

2013

## A New Simplified Federated Single Sign-on System

Chen Liang  
*Technological University Dublin*

Follow this and additional works at: <https://arrow.tudublin.ie/scienmas>

 Part of the [Computer and Systems Architecture Commons](#), [Data Storage Systems Commons](#), and the [Hardware Systems Commons](#)

---

### Recommended Citation

Liang, C.(2013) *A new simplified federated single sign-on system*. Masters Thesis. Technological University Dublin.

This Theses, Masters is brought to you for free and open access by the Science at ARROW@TU Dublin. It has been accepted for inclusion in Masters by an authorized administrator of ARROW@TU Dublin. For more information, please contact [yvonne.desmond@tudublin.ie](mailto:yvonne.desmond@tudublin.ie), [arrow.admin@tudublin.ie](mailto:arrow.admin@tudublin.ie), [brian.widdis@tudublin.ie](mailto:brian.widdis@tudublin.ie).



This work is licensed under a [Creative Commons Attribution-Noncommercial-Share Alike 3.0 License](#)



# **A New Simplified Federated Single Sign-on System**

By

Chen Liang B.Sc.

Supervisor: Dr. Fredrick Mtenzi

Supervisor: Mr. Paul Doyle

Thesis submitted to the Office of Postgraduate Studies and Research at the Dublin Institute of Technology in fulfilment of the requirements for the degree of Master of Philosophy (M.Phil.)

School of Computing

Dublin Institute of Technology

Kevin Street, Dublin 8, Ireland

# Abstract

Federated single sign-on (FSSO) provides single sign-on across multiple domains. The literature shows that current FSSO systems do not incorporate the authentication systems of desktop systems, web-based services/applications and non-web based services/applications. This is a challenge for the end-users because they need to be authenticated multiple times (by multiple authentication systems) to access all three services.

The work presented in this MPhil thesis addresses this challenge by developing a new simplified FSSO system that allows end-users to access desktop systems, web-based services/applications and non-web based services/applications using one authentication process. This new system achieves this using two major components: an “Authentication Infrastructure Integration Program (AIIP)” and an “Integration of Desktop Authentication and Web-based Authentication (IDAWA).” The AIIP acquires Kerberos tickets (for end-users who have been authenticated by a Kerberos single sign-on system in one network domain) from Kerberos single sign-on systems in different network domains without establishing trust between these Kerberos single sign-on systems. The IDAWA is an extension to the web-based authentication systems (i.e. the web portal), and it authenticates end-users by verifying the end-users’ Kerberos tickets.

This research also developed new criteria to determine which FSSO system can deliver true single sign-on to the end-users (i.e. allowing end-users to access desktop systems, web-based services/applications and non-web based services/applications using one authentication process). The evaluation shows that the new simplified FSSO system (i.e. the combination of AIIP and IDAWA) can deliver true single sign-on to the end-users. In addition, the evaluation shows the new simplified FSSO system has advantages over existing FSSO systems as it does not require additional modifications to network domains’ existing non-web based authentication infrastructures (i.e. Kerberos single sign-on systems) and their firewall rules.

## Declaration

I certify that this thesis which I now submit for examination for the award of *Master of Philosophy* is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my work.

This thesis was prepared according to the regulations for postgraduate study by research of the Dublin Institute of Technology and has not been submitted in whole or in part for another award in any other third level institution.

The work reported on in this thesis conforms to the principles and requirements of the DIT's guidelines for ethics in research.

DIT has permission to keep, lend or copy this thesis in whole or in part, on condition that any such use of the material of the thesis be duly acknowledged.

Signature \_\_\_\_\_ Date \_\_\_\_\_

Candidate

## Acknowledgements

I owe sincere and earnest thankfulness to my supervisors Dr. Fredrick Mtenzi and Mr. Paul Doyle for the support and guidance they showed me throughout my research and writing. In addition to directing my research, they advised me on how to overcome the difficulties during the process of my research.

I also would like to thanks Prof. Brendan O'Shea and Mr. Mark Deegan. This dissertation would not have been possible unless they helped me to secure my position as a research student in the School of Computing in the Dublin Institute of Technology.

I would like to show my gratitude to my parents for their support. They provided continue financial support that allowed me to achieved my goal.

# Table of Contents

<b>Abstract</b>	<b>i</b>
<b>Declaration</b>	<b>ii</b>
<b>Chapter 1: Introduction</b>	<b>1</b>
1.1 Research Background . . . . .	5
1.1.1 Authentication Models . . . . .	5
1.1.2 Federated Single Sign-on . . . . .	8
1.2 Research Aim . . . . .	11
1.3 Research Methodology . . . . .	12
1.4 Contributions . . . . .	14
1.5 Thesis Structure . . . . .	15
<b>Chapter 2: Authentication Systems</b>	<b>17</b>
2.1 Introduction . . . . .	17
2.2 Identity Management . . . . .	18
2.3 Authentication Systems . . . . .	20
2.3.1 Authentication Mechanisms . . . . .	20
2.3.2 Authentication Models . . . . .	24
2.4 Summary . . . . .	31
<b>Chapter 3: Federated Single Sign-on (FSSO) Systems</b>	<b>33</b>
3.1 Introduction . . . . .	33
3.2 Federated Single Sign-on Solutions in Web Space . . . . .	35
3.3 Federated Single Sign-on Solutions Beyond Web Space . . . . .	37

3.3.1	Project Moonshot . . . . .	38
3.3.2	Kerberos Secure Sharing . . . . .	39
3.3.3	SASL-SAML . . . . .	41
3.3.4	SAML-AAI/Kerberos . . . . .	43
3.4	Summary . . . . .	44
<b>Chapter 4: The Design of a New Simplified FSSO System</b>		<b>46</b>
4.1	Introduction . . . . .	46
4.2	Design Environment . . . . .	48
4.2.1	Simulation of Real World Applications . . . . .	48
4.2.2	Environment Specification . . . . .	52
4.3	Simulation Details . . . . .	56
4.3.1	End-user Database . . . . .	56
4.3.2	Kerberos Single Sign-on Simulation . . . . .	59
4.3.3	Ubuntu Desktop and Secure Shell Simulation . . . . .	65
4.3.4	SAML based FSSO System Simulation (Identity Provider) . . . . .	72
4.3.5	SAML based FSSO Simulation (Service Provider) . . . . .	73
4.4	New Simplified FSSO System Design . . . . .	73
4.4.1	Authentication Infrastructure Integration Program (AIIP) . . . . .	74
4.4.2	Integration of Desktop Authentication and Web-based Authentica- tion (IDAWA) . . . . .	81
4.4.3	Simplified Federated Single Sign-on System Life-cycles . . . . .	83
4.5	Summary . . . . .	90
<b>Chapter 5: Proof of Concept Implementation</b>		<b>91</b>
5.1	Introduction . . . . .	91
5.2	Technical Overview . . . . .	92
5.2.1	Development Environment and Programming Languages . . . . .	92
5.3	Implementation of Design Environment . . . . .	93
5.3.1	Kerberos Single Sign-on System . . . . .	93

5.3.2	Ubuntu Desktop and Secure Shell Protocol . . . . .	100
5.3.3	SAML based Federated Single Sign-on System Simulation . . . . .	107
5.4	Implementation of the New Simplified FSSO System . . . . .	110
5.4.1	Implementation of AIIP . . . . .	110
5.4.2	Implementation of IDAWA . . . . .	119
5.5	Demonstration of the New Simplified FSSO System . . . . .	119
5.5.1	Demonstration of AIIP . . . . .	122
5.6	Summary . . . . .	126
<b>Chapter 6:</b>	<b>Research Evaluation</b>	<b>128</b>
6.1	Introduction . . . . .	128
6.2	Criteria Development . . . . .	129
6.3	Evaluation using the Developed Criteria . . . . .	130
6.3.1	First Criterion . . . . .	130
6.3.2	Second Criterion . . . . .	131
6.3.3	Third Criterion . . . . .	133
6.3.4	Fourth Criterion . . . . .	135
6.3.5	Fifth Criterion . . . . .	137
6.4	Comparison Overview . . . . .	139
6.5	Summary . . . . .	143
<b>Chapter 7:</b>	<b>Discussion and Conclusion</b>	<b>144</b>
7.1	Introduction . . . . .	144
7.2	Contributions . . . . .	146
7.3	Critical Review . . . . .	147
7.4	Future Work . . . . .	148
<b>References</b>		<b>150</b>
<b>Appendices</b>		<b>159</b>
<b>Chapter A:</b>	<b>System Specifications</b>	<b>159</b>



A.1	Hardware Specifications . . . . .	159
A.2	Software Specifications . . . . .	159
<b>Chapter B: List of Security Technologies</b>		<b>162</b>
B.1	The Simple Authentication and Security Layer (SASL) . . . . .	162
B.2	Generic Security Service Application Program Interface (GSS-API) . . . . .	163
B.3	Extensible Authentication Protocol (EAP) . . . . .	164
B.4	Lightweight Directory Access Protocol (LDAP) . . . . .	164
B.5	ActiveDirectory . . . . .	165
<b>Chapter C: Life-cycles of the New Simplified FSSO System</b>		<b>168</b>
C.1	Life-cycle 1: end-user accesses non-web base application . . . . .	168
C.2	Life-cycle 2: end-user accesses web-based application . . . . .	170
C.3	Life-cycles 3: an end-user in domain Beta access domain Alpha . . . . .	170

# List of Figures

1.1	Desktop, web-based services/applications and non-web based services/applications have their own authentication systems . . . . .	2
1.2	KSS incorporates the authentication systems of desktop systems and non-web based services/applications . . . . .	3
1.3	SASL-SAML and SAML-AAI/Kerberos incorporates the authentication systems of both web-based and non-web based services/applications . . . .	3
1.4	The new simplified FSSO system incorporates the authentication systems of desktop systems, web-based services/applications and non-web based services/applications . . . . .	4
1.5	Silo model overview . . . . .	6
1.6	Centralised identity management overview . . . . .	6
1.7	Federated single sign-on overview . . . . .	7
1.8	Existing FSSO systems overview . . . . .	11
2.1	Authentication in Identity Management . . . . .	18
2.2	The silo model . . . . .	24
2.3	Silo Model Authentication Process . . . . .	25
2.4	An end-user must store credential in every domain . . . . .	26
2.5	Centralised Identity Management . . . . .	27
2.6	Federated single sign-on overview . . . . .	29
2.7	Federated Single Sign-on Process . . . . .	31
3.1	Federated Single Sign-on Model in Authentication Model . . . . .	33
3.2	SAML 2.0 incorporates three existing federation technologies and standards: SAML 1.1, ID-FF 1.2 and Shibboleth (Jason & Goode, 2012) . . . .	36

3.3	Project Moonshot Model (Howlett, 2011) . . . . .	38
3.4	Kerberos Secure Sharing (Gridwise Tech, 2010) . . . . .	40
3.5	SASL-SAML Model (Wierenga & Lear, 2012) . . . . .	42
3.6	Interoperating SAML-AAI and Kerberos (Papez, 2009) . . . . .	44
4.1	High level comparison of Kerberos simulation and real world system . . . .	49
4.2	High level comparison of SAML based FSSO system simulation and real world system . . . . .	50
4.3	High level comparison of Ubuntu desktop simulation and real world system	51
4.4	High level comparison of SSH protocol simulation and real world system .	52
4.5	Second and Third Federation Policies Written in XML . . . . .	55
4.6	Fourth and Fifth Federation Policies Written in XML . . . . .	55
4.7	Sample End-user Database Written in XML . . . . .	59
4.8	Database Tree Relation in Colour . . . . .	61
4.9	Simulate Kerberos Ticket Granting Ticket . . . . .	62
4.10	Sample Service Database Written in XML . . . . .	63
4.11	Sample Source String . . . . .	64
4.12	Sample Shared Key . . . . .	64
4.13	Sample Service Ticket . . . . .	65
4.14	Linux Shell and SSH simulation . . . . .	66
4.15	Sample SSH server simulation Entry in the Service Database . . . . .	70
4.16	Sample Shared Key between SSH Server and Kerberos Server Simulation .	70
4.17	Sample Message that includes a Username and Service Ticket . . . . .	71
4.18	Sample $AIIP_i$ Entry in the Service Database . . . . .	76
4.19	Sample Shared Key of $AIIP_i$ . . . . .	76
4.20	Sample Identity Assertion . . . . .	77
4.21	Side by side comparison of AIIP and KSS . . . . .	80
4.22	Sample Web-based Authentication System Simulation Entry . . . . .	82
4.23	Sample Shared Key between Web-based Authentication System Simulation and Kerberos Server Simulation . . . . .	82

4.24 Kerberos Ticket Granting Ticket Portal . . . . .	83
4.25 A New Simplified Federated Single Sign-on Life-cycle 1 . . . . .	85
4.26 A New Simplified Federated Single Sign-on Life-cycle 2 . . . . .	87
4.27 Life-cycle 3: an end-user in domain Beta access diverse service/applications domain Alpha . . . . .	89
5.1 Start Kerberos Server Simulation . . . . .	94
5.2 Kerberos single sign-on system thread flowchart . . . . .	95
5.3 Searching for a Matching Credentials . . . . .	97
5.4 Creating Ticket Granting Tickets . . . . .	97
5.5 Verifying Services Data . . . . .	98
5.6 Creating Shared Key . . . . .	98
5.7 Verifying Ticket Granting Ticket . . . . .	99
5.8 Creating Service Ticket . . . . .	99
5.9 Flowchart of Ubuntu desktop simulation . . . . .	101
5.10 Desktop Authentication and Requesting Ticket Granting Ticket . . . . .	101
5.11 Saving Ticket Granting Ticket . . . . .	102
5.12 Flowchart of SSH client simulation . . . . .	103
5.13 Requesting Service Ticket . . . . .	104
5.14 Sending Service Ticket . . . . .	104
5.15 Secure Shell Client Simulation . . . . .	104
5.16 Flowchart of a SSH server thread simulation . . . . .	105
5.17 SSH Server Simulation Requesting Shared Key . . . . .	106
5.18 SSH Server Simulation Opens the Shared Key . . . . .	106
5.19 Create Dedicated Directories . . . . .	107
5.20 Creating Cookie Value . . . . .	109
5.21 Extracting Cookie Value . . . . .	109
5.22 Web-based Service/Application Simulation in Domain foreign.virtual.vm .	110
5.23 Flowchart of identity provider end of AIIP . . . . .	112
5.24 Requesting Shared Key . . . . .	113

5.25	Accepting Service Tickets . . . . .	113
5.26	Creating/Sending Identity Assertion to the Service Provider End of AIIP .	114
5.27	Flowchart of a thread of service provider end of AIIP . . . . .	115
5.28	Extracting End-user's Identity Information from Identity Assertion Message	115
5.29	Mapping End-user's Credentials . . . . .	116
5.30	Requesting Ticket Granting Ticket on behalf of End-user . . . . .	116
5.31	Requesting Service Ticket Ticket on behalf of End-user . . . . .	117
5.32	Accessing SSH Servers in Separate Domains . . . . .	118
5.33	Flowchart of IDAWA . . . . .	120
5.34	IDAWA Requesting Shared Key . . . . .	121
5.35	Uploading Kerberos Ticket Granting Ticket onto IDAWA . . . . .	121
5.36	Web-based Authentication System Requests Service Ticket . . . . .	121
5.37	Web-based Authentication System Verifies the Service Ticket . . . . .	122
5.38	End-user login the Local Ubuntu desktop simulation in home.virtual.vm (i.e. domain Alpha) . . . . .	122
5.39	Start the remote SSH server simulation in domain foreign.virtual.vm (i.e. domain Beta) . . . . .	123
5.40	End-user attempts to access the remote SSH server in foreign.virtual.vm (i.e. domain Beta) . . . . .	123
5.41	$AIIP_i$ verifies the end-user's identity and sends an identity assertion to $AIIP_s$ . . . . .	124
5.42	$AIIP_s$ accepts the end-user's identity assertion and requests ticket granting ticket and service ticket on behalf of the end-user in domain foreign.virtual.vm	124
5.43	The remote Ubuntu desktop simulation accepts the request from the end-user	124
5.44	The end-user is granted access the to remote SSH server in domain for- eign.virtual.vm . . . . .	125
5.45	Use Kerberos Authentication Method in domain home.virtual.vm (i.e. do- main Alpha) . . . . .	125
5.46	Successful access to web-based service/application in domain foreign.virtual.vm	126

7.1	Existing FSSO systems overview . . . . .	145
A.1	Dell Optiplex GX260 configuration. . . . .	161
B.1	SASL is conceptually a framework that provides an abstraction layer between protocols and mechanisms(Melnikov & Zeilenga, 2006) . . . . .	163

# List of Tables

2.1	Differences Between Silo Model, Centralised Identity Management Model and Federated Single Sign-on Model . . . . .	29
3.1	Authority-centric and User-centric Federated Single Sign-on . . . . .	34
4.1	Network Domain Names . . . . .	52
4.2	Domain Variables . . . . .	58
6.1	Comparing approaches using the first criterion . . . . .	132
6.2	Comparing approaches using the second criterion . . . . .	134
6.3	Comparing approaches using the third criterion . . . . .	135
6.4	Comparing approaches with the fourth criterion . . . . .	138
6.5	Comparing approaches with the fifth criterion . . . . .	139
6.6	Comparison of the New Simplified FSSO System with Existing Approaches	142
A.1	IBM eServer Hardware Overview. . . . .	160
A.2	Sun Ultra 40M2 Workstation Configuration. . . . .	160
A.3	HP Probook 6550b Software Configuration. . . . .	161

# List of Acronyms

AES – Advanced Encryption Standard

AIIP – Authentication Infrastructure Integration Program

AJAX – Asynchronous JavaScript and XML

CIM – Centralised Identity Management

DOM – Document Object Model

EAP – Extensible Authentication Protocol

FSSO – Federated Single Sign-on

GSS-API – Generic Security Services Application Programming Interface

HTML – HyperText Markup Language

ID-FF – Liberty Identity Federation Framework

IDAWA – Integration of Desktop Authentication and Web-based Authentication

IdP – Identity Provider

IP – Internet Protocol

KDC – Key Distribution Centre

KSS – Kerberos Secure Sharing

OSI – Open Systems Interconnection

PHP – PHP: Hypertext Preprocessor

S4U2Proxy – Services for User to Proxy

S4USelf – Services For User to Self

SAML – Security Assertion Markup Language

SASL – Simple Authentication and Security Layer

SP – Service provider



SSH – Secure Shell Protocol

ST – Service Ticket

STS – Security Token Service

TCP – Transmission Control Protocol

TLS – Transport Layer Security

TGS – Ticket Granting Service

TGT – Ticket Granting Ticket

UCIM – User-Centric Identity Management

UFed SSO – User-Centric Federated Single Sign-on System

WB – Web Browser

WBAS – Web-based Authentication System

WSA – Web-based Services/Applications

XML – eXtensible Markup Language

# Chapter 1

## Introduction

It is currently not possible for end-users to access desktop systems, web-based services/applications and non-web based services/applications in multiple domains using one authentication session. This means end-users need to go through multiple authentication processes (i.e. entering their credentials at the authentication prompts), and they need to either remember an excessive number of passwords or recycle small numbers of passwords. This may cause end-users to enter their credentials at any authentication requests without identifying the requesting party, which leaves them vulnerable to phishing attacks.

