flag word is not present, then the extension invokes a browser window for each site whose credentials are included in the SSOcard. It then recovers the user credentials from the appropriate cookie⁴, and uses them to auto-populate the site login form, which it auto-submits.

Note that, unlike the other approaches, here the user credentials are not retrieved from a hidden form variable or from a URL, thereby avoiding the display of username and password values in the browser address bar. This provides protection against shoulder-surfing attacks.

11.2.2.3 Hidden Form Variables

In this approach the browser extension creates a separate invisible HTML form (containing hidden variables) for each site listed in the SSOcard. Each form is auto-filled using the user credentials and then auto-submitted. The extension opens a new browser window for each site contained in the SSOcard.

⁴If the cookie expires or is removed, the browser extension will fail to find an appropriate cookie and will then prompt the user to add the flag word to the end of each set of credentials in the relevant SSOcard.

To make this approach work, certain RP-specific information, notably the URL of the login server and the names given to the username and password fields, must be available to the browser extension independently of the SSOcard. This means that every time a new SSOcard is created, or an existing SSOcard is modified to include a new credential set, the browser extension must be modified to incorporate this information.

11.3 Implementation

We now describe three proof-of-concept prototypes implementing the SingleSigner scheme presented in section 11.2.2, one for each of the three described approaches to realising step 8. As is the case with the PassCard prototype (see section 10.3), the prototypes operate with both the CardSpace and the Higgins identity selectors without any modification.

11.3.1 Shared Properties

Each prototype is coded as a browser plug-in in JavaScript, executed using a C#-driven BHO (see section 6.4.2). In each case, SingleSigner can readily be enabled or disabled using the add-on manager in the Internet Explorer Tools menu.

11.3.1.1 SSOcard Format

The prototype permits credential sets to be stored in any of the 14 personal card fields with the exception of the *birthday* and *gender* fields (which cannot contain arbitrary strings). Credential sets must be stored in the format:

<URL> <username> <password>

where the fields are separated by a single space character.

11. USING AN INFORMATION CARD SYSTEM AS A PASSWORD-BASED SSO SYSTEM

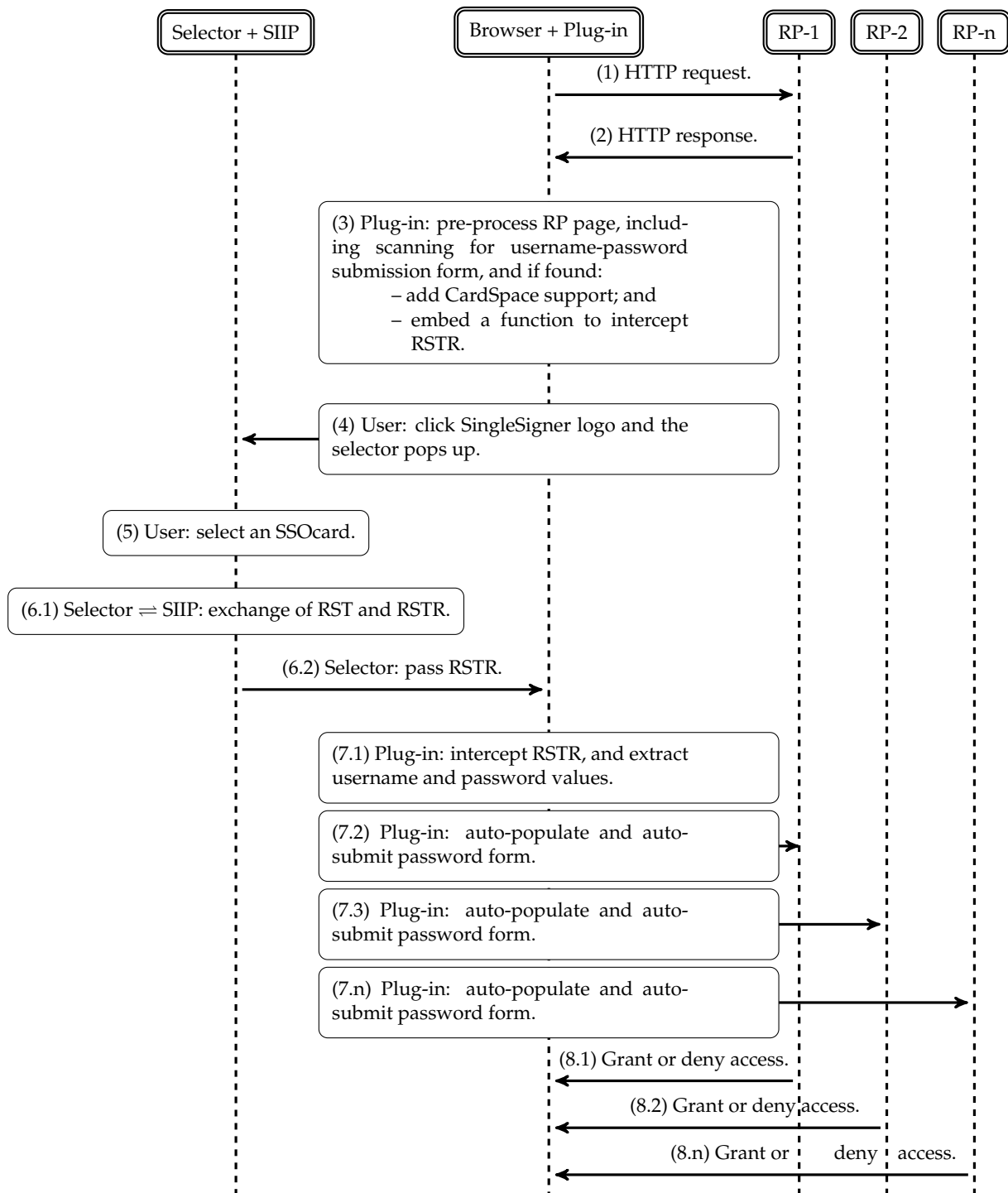


Figure 11.1: SingleSigner Operation

11.3.1.2 Operation

In step 3 of the protocol (see section 11.2.2) the plug-in processes the RP web page in the following way. The implementation of steps 3.1, 3.3, and 7.1 of

the SingleSigner prototype is precisely the same as steps 3a, 3.d, and 7.a, respectively, of the PassCard prototype given in section 10.3.2, and hence are not described again here.

3.2 It adds an HTML object tag that allows the user to invoke the selector. Within the object tag, it sets the param tags to indicate that the RP security policy requires SSOcards to contain at least one (mandatory) field, namely the *first name* field, and to also include 11 (optional) fields, namely *last name, email address, street, city, state, postal code, country/region, home phone, other phone, mobile phone, and web page*.

From a user side, marking at least one field as mandatory means less computation (as explained below) and, ultimately, a faster authentication process, hence helping user acceptability. In the SSOcard selection step, if the SSOcard only contains one credential set in a mandatory field then the user only needs to choose the SSOcard and click the *send* button. If the field was optional, then the user would first need to tick the optional field before clicking the *send* button. From an operational perspective, an RP security policy must contain at least one mandatory claim; of course this claim could be the PPID claim (see section 4.3.11).

3.4 It inserts the SingleSigner logo (see Fig. 11.2) in the login page, in such a way that it appears just before the submit button. The logo is associated with an *on-click* event, so that, if clicked, the selector is invoked (after calling the added function). As in PassCard, to cater for users with sight difficulties or web browsers configured not to display images, a text field can replace the SingleSigner logo.

In step 7, the plug-in performs the following steps.

7.2 It parses the intercepted RSTR message and extracts the value of the *first name* field as well as the values of any other optional fields present, thus learning the set of websites supported by the SSOcard.

11. USING AN INFORMATION CARD SYSTEM AS A PASSWORD-BASED SSO SYSTEM

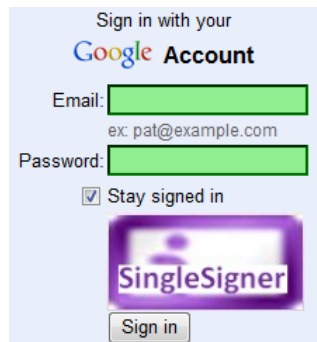


Figure 11.2: SingleSigner Logo

7.3 It auto-fills the username and password fields and auto-submits the login form of the currently visited website using the JavaScript *click()* method. It detects the correct username-password values for the visited site by comparing its domain name with the URLs contained in the specified fields of the RSTR.

11.3.2 The URL Query Parameters Prototype

11.3.2.1 Operation

Steps 3.1–7.3 are precisely the same as those described in section 11.3.1.2.

7.4 For each other site included in the SSOcard:

- (a) the plug-in invokes a new browser window using the JavaScript built-in method *open.window()*, that retrieves the site's login-page (using the URL provided in the SSOcard). Note that it sends the username-password values embedded in the URL (i.e. URL query parameters) as part of this HTTP request; and
- (b) using the login-page returned in the previous step, the plug-in:
 - (i) parses the URL query parameters to obtain the values of the username and password;
 - (ii) locates the username, password, and submit fields;

- (iii) auto-populates the username and password fields with the username and password values; and
- (iv) auto-submits the form using the JavaScript click method.

11.3.2.2 Protecting Credentials

This approach involves embedding the username and password in the URL. Thus, if they are embedded in clear text, they will be vulnerable to shoulder-surfing attacks. To address this potential problem, in step 7.4a the prototype encrypts these values using AES in CBC mode (see section 2.4.2.1), and decrypts them in step 7.4b.

The AES key used for username-password encryption is stored in the plug-in. This would be a security issue if the same key was used by every copy of the plug-in, but a unique random key can be generated at the time the plug-in is installed. The presence of the key on the user platform does not significantly increase the risks to credential secrecy, since the credentials must in any event be stored on the user platform.

Embedding the credential values in a URL could also cause problems because of URL size limitations (see section 2.5.3.8); however, this is not likely to be a major problem here since, much like PassCard, the amount of data involved is relatively small (see section 10.4.3). The use of encryption also results in a slight performance delay.

11.3.3 The Cookies Prototype

11.3.3.1 SSOcard Format

In this case the format of the SSOcard is identical to that described in section 11.3.1.1, except that, at the time of card creation, a flag word must be appended to each credential triple (see also section 11.2.2.2). The prototype expects to find a string of the form *cookie9*, where 9 indicates the lifetime (in days) of the persistent cookie created by the plug-in. After first use, this flag

11. USING AN INFORMATION CARD SYSTEM AS A PASSWORD-BASED SSO SYSTEM

word must be removed by the user (and added back if the credential cookie expires).

11.3.3.2 Operation

Steps 3.1–7.3 are precisely the same as those described in section 11.3.1.2. This is followed by step 7.4, which is precisely as in section 11.3.2.1, except that the following step is added between steps 7.4.b.iii and 7.4.b.iv:

- the plug-in (as described in section 11.2.2.2) first examines the RSTR for the flag word; if present it creates a persistent cookie containing the username and password. If it is not present then it recovers the username and password from the cookie.

11.3.3.3 Protecting Credentials

This approach involves storing the username and password in a cookie. Thus, if they are stored in clear text, they will be readable by anyone with temporary access to the user platform. As in section 11.3.2.2, this threat can be mitigated by encrypting the data in the cookie using a key known only to the browser plug-in.

11.3.4 The Hidden Form Fields Prototype

11.3.4.1 Initialisation

In this approach, the user must make certain modifications to the plug-in source code. The user must first obtain the URL of the login server for each website included in an SSOcard; this can be found by viewing the HTML source of the login web page and retrieving the action URL of the login form. The user must also obtain the names given to the username and password input fields⁵, which can also be found from the HTML source of the login web page. The user must then insert the URL and the names of the

⁵This is important since the site's login server will use these names to retrieve username-password values from the HTTP POST array.

username and password fields into the plug-in source code, in the specified way.

11.3.4.2 Operation

Steps 3.1–7.3 are precisely the same as those described in section 11.3.1.2.

7.4 For each other site included in the SSOcard, it:

- (a) creates an invisible HTML form containing at least two hidden input variables, and then auto-fills each variable with the corresponding username or password;
- (b) creates a new browser window using *open.window()*, a JavaScript built-in method; and
- (c) auto-submits the invisible HTML form.

11.3.4.3 Operational Issues

Prototype testing reveals that some website login servers impose restrictions on externally posted/submitted forms for security reasons. That is, if a user is currently visiting site *a.com* and the browser plug-in submits/posts a login form to site *b.com*, then access to a protected resource in domain *b.com* will not be granted even if the user credentials are correct.

11.4 Comparison

We next compare the three approaches in terms of usability and performance.

- **Usability.** The URL query parameters approach only requires entry of (username, password, URL) triples into SSOcards, and hence is clearly the most usable. The other two methods either require manual modifications to the plug-in source code (likely to be beyond most users) or

the additional overhead of adding flag words to SSOcard entries (and subsequently removing them).

- **Performance.** The hidden forms approach has the advantage that, once the browser windows are opened, no further processing is required. Less processing is required for the URL query parameters approach than the cookies approach.

11.5 Discussions

We now consider certain properties of SingleSigner.

11.5.1 Features

SingleSigner shares the security and usability properties of PassCard, as described in sections 10.4.1 and 10.4.2.

In addition, SingleSigner possesses the property that the compromise of any one password does not threaten the confidentiality of other passwords, or compromise user authentication at other sites (as would be the case, for example, if a password for an OpenID/Liberty IdP was compromised). That is, the only single point of failure for SingleSigner is the user platform itself. Failing to access a site whose credentials are included in an SSOcard will not impede access to other sites covered by the same card.

Like PassCard (see section 10.4.2), SingleSigner is flexible, since users can choose whether or not to use it simply by electing to click the SingleSigner logo (or not).

11.5.2 Limitations

SingleSigner shares the PassCard limitations, as described in section 10.4.3. As noted in section 11.2, SingleSigner as described in this chapter does not work as intended if the website at which the initial authentication takes place uses HTTPS. This is because, if such a website has a certificate, then

the selector will, by default, encrypt the RSTR using the public key of the requesting site. The plug-in does not have access to the site's private key, and hence will be unable to decrypt the token to obtain the username-password values. However, if the site at which the initial authentication takes place uses HTTP, then SingleSigner will work as intended even if all other sites included on the same SSOcard use HTTPS. As stated in section 11.2, this shortcoming can be avoided by adopting the approach described in section 10.2.2.2. That is, if the target site uses HTTPS, the user browser can be redirected to an arbitrary site using HTTP (the HS — see sections 10.2.3.2 and 10.3.3.4). The solution for SingleSigner works precisely as for PassCard.

Use of SSO systems in general, including SingleSigner, could be a threat to user privacy. As stated in section 3.6, user interactions on the web could be linked to build a unique user profile. For example, the HTTP referrer field and cookies could be employed to help build such a profile.

11.5.3 Enhancements

The fact that SingleSigner automatically creates a browser window for each site included in an SSOcard (as described in section 11.2.2.1) could be somewhat intrusive for the user, although whether or not this is a problem in practice depends partly on the number of sets of credentials included in a single SSOcard. We therefore propose an alternative method of operation. This alternative method operates as in step 8, except that, after invoking a new browser window for each site in the SSOcard, the browser extension stores (e.g. in a cookie) the URL of the page to which the user is granted access following a successful authentication. The browser extension then attempts to close (but not sign-out) each page it has invoked; the user-visited page will, of course, remain open since it was not invoked by the extension. When a user later visits a website included in the submitted SSOcard, the browser extension will auto-redirect the user to the logged-in page that the

11. USING AN INFORMATION CARD SYSTEM AS A PASSWORD-BASED SSO SYSTEM

extension stored earlier⁶ and then terminate. As long as the main browser session is still live, the user logged-in session at each site included in the SSOcard should still be valid; however, the session may be invalid if the main browser session is closed or if a certain period of inactivity, as determined by the site server, has elapsed. This method has been successfully tested with the URL query parameters approach.

11.6 Related Work

A very wide range of Internet SSO schemes⁷ have been proposed [78, 195]. We observe that, using the taxonomy of [195], SingleSigner is a local pseudo-SSO scheme in that the credentials are stored locally and the RPs are not aware of the operation of the scheme. Other examples of such schemes include Novell's SecureLogin⁸, Passlogix V-GO⁹ and Protocom's SecureLogin¹⁰. Automatic form-fillers, e.g. the automatic form completion functions of popular web browsers such as Internet Explorer and Firefox, can also be regarded as local pseudo-SSO schemes [195].

Like PassCard, perhaps the most distinctive feature of SingleSigner is its dependence on CardSpace, whereas other SSO systems are independent applications. SingleSigner can thus benefit from the CardSpace security features, which may give users greater confidence in its use.

⁶If such a page was not stored, then some website servers would prompt the user to re-authenticate.

⁷Examples include Passport, OpenID, Liberty Alliance (Kantara), Shibboleth, Facebook Connect (see sections 4.2, 4.5, 4.7, 4.8, and 4.6.4, respectively), Athens, Kerberos, AccessMatrix USO, Central Authentication Service (CAS), COMA, CoSign, Distributed Access Control System (DACS), Enterprise Sign On Engine, Evidian Enterprise SSO, FreeIPA, Global Login System, Imprivata OneSign, JBoss SSO, Open Source Single Sign On Server (JOSSO), myOneLogin, OneLogin, OpenAM, OpenASelect, Passlogix, Secure Network Communications, Smart card, Tiger OneConnect, and Ubuntu Single Sign On (see http://en.wikipedia.org/wiki/List_of_single_sign-on_implementations).

⁸<http://www.novell.com/products/securelogin/>

⁹<http://www.passlogix.com/sso>

¹⁰<http://www.protocom.cc>

11.7 Conclusions and Future Work

In this chapter we have proposed SingleSigner, a simple scheme that allows an Information Card system (such as CardSpace and Higgins) to be used as a password-based single sign on system. Three related approaches to achieving the single sign on functionality using CardSpace/Higgins were discussed. In each case users are able to store their credentials for a set of websites in a personal card, and use it to seamlessly sign-on to the relevant sites. The approaches do not require any changes to login servers or to identity selectors and, in particular, they do not require websites to support CardSpace or Higgins.

The schemes use the identity selector interface to seamlessly sign-on users to password-protected websites. It extends the use of personal cards to allow for such functionality, thereby both improving the usability and security of passwords as well as encouraging CardSpace/Higgins adoption.

Planned future work includes building a portable version of the Single-Signer scheme to help roaming users who might not have installation privileges or might not be able to use their personal machines, e.g. in Internet cafes, airport lounges, etc. In addition, we plan to investigate the possibility of extending SingleSigner to support single sign off.

Enhancing User Authentication in Information Card Systems

12.1 Introduction

This chapter describes a scheme designed to address a potential security limitation in Information Card-based identity management systems. It operates with a variety of such systems, including CardSpace and Higgins (see sections 4.3 and 4.4, respectively). For simplicity of presentation, in this chapter we describe its operation with CardSpace.

As discussed in section 4.3.14.1, one major limitation of CardSpace (and many other Information Card systems) is that anyone with access to a user account on the user platform can also access and use the InfoCards. That is, by default, CardSpace does not provide access protection for the identity selector. To address this issue, CardSpace allows individual InfoCards to be PIN-protected. The entire user account can also be password-protected. Whilst the use of passwords and PINs for InfoCard protection can help, it does not completely solve the problem, not least because one of the fundamental design goals of CardSpace is to reduce reliance on password authentication.

We address this limitation through the introduction of a second authentication factor to be used in conjunction with CardSpace authentication. This additional means of user authentication involves an OTP supplied to the user by a standard mobile device capable of receiving SMS messages. Such

devices are ubiquitous, making the system almost universally applicable. The system also provides two-factor authentication (see section 2.4.3.2), the first factor being possession of the PC containing the InfoCard and the second factor being possession of the appropriate mobile phone. Two factor authentication is typically considered a strong form of authentication [154].

In the scheme we propose, during the process of user authentication on a PC using an Information Card system, a random and short-lived OTP is sent to the user's mobile device; this must then be entered into the PC by the user when prompted. The scheme does not require any changes to login servers, identity selectors, or to the mobile device itself. Note that this scheme, if correctly implemented, would enhance the security of PassCard and SingleSigner, described in chapters 10 and 11, respectively.

The wide use of Windows, recent versions of which incorporate CardSpace, means that any enhancement to CardSpace security is likely to be of significance for large numbers of identity management users and RPs. In addition, the use of a mobile phone to enhance CardSpace-based authentication is attractive since users are neither required to remember any new passwords nor obliged to use any additional hardware. Furthermore, many RPs may not accept the burden of supporting a second authentication factor (e.g. SMS-based authentication), unless there is a significant financial incentive or if forced to do so for legal or regulatory reasons. As a result, a client-side technique for supporting SMS authentication for CardSpace-enabled RPs could be practically useful. Such a technique avoids any impact on the performance of the server since the additional overhead is handled by the client.

The remainder of the chapter is organised as follows. Section 12.2 describes the scheme. In section 12.3 we discuss implementation issues, and in section 12.4 we provide a security analysis. In section 12.5 we present a prototype realisation, and section 12.6 highlights possible areas for related work. Finally, section 12.7 concludes the chapter. Much of the material in

this chapter has been published [13].

12.2 The Scheme

We next describe the novel scheme, covering relevant operational aspects.

12.2.1 System Entities

The entities involved in the scheme are:

- a CardSpace-enabled RP;
- a CardSpace-enabled UA (e.g. a web browser capable of invoking the identity selector, such as Internet Explorer);
- software installed on the user platform (referred to throughout as the *adaptor*) implementing the scheme described in section 12.2.2 below; and
- a handheld device capable of receiving SMS messages (e.g. a mobile phone).

The SMS allows mobile phones and other cellular network devices to exchange short messages of at most 160 or 70 characters, depending on whether a Latin or non-Latin alphabet is used [106, 156]. SMS is supported by all GSM and 3G handsets, and it is very widely used.

The adaptor could be implemented as a browser extension, capable of scanning and modifying browser-rendered pages, and intercepting RSTR messages. In addition, it must be able to generate and send a random, short-lived OTP to the user's mobile phone, and provide a means for the user to enter the OTP. Prior to use of the protocol, the browser extension must be installed on the client PC and provided with the phone number of the user's mobile phone. Implementing the scheme as a browser extension means that the RP cannot employ the optional STS (see section 5.3.3).

12.2.2 Operation

The system operates as follows; a summary of the protocol is shown in Figs. 12.1 and 12.2. Steps 1, 2, 4–7, and 10 are the same as steps 1, 2, 3–6, and 8, respectively, of the CardSpace personal card protocol given in section 4.3.9.1.

3. Adaptor \rightarrow UA. The adaptor scans the login page to detect whether the RP website supports CardSpace. If so, it proceeds; otherwise it terminates. On proceeding, the adaptor processes the RP login page, including embedding a function to intercept the SIIP-issued RSTR.
8. Unlike in the standard case, the RSTR does not reach the RP; instead the adaptor performs the following steps.
 - a) Selector \rightarrow Adaptor: RSTR. The adaptor intercepts the RSTR and temporarily stores it.
 - b) Adaptor: generates OTP. The adaptor computes (and temporarily stores) a random, short-lived OTP.
 - c) Adaptor \rightarrow Mobile Phone: OTP. The adaptor sends the OTP to the user's mobile phone in an SMS message, sent via an HTTPS-protected connection to the SMS centre or SMS gateway of a wireless carrier or SMS service provider. This method is adopted because it does not require a special application to be installed on the user's mobile phone, which may not be possible in non-smart phones. In addition such an approach has a better transmission rate than other methods such as Bluetooth or infrared (see section 12.3.2.1).
9. User \rightleftharpoons UA \rightarrow RP. The adaptor prompts the user to enter the OTP, which the user reads from the phone display¹. The adaptor verifies

¹Note that if the mobile phone and/or the SIM card are PIN-protected, then the user must first enter the correct PIN(s); this constitutes a third authentication factor.

that the entered OTP matches the one it just generated. The OTP must be entered within a defined interval, e.g. of 10 minutes, after its generation, or else the adaptor will delete the RSTR and provide an error message to the user. If all the checks succeed, the protocol continues and the adaptor submits the RSTR to the RP.

- (3) A → UA: pre-process RP page, where A is the adapter, and UA is the user agent.
 (8) S → [A: generate OTP] → M, where S is the selector and M is the mobile device.
 More specifically:
 (8.1) S → A: RSTR;
 (8.2) A: generate OTP; and
 (8.3) A → M: OTP.
 (9) U → UA: OTP, where U is the user.

Figure 12.1: Summary of the Protocol

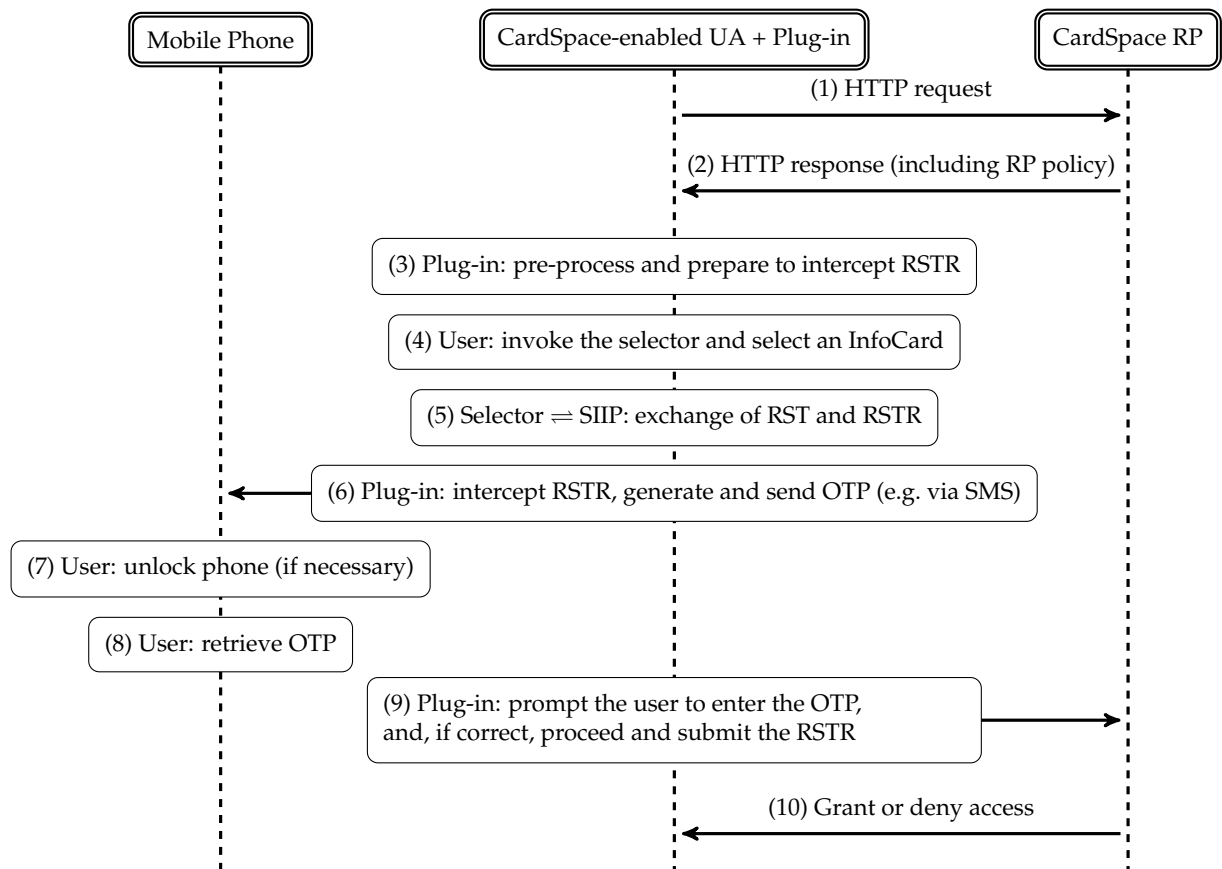


Figure 12.2: Protocol Exchanges

12.3 Discussion

We now consider implementation issues, possible variants and potential advantages of the scheme.

12.3.1 Implementation Issues

The length of the OTP must be carefully chosen to achieve an acceptable balance between security and usability. To maximise usability, we propose the use of a 4-character OTP made up of lower case letters and digits (excluding 0, i, j and o). This gives a total of 32^4 possible OTPs (i.e. just over a million), which is roughly 100 times the number of possible 4-digit PINs commonly used for bank cards.

12.3.2 Variants of the Scheme

12.3.2.1 OTP Transmission

In the scheme described above, the OTP is sent from the client to the mobile device in an SMS message. Whilst convenient, this has cost implications and may also involve a delay of a few seconds. Possible alternatives include sending it via Bluetooth, infrared or a USB/serial cable. Such approaches have the advantage of avoiding the SMS messaging costs but require both devices to support the relevant technologies. The main disadvantage of such approaches is the need to install a special application on the phone; this will rule out non-smart phones, and significantly increase the complexity of setting up the scheme.

A further alternative would be to use a messaging service other than SMS for the OTP transfer (e.g. instant messaging or email); like the use of the SMS service, such an approach would avoid the need to install additional applications on the phone, provided that the phone supports the relevant service.

12.3.2.2 OTP Entry

In the scheme as described above, the user manually enters the OTP, which is potentially inconvenient and time-consuming (although the use of a 4-digit OTP, as described in section 12.3.1, should minimise inconvenience). An alternative approach would be to send the OTP back automatically, e.g. via an SMS message sent to the SMS gateway, from where the adaptor could retrieve it. Whilst convenient, such a process could be costly, since use of the SMS gateway would incur additional messaging costs.

12.3.2.3 SAML Extension

As part of step 9 of section 12.2.2 the adaptor could create a new SAML token containing the SIIP-issued RSTR and an additional SAML field indicating that the user has been authenticated using an SMS-transmitted OTP. Of course, the RP would need to be modified to be able to process such a token, although this is likely to be straightforward. If the RP trusts that the correct adaptor is running unmodified on the user platform, then this authentication statement could potentially give the RP added assurance of user authenticity. However, this is a significant trust assumption (see section 5.5).

12.3.3 Advantages

Like other OTP-based authentication systems, the scheme reduces exposure to shoulder-surfing attacks and also helps to thwart key loggers.

The scheme does not require users to remember new passwords for each new account; this could reduce the risks arising from password re-use, writing passwords down in insecure ways, and use of easily-guessable passwords.

In addition to strengthening user authentication, the scheme could also serve as an intrusion detector. If the user receives an unexpected OTP, then

it could be deduced that there is a security breach.

Finally, the scheme operates transparently to external parties, and hence does not require any changes to RPs or identity selectors.

12.4 Security Analysis

12.4.1 Threats to the Mobile Device

If a mobile phone or SIM (Subscriber Identity Module) is lost, stolen or borrowed, then it might be possible to access an OTP from the SMS inbox. However, this will be of no value without access to the corresponding PC, and the OTP will expire a short time after generation. Moreover, a lost phone or SIM is likely to be reported by its owner, causing the SIM to be deactivated, which means that the usefulness of such a stolen device for impersonating a user will be very limited.

12.4.2 Threats to the Supporting Infrastructure

An attacker with temporary access to the PC, but without the mobile phone, could attempt to intercept the OTP whilst it is being transmitted from the PC to the phone. However, the communication link between the SMS gateway and the PC is protected using HTTPS, and the connection between the visited mobile network and the mobile phone is protected by the air interface encryption mechanism of the mobile network [106, 156]. This leaves the SMS gateway and the SMS network itself as the only sources of such a threat, and routinely compromising either the gateway or the SMS network for such a purpose seems unlikely to pose a significant threat in practice.

12.4.3 Threats to the PC

12.4.3.1 Exhausting the User's SMS Credit

An adversary who has access to the user's PC but does not possess the user's mobile phone could cause the system to repeatedly send SMS mes-

sages, resulting in exhaustion of the user's credit at the SMS gateway. This risk can be mitigated in the following ways.

1. If a user receives an unexpected SMS containing an OTP, then the user should immediately change their password at the SMS gateway. This will deny the adversary the ability to send any further SMS messages from the user's PC.
2. The browser extension could implement a simple, client-based, lock-out mechanism using cookies. That is, if the correct OTP is not entered within three attempts, the browser extension could write a persistent cookie (see section 2.3.4.1) to the client PC which will cause the current attempt to log-in to the RP to be terminated. The browser extension would then generate a special lock-out OTP and send it to the user's mobile phone. The next time that the user attempts to log-in to the same RP, the browser extension (before invoking the selector) would prompt the user to enter the lock-out OTP, and would only proceed if the correct OTP is entered. Although this solution may help to discourage an attacker, it is not foolproof, since cookies could be manually deleted on the user platform, and an attacker could arrange for OTP-bearing SMSs to be sent to a large number of different domains.

12.4.3.2 Disabling the Browser Extension

If the system is configured so that it is possible to disable the OTP adaptor, then a knowledgeable intruder could defeat the protection provided by the scheme. Therefore, a robust implementation of the scheme proposed in section 12.2.2 must not allow an adversary to disable it. That is, the system must be configured to oblige CardSpace users to use the OTP adaptor.

However, browser extensions can be enabled or disabled at will by anyone who has access to a Windows user account. So an adversary with access to the appropriate account on the PC could simply disable the browser ex-

tension and thereby cause CardSpace to operate without the OTP enhancement.

It may be possible to remove this threat, at least partially, by installing the browser extension so that administrator privileges are required to disable it, and also persuading the PC owner to log-in using a non-administrator account. It may also be possible to make use of UAC (see section 10.4.3), so that disabling a browser extension causes Windows to prompt the user for an administrator password.

Ultimately, it would be desirable to implement the scheme described in section 12.2.2 as an integral part of CardSpace, thereby negating this threat. In such a scenario, each InfoCard might be given a selectable field to indicate whether SMS-based authentication is required. A user could thus choose to SMS-protect an important InfoCard by simply selecting the appropriate field.

12.4.3.3 Exploiting CardSpace Backup Facilities

The CardSpace backup facilities could be exploited to allow an InfoCard to be exported from one PC to another, thereby avoiding the protection provided by the scheme proposed here. An attacker could, for example, export a personal card to a USB memory stick, and then reload the card onto his or her own PC in order to impersonate the card owner. An exported card could also be transferred as an email attachment. This risk could be mitigated using countermeasures similar to those discussed above.

12.5 Prototype Realisation

The prototype is coded as a browser plug-in in JavaScript, executed using a C#-driven BHO (see section 6.4.2). The prototype operates with both the CardSpace and the Higgins identity selectors without any modification.

12.5.1 User Registration

Prior to use, the prototype user must have accounts with a CardSpace-enabled RP and an SMS gateway service provider, e.g. Clickatell (<http://clickatell.com>). The prototype plug-in provides step-by-step instructions to assist the user to enter their mobile phone number and SMS account login details (e.g. username and password).

12.5.2 Prototype Operation

In this section we consider specific operational aspects of the prototype. We refer throughout to the numbered protocol steps given in section 12.2.2.

The implementation of step 3 is precisely the same as that of step 3 of the CardSpace-OpenID integration prototype described in section 8.4.2, except that steps 3.1 (c) to 3.1 (e) are skipped.

In step 8 the plug-in uses the DOM to perform the following steps.

- 8.1 It intercepts the RSTR sent by the selector using the interception function added by the plug-in.
- 8.2 It generates a 4-character, random OTP (see section 12.3.1). It also starts a 10-minute time counter.
- 8.3 It builds an HTTPS-based URL, bearing the user's mobile phone number, the user's account login details, and the OTP.
- 8.4 It automatically invokes the URL in a new, small browser window. This process will cause the OTP to be sent to the SMS gateway via a secure TLS/SSL channel. On receipt of the OTP, the SMS gateway delivers it to the user's mobile phone in an SMS message.

In step 9 the plug-in performs the following steps.

- 9.1 It prompts the user to enter the OTP, using a JavaScript pop-up box.

9.2 It verifies the user-entered OTP by comparing it with the version it previously generated (in step 8.2), ensuring that the OTP has been entered within the 10-minute time window. If the verification succeeds it proceeds to the next step. If the verification fails, the user is allowed to try again. However, if verification fails on three successive occasions, the plug-in terminates the login process and writes a persistent cookie to prevent the user from logging into this RP using the same browser for a defined time period, e.g. 24 hours. This process operates as follows.

On the first occasion that the system is used with a particular RP, or if the previously written cookie has expired and been deleted, the plug-in writes a persistent cookie containing the number of failed OTP entry attempts for this RP (i.e. either zero if the attempt is successful or one if the attempt fails) and with a lifetime of 24 hours. Whenever the system is used subsequently the presence of this cookie is checked; if it is present then the current number of failed OTP entry attempts it records is checked — if it is equal to three then no SMS is sent and the RSTR is blocked, i.e. the system is locked out and can only be unlocked if the user enters the special lockout OTP. If it is less than three then the system proceeds. If the OTP entry attempt succeeds then a new cookie is written containing the value zero; if the OTP entry attempt fails, then a new cookie is written containing a value one larger than the previous value.

9.3 It creates an invisible HTML form with method attribute set to POST.

9.4 It writes the entire RSTR into the invisible HTML form as a hidden variable, with the name attribute of this variable set to the name of the CardSpace object tag.

9.5 It writes the URL of the CardSpace-enabled RP into the action attribute of the invisible form.

9.6 Finally, it auto-submits the HTML form (transparently to the user), using the JavaScript inherent method *submit*.

The prototype has been successfully tested with *Clickatell* (see section 12.5.1), an experimental implementation of a CardSpace-enabled RP, the current (unmodified) CardSpace and Higgins identity selectors, and with a standard mobile phone.

12.5.3 Practical Issues

The plug-in must scan every HTML web page to check whether it supports CardSpace, and this may affect system performance. However, informal tests on the prototype suggest that this is not a serious issue. In addition, the plug-in can be configured so that it only operates with certain websites.

If the web browser is compromised, then an adversary could steal the RSTR and the OTP and use them to impersonate the user. However, as stated in section 5.6, the same risks apply when manually entering credentials (e.g. username and password) into a browser.

Finally, some older browsers may not be able to run the prototype plug-in, as it was built using JavaScript. However, as stated in section 6.4.4, most modern browsers support JavaScript, and so this seems unlikely to be a major usability obstacle.

12.6 Related Work

Using a mobile device as a means of user authentication is attractive because of the ubiquity of mobile phones, and many such schemes have been proposed. Examples of schemes in which a mobile phone is used to authenticate a user to a remote server include the following.

- Hart et al. [109] proposed a scheme in which user credentials (i.e. username and password) are stored in a Java-enabled SIM card. When the

user visits a website, the browser extension requests the user credentials for this site from an SMS gateway, which then sends a specially formatted SMS message to the appropriate SIM card. The SIM card responds with another SMS message containing the requested credentials, and the SMS gateway forwards them to the browser extension via an HTTPS channel. The browser extension then auto-submits them to the visited site. The scheme requires the user to possess a SIM capable of hosting an application, and for the user to load an appropriate application into it. It also has an SMS messaging cost at least twice that of the scheme described in this chapter.

- Wu et al. [230] and Jammalamadaka et al. [140] proposed schemes involving a combination of a third party proxy, which stores the user credentials, and a mobile phone. The schemes are designed for use in cases where an untrusted PC, e.g. in an Internet kiosk, is used to access a remote website, and they avoid the need for the user to enter long-term secret credentials into such a PC (see also [196]). The phone is used to explicitly authorise the proxy to release the credentials to the remote website. Unfortunately, not only is the use of a proxy a potential security and reliability threat, but the PC must be configured to use the proxy. This latter requirement is not only potentially inconvenient, but in some cases may be impossible to meet since the user may not have the necessary permissions to change the browser settings.
- Florêncio and Herley proposed URRSA (Universal Replay-resistant Secure Authentication) [93], an OTP-enhanced service (based on a reverse proxy [160]) that allows users to access password-protected websites. The URRSA service does not require changes to login servers. A list of 10 different encrypted copies of a long-term user password (effectively functioning as OTPs) is generated and sent to the user's mobile phone using SMS; the corresponding decryption keys are stored

at the URRSA server. A user wishing to access a protected site first navigates to the URRSA site and enters the URL and user ID of the account to be accessed. The user then enters the appropriate OTP from the current list, allowing the URRSA server to decrypt and temporarily store the real password. The URRSA server then fetches the previously registered login page and prompts the user to click the submit button; the login process then proceeds. The user process for this scheme is relatively complex, and new lists will need to be downloaded fairly frequently, increasing the burden on the user.

- Aloul et al. [18] proposed a system that involves using a PIN-protected mobile phone as a token for OTP generation. Additionally, an SMS-based mechanism is implemented as both a backup mechanism for retrieving the OTP and as a possible means of client-server synchronisation. This method requires both the client and server to pay to send SMS messages. Unlike the scheme described here, the mobile phone must support J2ME (Java 2 Platform, Micro Edition) [159], and, prior to use, the user must install a special application into the phone.
- Mannan et al. [163] and Alqattan et al. [20] proposed similar schemes in which the entry of user authentication credentials is accomplished using a trusted handheld device, e.g. a PIN-protected mobile phone. For example, in the MP-Auth (Mobile Password Authentication) scheme [163], the mobile device encrypts the password using the end server's public key before passing it via an untrusted machine to the remote server. However, unlike the scheme described in this chapter, these schemes require changes to login servers and also require users to possess J2ME-enabled mobile phones.
- Schuba et al. [212] proposed the *Internet ID* approach, in which a mobile phone is used to provide user authentication to a Liberty-enabled IdP. We outline the variant most similar to the scheme described in

this chapter. A Liberty IdP generates a random sequence of symbols, and sends them to the user's mobile phone in an SMS message. Simultaneously, these symbols are shown on the PC browser, and the user is required to confirm to the phone that the browser-displayed symbols are the same as those in the SMS message, e.g. by clicking a link on the WAP (Wireless Application Protocol) page on the mobile phone. Although this system does not require the user to type anything, it does require changes to the operation of Liberty IdPs.

- Jørstad et al. [144] proposed a scheme which supports interoperability between CardSpace and Liberty. It uses a mobile phone for user authentication to the IdP; the IdP sends an SMS message to the user, and, in order to be authenticated, the user must confirm receipt of the message (see also section 6.5). Much like the Internet ID approach [212], this method requires changes to the operation of the IdP.

Examples of schemes in which a mobile phone is used to authenticate a user to a local PC include the following.

- Lach [154] proposed *MOTH* (derived from *Mobile Authentication*), a scheme in which a workstation and a mobile device communicate using Bluetooth, and authentication is realised using digital signatures. Unlike in our scheme, the mobile device in the *MOTH* system must be able to run Java midlets. To avoid an attacker bypassing the scheme, a *MOTH*-conformant PC must be configured to only use the *MOTH* service for authentication, and not to fall back to password authentication. Analogously, the scheme described in this chapter must be configured to oblige the use of the adaptor with CardSpace (see section 12.4.3.2). In *MOTH*, binding a user to a public key remains a challenge.
- Abdulhameed et al. [1] proposed a method which uses a Bluetooth-enabled mobile phone. The user's PC communicates with the phone

via a Bluetooth link, and public-key cryptographic techniques are used to provide mutual authentication between the PC and the phone. The PC periodically senses the phone to ensure that the user is still present; if the mobile phone moves out of range, the PC is configured to take certain measures to raise the security level. It is unclear from the description provided [1] whether this form of authentication could be disabled by an attacker so that the PC reverts to password-based user authentication, a possible means of circumventing the scheme. Not only must the mobile phone be Bluetooth-enabled, but it must also support Java to provide certain cryptographic and authentication services.

Finally note that the scheme proposed in this chapter falls somewhere in between the two classes described above, in that it provides authentication to a local PC in such a way that it enables authentication to a remote site to continue in a more secure way.