Federated single sign-on (FSSO) is an authentication model that allows end-users to access computer services/applications from different network domains but the end-users are only required to enter their credentials (i.e. to be authenticated) once (Suriadi et al., 2009). This model requires the establishment of a federation (i.e. a circle of trust) between network domains (Wason et al., 2003); network domains allow end-users that have been authenticated by other network domains (in the same federation) to access their computer resources without requiring re-authentications.

Although FSSO allows the end-users to access computer resources in different network domains using one authentication session, it does not specify which authentication systems are used in the network domains. Desktop systems, web-based services/applications and non-web based services/applications use their own authentication

systems, and end-users need to be authenticated by these authentication systems before accessing the computer services/applications. For example, end-users need to enter their credentials (e.g. username and password) at the desktop login prompt (i.e. desktop authentication) in order to access the desktop system, then the same end-users need to enter their credentials again at the web-based login prompt in order to access web-based services/applications. A FSSO system will not deliver single sign-on unless it incorporates the authentication systems of desktop systems, web-based services/applications and non-web based services/applications (Figure 1.1).

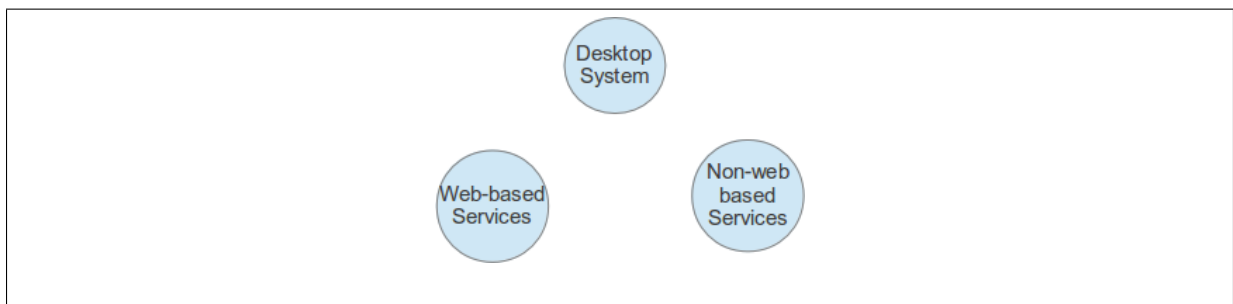


Figure 1.1: Desktop, web-based services/applications and non-web based services/applications have their own authentication systems

FSSO systems such as Project Moonshot (Howlett, 2011), Kerberos Secure Sharing (KSS) (Gridwise Tech, 2010), SASL-SAML (Wierenga & Lear, 2012) (i.e. the combination Simple Authentication and Security Layer with SAML) and SAML-AAI/Kerberos (Papez, 2009) (i.e. the combination of SAML's authentication authorisation infrastructure with Kerberos single sign-on infrastructure) partially incorporate the authentication systems. Project Moonshot only incorporates the authentication systems of non-web based services/applications. KSS incorporates the authentication systems of desktop systems and non-web based services/applications (Figure 1.2). SASL-SAML and SAML-AAI/Kerberos incorporates the authentication systems of both web-based and non-web based services/applications (Figure 1.3). These FSSO systems do not incorporate the authentication systems of desktop systems, web-based services/applications and non-web based services; therefore, a single end-user currently needs to go through multiple authentication processes to access desktop systems, web-based services/applications and non-web based services/applications.

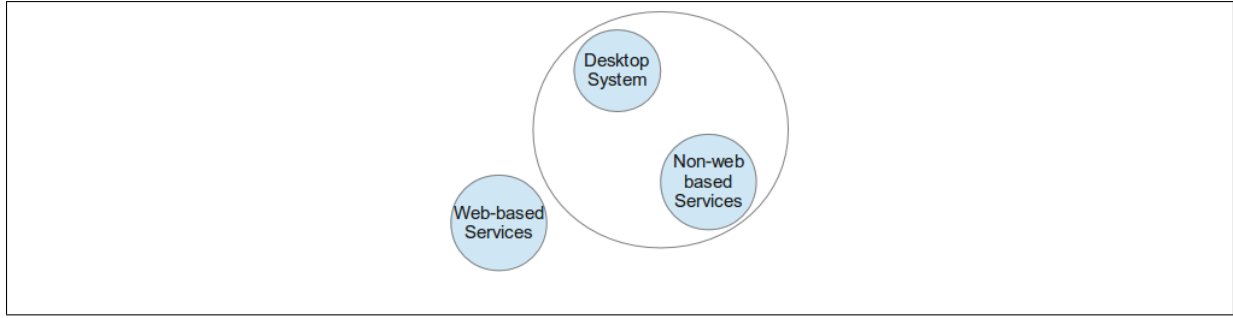


Figure 1.2: KSS incorporates the authentication systems of desktop systems and non-web based services/applications

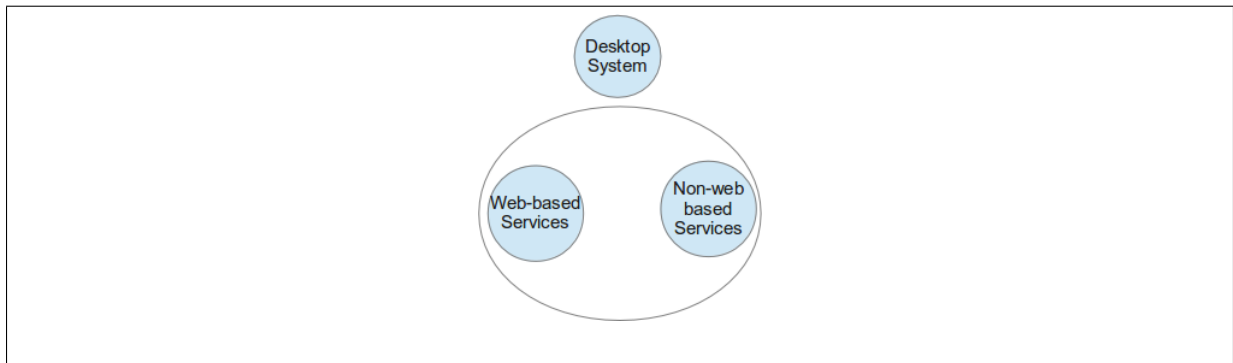


Figure 1.3: SASL-SAML and SAML-AAI/Kerberos incorporates the authentication systems of both web-based and non-web based services/applications

This research attempts to address this challenge by developing a new simplified federated single sign-on system that incorporates the authentication systems of desktop systems, web-based services/applications and non-web based services/applications (Figure 1.4). The new system allows end-users to access desktop systems web-based services/applications and non-web based services/applications (in different domains) using one authentication process, and it includes the developments of two major components: an “Authentication Infrastructure Integration Program (AIIP)” and an “Integration of Desktop Authentication and Web-based Authentication (IDAWA).”

Authentication Infrastructure Integration Program (AIIP) draws from KSS because KSS is the only system that incorporates the authentication systems of desktop systems and non-web based services/applications. Although both KSS and AIIP use Kerberos single sign-on systems (i.e. authentication systems) for providing FSSO, AIIP does not directly connect the Kerberos single sign-on systems in different network domains. The AIIPs (in different domains) seamlessly bridge the connections between end-users’ desktop systems in one domain and Kerberos single sign-on systems in the other domains.

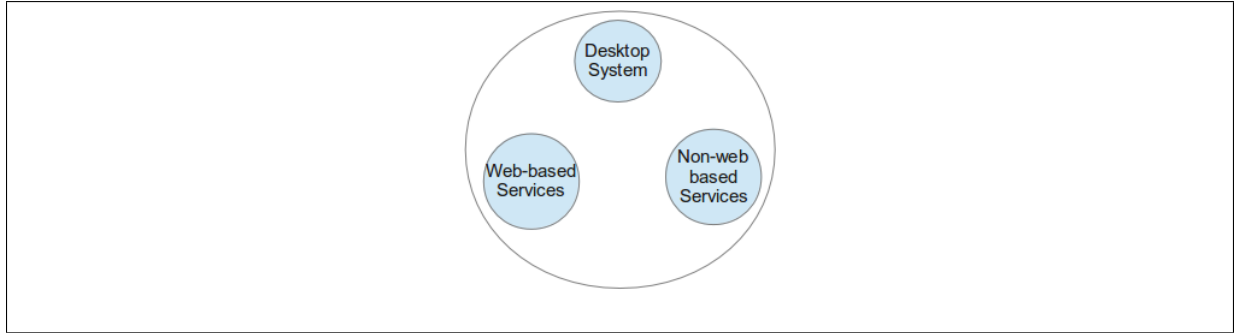


Figure 1.4: The new simplified FSSO system incorporates the authentication systems of desktop systems, web-based services/applications and non-web based services/applications

They seamlessly acquire Kerberos tickets in different domains for the end-users that have gained access to desktop systems, so that end-users can proceed to access non-web based services/applications (in different domains) using these Kerberos tickets (instead of requiring re-authentication). Unlike KSS, the AIIP does not expose the Kerberos single sign-on systems to the outside world (i.e. the Internet).

The IDAWA is an extension to the authentication systems of web-based services/applications. It allows end-users to use Kerberos tickets (which were acquired during desktop authentication) as means for web-based authentications; end-users, who gained access to desktop systems, don't need to be re-authenticated in order to access web-based services/applications.

The new simplified FSSO system combines AIIP and IDAWA to provide FSSO. It allows end-users to access desktop systems, web-based services/applications and non-web based services/applications using one authentication process. End-users who have gained access to desktop systems can proceed to access both web-based and non-web based services/applications (in different domains) without requiring re-authentication.

## 1.1 Research Background

### 1.1.1 Authentication Models

An authentication model consists of two elements: an identity provider and a service provider (Jøsang et al., 2007; Vacca et al., 2010). An identity provider manages end-users' identities, provides authentication and identity assertions for the authenticated end-users. A service provider, which relies on the identity providers for providing authentications, provides computer services to authenticated end-users.

There are three authentication models: the silo model, centralised identity management (CIM) and federated single sign-on (FSSO) (Wason et al., 2003; Jøsang & Pope, 2005; Jøsang et al., 2007; Mather et al., 2009; Vacca et al., 2010). Each model manages the identity provider and service provider differently (as described below). The authentication processes of each model are also different.

#### Silo Model

The silo model keeps the identity provider and service provider in the same space (Jøsang & Pope, 2005; Jøsang et al., 2007; Vacca et al., 2010). Each network domain has an identity provider and a service provider (Figure 1.5). A domain's identity provider authenticates end-users for that domain's service provider. A service provider may provide desktop systems, web-based services/applications or non-web based services/applications. The silo model does not provide any cross domain authentication. End-users need to be authenticated multiple times to access computer services/applications in multiple domains, and they need to remember multiple sets of credentials.

#### Centralised Identity Management Model

Centralised Identity Management (CIM) uses centralised identity provider approach to address the cross domain authentication challenge. It removes the identity providers in all of the domains, and instead it uses a single centralised identity provider to manage all

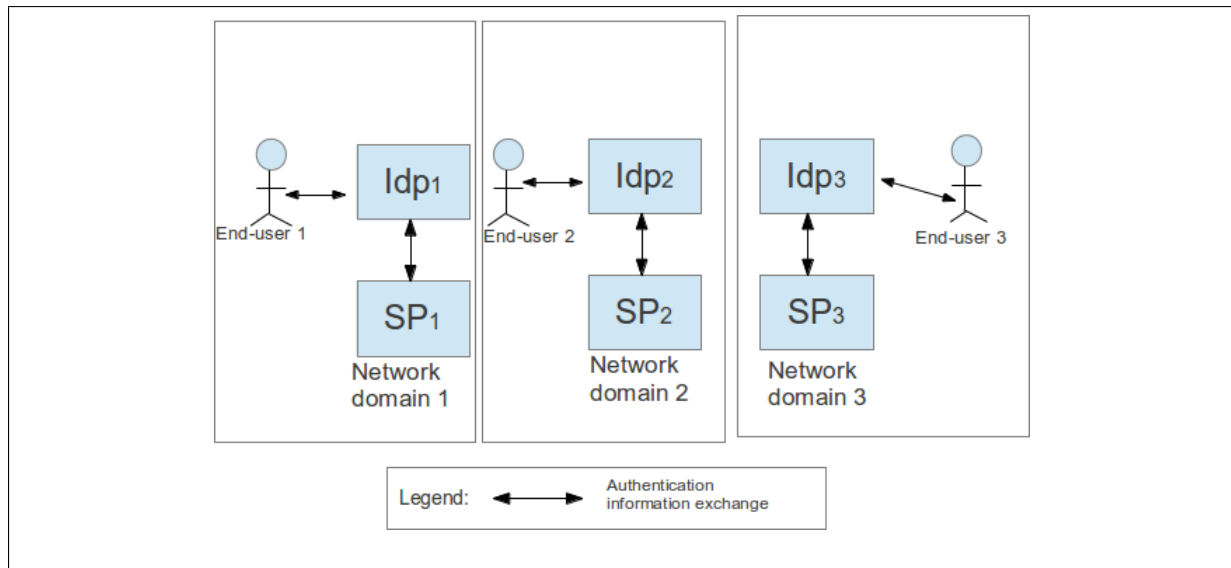


Figure 1.5: Silo model overview

the identities for every domain (Figure 1.6) (Jøsang et al., 2007). Although this approach allows end-users to access computer services in multiple domains by authenticating to the single identity provider using one set of credentials, this model has not been adopted by network domains due to privacy problems (i.e. network domains do not trust a single identity provider to manage all the identities in the world) (Chappell, 2006). Network domains continue to use the silo model for managing authentications, and there is a need for an authentication model that doesn't cause privacy issues.

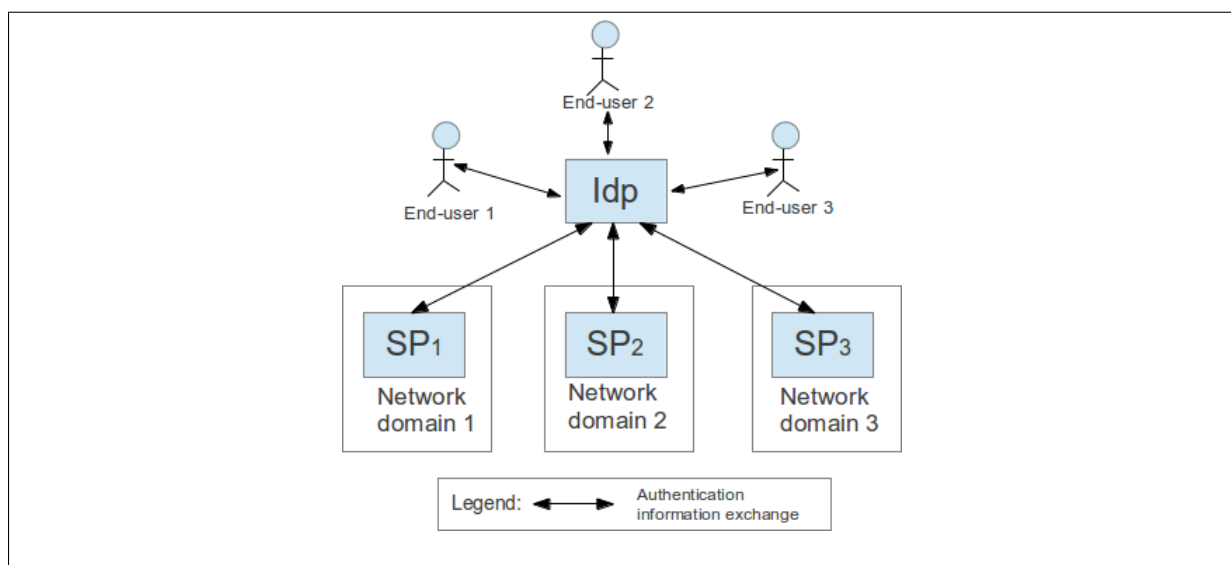


Figure 1.6: Centralised identity management overview

## Federated Single Sign-on Model

Federated single sign-on (FSSO) decentralises identity providers into individual network domains (i.e. each domain manages its own identity provider) similar to the silo model, and it builds a federation (i.e. a circle of trust) between domains (Figure 1.7). These trust relationships are created using out-of-band business and legal agreements between the federation participants (Buecker et al., 2008). These agreements must be in place before a federation can begin to operate (e.g. a security policy specified that network domains can provide computer services/applications to end-users without requesting authentication if the end-users belong to network domains in the same federation). End-users can access computer services/applications in network domains by claiming they are from one of the domains within the circle of trust. For example, an identity provider (in one domain within the federation) vouches for end-users when they require accesses to service providers (in the other domains), and the service providers will accept the accesses from the end-user due to the established federation.

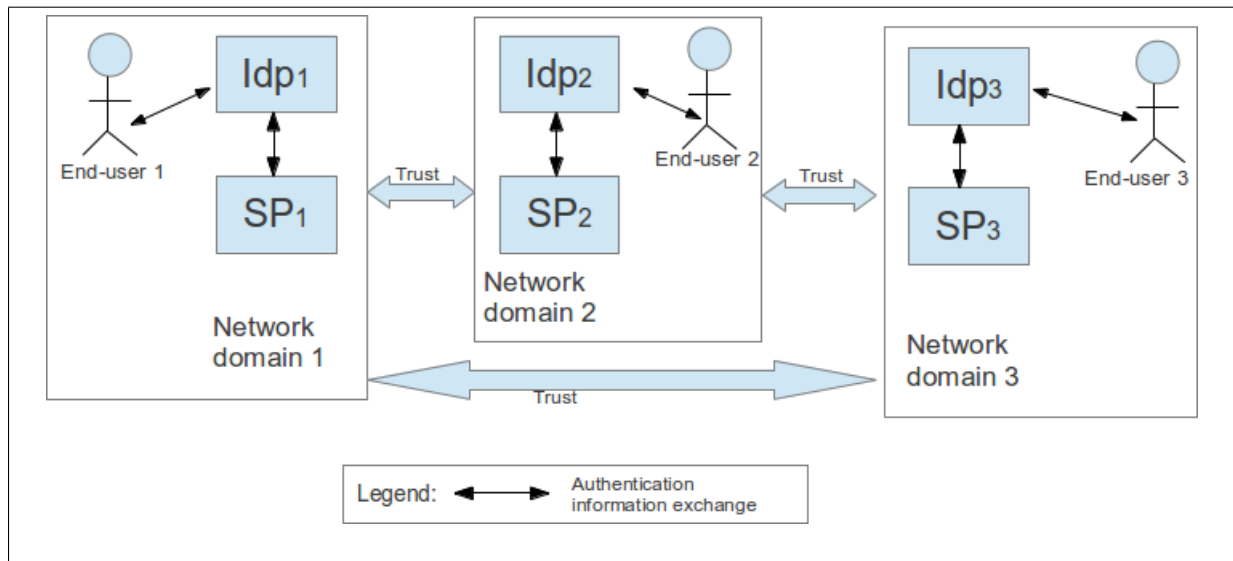


Figure 1.7: Federated single sign-on overview

FSSO has the potential to become the predominate authentication model that is used by network domains for managing cross domain single sign-on. By separating the identities into individual network domains, network domains adopt FSSO by building federation on top of their existing authentication models (i.e. the silo model). In addition,



FSSO is an improvement over CIM since it doesn't cause privacy issues.

The next section gives a in-depth review of the current FSSO systems: Project Moonshot (Howlett, 2011), Kerberos Secure Sharing (KSS) (Gridwise Tech, 2010), SASL-SAML (Wierenga & Lear, 2012) (i.e. the combination Simple Authentication and Security Layer with SAML) and SAML-AAI/Kerberos (Papez, 2009) (i.e. the combination of SAML's authentication authorisation infrastructure with Kerberos single sign-on infrastructure).

### 1.1.2 Federated Single Sign-on

Federated single sign-on (FSSO) was initially applied in web space. FSSO standards such as Security Assertion Markup Language (SAML) (OASIS, 2010; Madsen et al., 2005), Liberty Identity Federation Framework (ID-FF) (Wason et al., 2003) and WS-Federated (Microsoft, 2012b) are supported by web-based services/applications; end-users can access web-based computer services/applications (in different domains within a federation) after web-based authentication. These standards are not supported by non-web based services/applications; therefore, end-users need to be authenticated separately (i.e. multiple times) by different authentication systems when accessing desktop systems and non-web based services/applications. For example, SAML based FSSO systems have been used by major academic networks (e.g. HEAnet), universities/institutions (e.g. Dublin Institute of Technology, Trinity College and University College Dublin) and online libraries (e.g. IEEE Xplore Digital Library and ACM Digital Library). Students that were authenticated by Dublin Institute of Technology's web-based authentication system can access online libraries such as IEEE Xplore Digital Library and ACM Digital Library (using web browsers) without requiring re-authentication (HEAnet, 2008); however, the same students need to be authenticated again by remote desktop authentication systems when requesting access to remote desktop systems in Trinity college.

This research reviewed four projects that attempt to provide FSSO beyond web space: Project Moonshot, Kerberos Secure Sharing (KSS), SASL-SAML and SAML-

AAI/Kerberos. Project Moonshot aims to provide FSSO for accessing non-web based services/applications, KSS aims to provides FSSO that incorporates the authentication systems of desktop systems and non-web based services/applications, and SASL-SAML and SAML-AAI/Kerberos aim to provide FSSO that incorporates the authentication systems of both web-based and non-web based services/applications.

### **Project Moonshot**

Project Moonshot partially addresses the single sign-on challenge by supporting only non-web based services/applications. Desktop authentication, web-based authentication and non-web based authentication are separate authentication processes in this system. End-users need to be authenticated multiple times to access desktop systems, web-based services/applications and non-web based services/applications. For example, an end-user needs to be authenticated three times in order to access a local desktop system in DIT, IEEE Xplore Digital Library and a remote Linux desktop in Trinity college.

### **Kerberos Secure Sharing**

Keberos Secure Sharing (KSS) incorporates the authentication systems of desktop systems and non-web based services/applications, so that an end-user can access a desktop system and non-web based services using one authentication process. It directly connects network domains' Kerberos single sign-on systems, so that Kerberos single sign-on systems can support non-web based FSSO in addition to support local network single sign-on. The end-users, who have gained access to desktop systems (i.e. who have been authenticated by Kerberos single sign-on systems and acquired Kerberos tickets), can access non-web based services/applications using Kerberos tickets.

KSS also partially addresses the single sign-on challenge. This system does not incorporate web-based authentication systems; therefore, the end-users need to be authenticated again (by web-based authentication systems) in order to access web-based services/applications. For example, KSS allows students (who gained accesses to desktop

systems in DIT) to remotely access Linux desktops in Trinity college and UCD without requiring re-authentication. However, the same students need to be authenticated again (by the web-based FSSO system) when accessing IEEE Xplore Digital Library.

### **SASL-SAML**

SASL-SAML uses SASL to facilitate communications between non-web based services/applications and SAML based FSSO system. This allows SAML based FSSO systems (i.e. web-based FSSO systems) to manage single sign-on for both web-based and non-web based services/applications, and end-users can access both web-based and non-web based services after one authentication process.

SASL-SAML does not consider desktop authentication as part of its single sign-on process (i.e. it does not incorporate the desktop authentication systems into its FSSO system). This results in end-users needing to be authenticated multiple times to access desktop systems and (both web-based and non-web based) computer services/applications. For example, the students need to be authenticated in order to access local desktop systems in DIT. The same students need to be authenticated again in order to access IEEE Xplore Digital Library and remote Linux desktops in Trinity college.

### **SAML-AAI/Kerberos**

SAML-AAI/Kerberos also uses SAML based FSSO system for managing single sign-on. It requires web-based interfaces (accessible by web-browsers) to be implemented for non-web based services/applications. It develops a web-based software that integrates web-based FSSO sessions (in one domain) with non-web based single sign-on sessions (in the other domains). The end-users (who obtained web-based FSSO sessions in one domain) can access both web-based and non-web based computer services/applications (in the other domains) without requiring re-authentication.

SAML-AAI/Kerberos also does not consider desktop authentication as part of its single sign-on process. End-users need to be authenticated once by desktop authentication

tions systems to access desktop systems, then they need to be authenticated by web-based authentication systems to access web-based services/applications. For example, the students need to be authenticated in order to access local desktop systems in DIT. The same students need to be authenticated again in order to access IEEE Xplore Digital Library and the web-based interfaces of remote Linux desktops in Trinity college.

## Research Gap

The reviews of Project Moonshot, Kerberos secure sharing, SASL-SAML and SAML-AAI/Kerberos showed that each FSSO system partially incorporates the authentication systems of desktop systems, web-based services/applications and non-web based services/applications (Figure 1.8). There is a need for a FSSO system that incorporates the authentication systems of all three types of services.

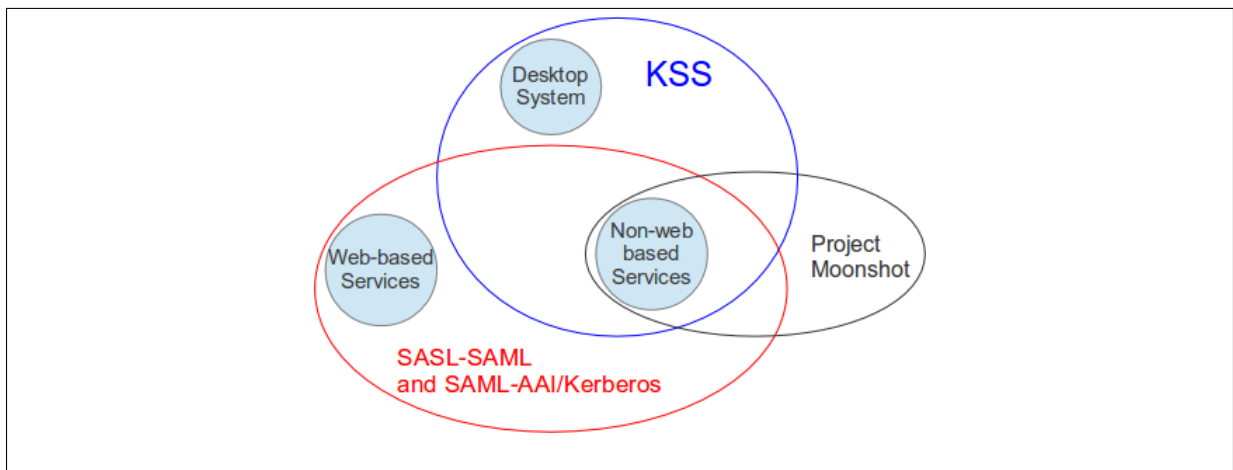


Figure 1.8: Existing FSSO systems overview

## 1.2 Research Aim

This research aims to develop a new simplified FSSO system in order to address the research gap (i.e. the lacking of a FSSO system that allows end-users to access desktop systems, web-based services/applications and non-web based services/applications using one authentication process). The new simplified FSSO system attempts to incorporate the authentication systems of desktop systems, web-based services/applications and non-

web based services/applications, so that end-users can access the above services using one authentication process.

Although trust is an important part of the federated single sign-on model, it is not the focus of this research. The new simplified FSSO system assumes that trust has been established between network domains. The management of trust (e.g. establishment, measurement, maintenance and revocation) is out of the scope of this research.

This research draws extensively from the approaches of Kerberos Secure Sharing (KSS) and SAML-AAI/Kerberos. KSS uses Kerberos to provide single sign-on for end-users to access desktop systems and non-web based services/applications, and SAML-AAI/Kerberos can acquire Kerberos tickets (from Kerberos single sign-on systems) on behalf of the end-users. The simplified FSSO system incorporates these elements into its design.

The new simplified FSSO system assumes that the network domains employ their own Kerberos single sign-on systems (i.e. non-web based authentication infrastructures) similar to KSS and SAML-AAI/Kerberos. It also assumes that a federation (i.e. a circle of trust) has been established between the network domains, and the network domains use SAML based FSSO system (i.e. web-based FSSO system).

The proof-of-concept system is evaluated using simulations of real world environments. The evaluation shows that the proof-of-concept system allows end-users to access desktop systems, web-based services/applications and non-web based services/applications using one authentication process. In addition, the proof-of-concept system does not require modifications to the existing Kerberos single sign-on systems, and it doesn't require additional modifications to network domains' firewall rules.

### 1.3 Research Methodology

This research found that currently the end-users need to be authentication multiple times in order to access desktop systems, web-based services/applications and non-web based

services/applications. For example, an end-users need to be authenticated multiple times in order to access a desktop systems in DIT, IEEE Xplore Digital Library and remote Linux desktop system in Trinity college.

In order to address this challenge, this research reviews relevant literature on authentication models (the silo model, centralised identity management and federated single sign-on), and it found that federated single sign-on (FSSO) has the potential to deliver single sign-on across multiple network domains.

The review of FSSO systems (e.g. Project Moonshot, Kerberos secure sharing, SASL-SAML and SAML-AAI/Kerberos) shows that the existing FSSO systems do not support desktop systems, web-based services/applications and non-web based services/applications. Although Kerberos secure sharing (KSS) is the only known FSSO system that allows end-users to access desktop system and non-web based services/applications using one authentication session, it does not support web-based services/applications. In addition, it requires trust to be established between Kerberos single sign-on systems, which means the Kerberos single sign-on systems (in different network domains) need to be directly connected (i.e. exposed to the outside world). The review also shows that although the SAML-AAI/Kerberos doesn't provide single sign-on for end-users to access desktop systems and both web-based and non-web based services, it has the components to acquire Kerberos tickets on behalf of the end-users, and these components can be used to improve KSS.

The results of the literature review are incorporated into the design of the new simplified FSSO system. The new simplified FSSO system includes the designs of "Authentication Infrastructure Integration Program (AIIP)" and "Integration of Desktop Authentication and Web-based Authentication (IDAWA)." The design of AIIP draws extensively from KSS and SAML-AAI/Kerberos, and it acquires Kerberos tickets from Kerberos single sign-on systems in different network domains without setting up trust between these Kerberos single sign-on systems. The IDAWA is deigned to be an extension to web-based authentication systems, and it uses Kerberos tickets as means for authentication. The combination of AIIP and IDAWA allows end-users to access desktop systems, web-based

services/applications and non-web based services/applications using one authentication process.

This research uses Linux based Operating System (e.g. Ubuntu Server 10.04.4 LTS), Python programming language, eXtensible Markup Language (XML) and PHP programming language to implement the evaluation environment and the proof-of-concept system. It also uses virtual machine technology (e.g. Xen server) to divide hardware machines (e.g. Sun Ultra 40M2 Workstation) into multiple virtual machines in order to host the evaluation environment and the proof-of-concept system. All of the virtual machines are on the same local network in order for them to communicate with each other, and Bind9 is used to divide virtual machines into different domains.

## 1.4 Contributions

**Contribution 1.** The development of a new simplified FSSO system represents an important contribution to the research of federated single sign-on. This work is a novel attempt to provide single sign-on for end-users to access desktop systems, web-based services/applications and non-web based services/applications. The new simplified FSSO system includes two major components: The Authentication Infrastructure Integration Program (AIIP) and the Integration of Desktop Authentication and Web-based Authentication (IDAWA).

- The Authentication Infrastructure Integration Program (AIIP) acquires Kerberos tickets (for end-users who have been authenticated by a Kerberos single sign-on system in one domain) from Kerberos single sign-on systems in different network domains without establishing trust between these Kerberos single sign-on systems.
- The Integration of Desktop Authentication and Web-based Authentication (IDAWA) is an extension to the web-based authentication systems (i.e. the web portal), and it authenticates end-users by verifying end-users' Kerberos tickets.

**Contribution 2.** The development of three new criteria for evaluating FSSO systems. They are used to evaluate the system that is presented in this research. The first criterion determines which FSSO system can deliver true single sign-on (i.e. allowing end-users to access desktop systems, web-based services/applications and non-web based services/applications using one authentication process). The second criterion determines which FSSO system requires the least amount of modification to network domains' non-web based authentication infrastructure (i.e. requesting the least active support and commitment from network domains). The third criterion determines which FSSO system doesn't exposes the network domains' non-web based authentication infrastructures to the outside world (i.e. lower risk to be compromised by attackers from the Internet).

## 1.5 Thesis Structure

The remaining sections of this document are organised as follows:

Chapter 2, Basic Concepts, focuses on the authentication functionality of identity management. The chapter starts from the introduction of identity management and its core functionalities (e.g. authentication, authorisation and auditing), then it focuses on reviewing the existing authentication models (e.g. silo, centralised identity management and FSSO) and determines that FSSO has the potential to be the predominant authentication models for cross domains single sign-on.

Chapter 3, Federated Single Sign-on (FSSO) Systems, focuses on reviewing FSSO systems. This chapter reviews the existing approaches and presents the strengths and weaknesses of these approaches. The limitations with existing approaches are identified and the need for a new simplified FSSO system is discussed.

Chapter 4, The Design of a New Simplified FSSO System, describes the design of the new simplified FSSO system. This chapter presents the technical specifications of the simulations of Kerberos single sign-on, SAML based single sign-on, Secure shell protocol, Unix/Linux desktop and a web-service. It then presents the design of a new simplified



FSSO system (including AIIP and IDAWA).

Chapter 5, Proof of Concept Implementation, describes the implementation of the new simplified FSSO system. This chapter presents the set-ups of hardware and software, the flowchart of the systems, the lessons learned during the proof of concept development.

Chapter 6, Research Evaluation, presents the evaluation of the new simplified FSSO system. This chapter presents five evaluation criteria (3 new criteria and 2 existing criteria). These criteria are used to compare the new simplified FSSO system with Project Moonshot, Kerberos Secure Sharing, SASL-SAML, SAML-AAI/Kerberos. This chapter also discusses the results of the evaluations.

Chapter 7, Discussion and Conclusion, concludes the thesis. This chapter summarises the main findings, presents the critical review and suggests further work.

# Chapter 2

## Authentication Systems

### 2.1 Introduction

Computer security maintains the confidentiality, integrity and availability (CIA) of computer systems (Bishop, 2003). It prevents potential security violations such as unauthorised information access (i.e. an unauthorised end-user is able to read and take advantage of information stored in the computer), unauthorised information modification (i.e. an unauthorised end-user is able to make changes in stored information) and unauthorised denial of use (i.e. an unauthorised end-user is able to prevent an authorised end-user from referring to or modifying information even though the intruder may not be able to refer to or modify the information) (Anderson, 1973; Saltzer & Schroeder, 1975).

Identity management has an important role in maintaining computer security. It provides the confidentiality, integrity and availability of computer systems via managing the end-users' identity information (e.g. personally identifiable information and permission). It has three core functionalities: Authentication, Authorisation and Auditing (AAA). Authentication is the process of verifying claims about holding specific identities (Benantar, 2005; Pfleeger & Pfleeger, 2006; Vacca et al., 2010). Authorisation is the process of empowering someone to perform an operation or to have access to restricted resources (Salomon, 2005). Auditing is the process of collecting and analysing information

in order to ensure a proper level of security, as well as compliance with the policies of an organisation (Salomon, 2005).

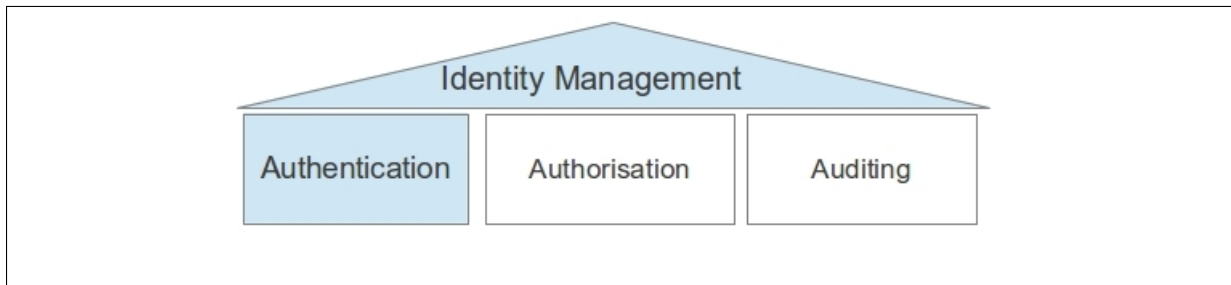


Figure 2.1: Authentication in Identity Management

This chapter focuses on the authentication aspect of identity management (Figure 2.1). Section 2.2 gives a brief overview of identity management and its core functionalities (e.g. Authentication, Authorisation and Auditing). Section 2.3 presents the background information on authentication systems (authentication mechanisms and authentication models). Section 2.4 summaries this chapter.

## 2.2 Identity Management

Digital identity represents an entity in a specific context (Miyata et al., 2006; El Maliki & Seigneur, 2007). It consists of traits, attributes, and preferences upon which one may receive personalised services (Project, 2003). It does not have to represent an entity’s real life identity (e.g. name, birthday) (Seigneur, 2005).

Identity management (in computing) can be defined as “the processes of creating, managing and decommissioning of end-users’ digital identities (Halperin & Backhouse, 2008; Lee et al., 2009; Thomas & Meinel, 2009; Balasubramaniam et al., 2009).” It faces challenges, for example, how to ensure the privacy of the digital identities (i.e. digital identity available only to the right individuals or services at the right time and place), how to avoid the abuse of digital identities, and how to make these provisions possible in a scalable, usable, and cost effective manner (Glasser & Vajihollahi, 2008; Bertino & Takahashi, 2010). It has an important role in maintaining computer security. It is a key research theme for the coming decades (Dunleavy et al., 2006; Halperin & Backhouse,

2008). It has three core functionalities: Authentication, Authorisation and Auditing (AAA) (Mont et al., 2003).

## **Authentication**

Benantar (2005); Pfleeger & Pfleeger (2006); Vacca et al. (2010) describe authentication as the process of verifying claims about holding specific identities. The process consists of obtaining the authentication information from an entity, analysing the data, and determine if it is associated with that entity (Pfleeger & Pfleeger, 2006). Failure to identify a legitimate end-user will deny the end-user's access to the system. Failure to identify an illegitimate end-user will threaten the security of the entire system.

## **Authorisation**

Authorisation is the process of empowering someone to perform an operation or to have access to restricted resources (Salomon, 2005). It manages the level of access of a digital identity. It's the next step after authentication in the identity management functionalities. Authorisation identifies the level of access of an end-user's digital identity after the end-user was authenticated.