12.7 Conclusions and Future Work

In this chapter we have proposed a simple scheme for using a mobile device to enhance user authentication in Information Card-based systems, such as CardSpace and Higgins. During the process of user authentication on a PC using an Information Card system, a random and short-lived one-time password is sent to the mobile device; this must then be entered into the PC by the user. The scheme does not require any changes to login servers, identity selectors, or to the mobile device itself.

Planned future work includes exploring the possibility of extending the scheme to operate with other client-enabled identity management systems, including password managers. We also plan to develop the prototype in various ways, including:

- preventing it being disabled by an unauthorised PC user;

12. ENHANCING USER AUTHENTICATION IN INFORMATION CARD SYSTEMS

- providing support for OTP transfer to the mobile via Bluetooth and/or infrared; and
- supporting automated OTP entry from the mobile device.

Part IV

Universality

Overview

Part IV of the thesis introduces a client-based identity management tool that can support a wide range of identity management systems using a single user interface. It consists of a single chapter, *chapter 13*, which describes the client-based tool. The tool is designed to provide a consistent user experience, whilst supporting a range of existing identity management technologies.

A Universal Client-based Identity Management Tool

13.1 Introduction

13.1.1 The Need for Authentication

Authentication of human users is a fundamental security requirement; indeed, it could be argued that it is *the* fundamental requirement [15]. Despite its importance, it is almost universally acknowledged that providing user authentication remains a huge practical problem. In practice, as many observers have noted (see, for example, Herley et al. [110]), we are still using passwords almost universally. Again as widely acknowledged, the use of passwords has many shortcomings, not least because users today have so many Internet relationships, all requiring authentication. In such a context, password re-use and use of weak passwords are almost inevitable.

A common approach to addressing this problem is to propose yet another new way of achieving user authentication, possibly involving a PKI (see section 2.4.4.1). However, there are already many good technological solutions. Perhaps the real problem is the insufficiently broad adoption of the solutions we already have. If so, this is partly a business and sociological issue, but perhaps it is also a problem which requires new technical thinking.

It is easy for those of us providing technological solutions to claim that this is not our problem. We provide the technology, and the business and

commercial world should just get on with adopting it. However, real life is not so simple. We, in the academic world, should be thinking about how to devise technological solutions which are easier to adopt. As always, key issues for adoption are transparency, ease of use, and backward compatibility, and these factors have played a large part in the design of the system we describe here.

13.1.2 Identity Management

As stated in chapters 3 and 4, identity management systems have been designed to simplify user authentication. An identity management system enables an IdP to support authentication of a user (and assertion of user attributes) to an RP. As discussed in chapter 4, recent years have seen the emergence of a wide range of such systems. Each system has its own set of protocols governing communications between the main parties. As well as its own protocols, each system may also have a unique supporting infrastructure, including public-key certificates, shared keys, passwords, etc. Some systems have gained traction recently, e.g. the use of OpenID in some sectors and Facebook's adoption of OAuth (in the form of Facebook Connect). However, the systems that have been most widely used also possess the most significant security issues (e.g. phishing vulnerabilities), and no system has broad penetration into the user community.

As discussed in sections 1.2.3, 3.5.2 and 5.1.2, many identity management systems are susceptible to phishing attacks, in which a malicious (or fake) RP redirects a user browser to a fake IdP. The user then reveals to the fake IdP secrets that are shared with a genuine IdP. This arises because, in the absence of a system-aware client agent, schemes rely on browser redirects.

A further problem faced by an end user, as stated in sections 1.2.4, 5.1.2 and 5.4.3, is that the user experience of every identity management system is different. It is widely acknowledged that users fail to make good security decisions, even when confronted with relatively simple decisions. The lack

of consistency is likely to make the situation much worse, with users simply not understanding the complex privacy- and security-relevant decisions that they are being asked to make.

Finally, as stated in section 5.4.4, when using third party IdPs which provide assertions about user attributes, there is a danger that a user will damage their privacy by revealing attributes unintentionally to an RP. In general, getting settings correct for systems handling PII is a non-trivial task.

13.1.3 A New Approach

It is tempting to try to devise another new scheme which has the practical advantages of OpenID and OAuth, but yet provides robust protection against phishing and privacy loss. That is, we might wish to devise a client-based scheme with the user convenience of other systems, but which somehow avoids the fate of CardSpace (as discussed in section 4.3.1). However, it seems that a new solution is highly unlikely to succeed when others have failed, especially given that systems such as CardSpace have had the support of a large corporation and incorporate very attractive features. Moreover, a new system is likely to create yet another different user experience, increasing the likelihood of serious mistakes by end users. This suggests that devising yet another new system may not be the right approach.

The goal of this chapter is to propose a new approach to the user authentication problem. It does not involve proposing any new protocols or infrastructures. The goal is to try to make it easier to use existing systems, and also to make their use more secure (including resistance to phishing) and privacy-enhancing, not least through the provision of a consistent user interface and an explicit user consent procedure.

The scheme we propose (which we call *IDSpace*) involves a client-based user agent. This is a single tool which supports a wide range of identity management systems yet provides a single interface to the user. The consistent user interface should maximise user understanding of what is happen-

ing and thereby reduce the risk of errors and increase user confidence. It also avoids the need for passive browser redirects, hence mitigating phishing attacks. Much of the material in this chapter has been published [9, 15].

13.1.4 CardSpace

One motivation for introducing the scheme arises from consideration of CardSpace (see section 4.3) and other Information Card systems such as Higgins (see section 4.4). We make the following observations.

- The user interface of CardSpace and the underlying communication protocols are not inherently tied together (see also [112]). It is thus possible in principle to keep the simple, intuitive user interface, and use it as the front end for a tool which manages user credentials in a consistent way regardless of the underlying identity management system. Credential sets can then identify with which identity management system(s) they should be used. For example, each credential set could be stored as a self-describing XML document. Indeed, these credential sets could include username-password pairs.
- Before issuing a security token, a CardSpace-enabled IdP will typically need to authenticate the user. This user authentication takes place via the local CardSpace software. There are two key advantages of such an approach: it provides a consistent user experience, and it helps to limit the possibility of phishing attacks.

These two observations provide the main motivation for the design of the IDSpace scheme.

13.1.5 Organisation

The remainder of the chapter is organised as follows. Section 13.2 introduces IDSpace, and, in section 13.3, we give a high-level architecture of

IDSpace. Section 13.4 discusses a number of functions that an IDSpace-conformant system must provide. Section 13.5 describes IDSpace operation, and section 13.6 shows how IDSpace operates with certain existing identity management systems. Section 13.7 outlines a possible implementation, and, finally, section 13.8 concludes the chapter, including highlighting possible areas for related work and listing future research directions.

13.2 IDSpace

We now describe IDSpace, the name of which pays homage to CardSpace. IDSpace is an architecture for a client-based identity management tool that operates in conjunction with a client web browser. A tool conforming to the architecture provides a user-intuitive and consistent means of managing a wide range of types of digital identities and credentials for web activities. The IDSpace architecture is designed to support a wide range of existing identity management protocols, and can be used to replace existing identity management client software, including the CardSpace/Higgins agents, Liberty-enabled client software, and client-based password managers.

It is important to note that IDSpace is not an identity management system, at least not in the normal sense of the term. Instead it is an architecture for a client system which enables the use of a multiplicity of identity management protocols with maximal transparency to the user (and avoiding the need to install multiple identity management clients). The IDSpace architecture is designed so that conformant tools are able to work with existing Internet RPs and IdPs without any changes to their current operation. That is, the system is transparent to third parties.

The IDSpace architecture is designed to be platform-independent, and a partial prototype implementation has been developed (described in section 13.7). Implementations should be capable of being deployed on Windows, Unix, Mac OS, and smart phone-based platforms with minimal changes.

Key parts of the IDSpace system can be instantiated as browser add-ons, e.g. written in C++ and/or JavaScript, thereby maximising portability.

As with any identity management tool, the primary purpose is to enable an end user to access a protected resource. Once installed on a user platform, IDSpace will execute whenever a user wishes to access a protected service using a web browser. It allows the user to select a particular identity management system from amongst those supported by the RP. It also allows the user to choose which set of credentials is to be used with this RP, where the network interactions with the RP and IdP will conform to the chosen identity management system.

IDSpace interacts with the user via a key component known as the *card selector*. This provides a visual representation of user credential sets in the form of virtual cards, referred to here as *credential cards* (*cCards*). The operation of this component is motivated by the CardSpace identity selector, whose virtual cards are known as *InfoCards* or *iCards*. Higgins, which originated as an open-source implementation of a CardSpace-like system (see section 4.4), also uses the term *InfoCards*.

A cCard can represent any of a wide range of types of user credential, including:

- ready-to-use tokens including password manager tokens containing a username-password pair, referred to as *local cCards*; and
- a pointer to a remote, credential-issuing party (an IdP), referred to as *remote cCards*.

Whilst IDSpace has a similar user interface to CardSpace and Higgins, it is important to note certain fundamental differences. Both CardSpace and Higgins support just one set of protocols for web interactions between the user platform and third party systems. If future versions of these systems support additional protocols, then this is likely to require corresponding modifications to RPs and/or IdPs. IDSpace, by contrast, is designed to work

with almost any conceivable identity management protocol suite, and its adoption does not require any changes to third party systems (including IdPs and RPs).

IDSpace is made up of a set of self-contained components interacting with each other in a pre-defined way, thus enabling modular implementation. Such an architectural design enables new identity management protocols to be supported in a simple way by adding new software modules to an existing implementation.

13.3 High-level Architecture

13.3.1 Context of Use

As stated above, IDSpace provides a user-intuitive means for managing digital identities and credentials for web activities, consistent across underlying identity management systems. The intended context of use is shown in Fig. 13.1.

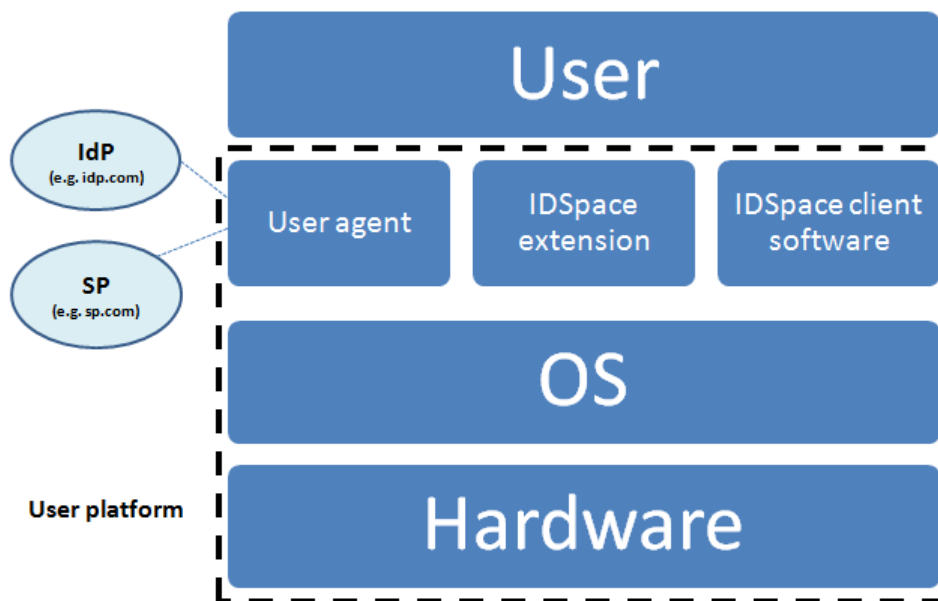


Figure 13.1: IDSpace Context

The parties involved, as shown in Fig. 13.1, include the following.

1. The *user* interacts with a *user platform* or *hardware platform* (e.g. a PC or mobile device) in order to access services provided across the Internet. This user platform is equipped with an *operating system (OS)* on which applications execute.
2. The *IdP* provides identity services to the user. This typically involves issuing a user-specific security token for use by an RP (where, although the token is intended for use by a specific user, the user's identity will not necessarily be revealed to the RP). This token will provide the RP with assurance regarding certain attributes of the user. The IdP is located either remotely or locally on the user platform; in the latter case the IdP is referred to as a *local identity provider (LIP)*. Examples of possible IdPs include Facebook and Google.
3. The *RP* provides services which the user wishes to access. In order to allow the user to access a protected resource, the RP will wish to be provided with verifiable statements regarding certain attributes of the user. This is typically achieved by supplying the RP with a user-specific credential or security token issued by a local or remote IdP. Examples of possible RPs include YouTube, Amazon, Facebook and Google (some parties may act as both IdPs and RPs).
4. The *UA* is a software component employed by a user to manage interactions between the user/user platform and remote entities (IdPs and RPs). This will typically be instantiated as a web browser, such as Internet Explorer or Firefox; indeed, for the sake of simplicity, in some subsequent discussions we refer to a web browser rather than a UA. The UA processes protocol messages on behalf of the user, and prompts the user to make decisions, provide secrets, etc.
5. The *IDSpace client software*, implementing part of the IDSpace architecture, interacts with the user via a graphical user interface (GUI). This

GUI allows the user to select a particular credential set (represented as a cCard) for use in a specific transaction with an RP. The application also interacts with a web browser, and, where necessary, with remote entities.

6. The *IDSpace extension* (or the *IDSpace browser extension*), implementing part of the IDSpace architecture, supplements the functionality of the UA. It is made up of a set of modules performing specific tasks, e.g. scanning a web page for a username-password login form. The IDSpace extension exchanges data with the client software via the browser, and, where necessary, interacts with the user.

13.3.2 IDSpace Components

Fig. 13.2 shows the relationships between the main components of IDSpace, including the primary information flows. The dotted line shows the limits of the browser extension. Note that, although shown as part of the browser extension, the *activator* could also be implemented as an independent component. This is because, in certain identity management systems e.g. CardSpace, the RP web page must implement certain X/HTML tags to enable this component to perform its task (see section 4.3.7). However, it is also possible for a browser extension to add such tags.

The remaining components, apart from the web browser and remote IdP, represent the *IDSpace client software*. Note that the boxes marked *other ...* refer to IDSpace components, which, although covered in the text, are not shown in Fig. 13.2.

The two primary elements of the IDSpace architecture, i.e. the IDSpace client software and the IDSpace extension (as introduced in section 13.3.1), are now discussed in greater detail.

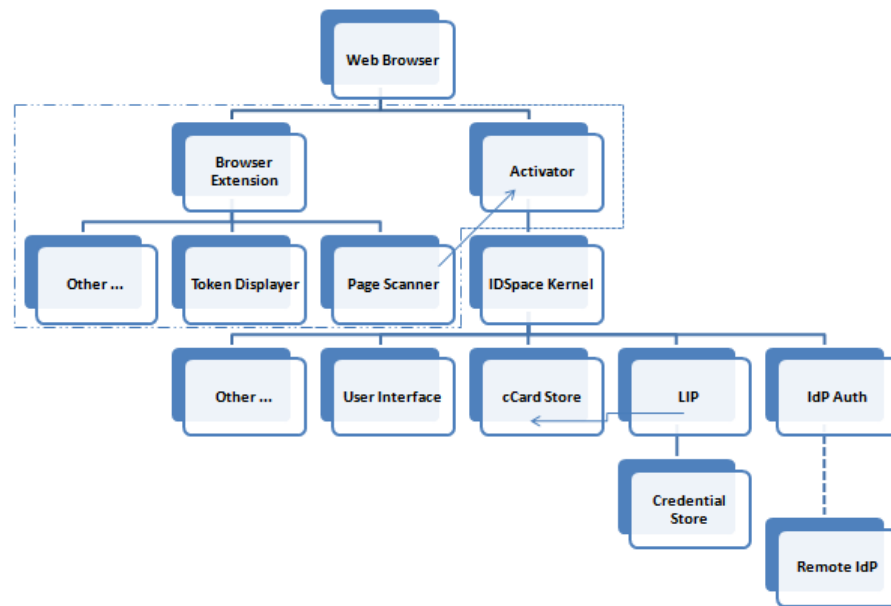


Figure 13.2: IDSpace Components

13.3.2.1 Client Software

The client software, a stand-alone application, is made up of the following components.

cCards A cCard is a relatively non-sensitive XML document corresponding to a set of user credentials (or, more generally, to a set of user private information). A cCard indicates the types of personal information in the set, and also the type (or types) of identity management system with which the cCard can be used. However, it does not contain the personal information itself. cCards can be *local*, in which case they are generated by the LIP, or *remote*, in which case they are generated by a remote IdP.

cCard Store This is a protected local store for cCards. The nature of the protection provided for stored cCards will depend on the implementation environment. For example, protection could involve the use of cryptography, physical protection and/or logical protection (as provided by the OS).

Credential Store This is a protected local store for sensitive data, such as personal information, certificates, user passwords, etc., associated with local cCards. It is used by the LIP. In practice, the credential store and the cCard store could be combined. As is the case for the cCard store, the nature of the protection provided will be implementation-dependent, and could involve the use of cryptography, physical protection and/or logical protection.

Settings Store This is a local store for relatively non-sensitive data such as system state, system/user settings, user preferences, etc.

IDSpace Kernel This is the central component of IDSpace, which runs locally on the user platform, handling communications with and between other components of IDSpace. In particular, it performs the following functions.

- It receives and processes the security policy provided by the activator.
- It retrieves the cCards from the cCard store, and checks which of them meet the requirements of the RP security policy.
- It invokes the IDSpace *user interface* (see below) in a private desktop window, and displays the cCards that meet the requirements of the RP policy.
- If a remote cCard is chosen, it retrieves the policy of the relevant remote IdP by initiating a connection with it.
- It communicates with the user-selected IdP to obtain a security token, where necessary using the *IdP authentication* module (see below).

User Interface This component, which incorporates the card selector, is the main means by which an end user interacts with the IDSpace client software. Its tasks include the following.

- It displays the identity of the RP to the user, and indicates whether the RP has been visited previously. If the RP is being visited for the first time then it allows the user to either continue or terminate.
- It displays the available cCards (it might display all the cCards and highlight those that meet the RP policy, or it might only display those meeting the policy). Note that the cCards are displayed in the card selector.
- It allows the user to review the contents of a cCard.
- It allows the user to generate and modify local cCards — in doing so it provides an interface to some of the functions of the LIP.
- It allows the user to import a cCard provided by a remote IdP that supports InfoCards.
- It allows the user to create a cCard for a remote IdP which does not support InfoCards.
- It asks a user for explicit consent before providing potentially sensitive information to an RP.
- It allows the user to set preferences for future operation of the system. These preferences are stored in the settings store.

LIP This provides the functionality of an IdP, but is resident on the user platform. Like any IdP, the LIP can generate security tokens. These tokens can be retrieved by the IDSpace kernel. The LIP stores user-attribute values and other sensitive user data in the credential store.

IdP Authentication This authenticates the user to a remote IdP, if a remote cCard is selected. It uses the user interface to prompt the user to enter the required credentials, e.g. username and password, and then submits them to the IdP. By doing so it enables a consistent and simple user authentication interface to be provided to the user, even when a range of different identity protocols are being used. It also supports

IdP-specific protocol interactions, e.g. to create requests for specific types of token.

Networker This initiates a direct online connection between the client software and a remote server (i.e. not involving the browser).

13.3.2.2 Browser Extension

The IDSpace extension, typically implemented as a browser add-on, includes the following modules.

Page Scanner This browser extension module scans the RP login page in order to discover which identity system(s) the RP supports. It passes the results of the scan to the *identity system selector* (see below).

Activator This is a (logical) bridge between the browser and the IDSpace kernel. Its tasks include the following.

- It informs the user that IDSpace can be used.
- It enables the user to activate the card selector.

Identity System Selector This browser extension module enables the user to select the identity management system to be used from amongst those supported by the RP website. The precise operation of this component will depend on the implementation of the IDSpace architecture.

If more than one identity system is available, the identity system selector could ask the user to either choose an identity system immediately or defer the selection until the point at which a cCard is selected (using the card selector). It might also provide a means to store the user answer (in the settings store) for future authentication attempts.

It passes the user response to the *data transporter* (see below).

Data Transporter This browser extension module provides the means to exchange data between components of the IDSpace architecture, including the following.

- It is responsible for the transfer of metadata regarding the RP (e.g. the discovered and selected identity system(s), the identity of the RP, the RP policy requirements, etc.) to the IDSpace kernel. For example, if the user indicates that IDSpace is to be used, it passes the security policy of the RP website to the IDSpace kernel.
- It transfers data from the IDSpace kernel to the browser. For example, if IDSpace obtains or generates a security token during the authentication process, it gives the token to the browser which dispatches it to the RP.

Token Displayer This browser extension module displays an indication of the contents of an IdP-generated security token to the user. This helps the user to decide whether or not to allow the token to be passed to the RP. This function can only be provided if the token is not:

- encrypted in such a way that only the RP can read it (e.g. using an RP's public key); and
- transmitted via a (direct) IdP-RP back-channel, i.e. the token must pass via the client platform.

13.4 Supporting Functionality

We next discuss a number of key functions that an IDSpace-conformant system must provide. For many of these functions we outline multiple approaches to implementation.

13.4.1 Identity System Discovery

IDSpace must be able to determine which identity management systems are supported by an RP website. This can be accomplished in a number of different ways, including the following.

1. IDSpace could scan the visited page for HTML/XHTML tags that are associated with specific identity management systems. For example, the string:
 - *application/x-informationCard* indicates support for CardSpace; and
 - *openid_url* or *openid_identifier* indicates support for OpenID.

The benefits of such an approach include complete transparency, albeit at the cost of performance (because IDSpace must scan every browser-rendered web page).

2. IDSpace could ask the user which identity management systems the page currently supports. The benefits of such an approach include accuracy and higher performance, at the cost of transparency and user convenience although the user's choice could be stored in the settings store for future logins.
3. IDSpace could employ a hybrid approach based on a combination of the above two options, e.g. so that if the first option fails then it resorts to the second option.