## **Auditing**

Auditing is the process of collecting and analysing information in order to ensure a proper level of security, as well as compliance with the policies of an organisation (Salomon, 2005). It manages (create, modify, and delete) the security policies that are followed by authentication and authorisation. It also manages the provisioning and de-provisioning of the end-users' digital identities.

## 2.3 Authentication Systems

An authentication system contains five elements: the end-user, the distinguishing characteristic of the end-user, the proprietor who is responsible for the system, the authentication mechanism and the access control mechanism (Smith, 2001). The distinguishing characteristics were assigned to end-users by the proprietor. The authentication mechanism authenticates the end-users. The access control mechanism ensures the level of access that the end-users have.

There are four types of authentication mechanisms: Username and Password, Challenge and Response, Digital Certificates and Biometrics (Smith, 2001; Bishop, 2005; Pfleeger & Pfleeger, 2006). Section 2.3.1 presents the strengths and weaknesses of these authentication mechanisms. There are three authentication models: Silo Model, Centralised Identity Management Model and Federated Single Sign-on Model (Wason et al., 2003; Jøsang & Pope, 2005; Jøsang et al., 2007; Mather et al., 2009; Vacca et al., 2010). Section 2.3.2 reviews these authentication models.

### 2.3.1 Authentication Mechanisms

Authentication mechanisms (e.g. Username and Password, One-time Password, Digital Certificates and Biometrics) use any of the following qualities to confirm an end-user's identity (Pfleeger & Pfleeger, 2006; Bishop, 2005):

- Something you know: password, PIN.
- Something you have: one-time-password.
- Something you are: voice, face, fingerprint.
- Where you are: "Location" as another important quality.
- Any combination of the four previous qualities.

The following section discusses the qualities that are used by each authentication mechanisms. It also reviews the strengths and weaknesses of these authentication mechanisms.

## Username and Password

The Username and Password uses passwords as the sole authentication mechanism. Passwords are code words (Ahituv et al., 1987; Pfleeger & Pfleeger, 2006). They are known only to the end-user and the system. The passwords are either chosen by end-users or assigned by the system. The length and format of the password vary from one system to another.

The Username and Password mechanism is widely used; however, it suffers from password guessing, lost or forgotten password, redundant password, theft of password and revocation of password (Smith, 2001; Pfleeger & Pfleeger, 2006).

**Password Guessing.** The strength of the password relies on its complexity. A complex password is difficult to guess, but it's hard to remember or create. If a password is too simple, it's vulnerable to guessing or brute force attack.

**Lost or forgotten password.** Operators or system administrators often cannot determine the contents of the passwords. If end-users lose their passwords, new ones must be assigned.

**Redundant passwords.** Supplying multiple passwords for different accesses can be inconvenient and time consuming. It is also difficult for end-users to manage multiple passwords. It usually results the loss of passwords.

**Theft of password.** A password is vulnerable to password file theft, keystroke sniffing and network sniffing attacks.

**Revocation of password.** If a password was shared between multiple end-users to access a common file, revocation of one end-user's password needs notification.

Although Username and Password faces the above problems, it remains a widely used authentication mechanism. This is due to Username and Password being easy to manage. End-users or system administrators can easily change or reset the password when facing the listed problems.

## Challenge and Response

Challenge and Response uses the one-time password scheme to address the password theft problem. An one-time password is a type of password that changes every time it is used (Haller & Metz, 1996). The system assigns a static mathematical function to an end-user (instead of a static password). The system provides an argument to the function, and the end-user computes and returns the function value. Such systems are also called challenge-response systems because the system presents a challenge to the end-user and judges the authenticity of the end-user by the end-user's response (Pfleeger & Pfleeger, 2006).

This authentication mechanism requires end-users to carry a (physical or electronic) token to compute new passwords. This token-based authentication is more secure than Username and Password. It relies on unique physical objects for authentication. End-users can tell if their tokens have been stolen. It is also difficult for end-users to share their tokens with the others and still be able to access the computer system.

Although Challenge and Response is more secure than Username and Password, it remains less popular than Username and Password. This is because Challenge and Response is hard to manage than Username and Password. Instead of remembering passwords, end-users must keep the tokens with them at all time. If a token was lost, the owner of that token will not be able to access the computer systems. If an unauthorised end-user uses the token, he/she can attempt to impersonate the owner of that token. In addition, all of the trouble shooting (e.g. reset password) must be manually done by system administrators, which increases their work load.

## Digital Certificates

A digital certificate (or public key certificate) uses public key infrastructure for authentication. It is frequently quoted as the solution that is critical to authentication of the network. It binds a public-key value to a set of information that identifies the entity (such as a person, an organisation, an account, or a site) associated with use of the corresponding private key (this entity is known as the “subject” of the certificate) (Chokhani

& Ford, 1999). The advantage of a digital certificate is that only the public key is distributed through the network. We can take the security advantage of digital certificates as long as the private key of a digital certificate is protected safely (Pfleeger & Pfleeger, 2006).

A digital certificate is harder to create and manage than passwords. They need to be unique and contain the correct attributes of the end-users. End-users need to fill a list of questions to generate a single certificate. Digital certificates are stored as a digital file. End-users are in charge of maintaining them on their computers. End-users' private keys are at risk if their computers are not secure (Ellison & Schneier, 2000). End-users also are too willing to accept invalid certificates, which makes them vulnerable to spoofing (i.e. End-users assume they are accepting the correct certificates; however, they are accepting certificates generated by unauthorised parties) (Schneier, 1998).

## **Biometrics**

Biometrics deal with the identification of individuals based on their biological or behavioural characteristics (Jain et al., 1999). Biometric authentication devices can recognise the following biometrics: fingerprints, hand geometry (shape and size of fingers), retina and iris (i.e. parts of the eye), voice, handwriting, blood vessels in the finger, and face (Pfleeger & Pfleeger, 2006). Biometrics are less likely to be forged, lost, borrowed or stolen than passwords. It can not be forgotten and it is always available. It's considered as a more secure authentication solution than "Username and Password" authentication mechanism.

Although biometrics has potential to replace passwords, it also presents new problems: inaccurate reading, single point of failure and forgeries. In addition, a compromised biometric is a huge problem for end-users and system administrators since end-users can not replace their biometric data (e.g. eye ball, fingers) (Jain et al., 1999; Smith, 2001; Schuman, 2006; Pfleeger & Pfleeger, 2006). Biometrics will not completely replace password unless these problems are addressed.



### 2.3.2 Authentication Models

An authentication model consists of two parties, an identity provider (IdP) and a service provider (SP). The identity provider manages end-users' identities; it authenticates end-users and provides identity assertions for authenticated end-users. The service provider only provides computer services; it relies on the identity provider to provide authentication. There are three authentication models available: the silo model, centralised identity management model (CIM) and federated single sign-on model (FSSO).

#### The Silo Model

The silo model (or Isolated User Identity Model) is the most common authentication model currently deployed on the Internet (Jøsang & Pope, 2005; Jøsang et al., 2007; Vacca et al., 2010). It combines the identity provider (IdP) and service provider (SP) together in one network domain; however, it is not mandatory for them to reside in a single computer system (Figure 2.2).

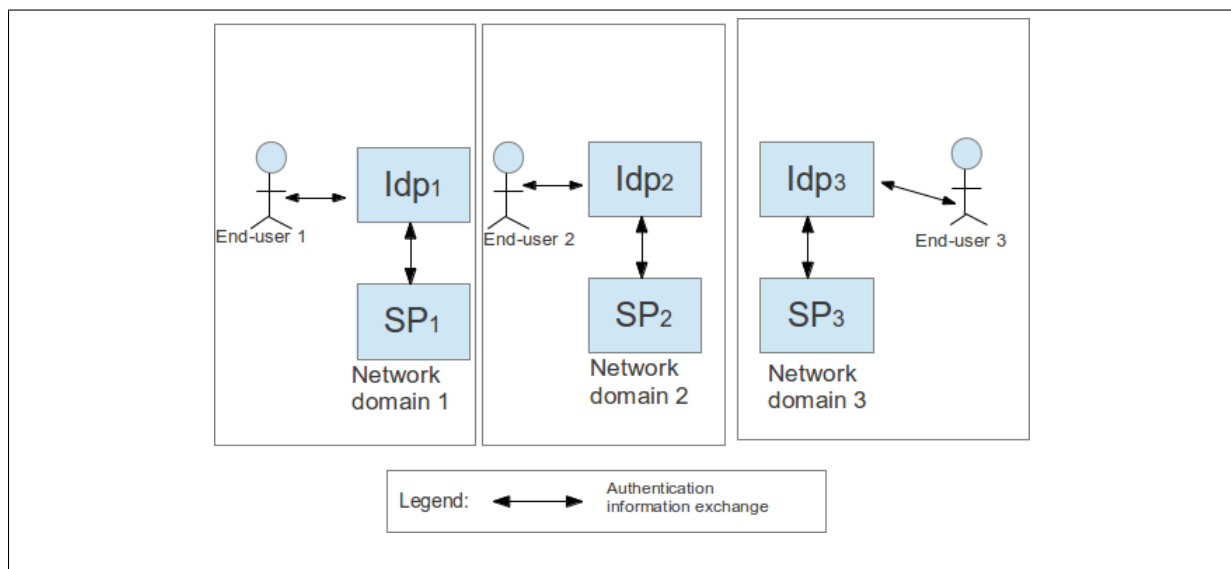


Figure 2.2: The silo model

The silo model is the easiest authentication model to implement and manage. The credentials of end-users are stored and managed in a single space. End-users can access computer services in a network domain with one set of credentials. For example, Microsoft Active Directory Domain Services (i.e. identity provider) manages end-users' credentials

for a network domain. It provides authentication for Microsoft services (e.g. Microsoft Windows XP Desktop Operating System). End-users need to be authenticated by Active Directory Domain Service before accessing Microsoft Windows XP Desktop Operating Systems (Figure 2.3). The process includes: (1) An end-user requests access to a Windows desktop, (2) Windows desktop requests the end-user to be authenticated, (3) the end-user sends its credential to the Windows desktop, (4) Windows desktop sends the credential to the Active Directory for verification, (5) Active Directory verifies the credential and replies the result back to the Windows desktop, and (6) Windows desktop accepts or rejects end-user's request based on the authentication result.

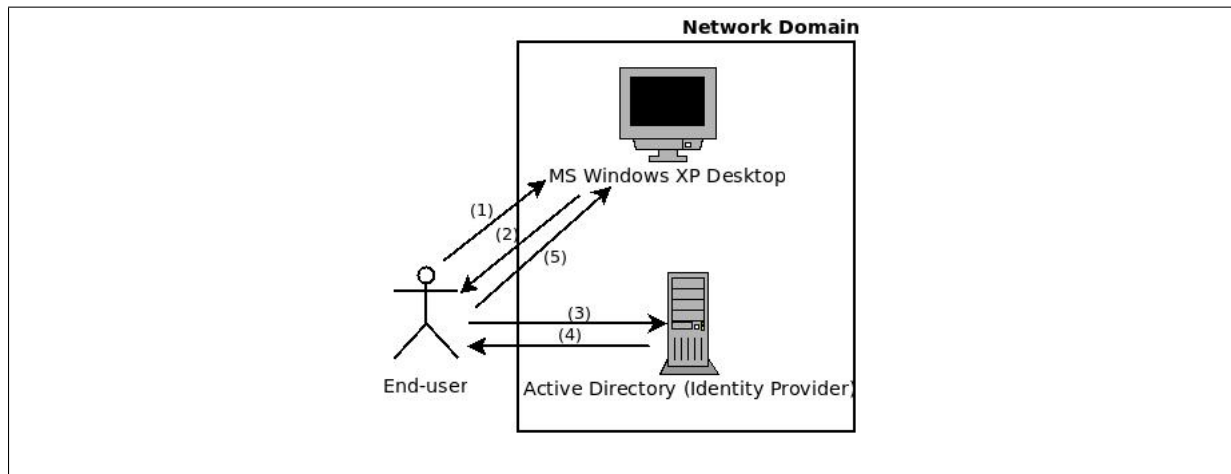


Figure 2.3: Silo Model Authentication Process

The silo model faces challenges when applying to a multi-domain environment (e.g. Internet). End-users need to create and remember a set of credentials for each network domain, which results in end-users are overloaded with identifiers and credentials (Figure 2.4). End-users are often required to memorise passwords, which unavoidably leads to users forgetting passwords. End-users usually use the same password combinations for every computer service, which means a compromised password effects every computer service. For example, end-users of Zappos.com use the same password for Zappos.com and other services (e.g. email service, computer desktop). The latest security breach of Zappos.com resulted in 24 million customer accounts being compromised (Rashid, 2012). Zappos.com reset these customers' passwords; however, customers' other services were still at risk. Customers needed to reset their password for all computer services to prevent

further security compromises.

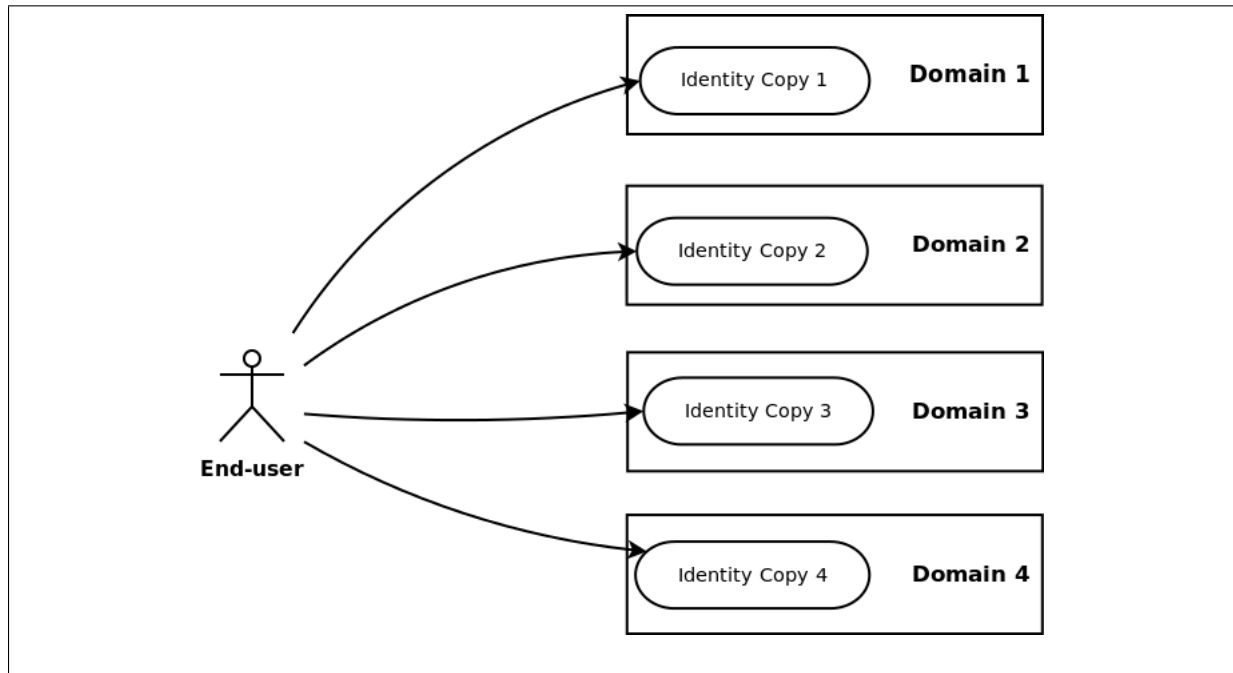


Figure 2.4: An end-user must store credential in every domain

### Centralised Identity Management (CIM)

Centralised Identity Management aims to address the problem of redundant credentials (in the multi-domain environment). It separates the identity providers from the service providers and merges them into a single central identity provider. The singular identity provider provides identity and authentication for every end-user in the world (Figure 2.5).

The primary example of CIM is Microsoft's Passport. It aims to provide a single and convenient method for identifying end-users across different websites (Jøsang et al., 2007). Microsoft will act as the central identity provider and hold end-users' personal information (e.g. credit card numbers) and make it available to all service providers (e.g. Google email, Amazon web store, Apple web store) whenever needed. The processes are: (1) an end-user requests email service from Google.com, (2) Google.com requests the end-user to be authenticated, (3) the end-user sends its credential to Google.com, (4) Google.com sends the end-user's credential to the identity provider (i.e. Microsoft.com), (5) Microsoft.com verifies the credential and replies the result back to Google.com, (6) Google.com accepts or rejects the end-user's request based on the authentication result.

End-users can have access to all service providers using the same set of identifiers and credentials (Bhargav-Spantzel et al., 2007).

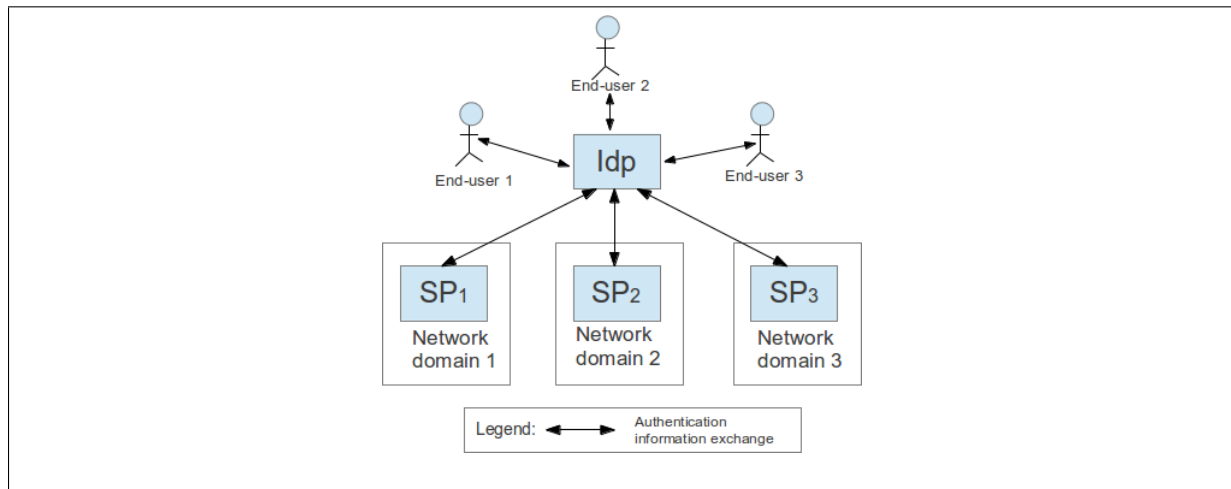


Figure 2.5: Centralised Identity Management

The main weakness of centralised identity management is privacy. A singular identity provider means all the credentials in the world are stored in a single space. It is not ideal for enterprises to store all of their credentials in one place. For example, Microsoft will be in charge of all the end-users' personal information in the world. It will have significant power to abuse these information, which is a violation of end-users' privacy. In the end, Microsoft acknowledged that "... no single organisation, not even one as big as Microsoft, could act as the sole identity provider for everything on the Internet (Chappell, 2006)."

Another weakness of CIM is single point of failure. The centralised identity management system contains credentials of every end-user in the world. It is a huge target for malicious attacks. Every service provider in the world will be affected when the identity provider experienced technical failures or been compromised.

### Centralised Single Sign-on

There may be many computer service providers in a single network domain; each service provider supports a unique computing task. The identity provider must identify legitimate end-users from all the service requests. End-users need to be authenticated for each computer service, which increases the number of authentication processes for a single end-user.

Centralised Identity Management (CIM) also proposed a centralised single sign-on model (i.e. one identity provider manages single sign-on sessions for multiple service provider). It addresses the redundant authentication processes by building a mutual trust between computer services providers and the central identity provider. It allows computer service providers to trust the authentication processes of the other computer service providers, and these computer service providers can grant accesses to the end-users without re-promoting the authentication challenge. It allows an end-user to access computer services using one authentication session. For example, an authentication session was created when an end-user was authenticated and gained access to an electronic mail service. The end-user can (continue to) access other computer services such as a printer service without requiring re-authentication.

Centralised single sign-on (or single sign-on) model has the same privacy problem in multi-domain environments as centralised identity management. For example, computer services provider (e.g. Google.com, Amazon.com and Apple.com) can not trust a single identity provider (e.g. Microsoft) to manage all of the authentication processes. This model can only be used in a single domain environment. For example, Google.com manages its end-users' credentials. It allows its end-users to access Google email services (i.e. mail.google.com), Google web calendar service (i.e. calendar.google.com) and Google document services (i.e. docs.google.com) using one authentication session.

### **Federated Single Sign-on**

Federated Single Sign-on model (FSSO) aims to deliver single sign-on across multiple domains without compromising the privacy of the identities (Patterson et al., 2004). It achieved it by de-centralising the identity providers to individual network domains (Table 2.1). It allows network domains to implement and manage their own identity providers (and credentials) unlike centralised identity management model.

Federated single sign-on model (Figure 2.6) relies on the establishment of a federation (i.e. a circle of trust) between network domains (Wason et al., 2003; Sullivan, 2005;

Management Types	Silo Model/Isolated User Identity Model	Centralised Identity Management Model	Federated Single Sign-on Model
Ownership of identity provider	Every domain manages its own identity providers.	An single identity provider is shared between domains.	Every domain manages its own identity providers.
Ownership and management of identities	Identities are owned and managed by their associate domains' identity providers.	Identities are owned by their associate domains but managed by a single identity provider.	Identities are owned and managed by their associate domains' identity providers.
Location of identities	One domain's identities are isolated from the other domains' identities.	Identities reside in a single space.	One domain's identities are isolated from the other domains' identities; however, identities' attributes can be shared between domains.

Table 2.1: Differences Between Silo Model, Centralised Identity Management Model and Federated Single Sign-on Model

Mather et al., 2009). These trust relationships are created using out-of-band business and legal agreements between the federation participants (Buecker et al., 2008). The federation manages the trust relationships established beyond the internal network boundaries or administrative domain boundaries among network domains (Mather et al., 2009; Ahn & Ko, 2007). It is expressed in the form of security policies. The policies indicates the level of access of end-users (from different domains) to the computer resources. Although trust is an important part of federated single sign-on model, the management of trust is out of the scope of this research.

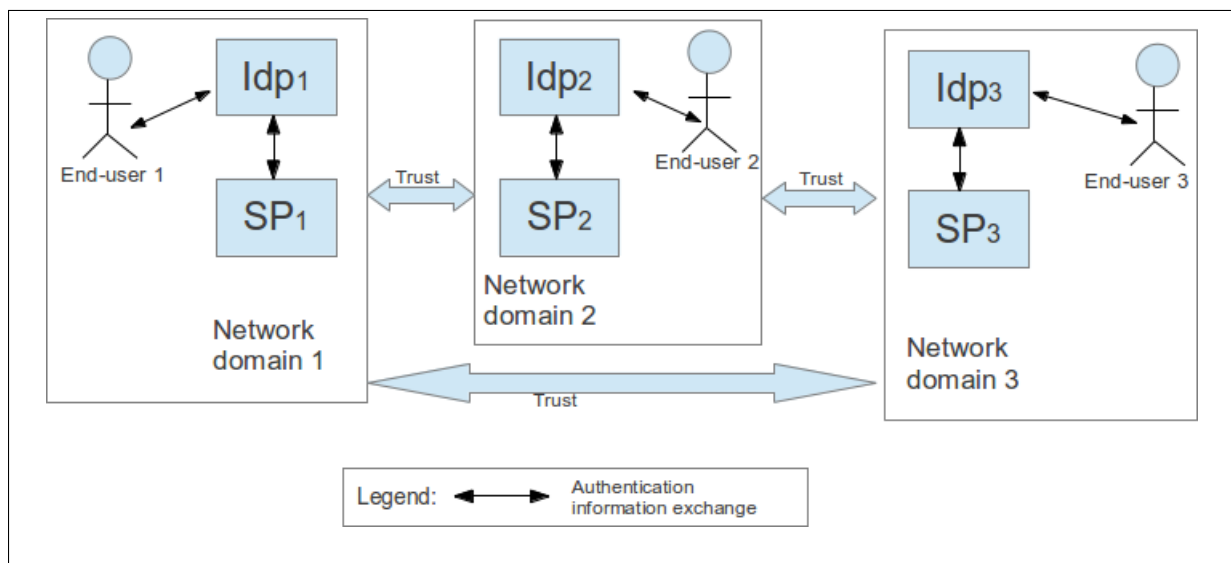


Figure 2.6: Federated single sign-on overview

The identity providers (in the FSSO) provide identity assertions for authenticated end-users. The identity assertions claim that the end-user belongs to network domains within the federation. The service providers allow end-users (who acquired identity assertions in one domain) to access computer services in the other domains (within the federation) through trust (without requiring re-authentication) (Ying, 2010; Takaaki et al., 2011; Sato & Nishimura, 2011; SWITCHaai, 2012; Google Inc., 2012). For example, a federation has been established between Dublin Institute of Technology (DIT) and Association for Computing Machinery (acm.org). The federation policies allow students (and staff) from DIT to access the online library of ACM.org (ACM Digital Library). End-users who obtained identity assertion (provided by identity provider in DIT) will be able to access the ACM Digital Library without requiring re-authentication (Figure 2.7). The federated single sign-on process consists of the following:

1. An end-user requests access to the ACM Digital Library (i.e. service provider).
2. ACM Digital Library asks which domain does the end-user belong to.
3. The end-user selects the domain (i.e. DIT).
4. ACM Digital Library re-directs the end-user to DIT's web portal (i.e. identity provider).
5. The end-user authenticates using its credential.
6. DIT will create an identity assertion for the end-user if the authentication was successful. The assertion is sent to the end-users, and it indicates that the end-user is a student (or staff) in DIT.
7. The end-user presents the identity assertion to the ACM Digital Library.
8. ACM Digital Library accepts the end-user's request and direct the end-user to its online contents.

Federated single sign-on model has advantages over centralised identity management (CIM) and the silo model: it does not present the privacy problems since each

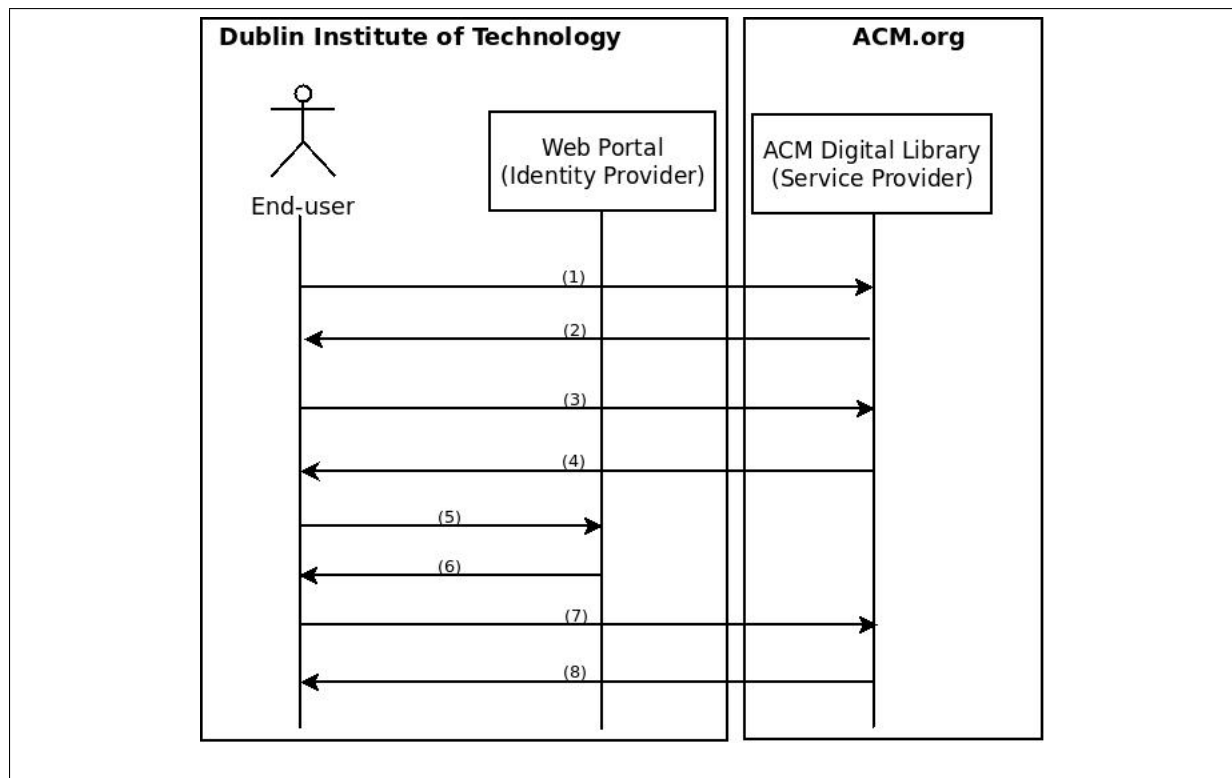


Figure 2.7: Federated Single Sign-on Process

network domain manages their own identity, and it can provide single sign-on across multiple domains using the federation established between the network domains. FSSO has the potential to become the predominate authentication model that is used by network domains to deliver cross domains single sign-on.

## 2.4 Summary

Authentication is a core functionality of identity management. It is the process of verifying the claims about holding specific identities. This chapter listed authentication mechanisms (e.g. “username and password”, “challenge and responses”, “digital certificates” and “Biometrics”) that are used by end-users for authentication. Although each authentication process may involve one or more authentication mechanisms, this research focuses on accessing computer services/applications using one authentication process. Therefore, evaluating the authentication mechanisms is out of the scope of this research.

This chapter focuses on reviewing the authentication models: the silo model, cen-



tralised identity management and federated single sign-on (FSSO). It found that FSSO model has the potential to deliver single sign-on (in a multi-domain environment) without causing privacy problem; therefore, FSSO has the potential to be the predominate authentication model (that is used by network domains) to deliver cross domain single sign-on. Chapter 3 reviews the strengths and weaknesses of the current federated single sign-on systems.

# Chapter 3

## Federated Single Sign-on (FSSO) Systems

### 3.1 Introduction

Federated single sign-on model (Figure 3.1) uses a claim based authentication model to address the cross domain single sign-on problems. It allows end-users, who have been successfully authenticated by domains within a federation, to access computer services/applications in different network domains (within the federation) without requiring re-authentication (Suriadi et al., 2009).

Federated single sign-on (FSSO) model de-centralised end-users' identities to their respective domains unlike centralised identity management model (CIM). End-users' identities are not stored and managed by a single party in this model; therefore, FSSO are not affected by single point of failure, and it does not present trust problems.

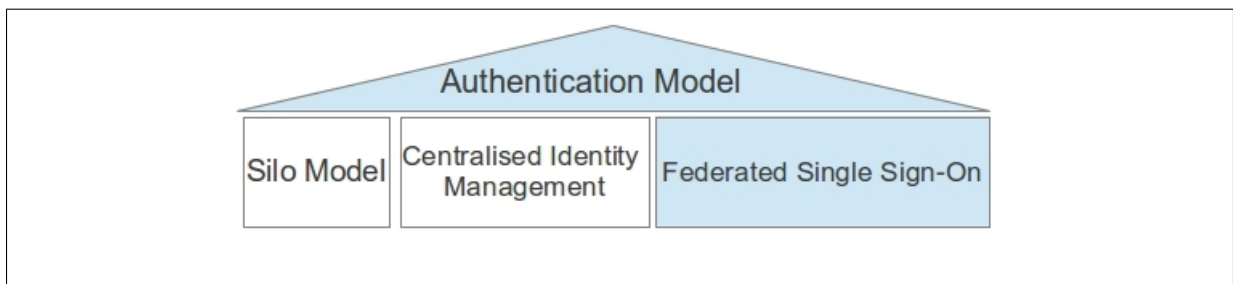


Figure 3.1: Federated Single Sign-on Model in Authentication Model

FSSO model requires a federation (i.e. a circle of trust) to be established between network domains (Wason et al., 2003; Sullivan, 2005; Mather et al., 2009). A federation (i.e. a circle of trust) allows network domains to exchange information about their end-users and computer resources, and it enables collaborations and transactions (Sullivan, 2005; Ahn & Ko, 2007). It manages the trust relationship established beyond the internal network boundaries or administrative domain boundaries among distinct organisations (Mather et al., 2009).

A federation is expressed in the form of security policies. These policies ensure the end-users have the correct level of accesses to network domains' computer resources. For example, a corporation named Medi Corp (medico.com) has an access control policy that states, in English: Any end-user with an e-mail name in the “medico.com” namespace is allowed to perform any action on any resources (Godik & Moses, 2003). The policies may be changed to machine readable language, so they can be managed by computer systems.

Beside the traditional “authority-centric” federated single sign-on system that was described, there is a “user-centric” federated single sign-on system (UFed SSO). The UFed SSO allows end-users to have an effective control over the use and management of their private identification information (Table 3.1). It adopts the user-centric identity management (UCIM), and it designs from the end-user's perspective (Jøsang & Pope, 2005; Suriadi et al., 2009). End-users manage their own private identification information in UFed SSO (Suriadi et al., 2009).

Management Type	Authority-centric FSSO	User-centric FSSO
Identity claim	The end-user was verified as who he/she claim to be (strong claim).	The end-user has an identity (weak claim).
Management of identity information	Identity information is managed by the identity providers.	Identity information is managed by the end-users.
Ownership of identity information	Identity information is owned by identity providers.	Identity information is owned by end-users.

Table 3.1: Authority-centric and User-centric Federated Single Sign-on

The UFed SSO does not provide strong identity assertions. Service providers are not obliged to trust any claims of any end-users without corroboration by a trusted third

party (Maler & Reed, 2008). The “Authority-centric” federated single sign-on is still favoured by business enterprises, governments and education faculties such as JANET (2012) and HEAnet (2012).

The focus of this research is “authority-centric” federated single sign-on. Section 3.2 gives a brief overview on federated single sign-on standards in web space. Section 3.3 identifies the strengths and weaknesses of existing approaches that aim to extend federated single sign-on beyond web space. Section 3.4 summaries this chapter.

## 3.2 Federated Single Sign-on Solutions in Web Space

Federation technologies first appeared with the aim of offering web-based single sign-on across network domains (Paul & Madsen, 2004; Don & Smith, 2008). Federated identity and web services were both complement and dependent on each other (Paul & Madsen, 2004). Federated identity standards such as Security Assertion Markup Language (SAML), Liberty Identity Federation Framework (ID-FF) and WS-Federation took advantages of the web services standards stack. They were seen as built on top of web services.

Security Assertion Markup Language (SAML) is an XML-based framework for exchanging security information (e.g. end-user’s authentication, entitlement and attribute information) (Maler et al., 2003). It was developed by the Security Services Technical Committee of the Organisation for the Advancement of Structured Information Standards (OASIS). It’s main objective is to deliver a single sign-on experience for end-users to access web-based services/applications. It allows an end-user that was authenticated in one domain to use web-based services/applications in the other domains without requiring re-authenticating. SAML based solution such as Shibboleth<sup>®</sup> system provides single sign-on capabilities and allows sites to make informed authorisation decisions for individual access of protected online resources in a privacy-preserving manner (Internet2, 2012).

The Liberty Alliance project aimed to develop open standards for federated net-

work identity management and identity-based services (Sun Microsystems, 2004). It was formed in September 2001. Its goals were to ensure interoperability, support privacy, and promote adoption of its specifications, guidelines, and best practices.

Liberty Identity Federation Framework (ID-FF) was developed by Liberty Alliance. It was built on top of existing XML-based standards. ID-FF enables identity federation and management through single sign-on, end-user account linking, and simple session management (Shim et al., 2005).

SAML 1.0/1.1, Shibboleth Project and Liberty Alliance combined their work to create SAML 2.0 (Figure 3.2) (Mather et al., 2009; Jason & Goode, 2012). SAML 2.0 emerged as the de-facto federated single sign-on standard for web-based applications. It has been broadly implemented by all major web access management vendors (Ragouzis et al., 2008). It is supported by major application server products (e.g. IEEE Xplore Digital Library and ACM Digital Library). It can be found among web services management and security vendors (e.g. Ping Identity, Shibboleth and OpenAthens).

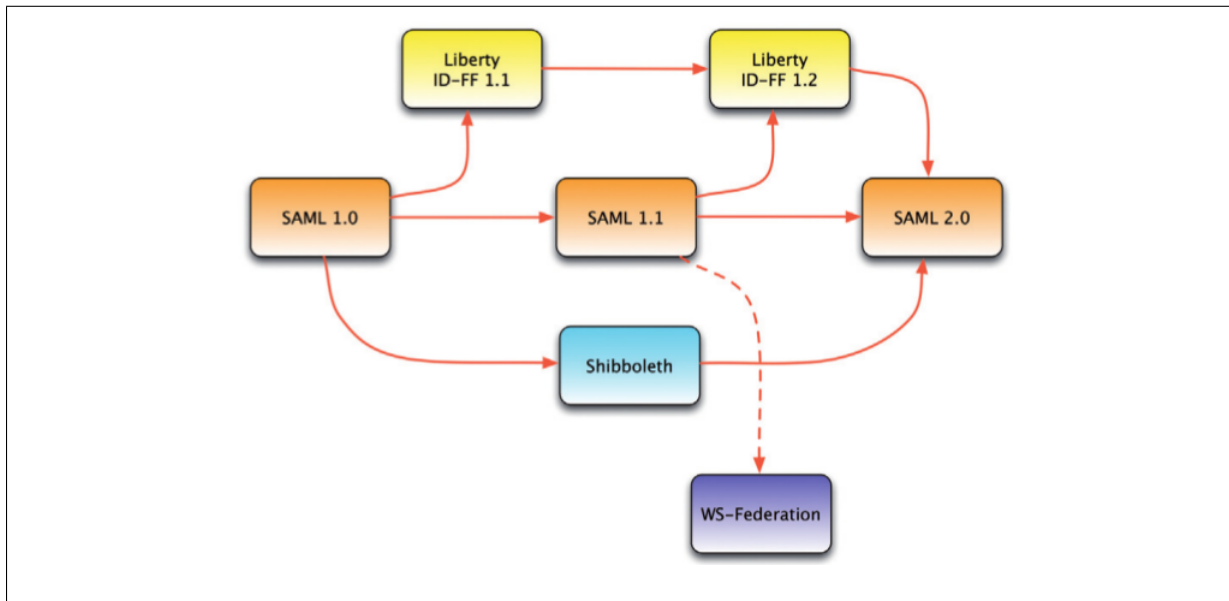


Figure 3.2: SAML 2.0 incorporates three existing federation technologies and standards: SAML 1.1, ID-FF 1.2 and Shibboleth (Jason & Goode, 2012)

WS-Federation is another federated identity standard. It was developed by Microsoft, IBM, RSA Security, VeriSign, and BEA. It relies on a security token service (STS) to broker trust of identities, attributes, and authentication between participating

web services. WS-Federation v1.2 has been approved by OASIS to become a federated single sign-on standard for web-based services/applications McRae (2009). WS-Federation based products (e.g. WS Federation Authentication Module) have been integrated into Microsoft's software. WS Federation Authentication Module can be used by network domains that use Microsoft products (Microsoft, 2012b).