13.4.2 Identity System Selection

Having learnt which identity management system(s) an RP website supports, IDSpace must allow the user to select which system to use for the current transaction. Such a process could take place before or after invocation of the card selector. We next consider these options in greater detail.

1. **Prior to Selector Invocation.** IDSpace could allow the user to choose the identity management system in one of the following ways.

- IDSpace could embed a descriptive icon (logo, image, link or button) in the web page for each available system, and require the user to select one (e.g. by clicking the selected icon). Whilst this approach is intuitive and transparent, it could damage the appearance of the page, particularly if there are many logos to embed.
- IDSpace could ask the user which system they wish to use by embedding forms in the page or by triggering pop-up boxes. The benefits of such an approach would include accuracy and higher performance, at the cost of minor user convenience.
- IDSpace could add an identity management system selection option to the in-page context menu (i.e. the menu that appears as a result of right-clicking on the mouse). Once such an option is selected, a list of identity management systems would be displayed, allowing the user to select one. Whilst this might be transparent, it might not be so intuitive to end users.
- IDSpace could enhance the browser frame¹, including adding a browser icon, bar or menu. Once the added icon (or bar or menu) has been selected, the user could choose one of the systems currently supported by the RP. Whilst this may be transparent, modifying the browser frame could be somewhat intrusive to the end user.

2. **After Selector Invocation.** The IDSpace card selector could display the currently supported identity management systems, allowing the user to select one. This choice could be combined with a display of the

¹Both the browser frame and the browser-displayed web page could be extended. Browser extensions could, for example, create lightweight buttons, menu extensions, and in-process BHOs. The browser frame could be extended using band objects, and the web page content could be enhanced with, for example, ActiveX Controls or similar technologies [74].

available cCards (if any) associated with each of the systems. In the latter case, the selector window could be partitioned so that each section displays an identity management system along with a previously used cCard for that system; a clickable option could be used to request the display of other available cCards. This approach would be transparent, convenient and would avoid making changes to web browsers or web pages. However, it would require more processing, and hence could adversely affect user platform performance.

13.4.3 Card Selector Invocation

In response to a user action, IDSpace must be able to invoke the card selector. This involves embedding IDSpace support in the RP web page using a browser extension (see above).

13.4.4 cCard Storage

The format of cCards must be sufficiently flexible and self-contained to allow cCard storage in a variety of locations, and to support portability. We assume that cCards will be protected while stored where, as stated previously, the nature of this protection will be implementation-dependent.

cCards could be stored on various media, including:

- local file systems, which would give good performance and allow fast retrieval;
- remote web servers (the cloud), which would give a roaming capability; and
- portable user platforms such as mobile phones or smart cards, which would also provide a roaming capability.

13.4.5 cCard Format

Each cCard will contain an identifier indicating the identity management system with which it can be used; in principle a cCard could have many such identifiers. We suppose here that cCards are encoded using XML, as is the case for CardSpace InfoCards. A single XML schema could be devised encompassing all supported identity management systems. This would have the advantage that the identity system identifier (discussed immediately above) could form part of the encoding of a cCard. Other methods of encoding could also be used, such as JSON² [73].

13.4.6 cCard Contents

The content of a cCard will vary depending on the identity management system with which it is to be used. However, the types of content listed below are likely to be contained in almost all cCards. Note that such contents are similar to the contents of the CardSpace InfoCards listed in section 4.3.2.

1. *A list of supported attribute types*, e.g. age, password, first name, last name, the values of which are known by the IdP, and for which the IdP will be prepared to generate a security token. The actual claim values are not stored by the card selector; they are either stored by the remote IdP or by the LIP. The LIP will store the values in the protected credential store. Protection could, for example, involve implementing the credential store on a separate device such as a smart card, or using a TPM [97, 104, 218] to provide encrypted storage.
2. *A list of supported token type(s)*, indicating which type(s) of security token (e.g. SAML, username-password) the IdP associated with the cCard is capable of issuing.
3. *IdP location*, including the URL(s) of the remote or local IdP(s).

²<http://www.json.org/>

4. *IdP authentication method(s)*, specifying the method(s) employed by the IdP to authenticate the user.
5. *Display information*, e.g. an image and/or a name for the cCard.

13.4.7 Process Isolation

Where possible, the IDSpace processes should be isolated from other processes to maximise the security and privacy of data handled by IDSpace. For example, on a Windows platform the IDSpace card selector could be invoked in a private desktop session.

13.4.8 Authentication Methods

The IDSpace architecture allows the user to be authenticated to an IdP using a wide range of different authentication methods. The ease with which additional methods can be supported depends on precisely how user authentication to a remote IdP is supported by IDSpace. We consider three main possibilities.

1. IDSpace could control all communications between the user and the remote IdP. That is, all requests for authenticating information by the IdP could be made to the user by IDSpace (specifically by the *IdP authentication* component, as described in section 13.3.2.1), and the supplied information could then be forwarded by IDSpace to the remote IdP. Adding a new authentication method would require adding functionality to the implementation of IDSpace executing on the user platform. This is the approach adopted by CardSpace, currently deployed versions of which support four authentication methods (see section 4.3.2).

New user authentication techniques could be added in a modular fashion, as and when they are required. Whilst this would clearly add to the cost of deploying and maintaining an IDSpace implementation, for

a widely deployed system this does not seem such an unreasonable approach (given that the number of authentication methods seems unlikely to grow very rapidly). Such an approach would have the advantage of user transparency and would enable the provision of a consistent user interface for the authentication process, and is hence the preferred option.

2. IDSpace could cause the task of user authentication to be performed at the IdP rather than via the IDSpace user interface (i.e. using the IdP authentication component). That is, whenever a remote IdP requires user authentication, e.g. prior to issuing a security token, IDSpace would redirect the UA (the web browser) to the IdP, allowing the IdP to directly authenticate the user using a method of the IdP's choice. Although such an approach would minimise the maintenance cost for IDSpace, the user would lose the consistent experience provided by the IDSpace user interface.
3. IDSpace could employ a hybrid approach. The default would be the first approach outlined above. IDSpace could support a set of widely-adopted (possibly standardised) authentication methods; new methods could be added as and when it is deemed appropriate. However, if an IdP wishes to use a technique not supported by IDSpace, then IDSpace could redirect the UA (web browser) to the IdP for direct authentication.

13.5 IDSpace Operation

13.5.1 Initialisation

Prior to use of IDSpace, the following preparatory steps must be performed.

- The IDSpace components, including the browser extension and the client software, must be installed on the user platform.

- The user must install cCards in the cCard store on the user platform. As noted above, these cCards can be created by either a local or a remote IdP. We briefly consider the two cases.

- Local cCards are created using the LIP. Once it has created a cCard, the LIP will insert it in the cCard store, and the corresponding user data will be added to the credential store. A user could also choose to create a local cCard during use of IDSpace.
- Remote cCards correspond to remote IdPs. The source of such cCards will depend on the IdP. In the case of IdPs supporting InfoCards, the creation of such a cCard will typically occur via an out-of-band process, i.e. a process completely independent of the operation of IDSpace, perhaps involving the user completing a registration process using the IdP website. The resulting cCard will be generated by the IdP and provided to the user, and the user can then arrange for it to be imported into IDSpace using the IDSpace user interface. Note that the creation of remote cCards by remote IdPs will only be possible if either the:

1. IdP is aware of IDSpace; or
2. identity management system in use already supports the notion of remote InfoCards (e.g. CardSpace).

In all other cases, a remote cCard will need to be created by the local IDSpace software, perhaps using a series of menus designed specifically for the purpose.

- For ease of identification, the user can personalise a cCard, e.g. by giving the cCard a meaningful name, and/or uploading an image representing the cCard to be displayed by the user interface.

13.5.2 Protocol Flows

We now describe the operation of IDSpace. It is important to note that some parts of the operation of IDSpace will vary depending on the specific identity management system in use. The operation of IDSpace in the case of two widely discussed identity management systems is described in section 13.6.

1. UA → RP: HTTP/S Request. A user employs a UA to navigate to an RP login page.
2. RP → UA: HTTP/S Response. A login page is returned to the UA.
3. IDSpace Browser Extension → UA: Page Processing. Certain IDSpace browser extension modules (as described below) perform the following processes on the login page provided by the RP.
 - a) Page Scanner → UA: Page Scanning. The page scanner module scans the login page to discover which identity management system(s) are supported by the RP (from amongst those supported by IDSpace). It passes the identifiers of the supported systems to the identity system selector. If no identity management system is identified, the page scanner could embed an icon in the browser frame to allow the user to inform IDSpace if there is an RP-supported identity system available that has been missed.
 - b) Identity System Selector → UA. The identity system selector module uses the results passed to it by the page scanner. If more than one identity management system has been discovered, then (depending on the implementation) the identity system selector could ask the user to select one. Alternatively, the decision could be deferred and made using the card selector. The advantages and disadvantages of the two approaches are discussed in section 13.4.2. A further alternative approach would involve the user deciding at which stage to make a choice.

The module might also offer to store any choices made by the user (in the settings store) for managing future authentication attempts. The module finally reports all the results to the data transporter module (see below).

- c) Activator \Rightarrow UA: Card Selector Activation. The activator module provides a means for the user to activate the card selector. How this is achieved is implementation-specific (options are discussed in sections 13.4.2 and 13.4.3). This involves embedding IDSpace-enabling tags and an IDSpace security policy in the login page. The embedded policy is subsequently used by the IDSpace user interface to help it decide which cCards should be displayed for possible use.
4. User \rightarrow UA: Card Selector Invocation. The user performs an action which invokes the card selector. The precise way in which this occurs is implementation-specific (options are discussed in section 13.4.2).
 5. Data Transporter \rightarrow Kernel: Passing Metadata. The data transporter module passes the necessary metadata (including the identified/selected identity system(s), the RP identity, and the RP policy requirements) to the IDSpace kernel.
 6. IDSpace Kernel \Rightarrow Card Selector: RP Identity. The IDSpace kernel examines the RP identity (as received from the data transporter module in the previous step), including noting whether or not the RP uses HTTPS and whether or not the user has visited this particular RP before. The IDSpace kernel uses the card selector to:
 - a) identify the RP to the user; and
 - b) ask the user whether to continue or terminate the protocol.

Depending on the user answer, IDSpace either continues or terminates. To assist in user decision-making, the card selector could indicate sig-

nificant security-relevant features of the RP to the user, e.g. using visual cues. In particular, it could indicate whether or not the RP:

- uses HTTPS (see section 2.5.5);
- possesses an EV certificate (see section 2.4.4.1);
- has been visited before; and/or
- requires a large number of, or particularly sensitive, types of user attributes.

The card selector could also offer the user a recommendation as to whether or not to continue, based on user policy settings and the RP's security properties.

7. IDSpace Kernel \Rightarrow IDSpace Components. The IDSpace kernel evaluates the received metadata in order to learn which actions to take. If the user has already chosen an identity management system, then the following processes take place.

- a) IDSpace Kernel \Rightarrow cCard Store: cCards Retrieval. The IDSpace kernel retrieves the appropriate cCards (possibly none) by comparing the received metadata with the available cCards. Note that the retrieved cCards are specific to the user-selected identity management system.
- b) IDSpace Kernel \rightarrow Selector: Displaying cCards. The IDSpace kernel passes the retrieved cCards to the card selector so that they can be displayed to the user. cCards previously used with this RP (if any) could be displayed more prominently than the others.

If the user has not yet chosen an identity management system, then the following processes take place.

- a) IDSpace Kernel \Rightarrow cCard Store: cCards Retrieval. The IDSpace kernel retrieves the appropriate cCard(s) by comparing the re-

ceived metadata with the available cCards. Note that cCards will be retrieved for all RP-supported identity management systems.

- b) IDSpace Kernel → Card Selector: Displaying RP-supported Identity Management Systems + cCards. The kernel passes the RP-supported identity management systems, along with the matching cCards (if any), to the card selector to be displayed to the user. The card selector displays the list of supported identity management systems, together with the available cCards, indicating which cCards have been used previously with this RP. It could also indicate which identity management systems have been previously used with this RP.

Depending on the implementation and the number of systems and cCards to be displayed, the card selector might only display the cCards previously used. In such a case it would need to indicate that other cCards are also available, and would need to provide a means to retrieve them.

In both cases, the card selector should also allow the user to create a new local cCard, if the relevant identity management system supports such cCards.

8. User → Card Selector: Selecting/Creating cCards. The user selects (or creates) a cCard.
9. Card Selector → IDSpace Kernel: User Action Results. The card selector reports the results of the user actions back to the IDSpace kernel.
10. IDSpace Kernel ⇌ IDSpace Components. The IDSpace kernel evaluates the results received from the card selector, and takes the appropriate steps.

If the user has chosen to select an existing cCard, then the following processes take place.

- a) The IDSpace kernel determines whether an IdP (local or remote) needs to be contacted. If not, control is passed to step 13. If so, the protocol continues.
- b) The IDSpace kernel determines the IdP (local or remote) that must be contacted in order to enable the user to obtain the security token required by the RP. This also includes determining the nature of the information regarding the user (e.g. login credentials) that must be supplied to this IdP.
- c) IDSpace Kernel \Rightarrow Card Selector: Displaying IdP Identity. If this IdP has not previously been used, or if it does not use HTTPS, the IDSpace kernel uses the card selector to obtain user consent before sending the IdP any information. This step is designed to mitigate the risks of phishing attacks. In such a case the card selector reports the user response back to the kernel.
- d) If user consent has been obtained, the kernel now passes a token request to the IdP (see step 11). Depending on the identity management system in use, this token request will have been either received from the RP or created by the IDSpace kernel.

If the user has chosen to create a local cCard, the following processes take place.

- a) IDSpace Kernel \Rightarrow Selector GUI. The kernel invokes a special card selector window to allow the user to enter the necessary data. This would typically include allowing the user to personalise the cCard, e.g. by uploading a cCard image, entering a cCard name, etc. Such steps would enable the cCard to be readily recognisable.
- b) IDSpace Kernel \Rightarrow cCard Creation Module (in the LIP): cCard Creation. The kernel instructs the cCard creation module to create an XML-based cCard using the user-inserted data. The cCard creation module returns the newly-created cCard to the kernel.

- c) IDSpace Kernel \rightleftharpoons cCard Store: cCard Storage. The kernel sends the cCard to the cCard store for permanent storage; the cCard store reports back to the kernel whether or not the operation has been successful.
 - d) IDSpace Kernel \rightleftharpoons Card Selector. The kernel treats the newly-created cCard as a user-selected cCard and step 10a repeats.
11. IDSpace Kernel \rightleftharpoons IdP. One of the following processes takes place, depending on whether the selected IdP is local or remote.
- If a remote IdP is selected, and if such information is required by the IdP (and is not already stored by IDSpace), then the IDSpace kernel prompts the user to enter the relevant IdP credentials using a special credential screen. If this fails, e.g. if the kernel does not support the IdP authentication method, or if the user-selected identity management system dictates that the UA must be redirected to the IdP, then the kernel redirects the UA (web browser) to the remote IdP along with an authentication request. In the latter case the IdP can authenticate the user directly using an authentication method of its choice.
If user authentication is successful, the IdP issues a security token.
 - If a local IdP is selected, then the kernel constructs a token request and sends it to the LIP. The LIP responds with an appropriate security token.
12. Token Displayer Module \rightleftharpoons User. If an identity management system other than an Information Card system such as CardSpace is in use, then the token displayer module intercepts, analyses, and displays information about the security token before releasing it to the RP, and seeks user consent for release. If consent is denied, then the protocol is terminated. As stated in section 13.3.2.2, this assumes that the token is

not end-to-end encrypted to the RP and that it is not sent via a direct IdP-RP channel.

If CardSpace is in use, then, as stated in section 4.3.6, the CardSpace IdP will send back a display token along with the real token, which the kernel can instruct the card selector to display to the user, prior to obtaining user consent.

13. IDSpace Kernel \rightarrow UA \rightarrow RP: Passing Security Token. The security token is passed to the UA, which forwards it to the RP.

14. RP \rightarrow UA: Grant/Deny Access. The RP verifies the token, and, if satisfied, grants access to the user.

13.6 Mapping Specific Protocol Architectures onto IDSpace

As discussed in sections 3.5.1 and 3.7.3, identity management systems can be classified according to how the RP communicates via the client with the IdP. There are two main ways in which this can be achieved, namely by using an HTTP redirect or involving an active client.

We now describe how two specific examples of identity management systems can be mapped onto the IDSpace architecture. We consider OpenID (see section 4.5) and Liberty (using a LEC) (see section 4.7) since they are widely discussed examples of a redirect-based and an active client-based system, respectively. We also briefly look at CardSpace support. These descriptions are intended as examples; it is important to note that this is not the only way in which the systems concerned could be supported using IDSpace.

13.6.1 IDSpace and OpenID

13.6.1.1 cCards

Either prior to, or during, use of IDSpace, the user must create an OpenID-specific cCard. This cCard must contain one required field, and may also contain an optional field, as follows.

1. The single *required field* must contain the user's OpenID identifier.
2. The *optional field* contains the URL of the user's (OpenID-enabled) IdP. This field is optional since it is not strictly necessary for IDSpace operation, as an OpenID-enabled RP will specify the URL of the IdP it needs to use. However, since this leaves open the possibility of a fake IdP attack by a malicious or fake RP, the use of the optional field, which enables cross-verification of the URL provided by the RP, is strongly recommended.

The cCard contains a unique, OpenID-specific identifier, and is stored in the secure cCard store, possibly in an OpenID-specific location (e.g. to allow faster retrieval).

13.6.1.2 Protocol

We now describe one way in which IDSpace could support OpenID. Steps 3b, 4–9, 10a–d (second series), 13 and 14 of the IDSpace-OpenID-specific protocol are the same as the corresponding steps of the generic IDSpace protocol given in section 13.5.2, and hence are not described here. Whenever prompted to select, create, or import a cCard, it is assumed that the user will select, create, or import an OpenID-specific cCard (an IDcard — see Fig. 13.7).

1. UA → RP: HTTP/S Request. The user navigates to an OpenID-enabled RP.

2. RP → UA: HTTP/S Response. A login page is returned containing an OpenID form.
3. IDSpace Browser Extension → UA: Page Processing. The browser extension performs the following processes on the login page provided by the RP.
 - a) Page Scanner Module → UA: Page Scanning. The page scanner module searches the login page for an OpenID login form; such a form can be identified by searching for an input field named *openid_url* or *openid_identifier*. The page scanner module also scans the page for triggers for other identity management systems supported by IDSpace. Finally, the module passes the search results to the identity system selection module.
 - c) Activator ⇌ UA: Card Selector Activation. The activator module performs the following processes.
 - i. It embeds IDSpace-enabling tags in the RP login page, including a security policy in the format required by IDSpace. This policy must request OpenID-specific cCards.
 - ii. It adds a special function to the RP-provided login page to receive the security token that will later be returned by the IDSpace kernel.
 - iii. It employs implementation-dependent means to enable the user to activate the IDSpace card selector (see sections 13.4.2 and 13.4.3); for example, it might cause a special icon to appear above the submit button with the property that clicking this icon invokes the selector.
10. IDSpace Kernel ⇌ IDSpace Components. The IDSpace kernel evaluates the results provided by the card selector and takes appropriate

actions. If the user has chosen to select an existing OpenID-specific cCard, then the following steps are performed.

- a) The kernel retrieves the cCard and passes it to the browser.
 - b) The IDSpace browser extension parses the received cCard, retrieving the value of the user's OpenID and (if present) the OpenID IdP.
 - c) The browser extension temporarily stores the OpenID IdP value.
 - d) The browser extension inserts the user's OpenID identifier in the OpenID form, and submits the form back to the RP.
 - e) The RP performs an IdP discovery process (see section 4.5.2). As soon as the OpenID IdP has been discovered, the RP generates an OpenID authentication request and attempts to redirect the user's browser to the IdP.
 - f) The browser extension intercepts the RP-initiated OpenID authentication request, and compares the value of the OpenID IdP in this request with the OpenID IdP value it stored in step 10c. If they match, the process continues (with redirection of the browser to the IdP). If not, the browser extension could either terminate or warn the user of a possible phishing threat and ask whether or not to continue.
 - g) From this point on, OpenID operates as it would do in the absence of IDSpace, except for the final check in step 12 (see also the discussion below). In particular the user experience is OpenID-specific, and the user will see the OpenID IdP's authentication page.
11. OpenID IdP \Rightarrow User. If necessary, the OpenID IdP authenticates the user. If successful, the OpenID IdP requests permission from the user to send the OpenID assertion token to the RP.

12. Token Displayer \Rightarrow User. When the OpenID IdP attempts to redirect the UA back to the RP, the token displayer module intercepts, analyses, and displays a summary of the contents of the OpenID security token to the user before releasing it to the RP. If user consent to proceed is given, then the protocol continues; otherwise it terminates. Note that this is possible since the OpenID token provided by the IdP is not encrypted.

The above example describes only a partial integration of OpenID with IDSpace. We believe that it is possible to replace direct authentication of the user by the OpenID IdP with a process mediated by IDSpace using the IdP authentication module. This would enhance the user experience by making the user authentication process consistent across different identity management systems. However, whilst the system described above has been prototyped, the latter enhancement has not been implemented, and hence its practicality remains untested.

13.6.2 IDSpace and LEC

13.6.2.1 LECcards

Either prior to, or during, use of IDSpace, the user must create a Liberty-specific cCard. This cCard must contain one required field, and may also contain one or more optional fields, as follows.

1. The single *required field* must contain the URL of the user's LEC IdP.
2. The *optional field(s)*, could contain other alternative backup LEC IdPs.

The cCard contains a unique, LEC-specific identifier, and is stored in the secure cCard store, possibly in a LEC-specific location (e.g. to allow faster retrieval).

13.6.2.2 IdP Authentication Functionality

The IdP authentication module is part of the IDSpace client software. When supporting Liberty (LEC profile) its functionality includes the ability to handle token requests in Liberty format (received from Liberty RPs and sent to Liberty IdPs) and also the means to parse and process token messages received from a Liberty IdP. It makes use of the networker module to communicate with the IdP and the RP.