Federated single sign-on standard such as SAML, ID-FF and WS-Federation partially address the single sign-on problem. They allow end-users to access web-based services in a single sign-on fashion. They do not allow end-users to access both web-based and non-web based computer services/applications in a single sign-on fashion.

This research proposes to integrate web-based FSSO standards with FSSO solutions that support non-web based services/applications. Although these standards can not address the single sign-on problem on their own, they can be used in conjunction with FSSO solutions that support non-web based services/applications. The integration can provide an single sign-on session for end-users to access both web-based and non-web based services.

### **3.3 Federated Single Sign-on Solutions Beyond Web Space**

The need of federated single sign-on beyond web-based services/applications has been emphasised by Papez (2009); Wierenga & Lear (2012); Howlett (2011). This research reviewed four projects that aim to extend federated single sign-on beyond web space: Project Moonshot, Kerberos Secure Sharing (KSS), SASL-SAML (i.e. the combination Simple Authentication and Security Layer with Security Assertion Markup Language) and SAML-AAI/Kerberos (i.e. the combination of Security Assertion Markup Language's authentication authorisation infrastructure with Kerberos single sign-on infrastructure). This section reviews the strengths and weaknesses of these approaches.

### 3.3.1 Project Moonshot

Project Moonshot is a federated single sign-on project initiated by JANET(UK) (Howlett, 2011). It attempts to address a number of problems that were not addressed by SAML based federated single sign-on systems. They include: SAML based federated single sign-on do not support non-web based applications; it is difficult for end-users (who belong to multiple domains) to be authenticated by the correct identity provider (and obtain the correct identity assertion); it is difficult to select a identity provider from a large list (i.e. a list that contains more than 100 identity providers).

Project moonshot proposes a new federated single sign-on system that supports non-web based applications. It combines Extensible Authentication Protocol (EAP) and Generic Security Services Application Programming Interface (GSS-API) to create a solution for federated authentication (Howlett, 2010). EAP provide federation and authentication mechanisms; it verifies the authentication, authorisation and auditing (AAA) information. GSS-API integrates non-web based services/applications with the new federated single sign-on system, so that these services/applications can support EAP. The authentication process (Figure 3.3) includes: (1) an end-user's user agent sends the end-user's credentials to the service provider via GSS-API; (2) the service provider sends the credentials to the identity provider via AAA transport protocol; (3) the identity provider verifies the credentials using EAP, and replies the result via AAA transport protocol; (4) the service provider accepts or rejects end-user's requests based on the authentication result.

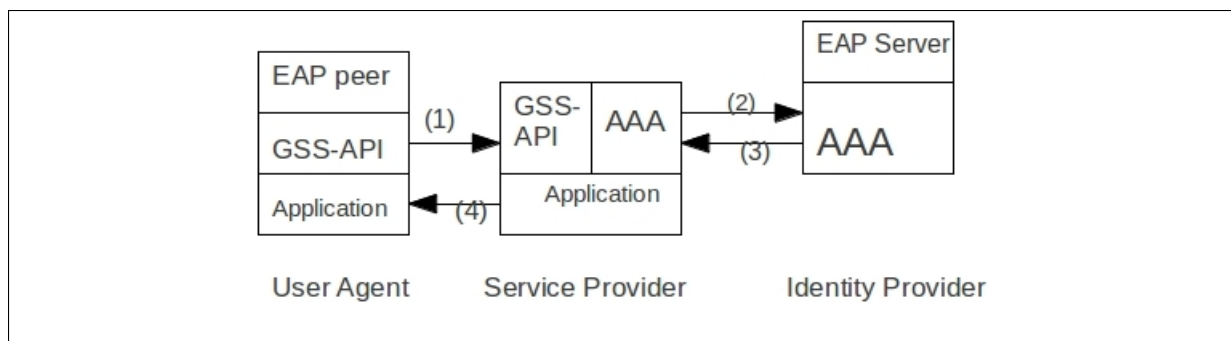


Figure 3.3: Project Moonshot Model (Howlett, 2011)

Project Moonshot makes several important assumptions about the environment

of the end-users (Painless Security, 2010). It assumes end-users have EAP supplicant software installed and configured. It assumes end-users have configured their supplicant with some information about the sets of identities they use. It also assumes end-users may be required to take some steps to enable support for Project Moonshot in their applications (e.g. replacing existing applications with ones that support project Moonshot).

It is clear that Project Moonshot is suitable for network domains that have the ability to configure the machines of all their affiliates. It is also suitable for experienced end-users who are able to follow directions and who see sufficient benefit in federated single sign-on. It is not suitable for end-users who use machines that they do not control or that have not been configured to meet the needs of their organisations.

Although Project Moonshot can deliver federated single sign-on for non-web based services/applications, it does not interoperate with web-based federated single sign-on technologies. This results in end-users need to be authenticated twice to access both web-based and non-web based services. In addition, Project Moonshot does not interoperate with network domains' existing non-web based authentication infrastructures, which means federated authenticated (by Project Moonshot) and desktop authentication are two separate authentication processes. End-users need to be authenticated at least three times to access desktop system, web-based services/applications and non-web based services/applications.

### **3.3.2 Kerberos Secure Sharing**

Kerberos Secure Sharing (Gridwise Tech, 2010) allows end-users to access desktop systems and non-web based applications using one authentication process (Figure 3.4). It proposes an architecture that implements federated single sign-on using LDAP, Kerberos and GridwiseTech's AdHoc. LDAP manages authorisation (i.e. end-users' access rights). Kerberos manages authentication and single sign-on. GridwiseTechs AdHoc is responsible for managing access policies to resources.

Kerberos enables network applications to securely identify their peers (Miller et al.,



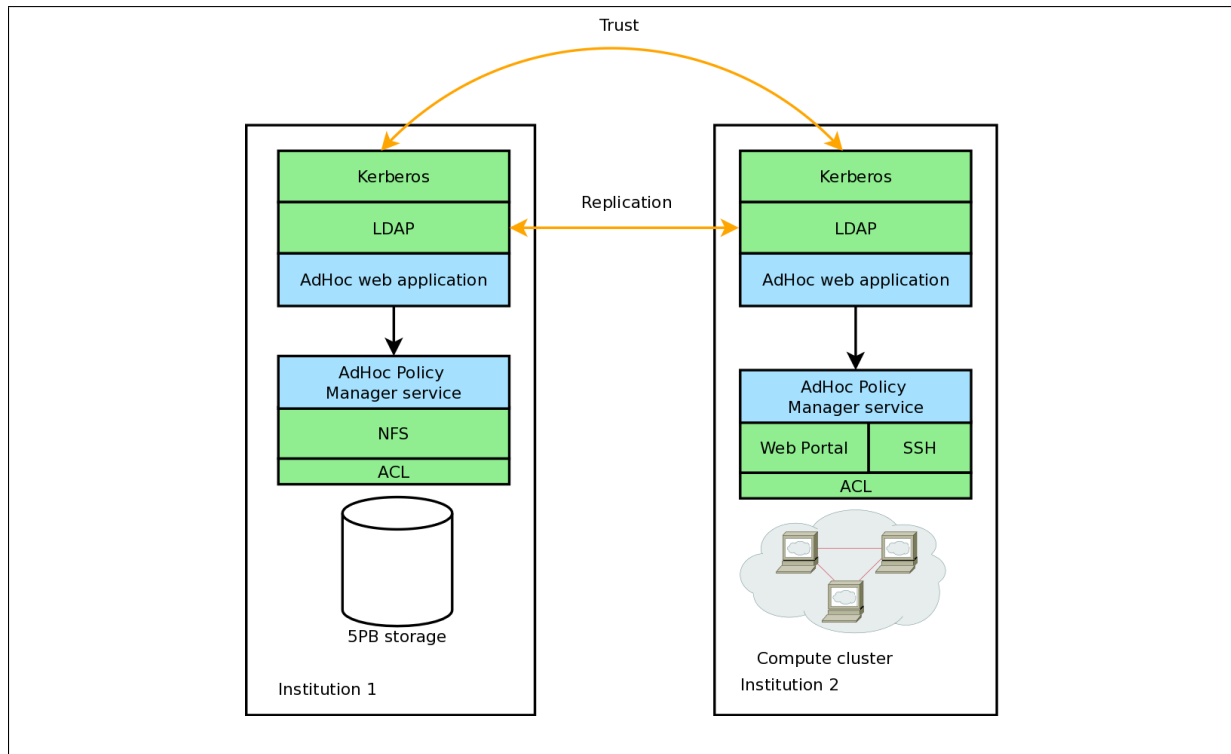


Figure 3.4: Kerberos Secure Sharing (Gridwise Tech, 2010)

1988). Kerberos has the following major components: Key Distribution Centre (KDC) and Ticket Granting Service (TGS). KDC is trusted to hold secret keys known by each client and server on the network (their secret keys are established out-of-band or through an encrypted channel). It also has access to the database that stores end-users' credentials (Needham & Schroeder, 1978). Ticket granting server (TGS) is logically distinct from KDC which provides the initial ticket granting ticket service. TGS runs on the same host and has access to the same database of clients and keys used by the KDC (Kohl et al., 1991; El-Hadidi et al., 1997).

Kerberos Secure Sharing (KSS) established federation by directly connecting network domains' Kerberos single sign-on systems (e.g. KDCs and TGSs). It requires each network domain's KDC to have records of the other KDCs. It does not deploy new federated single sign-on systems into network domains.

KSS has advantages over Project Moonshot: it allows end-users to log in their desktop (using Kerberos authentication) and access non-web based services/applications (which support Kerberos) without requiring re-authentication; it does not require new federated single sign-on systems to be deployed into network domains. However, KSS

is only suitable for network domains that trust each other and willing to expose their Kerberos servers. For example, KSS is suitable for domains such as youtube.com and google.com since they belong to the same owner (i.e. google.com). It is not suitable for network domains that are unwilling to expose their Kerberos servers. For example, google.com and apple.com are competing companies, they will not expose their internal authentication infrastructures to each other.

KSS also partially addresses the single sign-on problem (similar to Project Moonshot). This approach only focus on supporting desktop systems and non-web based applications. It does not allow end-users to access both web-based and non-web based services/applications without requiring re-authentication.

### 3.3.3 SASL-SAML

SASL-SAML attempts to provide federated single sign-on for both web-base and non-web based services/applications. It uses SAML based federated single sign-on systems to manage federated single sign-on for both web-based and non-web based services/applications (Wierenga & Lear, 2012). It does not use the existing authentication infrastructures (e.g. Kerberos). It integrates the existing SAML based federated single sign-on system (e.g. Shibboleth) with non-web based services/applications using Simple Authentication and Security Layer (SASL).

Simple Authentication and Security Layer (SASL) is a generalised mechanism for identifying and authenticating an end-user. It is also used for optionally negotiating a security layer for subsequent protocol interactions (Melnikov & Zeilenga, 2006). The effect is to make modular authentication, so that newer authentication mechanisms can be added as needed.

SASL-SAML relies on the exchange of SAML based messages (e.g. identity assertions and credentials) through SASL channels to achieve federated single sign on for non-web based applications (Figure 3.5). Non-web based services/applications (and their receptive user agents) are modified to support SASL. The modifications allow them

to exchange SAML based messages with the SAML based federated single sign-on systems; therefore, they can use SAML based federated single sign-on systems for federated single sign-on. This approach allows end-users who were authenticated (using SAML based federated single sign-on system) to access both web-based and non-web based services/applications without requiring re-authentication.

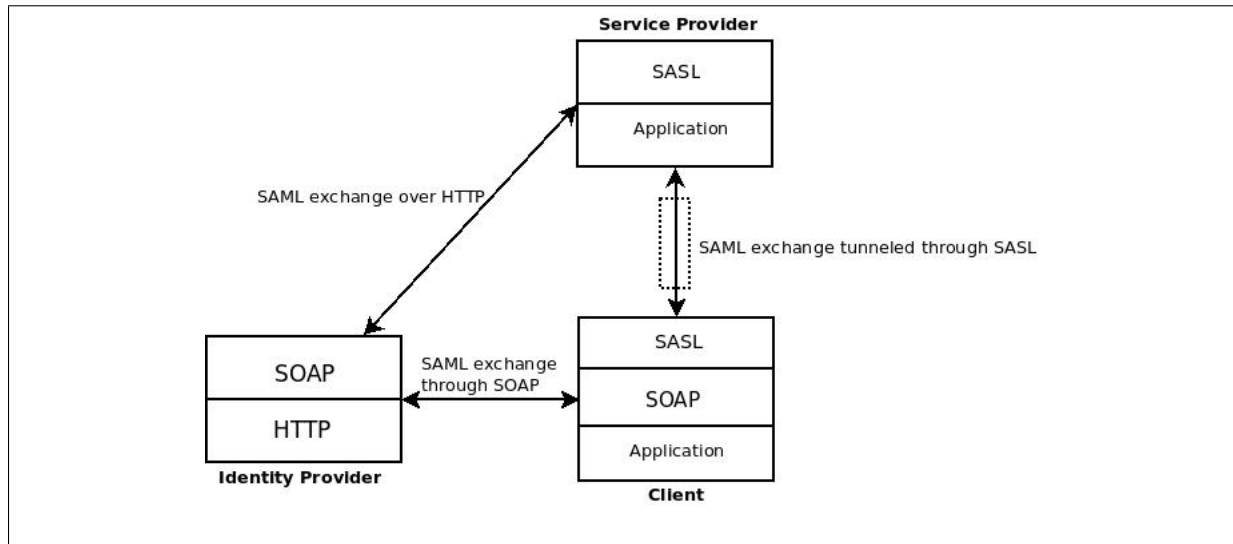


Figure 3.5: SASL-SAML Model (Wierenga & Lear, 2012)

Although SASL-SAML provides federated single sign-on sessions for end-users to access both web-based and non-web based services, it does not integrate federated single sign-on sessions with end-users' desktop authentication sessions. Therefore, the end-users need to be authenticated (by network domains' non-web based authentication infrastructures) to access desktop systems, then they need to be authenticated again (by SASL-SAML) to access both web-based and non-web based services/applications.

This approach requires active support and commitment from applications providers and network domains. It requires existing non-web based service/applications to be modified to support SASL. If applications providers and network domains are unwilling to modify their existing services/applications, this model will fail.

### 3.3.4 SAML-AAI/Kerberos

SAML-AAI/Kerberos (Papez, 2009) allows end-users to access non-web based services as web-based services, so that SAML based federated single sign-on systems can be used to manage federated single sign-on for all services/applications. It contains two major components: web-based software that integrates SAML based federated single sign-on systems with network domains' Kerberos single sign-on systems (Figure 3.6) and web-based interfaces for non-web based services/applications. The combination of the web-based software and web-based interfaces allow end-users to access both web-based and non-web based services/applications without requiring re-authentication.

The web-based software integrates end-users' web-based single sign-on sessions (created by SAML based federated single sign-on systems) with non-web based single sign-on sessions (created by Kerberos single sign-on systems). It requires end-users to be authenticated by SAML based federated single sign-on systems. It uses Kerberos extensions – “Services For User to Self (S4USelf)” and “Services for User to Proxy (S4U2Proxy)” – to acquire Kerberos tickets (i.e. ticket granting ticket and service ticket) on behalf of the authenticated end-users (Papez, 2009; Microsoft, 2012a). It seamlessly extends end-users' federated single sign-on sessions (created by SAML based federated single sign-on) into the other domains.

SAML-AAI/Kerberos requires application providers and network domains to develop web-based interfaces for non-web based applications. End-users can access non-web based services/applications via their web-based interfaces; therefore, end-users can access both web-based and non-web based services/applications using web browsers.

Although SAML-AAI/Kerberos partially leveraged network domains' authentication infrastructure, it also does not integrate federated single sign-on sessions with desktop authentication sessions (similar to SASL-SAML). This approach requires at least two authentication processes for an end-user to access a desktop system and (both web-based and non-web based) computer services/applications. For example, end-users need to be authenticated once to access their desktop, then they need to be authenticated again to

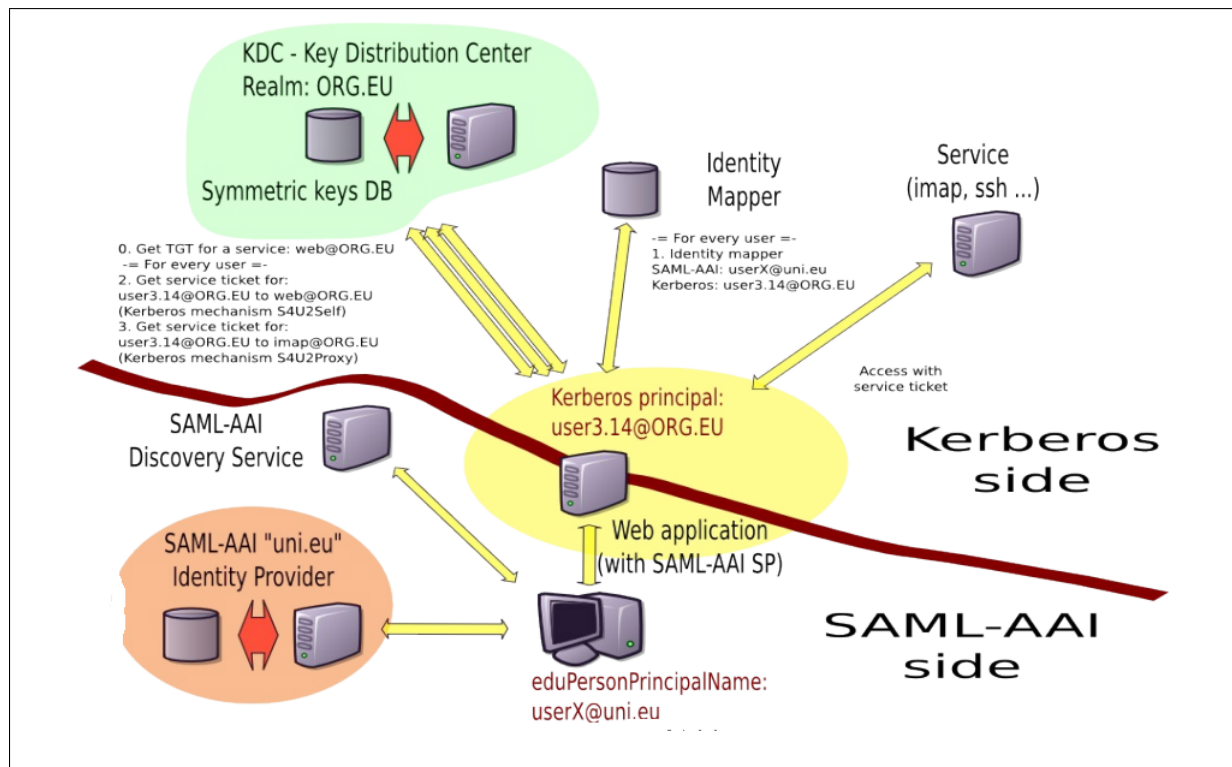


Figure 3.6: Interoperating SAML-AAI and Kerberos (Papez, 2009)

access web-based services/applications.