13.6.2.3 Protocol

We now describe one way in which IDSpace could act as a LEC. Steps 3(b,c), 4–9, 10a–d (second series), 13 and 14 of the IDSpace-LEC-specific protocol are the same as the corresponding steps of the generic IDSpace protocol given in section 13.5.2, and hence are not described again here. Whenever prompted to select, create, or import a cCard, we assume that the user will select, create, or import a Liberty-specific cCard.

1. UA → RP: HTTP/S Request. A user navigates to a LEC-enabled RP.
2. RP → UA: HTTP/S Response. A login page is returned containing an option (e.g. a button, link, or image) to use Liberty (we use Liberty here and below to mean Liberty using the LEC profile).
3. IDSpace Browser Extension → UA: Page Processing. The browser extension performs the following processes on the login page provided by the RP.
 - a) Page Scanner Module → UA: Page Scanning. The page scanner module searches the page for a distinguishing feature that indicates support for Liberty. The page scanner module also scans the page for triggers for other identity management systems supported by IDSpace. Finally, the module passes the search results to the identity system selection module.

10. IDSpace Kernel \Rightarrow IDSpace Components. The IDSpace kernel evaluates the results provided by the card selector and takes appropriate actions. If the user has chosen to select an existing Liberty-specific cCard, then the following steps are performed.
- a) The IDSpace kernel retrieves the cCard, and passes it to the IdP authentication module.
 - b) The IdP authentication module parses the received cCard, retrieving the values of the LEC IdP(s) and temporarily stores them.
 - c) IdP Authentication Module \rightarrow RP: HTTP Request. The IdP authentication module issues an HTTP request to the RP containing a Liberty-enabled header (or with a Liberty-enabled entry in the *User-Agent* header).
 - d) RP \rightarrow IdP Authentication Module: HTTP Response + Authentication Request. The RP generates a Liberty authentication request and sends it to the IdP authentication module in the body of the HTTP response. The RP could choose to include a list of IdPs it knows about in the request.
 - e) The IdP authentication module compares the received list of IdPs (if present) with the LEC IdP(s) retrieved from the selected cCard. If there is a non-empty intersection, then a cCard-specified IdP is contacted (this should be the primary IdP if possible); if not, then either the protocol terminates or the user could be asked to choose an IdP from amongst those in the RP list. The user could also be offered the choice to store the selected IdP in the settings store for future authentication attempts. If the RP does not specify a list of IdPs, then the cCard-associated IdP is contacted.
 - f) IdP Authentication Module \rightarrow Liberty IdP: Authentication Request. The IdP authentication module issues an HTTP POST to send a SOAP-based Liberty authentication request message to the

appropriate IdP. Note that this request must contain the authentication request as received from the RP.

11. Liberty IdP \Rightarrow User. If necessary, the IdP authenticates the user. Ideally this process would be mediated by IDSpace using the IdP authentication module to provide a user experience that is consistent across identity management systems. If successful, the IdP generates a SOAP-based, signed Liberty authentication response message and sends it to the IdP authentication module via an SSL/TLS channel.
12. Token Displayer \Rightarrow User. If the token is not end-to-end encrypted, the token displayer module displays a summary of the token and requests user consent to proceed. If consent is granted, the protocol continues; otherwise it terminates.

13.6.3 IDSpace and CardSpace

During or prior to use of IDSpace, the user must create a CardSpace-specific cCard (using the LIP) and/or import a CardSpace-managed InfoCard. The IDSpace generic protocol given in section 13.5.2, excluding step 12, could then be used to provide the functionality of CardSpace (see section 4.3).

13.7 Implementation

We now briefly describe a (Windows-based) partial implementation of the IDSpace architecture. This implementation employs *ActiveX controls* to act as a gateway between the web page the IDSpace browser extension is processing and the IDSpace client software. ActiveX controls allow the IDSpace browser extension to access properties, call functions, or, more generally, to communicate with the IDSpace client software.

Technically speaking, ActiveX controls³ [74] are simple OLE (Object Linking and Embedding) objects — i.e. in-process servers that must support the

³[http://msdn.microsoft.com/en-us/library/aa751972\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/aa751972(VS.85).aspx)

IUnknown interface. Such controls expose their functionality to the COM (Component Object Model), and host applications such as Internet Explorer can call on functional elements using *QueryInterface*. Although ActiveX controls are a Microsoft-specific technology, similar technologies can be used on other platforms. For example, Netscape, Mozilla Firefox and some other browsers use the *Netscape plug-in application programming interface* (NPAPI), which provides similar functionality [74].

As in the prototypes described in previous chapters of this thesis, the IDSpace browser extension is coded as a JavaScript browser plug-in, executed using a C#-driven BHO (see section 6.4.2).

Once an identity management system such as OpenID, CardSpace, or even a username-password form is discovered by the page scanner module of the IDSpace plug-in (see section 13.4.1), the plug-in's activator module embeds an HTML object tag specifying the *classid* (i.e. a GUID) of the IDSpace-specific ActiveX control. This instructs Internet Explorer to automatically download and install the control, and also exposes the control's public methods to the JavaScript-based plug-in.

The IDSpace user interface, including the card selector (see Fig. 13.3), is implemented using C#. cCards are implemented using XML, and are stored in encrypted form (see Fig. 13.4).

The prototype implementation of IDSpace could readily be adapted to provide the PassCard functionality (see chapter 10). An IDSpace-based PassCard would work in exactly the same way as the CardSpace-based PassCard, except that the:

- HS is not required, since the IDSpace browser plug-in will have access to the appropriate username and password regardless of whether HTTP or HTTPS is in use (see section 10.2); and
- username-password values can be stored in IDSpace cCard fields defined specifically for the purpose (see Fig 13.5).

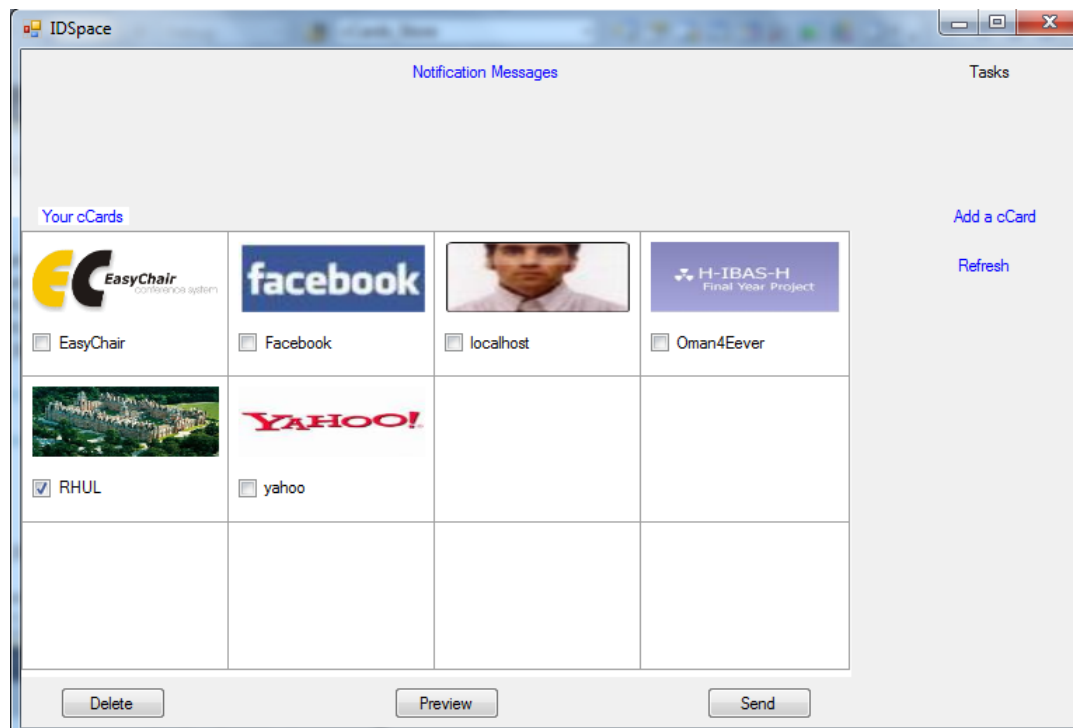


Figure 13.3: The IDSpace Card Selector

```

<?xml version="1.0" encoding="UTF-8"?>
- <PassCard cCard_PictureName="localhost" cCard_UserGivenName="localhost">
  - <EncryptedData xmlns="http://www.w3.org/2001/04/xmlenc#" Type="http://www.w3.org
    <EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#aes256-cbc"/>
    - <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
      - <EncryptedKey xmlns="http://www.w3.org/2001/04/xmlenc#">
        <EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
        - <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
          <KeyName>rsaKey</KeyName>
        </KeyInfo>
        - <CipherData>
          <CipherValue>XP2ejDbDzVWEEn6+plgnZEQNMIoH2gmApHISvY3rqSHQTRE2FF
        </CipherData>
        - <ReferenceList>
          <DataReference URI="#EncryptedElement1"/>
        </ReferenceList>
      </EncryptedKey>
    </KeyInfo>
    - <CipherData>
      <CipherValue>i/Z0eQ85kkSr7CVIYKeJSI6Zfx5nMr9Bn9dI4teXrVgyw8duZ995A12q5e
    </CipherData>
  </EncryptedData>
</PassCard>

```

Figure 13.4: Resized Screenshot of an XML-based, Encrypted cCard

Similarly, IDSpace could just as straightforwardly be configured to provide the SingleSigner functionality (see chapter 11).

If the page scanner module of the IDSpace plug-in discovers that an RP page supports OpenID, which is achieved by detecting the strings *openid.url*

13. A UNIVERSAL CLIENT-BASED IDENTITY MANAGEMENT TOOL

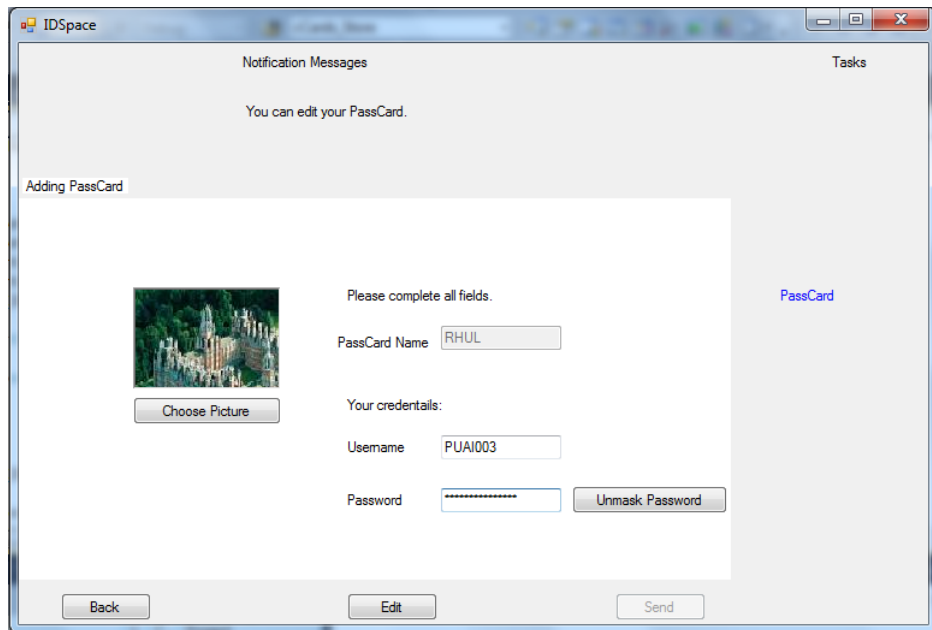


Figure 13.5: An IDSpace PassCard

or *openid_identifier* (see section 13.4.1), the IDSpace card selector lights up. This allows the user to select an IDcard (see Fig 13.6). As shown in Fig 13.7,

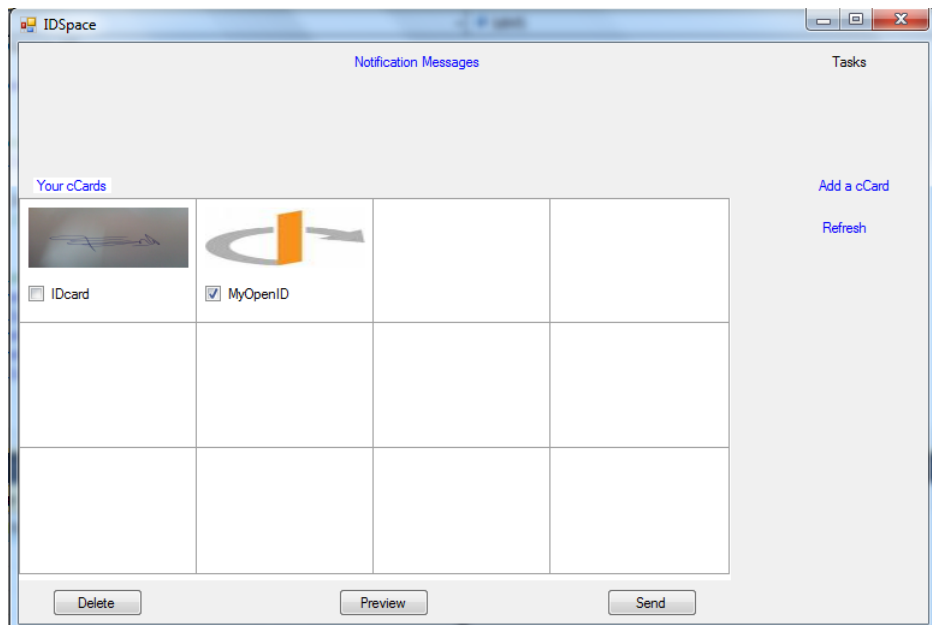
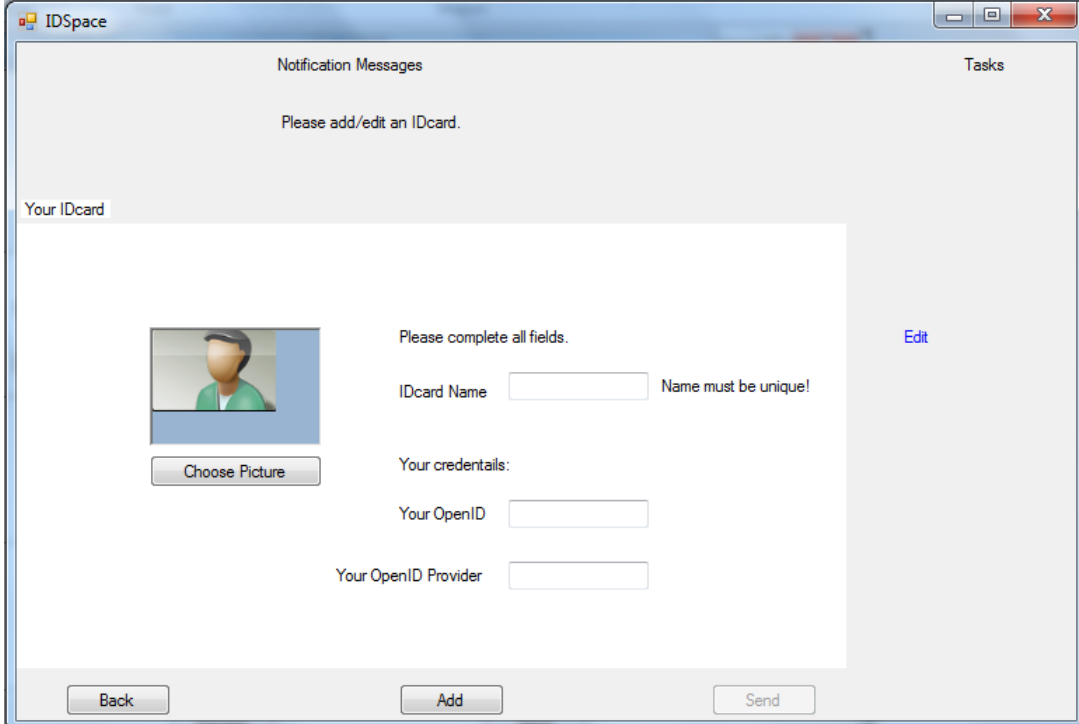


Figure 13.6: IDSpace IDcards

the IDcard requires the user to enter their OpenID identifier and, optionally, the URL of their OpenID-enabled IdP. Once the user selects and submits an

IDcard, IDSpace fills the OpenID form with the value of the user's OpenID identifier, and, from this point on, the OpenID native protocol continues in the normal way.



The screenshot shows a window titled "IDSpace" with a light gray background. At the top, there are two tabs: "Notification Messages" and "Tasks". Below the tabs, the text "Please add/edit an IDcard." is displayed. The main content area is titled "Your IDcard" and contains a form. On the left side of the form, there is a small square image placeholder showing a person's head and shoulders. Below the image is a button labeled "Choose Picture". To the right of the image, the text "Please complete all fields." is displayed. Below this, there are three input fields: "IDcard Name" with a text box and the label "Name must be unique!", "Your OpenID" with a text box, and "Your OpenID Provider" with a text box. To the right of the "Your OpenID" field, there is a blue link labeled "Edit". At the bottom of the form, there are three buttons: "Back", "Add", and "Send".

Figure 13.7: An IDSpace IDcard

13.8 Concluding Remarks

We have described an architecture for a client-based, platform-independent, protocol-agnostic identity management tool that operates in conjunction with a client web browser. A tool conforming to the architecture provides a user-intuitive means of managing digital identities and credentials for all web activities. A (partial) implementation of the architecture has been described.

13.8.1 Relationship to the Prior Art

13.8.1.1 CardSpace and Higgins

CardSpace (see section 4.3) shares certain features in common with IDSpace. In particular, it too is client-based and operates in conjunction with a web browser. However, CardSpace requires the IdPs and RPs to implement a specific set of protocols for inter-communication; we refer to these as the *CardSpace protocols*, although many are based on WS-* standards (see section 2.5.6). Although CardSpace supports a wide range of security token formats, these tokens must be sent using a very specific protocol suite.

This gives rise to a classic *chicken and egg problem* — without an established identity infrastructure of IdPs, there is little incentive for RPs to make the changes necessary to support CardSpace. Similarly, without any customer RPs, there is little incentive to set up a CardSpace-specific IdP infrastructure.

By contrast, IDSpace gives the convenience and intuitive user experience of CardSpace, without requiring RPs and IdPs to change the way they work. That is, IDSpace enables convenient and more secure operation by end users, without any changes to the existing identity infrastructures or service providers. Moreover, once deployed, IDSpace will enable much simpler deployment of more sophisticated systems such as the CardSpace protocols (and the many other systems currently emerging).

Higgins, which originated with the goal of providing CardSpace-like functionality on non-Windows platforms (see section 4.4), has somewhat similar objectives to IDSpace.

13.8.1.2 Other Schemes

In chapters 10 and 11 we described how to build browser extensions which enable CardSpace/Higgins identity selectors to support password management and password-based SSO without requiring any changes to RPs or to

identity selectors. Operational, open-source prototypes were also described. These prototypes demonstrate the workability of certain aspects of IDSpace.

13.8.2 Novel Features

The main novel feature of IDSpace, as intimated above, is the proposal of an architecture for a client-based system which supports multiple identity management systems transparently to RPs and IdPs. That is, it combines the convenience and intuitiveness of the CardSpace user interface with support for multiple systems, without requiring any changes to existing RPs and IdPs. To our knowledge, the only previous work permitting client-based support for multiple identity management systems requires the RPs and IdPs to adopt new protocols. The IDSpace architecture incorporates novel components, including the *page scanner*, *activator*, *identity system selector* and *token displayer*, which are not found in the CardSpace or Higgins architectures.

13.8.3 Future Work

Our main initial goal is to complete an operational prototype of IDSpace, which we plan to make available for public scrutiny and testing. We intend that the initial version should support all the identity management systems discussed in this chapter.

A variety of future directions for this research present themselves, a few of which we briefly mention.

- Apart from the identity management systems mentioned previously, it would also be desirable if IDSpace could provide support for protocols providing a high degree of privacy protection for end users, notably U-Prove and IdeMix (see sections 4.9.2 and 4.9.3, respectively). This remains a topic of ongoing research.

- In chapters 6 to 9 we investigated using a client-based tool to support interoperation between different identity management systems. It would be attractive (and straightforward) to build this functionality into an IDSpace implementation.
- We would like to investigate the possibility of configuring IDSpace to support user reputation. Instead of user reputation being managed by IdPs, as proposed by Agudo et al. [4], it appears likely to be possible to extend the IDSpace client software to support the management of user reputation; this would enable IDSpace to provide interested RPs with user reputation values. IDSpace could be set to retrieve user reputation values from multiple sources, including social networking sites, blog entries, specialised websites, financial institutions, or governmental agencies. This would remove the need to make changes to IdP servers. Of course, such a change would raise significant trust issues, since the consumer of reputation values would need to trust that the client software has not been manipulated to report incorrectly *high* reputation values; however, it may be possible to address this concern through the use of trusted execution environments on the user platform, e.g. through the use of the TPM (see section 5.5).
- Finally, in future work we intend to study variants of the architecture presented here to further enhance the security and privacy of user authorisation, whilst maintaining transparency to third parties.

Part V

Conclusions

Overview

Part V concludes the thesis by summarising the main contributions as well as highlighting possible areas for future work. This part of the thesis consists of a single chapter, *chapter 14*.

Conclusions and Future Work

14.1 Summary and Conclusions

Including this part, the thesis is divided into five parts, as follows.

The first part of the thesis contains background material and a review of relevant literature. It contains three chapters. Chapter 2 gave an introduction to, and definitions of, the concepts of identity, privacy, and security, as well as outlining associated protocols. Chapter 3 provided an introduction to identity management, covering related topics such as single sign on and Cameron's identity laws. It also gave an abstract model for identity management, and considered a range of properties which may be possessed by an identity management system. Chapter 4 provided a detailed description of those identity management systems of greatest relevance to this thesis, namely CardSpace, Higgins, OpenID, OAuth, Liberty, and Shibboleth. The chapter also gave an overview of certain other systems of background importance, namely Microsoft Passport, U-Prove, and IdeMix.