This approach also requires active support and commitment from applications providers and network domains similar to SASL-SAML. It requires them to create/deploy new web-based interfaces for their non-web based applications, so that end-users can access non-web based services/applications as they are web-based services/applications. This approach will fail if applications providers and network domains are unwilling to create new web-based interfaces for their non-web based services/applications.

### 3.4 Summary

The chapter found that there is a lack of federated single sign-on system that incorporates the authentication systems of desktop systems, web-based services/applications and non-web based services/applications. Project Moonshot supports non-web based services/applications, Kerberos secure sharing supports desktop systems and non-web based services/applications, and SASL-SAML and SAML-AAI/Kerberos support both

web-based and non-web based services/applications. These systems can not allow end-users to access desktop systems, web-based services/applications and non-web based services/applications using one authentication process. Chapter 4 proposes a new simplified federated single sign-on system that attempts to address this problem.

# Chapter 4

## The Design of a New Simplified FSSO System

### 4.1 Introduction

This chapter presents the design of the new simplified federated single sign-on (FSSO) system. The research found that both Kerberos secure sharing (KSS) and SAML-AAI/Kerberos leverage network domains' existing authentication infrastructures. KSS provides federated single sign-on by directly connecting network domains' Kerberos servers (i.e. non-web based authentication infrastructures). SAML-AAI/Kerberos provides federated single sign-on by connecting SAML-based federated single sign-on systems and service providers' Kerberos servers.

The proposed system aims to allow end-users to access desktop systems (in the same domain as the end-users), web-based services/applications and non-web based services/applications (in multiple domains) using one authentication process. It's designed to incorporate network domains' existing authentication infrastructures without exposing them to the outside world.

The design of the new simplified federated single sign-on system includes two main components: the design of a novel "Authentication Infrastructure Integration Program"

(AIIP) and the design of the “Integration of Desktop Authentication and Web-based Authentication” (IDAWA). AIIP is designed to facilitate federated single sign-on for end-users who have gained access to desktop systems (i.e. who have been authenticated by network domains’ Kerberos single sign-on systems). IDAWA is designed to integrate end-users’ desktop authentication sessions with web-based federated single sign-on sessions.

AIIP draws from Kerberos secure sharing and SAML-AAI/Kerberos. It also uses network domains’ Kerberos single sign-on systems for desktop authentications and local network single sign-on; however, it does not directly connect Kerberos single sign-on systems in different domains. AIIPs in different domains facilitate federated single sign-on by seamlessly bridging connections between end-users’ desktop systems in one domain and Kerberos single sign-on systems in the other network domains. AIIP in one domain acquires Kerberos tickets from Kerberos single sign-on (in the same domain) and sends the tickets to the end-users (in another domain). AIIPs allows end-users to use these Kerberos tickets to access non-web based services/applications (in different domain) in a single sign-on fashion.

IDAWA is an extension of SAML based federated single sign-on systems. It modifies web-based authentication systems (i.e. identity provider end of SAML based federated single sign-on systems) to support Kerberos tickets. It allows end-users who have acquired Kerberos tickets (i.e. who have been authenticated by Kerberos single sign-on systems) to access web-based services/applications without requiring re-authentication.

The new simplified federated single sign-on system assumes the following: a federation has been established between network domains; network domains employ their own Kerberos single sign-on systems (i.e. authentication infrastructures); they support SAML based federated single sign-on systems (i.e. web-based federated single sign-on systems); network messages are encrypted during transfer.

The assumptions are realised in the design environment. The design environment consists of simulations of the following: Kerberos single sign-on systems, SAML based federated single sign-on systems, Linux desktop systems and Secure Shell protocol. The



simulations were implemented with a goal to demonstrate the new simplified federated single sign-on system. They are the striped down versions of the real world systems.

Section 4.2 presents the design environment. It discusses the use of simulations in the design environment. Section 4.3 presents the technical details of the the simulations (including Kerberos servers, SAML based federated single sign-on systems, Linux desktops and Secure Shell protocol). Section 4.4 presents the design of the new simplified federated single sign-on system. It includes the design of two major components: “Authentication Infrastructure Integration Program” (AIIP) in section 4.4.1, and the “Integration of Desktop Authentication and Web-based Authentication” (IDAWA) in section 4.4.2. It then presents the life-cycles of the proposed system in section 4.4.3. Section 4.5 summaries this chapter.

## 4.2 Design Environment

### 4.2.1 Simulation of Real World Applications

Real world systems are production environments. They are managed by software programmers or system administrators. They can be modified by them. For example, administrators can change the IP addresses of servers. If the research uses these servers, it must change its design variables and re-implement its systems to support the new changes. New evaluations need to be conducted as well, which results in lost time. It is not advisable to conduct research in these environments.

This research uses simulations of real world systems: Kerberos single sign-on systems, SAML based federated single sign-on systems, Linux desktop systems and Secure Shell protocol. The simulations provide the necessary functionalities for designing the proposed system. They partially imitates the real world systems. Their goals are to present the life-cycle of a real world authentication processes in a clean manner.

The benefits of using simulations are the simulations are small in size, they utilise a small amount of computer resources, all processes (e.g. communication processes and

authentication processes) can be monitored and all communication messages and stored files (e.g. Kerberos tickets and databases) are created in human readable language. They are more manageable than real world systems.

### The Simulation of Kerberos Single Sign-on System

The simulation of Kerberos single sign-on system has the following core functionalities: authenticating end-users, creating/verifying ticket granting ticket, creating service ticket. These functionalities are implemented based on the real world systems. They are necessary to provide desktop authentication and local network single sign-on, and they provide the design environment for designing AIIP.

The simulation does not provide any encryption functionalities; it assumes the tickets are encrypted during transfer and storage. It also does not provide any administration functionalities. System configurations (e.g. create, modify or remove credentials in the end-user database) need to be done by directly editing the source code or the text files. Although these functionalities will be in the production system, they are not necessary for demonstrating the proof of concept implementation. Therefore, they are not included in the simulation (figure 4.1).

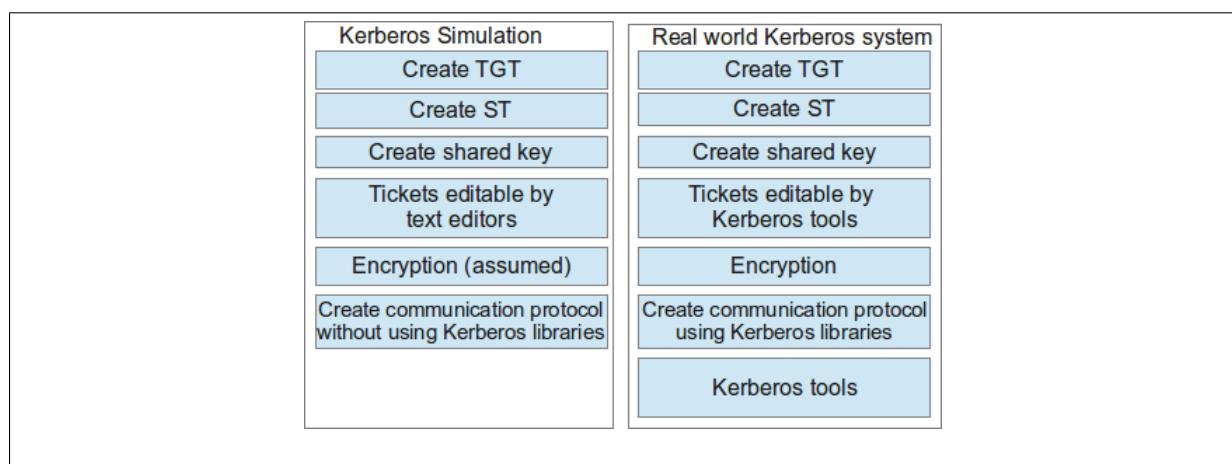


Figure 4.1: High level comparison of Kerberos simulation and real world system

The real world Kerberos system's communication protocol and Kerberos tickets can only be used by Kerberos tools. To have better management over the design environment, the simulation of Kerberos single sign-on system uses its own set of communication

protocols. In addition, the simulation's Kerberos tickets can be created, modified and removed by any text editors.

### The Simulation of SAML based Federated Single Sign-on System

The simulation of SAML based federated single sign-on system focuses on providing functionalities that are necessary for designing IDAWA (Figure 4.2). These functionalities include verifying end-users' credentials, creating identity assertions, storing identity assertions in web-browsers' cookies and verifying identity assertions. These functionalities are implemented based on real world systems.

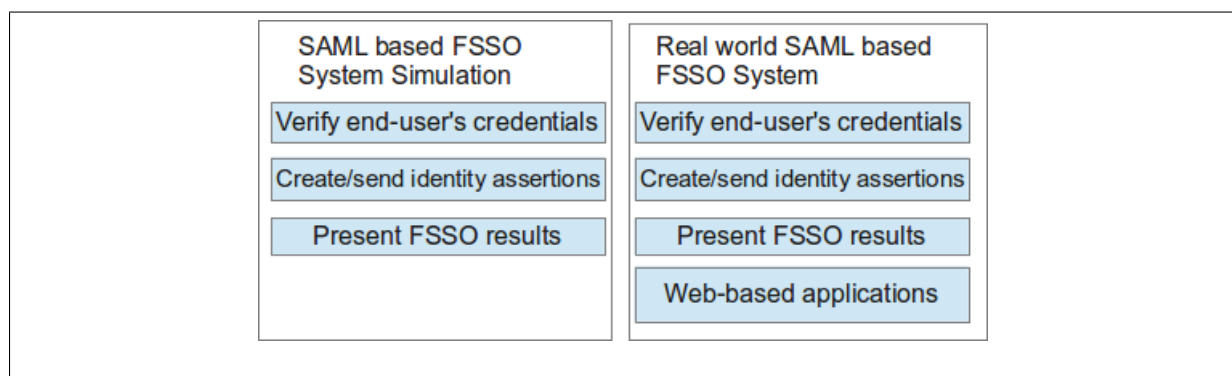


Figure 4.2: High level comparison of SAML based FSSO system simulation and real world system

The simulation will present the result of authentication (e.g. successful or unsuccessful). It does not provide any real world web-based services/applications. The identity message (used by the simulation) is a strip down version of the real world SAML based assertion message. It focuses on elements that are necessary for federated single sign-on processes. These elements include username, role, domain name and authentication result.

### The Simulations of Linux Desktop System and Secure Shell Protocol

The simulation of Linux desktop system is based on Ubuntu desktop. Ubuntu desktop is a free and open source Linux desktop system. It supports Kerberos single sign-on system, and it can use Kerberos as primary desktop authentication system. The end-user can use

it with or without a graphical user interface.

In order to demonstrate authentication and single sign-on, it is important to select functionalities (from real world systems) that relate to authentication and single sign-on. The simulation of Ubuntu desktop provides functionalities such as verifying end-users' credentials, storing the simulations of Kerberos tickets, sending the simulations of Kerberos ticket, the simulation of secure shell client and the simulations of secure shell server (Figure 4.3). The simulation of secure shell protocol provides functionalities such as verifying the simulation of Kerberos tickets and creating a connection between a client and server (Figure 4.4).

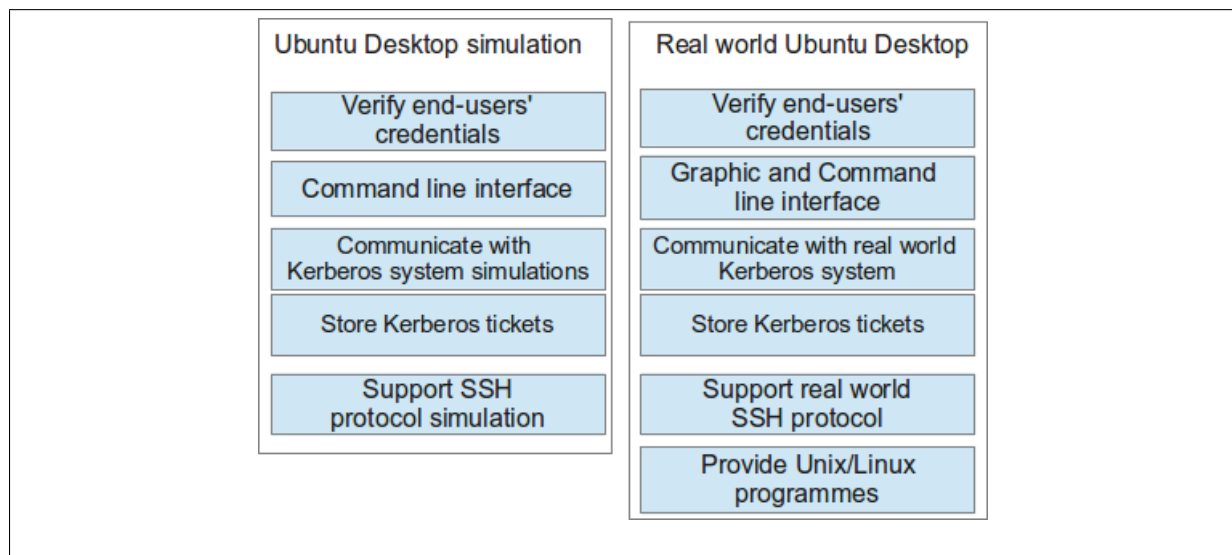


Figure 4.3: High level comparison of Ubuntu desktop simulation and real world system

These simulations uses the simulations of Kerberos single sign-on system for authentication and single sign-on. They support the communication protocols of the simulation of Kerberos single sign-on systems, and they do not interoperate with the real world system. The simulation of Ubuntu desktop does not provide any non-web based services/applications. The simulation of secure shell protocol assumes network communications are encrypted.

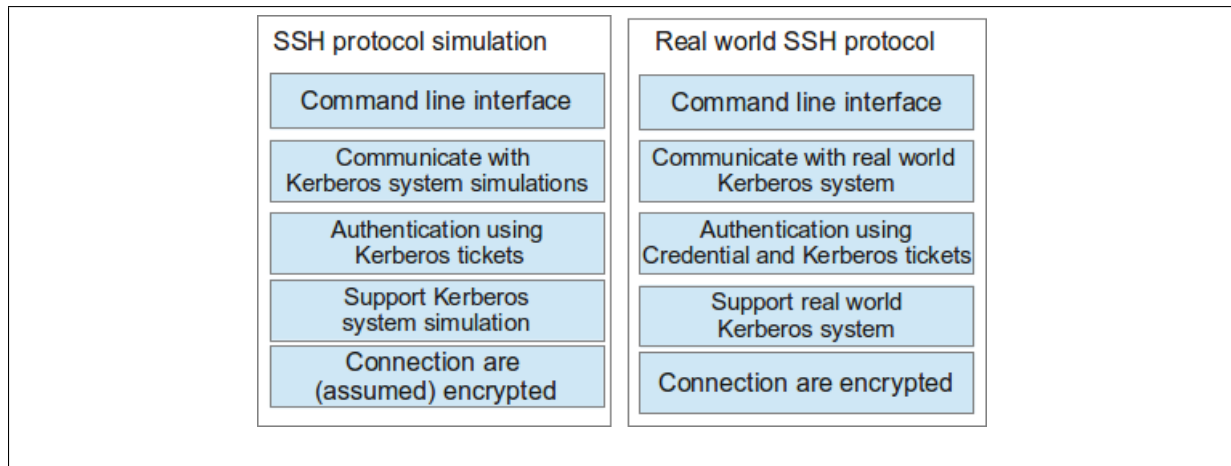


Figure 4.4: High level comparison of SSH protocol simulation and real world system

## 4.2.2 Environment Specification

### Three Sample Network Domains

The design environment specifies three sample network domains. They are named domain Alpha, domain Beta and domain Gamma (Table 4.1). Domain Alpha is assigned the domain name: “home.virtual.vm”, domain Beta is assigned the domain name: “foreign.virtual.vm” and domain Gamma is assigned the domain name: “third.virtual.vm”.

Domain Attributes	Domain Alpha	Domain Beta	Domain Gamma
Domain Name	home.virtual.vm	foreign.virtual.vm	third.virtual.vm
Provide computer Services/Applications	Yes	Yes	Yes
Provide end-users' identities	Yes	Yes	Yes

Table 4.1: Network Domain Names

The number of network domains will not effect the out come of the research. Each federated single sign-on process requires two parties: an identity provider and a service provider. A network domain can be an identity provider, a service provider or both at the same time. There can be more than three domains; however, every federated single sign-on process only involves an identity provider (in one domain) and a service provider (in another domain).

Although a new federated single sign-on process is created for accessing a new service/application, end-users only need to be authenticated for the first federated single

sign-on process. The rest of the federated single sign-on processes are seamless to the end-users; therefore, end-users view multiple federated single sign-on processes as one federated single sign-on process.

## Domain Setups

All three sample domains have the same setup. They have the simulations of the following systems: Kerberos single sign-on systems, SAML based federated single sign-on systems, Ubuntu desktop systems and secure shell protocol (SSH). The simulations of Kerberos single sign-on systems manage single sign-on inside local network domains, and they provide authentications for the simulations of both Ubuntu desktops and SSH protocol. The simulations of SAML based federated single sign-on systems facilitate federated single sign-on for web-based services/applications.

The goal of “Authentication Infrastructure Integration Program” (AIIP) is facilitating federated single sign-on by bridging communications between Kerberos single sign-on systems in different domains. The simulations of Kerberos single sign-on systems are essential for designing the AIIP. The Kerberos system simulations (in different domains) do not communicate or have any knowledge of each other.

The simulations of SAML based federated single sign-on system facilitate federated single sign-on between two ends: simulation of web-based authentication system (i.e. identity provider) and simulations of web-based services/applications (i.e. service provider). The simulation of web-based authentication system provides identity assertions for authenticated end-users. The simulation of web-based service/application displays the results of successful federated single sign-on.

The “Integration of Desktop Authentication and Web-based Authentication” (IDAWA) is an extension to the SAML based federated single sign-on system. It aims to allow the web-based authentication systems to use Kerberos tickets as mean of authentication. The simulation of SAML based federated single sign-on system is essential for designing the IDAWA.