In the second part of the thesis, we described a novel approach to supporting interoperation between a wide range of identity management systems. This part contains a total of five chapters. First, in chapter 5, we described a general model for interoperation between an Information Card-based identity management system and almost any other existing identity management system. Using this model, Information Card users are able to obtain a security token from an identity provider not supporting Information Cards; the contents of such a token can be processed by an Information Card-enabled relying party. We then went on to describe four specific in-

stantiations of this model, that enable interoperation between an Information Card system and:

- Liberty (chapter 6);
- Shibboleth (chapter 7);
- OpenID (chapter 8); and
- OAuth (chapter 9).

In the third part of the thesis, we introduced three novel schemes designed to enhance the practicality and security of identity management systems. This part contains three chapters, as follows.

- Chapter 10 described PassCard, a novel scheme enabling an Information Card system to be used as a password manager. Usernames and passwords are stored in personal cards, and these cards can be used to sign-on transparently to corresponding websites. The scheme does not require any changes to login servers, default browser security settings, or to identity selectors; in particular, it does not require websites to support an Information Card system. The chapter also gave details of a proof-of-concept prototype, together with security and usability analyses.
- Chapter 11 introduced SingleSigner, a related scheme that allows an Information Card system to be used as a password-based single sign on system. The chapter described three approaches to implementing SingleSigner. In each case users are able to store credentials for a set of websites in a single personal card, and use it to seamlessly sign-on to all these websites. The approaches do not require any changes to login servers or to identity selectors and, in particular, they do not require websites to support Information Cards. The chapter also described

three proof-of-concept prototypes and gave usability, security and performance analyses. Chapters 10 and 11 are concerned with techniques intended to help improve the usability and security of password use, as well as potentially encouraging adoption of Information Card systems.

- Chapter 12 described a scheme that uses a mobile device to enhance user authentication in Information Card systems. During the process of user authentication on a computer using an Information Card system, a random and short-lived one-time password is sent to the user's mobile device; this must then be entered into the computer by the user when prompted. The scheme does not require any changes to login servers, identity selectors, or to the mobile device itself. Details of a proof-of-concept prototype, together with security and operational analyses, were also provided.

In the fourth part of the thesis, we introduced IDSpace, a universal identity management tool designed to support a wide range of identity management systems using a single user interface. IDSpace is intended both to enhance user privacy and to address a range of security issues, notably phishing attacks. This part consists of a single chapter, chapter 13, which described IDSpace in detail. The goal of IDSpace is to simplify the use of a wide range of existing identity technologies, helping to encourage their use whilst imposing no additional burden on relying parties and identity providers. The chapter also described examples of the operation of the scheme with certain existing identity management systems.

In order to maximise the practicality and applicability of the schemes proposed in this thesis, we have endeavoured to ensure that the schemes are either completely transparent, and hence immediately deployable, or require only minimal changes to the existing identity management infrastructure. Application of the proposed schemes would enable interoperation

between identity management systems, increase the rate of their adoption, improve user security (particularly in enhancing user authentication and defeating phishing attacks), and provide a consistent user experience. The research results documented in this thesis have been published in a series of 11 research papers, articles and patents (see section 1.5).

14.2 Possible Future Work

We conclude the thesis by highlighting possible areas for future work.

We plan to explore the possibility of designing a client-based interoperation model similar to that described in chapter 5 to support interoperation between an Information Card identity provider and relying parties conforming to other identity management systems.

Future work also includes building a scheme that enables the use of the PassCard and SingleSigner systems (described in chapters 10 and 11) in smart phones, such as Apple's iPhone, Samsung's Galaxy, or HTC desire. A further possible topic for future work would be to investigate the possibility of building a portable version of PassCard and SingleSigner to support users who do not have installation privileges or are forced to use untrusted machines, e.g. when travelling. In addition, we plan to investigate the possibility of extending SingleSigner to support single sign off.

Further possible future work includes exploring the possibility of extending the CardSpace-mobile scheme described in chapter 12 to operate with client-based identity management systems other than CardSpace, including password managers. We also plan to extend the prototype described in chapter 12 in various ways, including:

- preventing it being disabled by an unauthorised computer user;
- providing support for one-time password transfer to the mobile via Bluetooth and/or infrared; and

- supporting automated one-time password entry from the mobile device.

The main initial goal for the IDSpace system described in chapter 13 is to complete a fully functional prototype, which we plan to make available for public scrutiny and testing. A variety of future directions for this research present themselves, including the following.

- Apart from the identity management systems discussed in chapter 13, it would also be desirable if IDSpace could provide support for protocols providing a high degree of privacy protection for end users, notably U-Prove and IdeMix (see sections 4.9.2 and 4.9.3).
- In chapters 6 to 9 we investigated using a client-based tool to support interoperation between different identity management systems. It would be attractive (and apparently straightforward) to build this functionality into an IDSpace implementation.
- Finally, we could study variants of the IDSpace architecture to further enhance the security and privacy of user authorisation whilst maintaining transparency to third parties.

Bibliography

- [1] Rania Abdelhameed, Sabira Khatun, Borhanuddin Mohd Ali, and AbdulRahman Ramli. Authentication model based Bluetooth-enabled mobile phone. *Journal of Computer Science*, 1(2):200–203, 2005. 330, 331
- [2] Renato Accornero, Daniele Rispoli, and Francesco Bergadano. Privacy-enhanced identity via browser extensions and linking services. In Pierangela Samarati, Sara Foresti, Jiankun Hu, and Giovanni Livraga, editors, *Proceedings of NSS '11 — the 5th International Conference on Network and System Security, September 6–8, 2011, Milan, Italy*, pages 89–96. IEEE, New York, 2011. 218
- [3] Carlisle Adams and Steve Lloyd. *Understanding PKI: Concepts, Standards, and Deployment Considerations*. Addison Wesley, Reading, Massachusetts, 2nd edition, 2002. 76
- [4] Isaac Agudo, M. Carmen Fernández Gago, and Javier Lopez. A multidimensional reputation scheme for identity federations. In Fabio Martinelli and Bart Preneel, editors, *Proceedings of EuroPKI '09 — the 6th European Workshop on Public Key Infrastructures, Services and Applications, September 10–11, 2009 Pisa, Italy. Revised Selected Papers*, volume 6391 of *LNCS*, pages 225–238. Springer, 2009. 378
- [5] Gail-Joon Ahn, Moonam Ko, and Mohamed Shehab. Portable user-centric identity management. In Sushil Jajodia, Pierangela Samarati, and Stelvio Cimato, editors, *Proceedings of IFIP SEC '08 — the IFIP TC-11 23rd International Information Security Conference, IFIP 20th World*

- Computer Congress, September 7–10, 2008, Milano, Italy*, volume 278 of *IFIP*, pages 573–587. Springer, Berlin, Heidelberg, 2008. 95
- [6] Haitham S. Al-Sinani. Integrating OAuth with Information Card systems. In Ajith Abraham, Daniel Zeng, Dharma Agrawal, Mohd Faizal Abdollah, Emilio Corchado, Valentina Casola, and Choo Yun Huoy, editors, *Proceedings of IAS '11 — the 7th International Conference on Information, Assurance, and Security, December 5–8, 2011, Malacca, Malaysia*, pages 198–203. IEEE, New York, 2011. [Full version available at: <http://www.ma.rhul.ac.uk/static/techrep/2011/RHUL-MA-2011-15.pdf>]. 257
- [7] Haitham S. Al-Sinani. Supporting interworking between OAuth and Information Card systems. *Journal of Information Assurance and Security (to appear)*, 7, 2012. 257
- [8] Haitham S. Al-Sinani, Waleed A. Alrodhan, and Chris J. Mitchell. CardSpace-Liberty integration for CardSpace users. In Ken Klingenstein and Carl M. Ellison, editors, *Proceedings of IDtrust '10 — the 9th Symposium on Identity and Trust on the Internet, April 13–15, 2010, Gaithersburg, Maryland*, pages 12–25. ACM, New York, 2010. 209
- [9] Haitham S. Al-Sinani and Chris J. Mitchell. *Method and apparatus for enabling authorised users to access computer resources*. UK patent application GB1115866.4, filed 14th September 2011. 340
- [10] Haitham S. Al-Sinani and Chris J. Mitchell. Using CardSpace as a password manager. In Elisabeth de Leeuw, Simone Fischer-Hübner, and Lothar Fritsch, editors, *Proceedings of IFIP IDMAN '10 — the 2nd IFIP WG 11.6 Working Conference on Policies and Research in Identity Management, November 18–19, 2010, Oslo, Norway*, volume 343 of *IFIP Advances in Information and Communication Technology*, pages 18–30. Springer, Boston, 2010. 274

-
- [11] Haitham S. Al-Sinani and Chris J. Mitchell. CardSpace-Shibboleth integration for CardSpace users. In *ACNS '11 [industrial track proceedings], the 9th International Conference on Applied Cryptography and Network Security, June 7–10, 2011, Nerja, Malaga, Spain*, pages 49–66, 2011. [Full version available at: <http://www.isg.rhul.ac.uk/cjm/Papers/cssifc.pdf>]. 231
- [12] Haitham S. Al-Sinani and Chris J. Mitchell. Client-based CardSpace-OpenID interoperation. In Erol Gelenbe, Ricardo Lent, and Georgia Sakellari, editors, *Proceedings of ISCIS '11 — the 26th International Symposium on Computer and Information Sciences, September 26–28, 2011, London, UK*, Lecture Notes in Electrical Engineering (LNEE), pages 387–393. Springer, London, 2011. [Full version available at: <http://www.ma.rhul.ac.uk/techreports/2011/RHUL-MA-2011-12.pdf>]. 241
- [13] Haitham S. Al-Sinani and Chris J. Mitchell. Enhancing CardSpace authentication using a mobile device. In Yingjiu Li, editor, *Proceedings of DBSEC '11 — the 25th IFIP WG 11.3 Conference on Data and Applications Security and Privacy, July 11–13, 2011, Richmond, Virginia*, volume 6818 of *LNCS*, pages 201–216. Springer-Verlag, Berlin, 2011. 317
- [14] Haitham S. Al-Sinani and Chris J. Mitchell. Extending the scope of CardSpace. In Mehmet A. Orgun, Atilla Elçi, Oleg B. Makarevich, Sorin A. Huss, Josef Pieprzyk, Lyudmila K. Babenko, Alexander G. Chefranov, and Rajan Shankaran, editors, *Proceedings of SIN '11 — the 4th International Conference on Security of Information and Networks, November 14–19, 2011, Sydney, Australia*, pages 235–238. ACM, New York, 2011. [Full version available at: <http://www.ma.rhul.ac.uk/techreports/2011/RHUL-MA-2011-15.pdf>]. 274

- [15] Haitham S. Al-Sinani and Chris J. Mitchell. A universal client-based identity management tool. In *Proceedings of EuroPKI '11 — the 8th European Workshop on Public Key Infrastructures, Services and Applications, September 15–16, 2011, Leuven, Belgium (to appear)*, LNCS. Springer-Verlag, Berlin, 2011. 337, 340
- [16] Haitham S. Al-Sinani and Chris J. Mitchell. Enabling interoperability between Shibboleth and Information Card systems. *Security and Communication Networks (to appear)*, 2012. 231
- [17] Haitham S. Al-Sinani, Chi Nguyen, and Branislav Vuksanovic. H-IBAS-H — Authentication system for university student portal using images. In *Proceedings of ICCCP '09 — the International Conference on Communication, Computer and Power, February 15–18, 2009, Muscat, Oman*. Sultan Qaboos University and IEEE-Oman Section, 2009. <http://icccp.net/proceedings/2009/Papers/ICCCP09-045.pdf>. 73
- [18] Fadi Aloul, Syed Zahidi, and Wassim El-Hajj. Two factor authentication using mobile phones. In *Proceedings of AICCSA '09 — the IEEE/ACS International Conference on Computer Systems and Applications*, pages 641–644. IEEE, New York, 2009. 329
- [19] Gergely Alpár, Jaap-Henk Hoepman, and Johanneke Siljee. The identity crisis — security, privacy and usability issues in identity management. *CoRR*, abs/1101.0427, 2011. 47, 49
- [20] Abdullah Alqattan, Nima Kaviani, Patrick Lewis, and Nicholas Pearson. *A Two-factor Authentication System Using Mobile Devices to Protect against Untrusted Public Computers*. University of British Columbia, Canada, 2007. 329
- [21] Waleed A. Alrodhan. Privacy and practicality of identity management systems. Technical Report RHUL-MA-2010-14, Depart-

-
- ment of Mathematics, Royal Holloway, University of London, 2010. <http://www.ma.rhul.ac.uk/static/techrep/2010/RHUL-MA-2010-14.pdf>. 47, 60, 74, 92, 94, 95, 99, 105, 131, 144, 153, 155, 161, 163, 165, 167, 203, 227
- [22] Waleed A. Alrodhan and Chris J. Mitchell. A client-side CardSpace-Liberty integration architecture. In Kent E. Seamons, Neal McBurnett, and Tim Polk, editors, *Proceedings of IDtrust '08 — the 7th Symposium on Identity and Trust on the Internet, March 4–6, 2008, Gaithersburg, Maryland*, volume 283 of *ACM International Conference Proceeding Series*, pages 1–7. ACM, New York, 2008. 153, 205, 226
- [23] Murray Altheim and Shane McCarron (editors). *XHTML 1.1 — Module-based XHTML*. W3C Recommendation, 2010. <http://www.w3.org/TR/xhtml11/>. 90
- [24] Steve Anderson et al. *Web Services Trust Language (WS-Trust)*, 2005. <http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-trust/ws-trust.pdf>. 91
- [25] Mikaël Ates, Christophe Gravier, Jérémy Lardon, Jacques Fayolle, and Bruno Sauviac. Interoperability between heterogeneous federation architectures: Illustration with SAML and WS-Federation. In *Proceedings of SITIS '07 — the 3rd International Conference on Signal-Image Technologies and Internet-based Systems, December 16–18, 2007, Shanghai, China*, pages 1063–1070. IEEE, New York, 2007. 98, 205
- [26] Mikaël Ates, Christophe Gravier, Jérémy Lardon, Jacques Fayolle, and Bruno Sauviac. Interoperability between heterogeneous federation architectures: Illustration with SAML and WS-Federation. *CoRR*, abs/0812.2094, 2008. 98, 205
- [27] Siddharth Bajaj et al. *Web Services Policy Framework (WS-Policy)*, 2006. <http://download.boulder.ibm.com/ibmdl/pub/>

software/dw/specs/ws-polfram/ws-policy-2006-03-01.pdf. 90

- [28] Keith Ballinger et al. *Web Services Metadata Exchange (WS-MetadataExchange)*, 2006. <http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-mex/metadataexchange.pdf>. 91
- [29] Adam Barth. *HTTP State Management Mechanism*. IETF: RFC 6265, 2011. <http://tools.ietf.org/html/rfc6265#section-3>. 85
- [30] Messaoud Benantar. *Access Control Systems: Security, Identity Management and Trust Models*. Springer, New York, Softcover reprint of hardcover 1st ed. 2006 edition (2010). 47
- [31] Tim Berners-Lee, Roy T. Fielding, and Larry Masinter. *Uniform Resource Identifier (URI): Generic Syntax*. IETF: RFC 3986, 2005. <http://ietf.org/rfc/rfc3986.txt>. 85, 86
- [32] Elisa Bertino and Kenji Takahashi. *Identity Management: Concepts, Technologies, and Systems*. Artech House Publishers, Norwood, Massachusetts, 2011. 47, 74, 75, 100, 138, 144
- [33] Vittorio Bertocci. *Programming Windows Identity Foundation*. Microsoft Press, Redmond, Washington, 2010. 101
- [34] Vittorio Bertocci, Garrett Serack, and Caleb Baker. *Understanding Windows CardSpace: An Introduction to the Concepts and Challenges of Digital Identities*. Addison-Wesley, Reading, Massachusetts, 2008. 77, 92, 95, 110, 111, 115, 119, 121, 122, 125
- [35] Karthikeyan Bhargavan, Cédric Fournet, Andrew D. Gordon, and Nikhil Swamy. Verified implementations of the Information Card federated identity-management protocol. In *Proceedings of ASIACCS '08*

-
- *the ACM symposium on Information, Computer and Communications Security*, pages 123–135. ACM, New York, 2008. 121, 123, 168
- [36] Matt Bishop. *Computer Security: Art and Science*. Addison-Wesley Professional, Boston, Massachusetts, 2002. 47, 64, 65
- [37] Joshua B. Bolten. *E-Authentication Guidance for Federal Agencies — M-04-04*. Office of Management and Budget (OMB), Executive Office of the President, the White House, Washington DC, 2003. 74
- [38] David Booth and Canyang Kevin (editors). *Web Services Description Language (WSDL) Version 2.0 Part 0: Primer*. W3C Recommendation, 2007. <http://www.w3.org/TR/wsdl20-primer/>. 89
- [39] Pete Bramhall, Marit Hansen, Kai Rannenber, and Thomas Roessler. User-centric identity management: New trends in standardisation and regulation. *IEEE Security & Privacy*, 5(4):84–87, 2007. 95
- [40] Stefan Brands. *Rethinking Public Key Infrastructures and Digital Certificates: Building in Privacy*. MIT Press, Cambridge, Massachusetts, 2000. 167, 169
- [41] Stefan Brands. *U-Prove Technology Overview*. Microsoft, 2010. 167, 169, 174
- [42] Stefan Brands, Liesje Demuyne, and Bart De Decker. A practical system for globally revoking the unlinkable pseudonyms of unknown users. In Josef Pieprzyk, Hossein Ghodosi, and Ed Dawson, editors, *Proceedings of ACISP '07 — the 12th Australasian Conference on Information Security and Privacy, July 2–4, 2007, Townsville, Australia*, volume 4586 of LNCS, pages 400–415. Springer, Berlin, Heidelberg, 2007. 168
- [43] Stefan Brands and Christian Paquin. *U-Prove Cryptographic Specification V1.0*. Microsoft, 2010. 170, 172, 173

- [44] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, and François Yergeau (editors). *eXtensible Markup Language (XML) 1.0*. W3C Recommendation, 5th edition, 2008. <http://www.w3.org/TR/2008/REC-xml-20081126/>. 89
- [45] James Brown, Phil Stradling, and Craig H. Wittenberg. *U-Prove CTP R2 White Paper — Revision 17*. Microsoft, 2011. 171
- [46] Bud P. Bruegger, Detlef Hühnlein, and Michael Kreutzer. Towards global eID-interoperability. In Arslan Brömme, Christoph Busch, and Detlef Hühnlein, editors, *Proceedings of BIOSIG '07 — Biometrics and Electronic Signatures, Special Interest Group on Biometrics and Electronic Signatures, July 12–13, 2007, Darmstadt, Germany*, volume 108 of LNI, pages 127–140. GI, 2007. 204
- [47] Bud P. Bruegger, Detlef Hühnlein, and Jörg Schwenk. TLS-Federation — a secure and relying party-friendly approach for federated identity management. In Arslan Brömme, Christoph Busch, and Detlef Hühnlein, editors, *Proceedings of BIOSIG '08 — Biometrics and Electronic Signatures, Special Interest Group on Biometrics and Electronic Signatures, September 11–12, 2008, Darmstadt, Germany*, volume 137 of LNI, pages 93–106. GI, 2008. 204
- [48] William E. Burr, Donna F. Dodson, and W. Timothy Polk. *Electronic Authentication Guideline — Special Publication 800–63 — Version 1.0.2*. Recommendations of NIST, 2006. 74, 75
- [49] Jan Camenisch and Ivan Damgård. Verifiable encryption, group encryption, and their applications to separable group signatures and signature sharing schemes. In Tatsuaki Okamoto, editor, *Proceedings of ASIACRYPT 2000 — the 6th International Conference on the Theory and Application of Cryptology and Information Security, December 3–7, 2000*,

- Kyoto, Japan*, volume 1976 of LNCS, pages 331–345. Springer, Berlin, Heidelberg, 2000. 168
- [50] Jan Camenisch, Markulf Kohlweiss, and Claudio Soriente. An accumulator based on bilinear maps and efficient revocation for anonymous credentials. In Stanislaw Jarecki and Gene Tsudik, editors, *Proceedings of PKC '09 — the 12th International Conference on Practice and Theory in Public Key Cryptography, March 18–20, 2009, Irvine, California*, volume 5443 of LNCS, pages 481–500. Springer, Berlin, Heidelberg, 2009. 168
- [51] Jan Camenisch and Anna Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In Birgit Pfitzmann, editor, *Advances in Cryptology — EuroCrypt 2001, International Conference on the Theory and Application of Cryptographic Techniques, Innsbruck, Austria, May 6–10, 2001, Proceeding*, volume 2045 of LNCS, pages 93–118. Springer, Berlin, Heidelberg, 2001. 168, 176, 178
- [52] Jan Camenisch and Anna Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In Matthew K. Franklin, editor, *Advances in Cryptology — CRYPTO '04, the 24th Annual International Cryptology Conference, Santa Barbara, California, August 15–19, 2004, Proceedings*, volume 3152 of LNCS, pages 56–72. Springer, Berlin, Heidelberg, 2004. 168
- [53] Jan Camenisch, Abhi Shelat, Dieter Sommer, Simone Fischer-Hübner, Marit Hansen, Henry Krasemann, Gérard Lacoste, Ronald Leenes, and Jimmy Tseng. Privacy and identity management for everyone. In *Proceedings of DIM '05 — the 2005 workshop on Digital identity management, Fairfax, Virginia*, pages 20–27. ACM, New York, 2005. 168

- [54] Jan Camenisch and Victor Shoup. Practical verifiable encryption and decryption of discrete logarithms. In Dan Boneh, editor, *Advances in Cryptology — CRYPTO '03, the 23rd Annual International Cryptology Conference, Santa Barbara, California, August 17–21, 2003, Proceedings*, volume 2729 of LNCS, pages 126–144. Springer, Berlin, Heidelberg, 2003. 168
- [55] Jan Camenisch and Els Van Herreweghen. Design and implementation of the IdeMix anonymous credential system. In Vijayalakshmi Atluri, editor, *Proceedings of CCS '02 — the 9th ACM Conference on Computer and Communications Security, November 18–22, 2002, Washington DC*, pages 21–30. ACM, New York, 2002. 167, 176
- [56] Kim Cameron. *The laws of Identity*. Microsoft, 2005. <http://www.identityblog.com/stories/2005/05/13/TheLawsOfIdentity.pdf>. 104, 109
- [57] Scott Cantor. *User Authentication and Subject Identifiers in Shibboleth*, 2008. <https://wiki.shibboleth.net/confluence/display/SHIB/IdPUserAuthnConfig>. 163
- [58] Scott Cantor, John Kemp, and Darryl Champagne (editors). *Liberty ID-FF bindings and profiles specification*. Liberty Alliance Project, 2004. <http://www.projectliberty.org/liberty/content/download/319/2369/file/draft-liberty-idff-bindings-profiles-1.2-errata-v2.0.pdf>. 154, 155, 157, 213, 214
- [59] Scott Cantor, John Kemp, Rob Philpott, and Eve Maler (editors). *Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0*. OASIS, 2005. <http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf>. 92, 161

- [60] Scott Cantor and John Kemp (editors). *Liberty ID-FF protocols and schema specification*. Liberty Alliance Project, 2005. http://www.projectliberty.org/resource_center/specifications/liberty_alliance_id_ff_1_2_specifications. 155, 213, 215
- [61] Scott Cantor (editor). *Shibboleth Architecture — Conformance Requirements*. Internet2, 2005. 162
- [62] Scott Cantor (editor). *Shibboleth Architecture — Protocols and Profiles*. Internet2, 2005. <http://shibboleth.internet2.edu/shibboleth-documents.html>. 162, 167
- [63] David Chadwick. FileSpace: an alternative to CardSpace that supports multiple token authorisation and portability between devices. In Kent Seamons, Neal McBurnett, and Tim Polk, editors, *Proceedings of IDtrust '09 — the 8th Symposium on Identity and Trust on the Internet, April 14–16, 2009, Gaithersburg, Maryland*, pages 94–102. ACM, New York, 2009. 111
- [64] David W. Chadwick. Federated identity management. In Alessandro Aldini, Gilles Barthe, and Roberto Gorrieri, editors, *Foundations of Security Analysis and Design V, FOSAD 2007/2008/2009 Tutorial Lectures*, volume 5705 of LNCS, pages 96–120. Springer, Berlin, Heidelberg, 2009. 47, 74, 107, 109
- [65] David W. Chadwick and George Inman. Attribute aggregation in federated identity management. *IEEE Computer*, 42(5):33–40, 2009. 128
- [66] David W. Chadwick, George Inman, and Paul Coxwell. CardSpace in the cloud. In Ehab Al-Shaer, Angelos D. Keromytis, and Vitaly Shmatikov, editors, *Proceedings of CCS '10 — the 17th ACM Conference on Computer and Communications Security, October 4–8, 2010, Chicago, Illinois*, pages 657–659. ACM, New York, 2010. 129

- [67] David W. Chadwick, George Inman, and Nate Klingenstein. A conceptual model for attribute aggregation. *Future Generation Computer Systems*, 26(7):1043–1052, 2010. 128
- [68] David Chaum. Security without identification: Transaction systems to make big brother obsolete. *Communications of the ACM*, 28(10):1030–1044, 1985. 167
- [69] Roberto Chinnici, Hugo Haas, Amelia A. Lewis, Jean-Jacques Moreau, David Orchard, and Sanjiva Weerawarana (editors). *Web Services Description Language (WSDL) Version 2.0 Part 2: Adjuncts*. W3C Recommendation, 2007. <http://www.w3.org/TR/wsdl20-adjuncts/>. 89
- [70] Roberto Chinnici, Jean-Jacques Moreau, Arthur Ryman, and Sanjiva Weerawarana (editors). *Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language*. W3C Recommendation, 2007. <http://www.w3.org/TR/wsdl20/>. 89
- [71] John H. Clippinger. *Higgins Towards a Foundation Layer for the Social Web*. Higgins — working draft, 2011. <http://www.socialphysics.org/images/Higgins6.04.06.doc>. 131, 132
- [72] Art Conklin, Glenn Dietrich, and Diane Walz. Password-based authentication: a system perspective. In *Proceedings of HICSS '04 — the 37th Annual Hawaii International Conference on System Sciences — Track 7*. IEEE Computer Society, Los Alamitos, California, 70170b, 2004. 273
- [73] Douglas Crockford. *The application/json Media Type for JavaScript Object Notation (JSON)*. IETF: RFC 4627, 2006. <http://tools.ietf.org/html/rfc4627>. 354

- [74] Matthew Crowley. *Pro Internet Explorer 8 & 9 Development: Developing Powerful Applications for the Next Generation of IE*. Apress, New York, 2010. 352, 371, 372
- [75] Ivan Damgård. Payment systems and credential mechanisms with provable security against abuse by individuals. In Shafi Goldwasser, editor, *Proceedings on Advances in cryptology, Santa Barbara, California, CRYPTO '88*, pages 328–335. Springer-Verlag, New York, 1990. 167
- [76] Neil Daswani, Christoph Kern, and Anita Kesavan. *Foundations of Security: What Every Programmer Needs to Know*. Apress, Berkeley, California, 2007. 66, 81, 86, 221
- [77] Joshua Davies. *Implementing SSL/TLS Using Cryptography and PKI*. John Wiley & Sons, New York, 2011. 88
- [78] Jan De Clercq. Single sign-on architectures. In George I. Davida, Yair Frankel, and Owen Rees, editors, *Proceedings of InfraSec'02 — the International Conference on Infrastructure Security, October 1–3, 2002, Bristol, UK*, volume 2437 of LNCS, pages 40–58. Springer-Verlag, Berlin, Heidelberg, 2002. 100, 312
- [79] Marco De Luca. *Password Management for Distributed Environments*. VDM Verlag, Saarbrücken, 2008. 273
- [80] Giovanni Della-Libera et al. *Web Services Security Policy Language (WS-Security Policy)*, 2005. <http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-secpol/ws-secpol.pdf>. 90
- [81] Alexander W. Dent and Chris J. Mitchell. *User's Guide To Cryptography And Standards*. Artech House, New York, 2004. 66

- [82] Tim Dierks and Christopher Allen. *The TLS Protocol — Version 1.0*. IETF: RFC 2246, 1999. <http://www.ietf.org/rfc/rfc2246.txt>. 88
- [83] Tim Dierks and Eric Rescorla. *The Transport Layer Security (TLS) Protocol — Version 1.2*. IETF: RFC 5246, 2008. <http://tools.ietf.org/html/rfc5246>. 88
- [84] Donald E. Eastlake and Paul E. Jones. *US Secure Hash Algorithm 1 (SHA1)*. IETF: RFC 3174, 2001. <http://www.ietf.org/rfc/rfc3174.txt>. 70
- [85] Chad La Joie (editor). *WS-Trust 1.3 Interoperability Profile — Working Draft 01*. SWITCH, 2008. <http://www.switch.ch/grid/support/documents/wst-interop-wd01.pdf>. 203
- [86] Eran Hammer-Lahav (editor). *The OAuth 1.0 Protocol*. IETF: RFC 5849, 2010. <http://tools.ietf.org/html/rfc5849>. 147
- [87] Ian Hickson (editor). *HTML5 — A vocabulary and associated APIs for HTML and XHTML*. W3C Working Draft, 2011. <http://www.w3.org/TR/html5/>. 79
- [88] Tim Moses (editor). *eXtensible Access Control Markup Language (XACML) — Version 2.0*. OASIS standard, 2005. http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf. 63
- [89] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Advances in Cryptology: Proceedings of CRYPTO '84, Santa Barbara, California*, volume 196 of LNCS, pages 10–18. Springer-Verlag, Berlin, 1984. 69
- [90] Ahmed El-Rabbany. *Introduction to GPS: The Global Positioning System*. Artech House, New York, 2nd edition, 2006. 56

-
- [91] Roy T. Fielding, James Getty, Jeffrey Mogul, Henrik Frystyk, Larry Masinter, Paul Leach, and Tim Berners-Lee. *Hypertext Transfer Protocol — HTTP/1.1*. IETF: RFC 2616, 1999. <http://tools.ietf.org/html/rfc2616>. 82, 83, 87
- [92] Brad Fitzpatrick, David Recordon, Johnny Bufu, and Josh Hoyt. *OpenID Authentication 2.0 — Final*, 2007. http://openid.net/specs/openid-authentication-2_0.html. 136, 137, 142
- [93] Dinei Florêncio and Cormac Herley. One-time password access to any server without changing the server. In Tzong-Chen Wu, Chin-Laung Lei, Vincent Rijmen, and Der-Tsai Lee, editors, *Proceedings of ISC '08 — the 11th International Conference on Information Security, September 15–18, 2008, Taipei, Taiwan*, volume 5222 of LNCS, pages 401–420. Springer-Verlag, Berlin, Heidelberg, 2008. 289, 328
- [94] Seth Fogie, Jeremiah Grossman, Robert Hansen, Anton Rager, and Petko D. Petkov. *XSS Attacks: Cross Site Scripting Exploits and Defence*. Syngress, Waltham, Massachusetts, 2007. 58, 150
- [95] Ned Freed and Nathaniel S. Borenstein. *Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies*. IETF: RFC 2045, 1996. <http://www.ietf.org/rfc/rfc2045.txt>. 215
- [96] Sebastian Gajek, Jörg Schwenk, Michael Steiner, and Chen Xuan. Risks of the CardSpace protocol. In Pierangela Samarati, Moti Yung, Fabio Martinelli, and Claudio Agostino Ardagna, editors, *Proceedings of ISC '09 — the 12th International Conference on Information Security, September 7–9, 2009, Pisa, Italy*, volume 5735 of LNCS, pages 278–293. Springer-Verlag, Berlin, Heidelberg, 2009. 111
- [97] Eimear Gallery. An overview of trusted computing technology. In C. J. Mitchell, editor, *Trusted Computing*, chapter 3, pages 29–114. IEE Press, London, 2005. 200, 354

- [98] Sergio Sánchez García and Ana Gómez Oliva. Solving identity management and interoperability problems at pan-European level. In Robert Meersman, Pilar Herrero, and Tharam S. Dillon, editors, *Proceedings of OTM '09 — On the Move to Meaningful Internet Systems, Confederated International Workshops and Posters, November 1–6, 2009, Vilamoura, Portugal*, volume 5872 of LNCS, pages 805–809. Springer-Verlag, Berlin, Heidelberg, 2009. 204
- [99] Sergio Sánchez García and Ana Gómez Oliva. Improvements of pan-European IDM architecture to enable identity delegation based on X.509 proxy certificates and SAML. In Pierangela Samarati, Michael Tunstall, Joachim Posegga, Konstantinos Markantonakis, and Damien Sauveron, editors, *Proceedings of WISTP '10 — the 4th IFIP WG 11.2 International Workshop on Information Security Theory and Practices. Security and Privacy of Pervasive Systems and Smart Devices, April 12–14, 2010, Passau, Germany*, volume 6033 of LNCS, pages 183–198. Springer, 2010. 204
- [100] Sergio Sánchez García, Ana Gómez Oliva, Emilia Pérez Belleboni, and Iván Pau de la Cruz. Solving identity delegation problem in the e-government environment. *International Journal of Information Security*, 10(6):351–372, 2011. 204
- [101] Simson Garfinkel. *PGP: Pretty Good Privacy*. O'Reilly Media, Sebastopol, California, 1994. 69
- [102] Britta Glade (editor). *Identity Assurance Framework: Assurance Levels*. Kantara Initiative, 2009. <http://kantarainitiative.org/confluence/download/attachments/38371432/Kantara+IAF-1200-Levels+of+Assurance.pdf>. 73, 74, 75
- [103] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems (Extended Abstract). In

- Robert Sedgewick, editor, *Proceedings of STOC '85 — the 17th annual ACM symposium on Theory of computing, May 6–8, 1985, Providence, Rhode Island*, pages 291–304. ACM, New York, 1985. 168
- [104] David Grawrock. *Dynamics of a Trusted Platform: A Building Block Approach*. Intel Press, Hillsboro, Oregon, 2009. 70, 200, 354
- [105] Martin Gudgin, Marc Hadley, Noah Mendelsohn, Jean-Jacques Moreau, Henrik Frystyk Nielsen, Anish Karmarkar, and Yves Lafon (editors). *SOAP Version 1.2 Part 1: Messaging Framework*. W3C Recommendation, 2007. <http://www.w3.org/TR/soap12-part1/>. 89
- [106] Scott B. Guthery and Mary J. Cronin. *Mobile Application Development with SMS and SIM Toolkit*. McGraw-Hill, New York, 2002. 317, 322
- [107] Eran Hammer-Lahav, David Recordon, and Dick Hardt (editors). *The OAuth 2.0 Authorization Protocol — draft-ietf-oauth-v2-20*, 2011. <http://tools.ietf.org/html/draft-ietf-oauth-v2-20>. 147, 149
- [108] Dick Hardt, Johnny Bufu, and Josh Hoyt. *OpenID Attribute Exchange 1.0 — Final*. Sxip Identity and JanRain, 2007. http://openid.net/specs/openid-attribute-exchange-1_0.html. 144
- [109] Jonathan Hart, Konstantinos Markantonakis, and Keith Mayes. Website credential storage and two-factor web authentication with a Java SIM. In Pierangela Samarati, Michael Tunstall, Joachim Posegga, Konstantinos Markantonakis, and Damien Sauveron, editors, *Proceedings of WISTP '10 — the 4th IFIP WG 11.2 International Workshop on Information Security Theory and Practices. Security and Privacy of Pervasive Systems and Smart Devices, April 12–14, 2010, Passau, Germany*, volume 6033 of LNCS, pages 229–236. Springer, Berlin, Heidelberg, 2010. 201, 327

- [110] Cormac Herley, Paul C. van Oorschot, and Andrew S. Patrick. Passwords: If we're so smart, why are we still using them? In Roger Dingledine and Philippe Golle, editors, *Proceedings of FC '9 — the 13th International Conference on Financial Cryptography and Data Security, February 23–26, 2009, Accra Beach, Barbados. Revised Selected Papers*, volume 5628 of LNCS. Springer-Verlag, Berlin, Heidelberg, 230–237, 2009. 34, 273, 337
- [111] Kipp E. B. Hickman. *The SSL protocol*. Netscape, 1995. <http://tools.ietf.org/pdf/draft-hickman-netscape-ssl-00.pdf>. 88
- [112] Long Nguyen Hoang, Pekka Laitinen, and N. Asokan. Secure roaming with identity metasystems. In Kent E. Seamons, Neal McBurnett, and Tim Polk, editors, *Proceedings of IDtrust '08 — the 7th Symposium on Identity and Trust on the Internet, March 4–6, 2008, Gaithersburg, Maryland*, volume 283 of ACM International Conference Proceeding Series, pages 36–47. ACM, New York, 2008. 129, 340
- [113] Jeff Hodges. *Technical Comparison: OpenID and SAML — Draft 07a (Whitepaper)*, 2009. <http://identitymeme.org/doc/draft-hodges-saml-openid-compare.html>. 144
- [114] Dennis Hofheinz and Eike Kiltz. Secure hybrid encryption from weakened key encapsulation. In Alfred Menezes, editor, *Proceedings of CRYPTO '07 — the 27th Annual International Cryptology Conference on Advances in Cryptology, August 19–23, 2007, Santa Barbara, California*, volume 4622 of LNCS, pages 553–571. Springer, Berlin, Heidelberg, 2007. 70
- [115] Arnaud Le Hors, Philippe Le Hégarret, Lauren Wood, Gavin Nicol, Jonathan Robie, Mike Champion, and Steve Byrne (editors). *Document*

-
- Object Model (DOM) Level 2 Core Specification*. W3C Recommendation, 2000. <http://www.w3.org/TR/DOM-Level-2-Core/>. 81
- [116] Josh Hoyt, Jonathan Daugherty, and David Recordon. *OpenID Simple Registration Extension 1.0*. Jan-Rain and VeriSign, 2006. http://openid.net/specs/openid-simple-registration-extension-1_0.html. 142, 245
- [117] Mohammed Hussain. *The Design and Applications of a Privacy-Preserving Identity and Trust-Management System*. PhD thesis, Queen's University, Kingston, Ontario, Canada, 2010. http://qspace.library.queensu.ca/bitstream/1974/5520/1/Hussain_Mohammed_201004_PhD.pdf. 47, 104, 168
- [118] International Organization for Standardisation, Genève, Switzerland. *ISO/IEC Second CD 24760 — Information technology — Security techniques — A framework for identity management*, 2010. 99
- [119] International Telecommunication Union — Telecommunication Standardisation Sector (ITU-T). *Baseline capabilities for enhanced global identity management trust and interoperability — Draft Recommendation — ITU-T X.1250 (X.idmreq)*, 2009. 47
- [120] International Telecommunication Union — Telecommunication Standardisation Sector (ITU-T). *Baseline identity management terms and definitions — Recommendation — ITU-T X.1252*, 2010. <http://kantarainitiative.org/confluence/download/attachments/45059055/T-REC-X.1252-201004-I!!PDF-E.pdf>. 47, 48
- [121] International Telecommunication Union — Telecommunication Standardisation Sector (ITU-T). Y.2720 (Y.ngnIdMframework). *NGN Identity management framework — Draft Recommendation*, 2008. 47, 93

- [122] ISO/IEC 10118-1:2000. *Information technology — Security techniques — Hash-functions — Part 1: General*, 2000. http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=31143. 70
- [123] ISO/IEC 10118-2:2010. *Information technology — Security techniques — Hash-functions — Part 2: Hash-functions using an n-bit block cipher*, 2010. http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=44737. 70
- [124] ISO/IEC 10118-3:2004. *Information technology — Security techniques — Hash-functions — Part 3: Dedicated hash-functions*, 2004. http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=39876. 70
- [125] ISO/IEC 10118-4:1998. *Information technology — Security techniques — Hash-functions — Part 2: Hash-functions using modular arithmetic*, 1998. http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=25429. 70
- [126] ISO/IEC 14888-1:2008. *Information technology — Security techniques — Digital signatures with appendix — Part 1: General*, 2008. http://www.iso.org/iso/iso_catalogue/catalogue_ics/catalogue_detail_ics.htm?csnumber=44226. 69
- [127] ISO/IEC 14888-2:2008. *Information technology — Security techniques — Digital signatures with appendix — Part 2: Integer factorisation based mechanisms*, 2008. http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=44227. 69
- [128] ISO/IEC 14888-3:2006. *Information technology — Security techniques — Digital signatures with appendix — Part 3: Discrete logarithm based mechanisms*, 2006. http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=43656. 69

- [129] ISO/IEC 18033–2:2006. *Information technology — Security techniques — Encryption algorithms — Part 2: Asymmetric ciphers*, 2006. http://www.iso.org/iso/catalogue_detail.htm?csnumber=37971. 69
- [130] ISO/IEC 18033–3:2010. *Information technology — Security techniques — Encryption algorithms — Part 3: Block ciphers*, 2010. http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=54531. 67
- [131] ISO/IEC 18033–4:2011. *Information technology — Security techniques — Encryption algorithms — Part 4: Stream ciphers*, 2011. http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=54532. 67
- [132] ISO/IEC 24760-1:2011(E). *Information technology — Security techniques — A framework for identity management — Part 1: Terminology and concepts*, 2012. 47, 48, 53, 54, 61, 94
- [133] ISO/IEC 27000:2009(E). *Information technology — Security techniques — Information security management systems — Overview and vocabulary*, 2009. http://www.iso.org/iso/catalogue_detail?csnumber=41933. 64, 65, 72
- [134] ISO/IEC 27001:2005(E). *Information technology — Security techniques — Information security management systems — Requirements*, 2005. http://www.iso.org/iso/catalogue_detail?csnumber=42103. 64
- [135] ISO/IEC 9796–2:2010. *Information technology — Security techniques — Digital signature schemes giving message recovery — Part 2: Integer factorisation based mechanisms*, 2010. http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=54788. 69

- [136] ISO/IEC 9796-3:2006. *Information technology — Security techniques — Digital signature schemes giving message recovery — Part 3: Discrete logarithm based mechanisms*, 2006. http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=42228. 69
- [137] ISO/IEC 9797-1:2011. *Information technology — Security techniques — Message Authentication Codes (MACs) — Part 1: Mechanisms using a block cipher*, 2011. http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=50375. 67
- [138] ISO/IEC 9797-2:2011. *Information technology — Security techniques — Message Authentication Codes (MACs) — Part 2: Mechanisms using a dedicated hash-function*, 2011. http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=51618. 67
- [139] ITU-T Recommendation X.509. *Information technology — Open systems interconnection — The Directory: Public-key and attribute certificate frameworks*, 2008. <http://www.itu.int/rec/T-REC-X.509-200811-I/en>. 76
- [140] Ravi Chandra Jammalamadaka, Timothy W. van der Horst, Sharad Mehrotra, Kent E. Seamons, and Nalini Venkasubramanian. Delegate: A proxy based architecture for secure website access from an untrusted machine. In *Proceedings of ACSAC '06 — the 22nd Annual Computer Security Applications Conference, December 11–15, 2006, Miami Beach, Florida*, pages 57–66. IEEE Computer Society, Washington DC, 2006. 328
- [141] Hosung Jo, Hwanjin Lee, Kilsoo Chun, and Heejin Park. Interoperability and anonymity for ID management systems. In *Proceedings of ICACT '09 — the 11th International Conference on Advanced Communi-*

- cation Technology, February 15–18, 2009, Phoenix Park, Dublin, Ireland, volume 2, pages 1257–1260. IEEE, New York, 2009. 205*
- [142] Michael B. Jones. *A Guide to Using the Identity Selector Interoperability Profile V1.5 within Web Applications and Browsers*. Microsoft, 2008. 111, 112, 118, 119, 121, 123, 124, 222
- [143] Michael B. Jones and Michael McIntosh (editors). *Identity Metasystem Interoperability Version 1.0 (IMI 1.0)*. OASIS Standard, 2009. <http://docs.oasis-open.org/imi/identity/v1.0/identity.html>. 101, 110, 111, 115, 116, 121, 122, 124, 133
- [144] Ivar Jørstad, Do Van Thuan, Tore Jønvik, and Do Van Thanh. Bridging CardSpace and Liberty Alliance with SIM authentication. In *Proceedings of ICIN '07 — the 10th International Conference on Intelligence in Next Generation Networks*, pages 8–13. Adera, Pessac, 2007. 206, 227, 330
- [145] Audun Jøsang and Simon Pope. *User Centric Identity Management*. Proceedings of AusCERT '05 — the Australian Computer Emergency Response Team Conference, 2005. <http://persons.unik.no/josang/papers/JP2005-AusCERT.pdf>. 95
- [146] Audun Jøsang, Mohammed Al Zomai, and Suriadi Suriadi. Usability and privacy in identity management architectures. In Ljiljana Brankovic, Paul D. Coddington, John F. Roddick, Chris Stekete, James R. Warren, and Andrew L. Wendelborn, editors, *ACSW Frontiers 2007, proceedings of the 5th Australasian Symposium on Grid Computing and e-Research (AusGrid 2007), the Fifth Australasian Information Security Workshop (Privacy Enhancing Technologies) (AISW '07), and the Australasian Workshop on Health Knowledge Management and Discovery (HKMD '07). Proceedings, January 30 to February 2, 2007, Ballarat, Victoria, Australia*, volume 68 of CRPIT, pages 143–152. Australian Computer Society, 2007. 103

- [147] Phil Karn, Perry Metzger, and William Allen Simpson. *The ESP Triple DES Transform*, 1995. <http://tools.ietf.org/html/rfc1851>. 67
- [148] Kalle Kaukonen and Rodney Thayer. *A Stream Cipher Encryption Algorithm Arcfour*. IETF: RFC 1851, 1997. <http://tools.ietf.org/html/draft-kaukonen-cipher-arcfour-01>. 67
- [149] Sampo Kellomäki and Rob Lockhart (editors). *Liberty ID-SIS employee profile service specification*. Liberty Alliance Project, 2005. <http://www.projectliberty.org/liberty/content/download/1031/7155/file/liberty-idsis-ep-v1.1.pdf>. 152
- [150] Seung Hyun Kim et al. *OpenID Authentication Method Using Identity Selector*. United States, Patent Application Publication, Pub. No. US 2009/0249078 A1, 2009. 206, 254
- [151] Adrian Kingsley-Hughes, Kathie Kingsley-Hughes, and Daniel Read. *VBScript Programmer's Reference*. Wrox, 3rd edition, 2007. 221
- [152] Hristo Koshutanski, Michaela Ion, and Luigi Telesca. Towards user-centric identity interoperability for digital ecosystems. *International Journal on Advances in Security*, 1(1):26–38, 2009. 96, 204
- [153] David Kristol. *HTTP State Management Mechanism*. IETF: RFC 2045, 2000. <http://tools.ietf.org/html/rfc2965>. 57
- [154] Jacek Lach. Using mobile devices for user authentication. In Andrzej Kwiecien, Piotr Gaj, and Piotr Stera, editors, *Proceedings of CN '10 — the 17th Conference on Computer Networks, June 15–19, 2010, Ustrón, Poland*, volume 79 of *Communications in Computer and Information Science*, pages 263–268. Springer, Berlin, Heidelberg, 2010. 316, 330

- [155] Marc Langheinrich (editor). *A P3P Preference Exchange Language 1.0 (APPEL1.0)*. W3C, 2002. <http://www.w3.org/TR/P3P-preferences/>. 62
- [156] Gwenaël Le Bodic. *Mobile Messaging Technologies and Services SMS, EMS and MMS*. Wiley, Chichester, 2003. 317, 322
- [157] John Leach. Improving user security behaviour. *Computers & Security*, 22:685–692, 2003. 35, 198
- [158] HwanJin Lee, InKyung Jeun, Kilsoo Chun, and Junghwan Song. A new anti-phishing method in OpenID. In *Proceedings of SECURWARE — the 2nd International Conference on Emerging Security Information, Systems and Technologies, August 25–31, 2008, Cap Esterel, France*, pages 243–247. IEEE, New York, 2008. 144
- [159] Sing Li and Jonathan Knudsen. *Beginning J2ME From Novice to Professional*. Apress, New York, 3rd edition, 2005. 329
- [160] Ari Luotonen. *Web Proxy Servers*. Prentice Hall PTR, New Jersey, 1997. 157, 289, 328
- [161] Eve Maler, Prateek Mishra, and Rob Philpott (editors). *Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML) V1.1*. OASIS, 2003. <http://www.oasis-open.org/committees/download.php/3406/oasis-sstc-saml-core-1.1.pdf>. 92
- [162] Nazir A Malik and Allan Tomlinson. Web-services architecture for pervasive computing environment. *Pakistan Journal of Science*, 61(3):153–157, 2009. 90
- [163] Mohammad Mannan and P. C. Van Oorschot. Using a personal device to strengthen password authentication from an untrusted computer. In Sven Dietrich and Rachna Dhamija, editors, *Financial Cryptography and Data Security, the 11th International Conference, FC '07, and the 1st*

- International Workshop on Usable Security, USEC '07, February 12–16, 2007, Scarborough, Trinidad and Tobago. Revised Selected Papers*, volume 4886 of LNCS, pages 88–103. Springer, Berlin, Heidelberg, 2007. 329
- [164] Erika McCallister, Tim Grance, and Karen Scarfone. *Guide to Protecting the Confidentiality of Personally Identifiable Information (PII) — NIST Special Publication 800–122. Recommendations of NIST*, 2010. <http://csrc.nist.gov/publications/nistpubs/800-122/sp800-122.pdf>. 52
- [165] Aleecia M. McDonald and Lorrie Faith Cranor. A survey of the use of Adobe flash local shared objects to respawn HTTP cookies. Technical Report CMU-CyLab–11–001, Carnegie Mellon, CyLab, Carnegie Mellon University, Pittsburgh, 2011. <http://www.casos.cs.cmu.edu/publications/papers/CMUCyLab11001.pdf>. 58
- [166] Mark McGloin and Phil Hunt. *OAuth 2.0 Threat Model and Security Considerations — draft-ietf-oauth-v2-threatmodel-00*, 2011. <http://tools.ietf.org/html/draft-ietf-oauth-v2-threatmodel-00>. 150
- [167] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, Boca Raton, Florida, 1996. 66, 68
- [168] Marc Mercuri. *Beginning Information Cards and CardSpace: From Novice to Professional*. Apress, New York, 2007. 109, 110, 111, 115, 203
- [169] Microsoft. *Microsoft's Vision for an Identity Metasystem*, 2005. <http://msdn.microsoft.com/en-us/library/ms996422.aspx>. 104, 109, 111

- [170] Microsoft and Ping Identity. *An Implementer's Guide to the Identity Selector Interoperability Profile V1.5.*, 2008. <http://msdn.microsoft.com/en-us/windows/aa663320.aspx>. 124
- [171] Marino Miculan and Caterina Urban. Formal analysis of Facebook Connect single sign-on authentication protocol. In *SOFSEM '11 (Software Seminar): Theory and Practice of Computer Science — the 37th Conference on Current Trends in Theory and Practice of Computer Science, January 22–28, 2011, Slovakia. Proceedings of Student Research Forum*, pages 99–116, 2011. 151
- [172] Joaquin Miller (editor). *Yadis Specification — Version 1.0*, 2006. <http://yadis.org/wiki/>. 138
- [173] R. L. 'Bob' Morgan, Scott Cantor, Steven Carmody, Walter Hoehn, and Ken Klingenstein. Federated Security : The Shibboleth Approach. *EDUCAUSE Quarterly*, 27(4):12–17, 2004. 162
- [174] F Nachira, P Dini, A Nicolai, M Le Louarn, and L Rivera Lèon (editors). *Digital Business Ecosystems*. European Commission, 2007. <http://www.digital-ecosystems.org/book/de-book2007.html>. 204
- [175] Anthony Nadalin, Marc Goodner, Martin Gudgin, Abbie Barbir, and Hans Granqvist (editors). *WS-Trust 1.4*, 2009. <http://docs.oasis-open.org/ws-sx/ws-trust/v1.4/ws-trust.html>. 173
- [176] Anthony Nadalin, Chris Kaler, Ronald Monzillo, and Phillip Hallam-Baker (editors). *Web Services Security: SOAP Message Security 1.1 (WS-Security 2004)*. OASIS Standard Specification, 2006. <http://docs.oasis-open.org/wss/v1.1/wss-v1.1-spec-os-SOAPMessageSecurity.pdf>. 91

- [177] Toru Nakanishi, Hiroki Fujii, Yuta Hira, and Nobuo Funabiki. Revocable group signature schemes with constant costs for signing and verifying. In Stanislaw Jarecki and Gene Tsudik, editors, *Proceedings of PKC '09 — the 12th International Conference on Practice and Theory in Public Key Cryptography, March 18–20, 2009, Irvine, California*, volume 5443 of LNCS, pages 463–480. Springer, Berlin, Heidelberg, 2009. 168
- [178] Arun Nanda and Michael B. Jones. *Identity Selector Interoperability Profile V1.5*. Microsoft, 2008. 123
- [179] National Institute of Standards and Technology (NIST). *Announcing the Advanced Encryption Standard (AES), FIPS 197*, 2001. <http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf>. 67
- [180] Tom Negrino and Dori Smith. *JavaScript and Ajax for the Web: Visual QuickStart Guide*. Peachpit Press, Berkeley, California, 7th edition, 2008. 58, 78, 81, 220
- [181] Eric Newcomer. *Understanding Web Services: XML, WSDL, SOAP and UDDI*. Addison Wesley, Reading, Massachusetts, 2002. 89, 90
- [182] NIST. *FIPS PUB 180–2: Secure Hash Standard*, 2002. <http://csrc.nist.gov/publications/fips/fips180-2/fips180-2.pdf>. 70
- [183] NIST. *FIPS PUB 198: The Keyed-Hash Message Authentication Code (HMAC)*, 2002. <http://csrc.nist.gov/publications/fips/fips198/fips-198a.pdf>. 67, 70
- [184] Information Sciences Institute, University of Southern California. *Internet Protocol*. IETF: RFC 791, 1981. <http://tools.ietf.org/html/rfc791>. 95

- [185] Rolf Oppliger, Sebastian Gajek, and Ralf Hauser. Security of Microsoft's identity metasytem and CardSpace. In *Proceedings of KiVS '07 — the Kommunikation in Verteilten Systemen*. VDE Publishing House, Berlin, 63–74, 2007. 119
- [186] Organisation for Economic Co-operation and Development (OECD). *OECD guidelines on the protection of privacy and transborder flows of personal data*, 1980. [http://www.oecd.org/document/18/0,3746,en_2649_34255_1815186_1_1_1_1,00&en-US\\$01DBC.html](http://www.oecd.org/document/18/0,3746,en_2649_34255_1815186_1_1_1_1,00&en-US$01DBC.html). 60
- [187] Organisation for Economic Co-operation and Development (OECD). *At a Crossroads: 'personhood' and Digital Identity in the Information Society*, 2008. <http://www.oecd.org/dataoecd/31/6/40204773.doc>. 49
- [188] Suhas Pai, Yash Sharma, Sunil Kumar, Radhika M. Pai, and Sanjay Singh. Formal verification of OAuth 2.0 using Alloy framework. In *Proceedings of CSNT '11 — the International Conference on Communication Systems and Network Technologies, June 3–5, 2011, Katra, Jammu, India*, pages 655–659. IEEE Computer Society, Los Alamitos, California, 2011. 147
- [189] John Palfrey and Urs Gasser. *Digital Identity Interoperability and eInnovation*. Berkman Publication Series, 2007. <http://cyber.law.harvard.edu/interop/pdfs/interop-digital-id.pdf>. 73, 206
- [190] Christian Paquin. *U-Prove Technology Integration into the Identity Metasytem V1.0*. Microsoft, 2010. 170, 172, 175, 176
- [191] Christian Paquin. *U-Prove Cryptographic Specification V1.1 — Draft Revision 1*. Microsoft, 2011. 170

- [192] Christian Paquin. *U-Prove Technology Overview V1.1 — Draft Revision 1*. Microsoft, 2011. 171
- [193] Christian Paquin. *U-Prove WS-Trust Profile V1.0 — Draft Revision 1*. Microsoft, 2011. 171
- [194] Christian Paquin and Greg Thompson. *U-Prove CTP White Paper*. Microsoft, 2010. 169
- [195] Andreas Pashalidis and Chris J. Mitchell. A taxonomy of single sign-on systems. In Rei Safavi-Naini and Jennifer Seberry, editors, *Proceedings of ACISP '03 — the 8th Australasian conference on Information security and privacy, July 9–11, 2003, Wollongong, Australia*, volume 2727 of *LNCS*, pages 249–264. Springer-Verlag, Berlin, Heidelberg, 2003. 100, 108, 312
- [196] Andreas Pashalidis and Chris J. Mitchell. Impostor: A single sign-on system for use from untrusted devices. In *Proceedings of IEEE Globecom '04 — the Global Telecommunications Conference, November 29 to December 3, 2004, Dallas, Texas*, volume 4, pages 2191–2195. IEEE Press, 2004. 328
- [197] Andreas Pfitzmann and Marit Hansen. *A terminology for talking about privacy by data minimization: Anonymity, Unlinkability, Undetectability, Unobservability, Pseudonymity, and Identity Management (v0.34)*, 2010. http://dud.inf.tu-dresden.de/literatur/Anon_Terminology_v0.34.pdf. 47, 53, 54
- [198] Thomas A. Powell and Fritz Schneider. *Javascript: The Complete Reference*. McGraw-Hill Osborne Media, Berkeley, California, 2nd edition, 2004. 220
- [199] Calvin Powers and Matthias Schunter (editors). *Enterprise Privacy Authorization Language (EPAL 1.2)*. W3C Member Submission 10

- November 2003, 2003. <http://www.w3.org/Submission/2003/SUBM-EPAL-20031110/>. 63
- [200] Bart Priem, Ronald Leenes, Eleni Kosta, and Aleksandra Kuczerawy. The identity landscape. In Jan Camenisch, Ronald Leenes, and Dieter Sommer, editors, *Digital Privacy — PRIME (Privacy and Identity Management for Europe)*, volume 6545 of *LNCS*, pages 33–51. Springer, Berlin, Heidelberg, 2011. 47
- [201] Dave Raggett. *HTML 3.2 Reference Specification*. W3C Recommendation, 1997. <http://www.w3.org/TR/REC-html32.html>. 79, 87
- [202] Dave Raggett, Arnaud Le Hors, and Ian Jacobs (editors). *HTML 4.01 Specification*. W3C Recommendation, 1999. <http://www.w3.org/TR/html401/>. 78, 79, 87
- [203] David Recordon and Brad Fitzpatrick. *OpenID Authentication 1.1*, 2006. http://openid.net/specs/openid-authentication-1_1.html. 136, 137
- [204] David Recordon, Michael B. Jones, and Nat Sakimura. *OpenID Provider Authentication Policy Extension 1.0*, 2008. http://openid.net/specs/openid-provider-authentication-policy-extension-1_0.html. 139
- [205] Drummond Reed and Dave McAlpin (editors). *eXtensible Resource Identifier (XRI) Syntax V2.0*. OASIS, 2005. <http://www.oasis-open.org/committees/download.php/15377>. 137
- [206] Eric Rescorla. *Diffie-Hellman Key Agreement Method*. IETF: RFC 2631, 1999. <http://www.ietf.org/rfc/rfc2631.txt>. 77
- [207] Eric Rescorla. *HTTP Over TLS*. IETF: RFC 2818, 2000. <http://www.ietf.org/rfc/rfc2818.txt>. 88

- [208] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978. 69
- [209] Jonathan Rosenberg, Henning Schulzrinne, Gonzalo Camarillo, Alan Johnston, Jon Peterson, Robert Sparks, Mark Handley, and Eve Schooler. *SIP: Session Initiation Protocol*. IETF: RFC 3261, 2002. <http://www.ietf.org/rfc/rfc3261.txt>. 111
- [210] Mary C. Rundle and Paul Trevithick. Interoperability in the new digital identity infrastructure. *SSRN eLibrary*, 2007. <http://ssrn.com/paper=962701>. 111, 203, 206
- [211] Tom Scavo and Scott Cantor (editors). *Shibboleth Architecture — Technical Overview*. Internet2, 2005. <http://shibboleth.internet2.edu/docs/draft-mace-shibboleth-tech-overview-latest.pdf>. 162, 163
- [212] Marko Schuba, Volker Gerstenberger, and Paul Lahaije. *Internet ID — Flexible Re-use of Mobile Phone Authentication Security for Service Access*, 2004. http://www.ericsson.com/res/thecompany/docs/journal_conference_papers/service_layer/internet_id_nordsec.pdf. 329, 330
- [213] Robert W. Shirey. *Internet Security Glossary*. IETF: RFC 2828, 2000. <http://www.ietf.org/rfc/rfc2828.txt>. 65, 71, 72, 76
- [214] William Stallings. *Network Security Essentials: Applications and Standards*. Pearson Education, New Jersey, 4th edition, 2010. 88
- [215] William Stallings. *Cryptography and Network Security — Principles and Practice*. Pearson Education, New Jersey, 5th edition, 2011. 64, 66, 68, 70, 72, 77, 88

- [216] Latanya Sweeney. *Uniqueness of Simple Demographics in the U.S. Population*. LIDAP-WP4 Carnegie Mellon University, Laboratory for International Data Privacy, Pittsburgh, PA: 2000, 2000. 53
- [217] Stephen A. Thomas. *SSL and TLS Essentials: Securing the Web*. John Wiley & Sons, New York, 2000. 88
- [218] Allan Tomlinson. Introduction to the TPM. In *Smart Cards, Tokens, Security and Applications*, pages 155–172. Springer, 2008. 200, 354
- [219] Jonathan Tourzan and Yuzo Koga (editors). *Liberty ID-WSF web services framework overview*. Liberty Alliance Project, 2005. <http://www.projectliberty.org/liberty/content/download/1307/8286/file/liberty-idwsf-overview-v1.1.pdf>. 152, 153
- [220] Paul Trevithick. *From Information Cards to Relationship Cards*. (IIW IX) Internet Identity Workshop, November 3, 2009. <http://www.slideshare.net/idworkshop/relationship-cards-iiw-nov-3-2009>. 131
- [221] US Code. *Title 44 — Public Printing and Documents*, 1968. <http://uscode.house.gov/pdf/2006/2006usc44.pdf>. 64
- [222] US Government Accountability Office (GAO). *Privacy: Alternatives Exist for Enhancing Protection of Personally Identifiable Information (GAO Report 08–536)*, 2008. <http://www.gao.gov/new.items/d08536.pdf>. 52
- [223] Bart van Delft and Martijn Oostdijk. A security analysis of OpenID. In Elisabeth de Leeuw, Simone Fischer-Hübner, and Lothar Fritsch, editors, *Proceedings of IFIP IDMAN '10 — the 2nd IFIP WG 11.6 Working Conference on Policies and Research in Identity Management, November 18–*

- 19, 2010, Oslo, Norway, volume 343 of *IFIP Advances in Information and Communication Technology*, pages 73–84. Springer, Boston, 2010. 144
- [224] Branislav Vuksanovic and Haitham S. Al-Sinani. Two proposals for improving the image-based authentication system: H-IBAS-H. In *Proceedings of INTERNET '09 — the First International Conference on Evolving Internet, August 23–29, 2009, Cannes/La Bocca, French Riviera, France*, pages 168–171. IEEE Computer Society, Washington DC, 2009. 73
- [225] Gabe Wachob, Drummond Reed, Les Chasen, William Tan, and Steve Churchill (editors). *eXtensible Resource Identifier (XRI) Resolution Version 2.0*. OASIS, 2008. <http://docs.oasis-open.org/xri/2.0/specs/xri-resolution-V2.0.html>. 138
- [226] Thomas Wason (editor). *Liberty ID-FF architecture overview*. Liberty Alliance Project, 2005. <http://projectliberty.org/liberty/content/download/318/2366/file/draft-liberty-idff-arch-overview-1.2-errata-v1.0.pdf>. 152, 153, 156
- [227] Rigo Wenning and Matthias Schunter (editors). *The Platform for Privacy Preferences 1.1 (P3P1.1) Specification*. W3C, 2006. <http://www.w3.org/TR/P3P11/>. 54, 62
- [228] Graham Williamson, David Yip, Ilan Sharoni, and Kent Spaulding. *Identity Management: A Primer*. MC Press, Big Sandy, Texas, 2009. 47, 74, 100
- [229] Phillip J. Windley. *Digital Identity*. O'Reilly Media, Sebastopol, California, 2005. 47, 50, 52
- [230] Min Wu, Simson Garfinkel, and Rob Miller. Secure web authentication with mobile phones. In *DIMACS Workshop on Usable Privacy and Se-*

- curity Systems*, 2004. <http://homepages.mcs.vuw.ac.nz/~ian/shared/papers/secureweb.pdf>. 328
- [231] Po-Wah Yau and Allan Tomlinson. Towards privacy in a context-aware social network based recommendation system. In *Proceedings of PASSAT/SocialCom '11 — the 3rd International Conference on Privacy, Security, Risk and Trust, and the 3rd International Conference on Social Computing, October 9–11, 2011, Boston, Massachusetts*, pages 862–865. IEEE Computer Society, 2011. 54