### **Federated Single Sign-on Policy Specifications**

Federated single sign-on requires the establishment of a federation (or a circle of trust) between network domains. The federation is expressed in the form of security policies. This design establishes the following sample policies:

1. End-users from any domain can access any services/applications in the other domains.
2. The role “student” in domain Alpha has the role “guest” in domain Beta and domain Gamma. The role “guest” can use services/applications and save end-user contents on them. It does not have the permission to administrate the services/applications (i.e. create, modify and delete services/applications).
3. The role “lecture” in domain Alpha has the role “power” in domain Beta and domain Gamma. The role “power” can use, create, modify and delete services/applications.
4. The role “guest” in domain Beta and domain Gamma has the role “student” in domain Alpha. The role “student” can use services/applications and save end-user contents on them. It does not have the permission to administrate the services/applications (i.e. create, modify and delete services/applications).
5. The role “power” in domain Beta and domain Gamma has the role “lecture” in domain Alpha. The role “lecture” can use, create, modify and delete services/applications.

The first policy is the basic policy for federated single sign-on. The identity provider in one domain provides identity assertions for authenticated end-users. The identity assertions indicate that the end-users have been authenticated by one domain within the federation. The policy allows the service providers in the other domains to accept the end-users’ service requests. This policy is applicable to all three sample domains.

Although authorisation is not the focus of this research, the second and third policies were required to support the authentication systems, and to demonstrate the changes of identities in different domains. They define the changes of roles from domain

Alpha to domain Beta/Gamma. They are applicable to domain Beta and domain Gamma. They are expressed in the form of XML document (Figure 4.5).

```
<?xml version="1.0" encoding="iso-8859-1"?>
<policy>
  <student>
    <role>guest</role>
  </student>
  <lecture>
    <role>power</role>
  </lecture>
</policy>
```

Figure 4.5: Second and Third Federation Policies Written in XML

The fourth and fifth policies define the changes of roles from domain Beta/Gamma to domain Alpha. They are applicable to domain Alpha. They are expressed in figure 4.6.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<policy>
  <guest>
    <role>student</role>
  </guest>
  <power>
    <role>lecture</role>
  </power>
</policy>
```

Figure 4.6: Fourth and Fifth Federation Policies Written in XML

The goals of these policies are to set up trust between network domains. The first policy is the most important one because it defines the basic trust between network domains, and it is used by the new simplified federated single sign-on system to facilitate federated single sign-on. The next four policies are used to demonstrate the changes of role when end-users access different services/applications in different domains, and they will not effect the out come of this research.

## Application Specification

Simulations of Kerberos single sign-on systems need to be executed on operating systems. This research requires operating systems (e.g. GNU/Linux) to be installed on three separate machines. Each machine represents a domain (Alpha, Beta or Gamma). Each machine hosts a simulation of the Kerberos system.



The simulation of SAML based federated single sign-on system operates on top of web server software. It requires web server software (e.g. Apache HTTP server) to be installed on the operating systems. Each domain (i.e. each machine) needs to install a copy of the web server software.

## 4.3 Simulation Details

This section presents the technical specifications of end-user databases, Kerbero single sign-on simulation, SAML based federated single sign-on simulation, Ubuntu desktop simulation and secure shell protocol simulation. All three domains have the same setups; therefore, the technical specifications will use domain Alpha as the sample domain.

### 4.3.1 End-user Database

The end-user database aims to store information that support the authentication systems. It is used by the federated single sign-on systems and Kerberos single sign-on systems. Federated single sign-on systems and Kerberos single sign-on systems require four basic types of information: username, password, role and domain name.

- Username, it is the unique identifier of the end-user. It represents an individual end-user. There should not be any repeating usernames in the database.
- Password, it is a combination of characters and numbers. It is the distinguishing characteristic that differentiates that particular end-user or group from others (Smith, 2001).
- Role, in this research, it represents an end-user's character, place and status in the network domain.
- Domain name, it is the name of the domain where the end-users reside in.

Each network domain will have an end-user database. These databases are independent from each other. The credentials in a network domain's end-user database are

provisioned to the other domains when a federation was established between these domains. For example, credentials in domain Alpha ( $D_\alpha$ ) are provisioned into domain Beta ( $D_\beta$ ) and domain Gamma ( $D_\gamma$ ). The same applies to domain Beta and domain Gamma. Subsection 4.3.1 presents the variables in the database, provisioning of credentials, and the technology used to construct the databases.

### Credential Provisioning

SAML-AAI/Kerberos provisions separate end-user databases in different domains. It proposed an identity mapping model that maps the identities in one domain into identities in the other domains. End-users seamlessly use different credentials to access computer services/applications in different domains.

This research manually provisions the credentials. It adopts the approach of SAML-AAI/Kerberos; end-users have different credentials in domain Alpha, domain Beta and domain Gamma. The credentials in these domains have the same username; however, they have different passwords, roles and domain names. The identities were mapped the identities from one domain into the other domains using SAML-AAI/Kerberos' mapping model. This research demonstrates federated single sign-on for three end-users. Their usernames are "chen", "fred", and "paul" (Table 4.2). These three credentials were provisioned from domain Alpha to domain Beta and domain Gamma. The end-users' passwords are different in different domains. For example, "chen" has the password "alpha123" in domain Alpha. The provisioned credentials "chen" has the password "beta123" in domain Beta .

Role represents an end-user's character, place and status in the network domain. For example, "chen" has the role of "student" in domain Alpha, and "chen" has the role of "guest" in domain Beta and domain Gamma. "chen" can use service applications in domain Beta/Gamma, but he can not create, modify or delete the service applications. "fred" and "paul" have the role of "lecture" in domain Alpha; therefore, they have the role "power" in domain Beta and domain Gamma. They can use, create, modify and

delete the service applications in domain Beta/Gamma.

The Kerberos single sign-on servers only authenticate end-users in their respective domains. For example, the Kerberos single sign-on server in domain Beta only authenticates end-users who belong to domain Beta (i.e. who have the domain name: “foreign.virtual.vm”). End-users’ credentials in domain Alpha need to be changed from “home.virtual.vm” to “foreign.virtual.vm” in domain Beta and to “third.virtual.vm” in domain Gamma.

Attributes	Domain Alpha	Domain Beta	Domain Gamma
Username	chen	chen	chen
Password	alpha123	beta123	gamma123
Role	Student	Guest	Guest
Domain Name	home.virtual.vm	foreign.virtual.vm	third.virtual.vm
Username	fred	fred	fred
Password	alpha123	beta123	gamma123
Role	Lecture	Power	Power
Domain Name	home.virtual.vm	foreign.virtual.vm	third.virtual.vm
Username	paul	paul	paul
Password	alpha123	beta123	gamma123
Role	Lecture	Power	Power
Domain Name	home.virtual.vm	foreign.virtual.vm	third.virtual.vm

Table 4.2: Domain Variables

## Database Technology

The Extensible Markup Language (XML) has become the de facto standard for representing, storing, and exchanging electronic data (Bray et al., 2008; Zhang et al., 2011). It is used to construct the simulation of end-users’ databases.

The end-users’ credentials are stored in the database using XML. The end-user databases are stored as text files. They can only be read and modified locally (as a text file). They do not support network connection, and they do not provide database administration functions.

A database has three entries (i.e. one for each end-user’s credentials). This research creates a XML tag <user> to mark each credential (Figure 4.7). The credentials contain four elements: username, password, role and domain name. Four XML tags were created

<username>, <password>, <role> and <domain> to mark these four elements. The XML tag <users> is used to mark the name of the database.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<users>
  <user>
    <username>chen</username>
    <password>alpha123</password>
    <role>student</role>
    <authenticated>true</authenticated>
    <domain>home.virtual.vm</domain>
  </user>
  <user>
    <username>fred</username>
    <password>alpha123</password>
    <role>lecture</role>
    <authenticated>true</authenticated>
    <domain>home.virtual.vm</domain>
  </user>
  <user>
    <username>paul</username>
    <password>alpha123</password>
    <role>lecture</role>
    <authenticated>true</authenticated>
    <domain>home.virtual.vm</domain>
  </user>
</users>
```

Figure 4.7: Sample End-user Database Written in XML

### 4.3.2 Kerberos Single Sign-on Simulation

This research assumes each network domain employs their own Kerberos single sign-on system. The simulation of Kerberos single sign-on system is based on the designs that were presented in Kohl et al. (1991); Neuman & Ts'o (1994); Walla (2000). They present the detailed descriptions of Kerberos single sign-on system's functions (e.g. creating/verifying tickets) and life-cycles.

A Kerberos system contains a key distribution centre (KDC) and ticket granting server (TGS). According to Kohl et al. (1991), KDC and TGS reside in the same physical server. They have access to the same end-user database. The differences between KDC and TGS are their functions. KDC authenticates the end-users' credentials and issues ticket granting tickets (TGT). KGS verifies the TGT and issues service tickets (ST).

The simulation includes the functions of KDC and TGS. Since the differences between KDC and TGS are logical, they are merged into a single component (in the simulation). The Kerberos system simulations are implemented based on the real world

system. They have the following core functionalities:

- Verifying end-user's credentials.
- Creating ticket granting ticket.
- Creating shared key between service applications and Kerberos servers.
- Verifying ticket granting ticket.
- Creating service ticket.
- Creating and maintain single sign-on session.

### **Kerberos Simulation Daemon**

Kerberos system simulation authenticates the end-users, and it issues ticket granting tickets and service tickets. They also issue shared keys to services/applications. They require static IP addresses and a specific network port for accepting incoming requests. The real world Kerberos single sign-on systems use port 88 by default; the simulation uses port 88 as well.

Kerberos system simulations use network sockets for network communication. Network sockets allow applications to communicate using standard mechanisms built into network hardware and operating systems. They were created using the identified IP addresses and network ports. This research uses Transmission Control Protocol (TCP) as the socket protocol similar to real world systems. TCP employs error detection and provides reliable network communications.

### **Database Functions**

Extracting the credential data from the database is one of the tasks of the Kerberos system simulations. It allows Kerberos system simulations to authenticate the end-users by comparing the credential information in the database and the credential information

of the end-users. The end-user database was constructed with XML language (Section 4.7); therefore, the database functions include the following:

- Reading the entire content of the XML file.
- Searching through the content to find the credential data.

The contents of a database are organised in a tree format (Figure 4.8). The searching function starts from the root element of the tree (i.e. `<users>`) and travels to child element (i.e. `<user>`) and sub child elements (i.e. `<username>`, `<password>`, `<role>` and `<domain name>`).

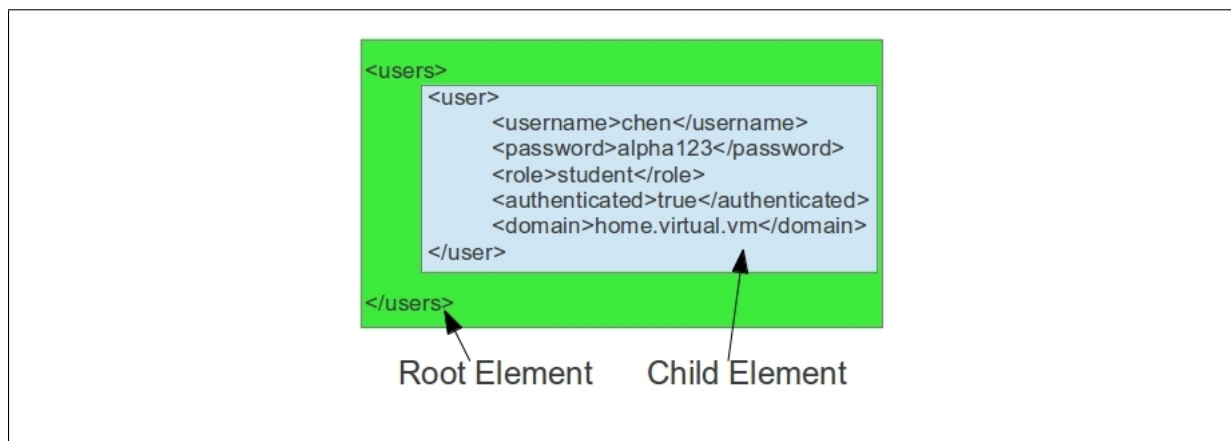


Figure 4.8: Database Tree Relation in Colour

## Authentication Function

Authentication is one of the core functionalities of Kerberos system simulation. It ensures that the end-users are who they claim to be. The authentication function involves two elements in the database: username and password.

The authentication function authenticates end-user by verifying two hash keys: one hash key is sent by an end-user's desktop system and created by putting the end-users' usernames and passwords through a SHA-2 function, and the second hash key is created from the usernames and password that are stored in the database. The authentication function will return a success message to the end-user and the Kerberos server simulation if the hash keys match.

SHA-2 is a stronger hash function than SHA-1, and NIST had decided to phase out SHA-1 in favour of SHA-2 (NIST, 2004); therefore, hash keys are created using the SHA-2 function in this research. A string that contains the username, role, authentication result, domain name, issue date and expire date is used as input for the hash function. The SHA-2 hash function generates a single fixed length string from the input string. The generated hash is appended to the end of the input string to create a ticket granting ticket.

### Create and Verify Ticket Granting Ticket

Ticket granting ticket (TGT) is one of the core components of Kerberos system simulation. It shows that the holder of the TGT has been successfully authenticated by the Kerberos system. The simulation of the TGT is based on the design specifications that were presented in Needham & Schroeder (1978); Kohl et al. (1991). A TGT contains the shared key between the Kerberos server and the end-user, and it contains the expired time of the tickets.

The simulation of TGT is a string that contains the username, role, authentication result, domain name, issue date, expire date and hash key. These contents are separated by semicolons (Figure 4.9). Kerberos single sign-on system performs a one-way hash function on end-users' credentials to create the hash key.

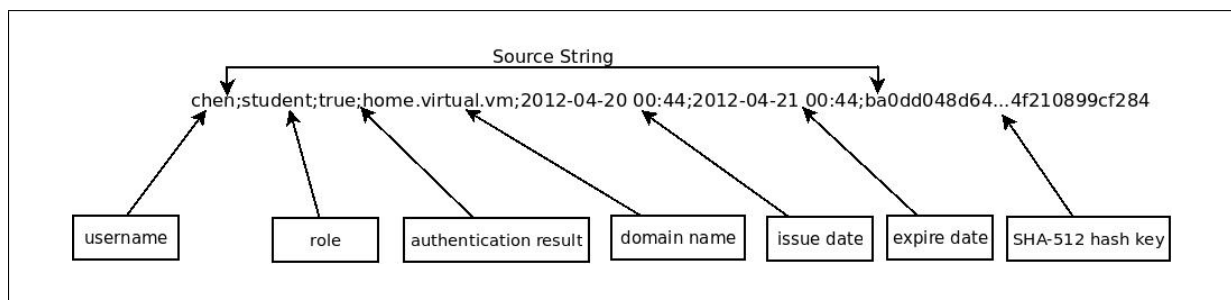


Figure 4.9: Simulate Kerberos Ticket Granting Ticket

Kerberos system simulation saves a copy of the TGT on the system. The copy is saved in the form of a text file. It's named after the end-user's username. The original TGT is sent to the end-user. The end-user saves the TGT in a text file for later use (e.g.

obtaining service ticket).

### Create Shared Key between Services and Kerberos Server

Kerberos server simulations create shared keys for service/applications simulations. The shared keys have two goals: They show that the services/applications have been registered in the Kerberos server simulation; they are also used to verify the end-users' service tickets.

A service/application simulation is manually registered in a service database. The service database is constructed using XML language. The variables of a service include: service name, domain name and IP address. For example, a sample service has the service name “ssh” (Figure 4.10). It has the sample domain name “home.virtual.vm” and sample IP address “127.0.0.1” (i.e. the local machine's IP address).

These variables are stored in a tree format. The tag `<services>` is the top of the hierarchy. Every service is under the section `<service>`. New service can be added into the service database by adding new `<service>` sections.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<services>
  <service>
    <serviceName>ssh</serviceName>
    <domain>home.virtual.vm</domain>
    <ipAddress>127.0.0.1</ipAddress>
  </service>
</services>
```

Figure 4.10: Sample Service Database Written in XML

A Kerberos server simulation needs to verify the service simulation to create a shared key. It receives the request for shared key, along with the service simulation's service name, domain name and IP address. Kerberos server simulation verifies the service simulation by searching for a matching content in the service database. It compares the content of `<serviceName>` and `<domain>` (under every service section) with the ones provided by the service simulation. If the service simulation was found in the service database, it will then create a shared key for the service simulation.

A shared ticket has three elements: service name, domain name and IP address. They are presented as strings similar to TGT. The service name, domain name and IP



address are merged into a single string (source string). They are separated by semicolons (Figure 4.11). This source string is used to create a shared key.

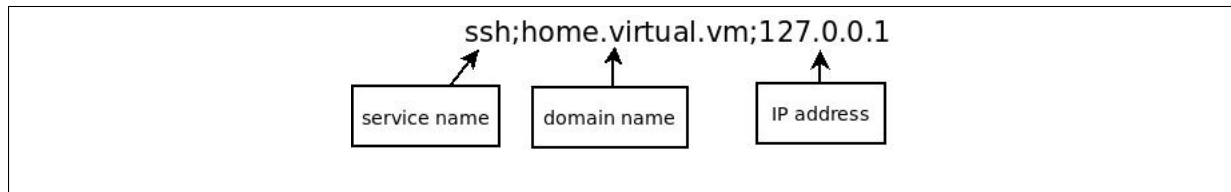


Figure 4.11: Sample Source String

SHA-2 hash function is also used to create shared keys between Kerberos server and the services. A hash key is created by putting the source string through SHA-2 hash function. The hash key is then appended to the end of the source string to create the shared key (Figure 4.12). The shared key is then manually input into the service simulation. The service simulation stores the shared key in the form of a text file.

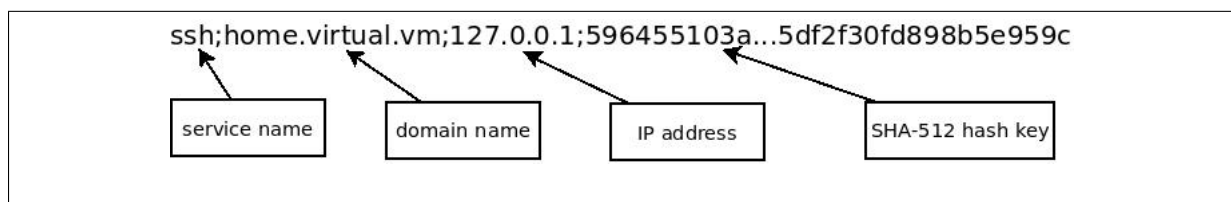


Figure 4.12: Sample Shared Key

### Create Service Ticket

End-users' ticket granting tickets (TGT) need to be verified by the Kerberos system simulations in order for them to obtain service ticket (ST). The end-users' user agents read the contents of ticket granting tickets (from the text files) and send them to the Kerberos server simulations. They also send the names of the requested services to the Kerberos server simulations.

Kerberos server simulations will search for the correspondent copies on the servers upon receiving the ticket granting tickets. They will continue with the TGT verification processes if matching copies were found on the system; otherwise, the end-users' requests will be rejected. The ticket granting ticket (TGT) verification processes include the verifications of username, domain name and hash key. They also include ensuring the issue date has expired.

A Kerberos server simulation extracts the necessary variables (e.g. username, domain name and hash key, issue date and expire date) from the TGT and from the correspondent copy. It ensures the username, domain name and hash key in the TGT match the corresponding variables in the copy. It also ensures the issue date on the TGT does not exceed the expired date on the copy. The end-user's request will be rejected if any of these condition was not met.

The creation of service ticket is the same as creating the shared key between a service/application simulation and Kerberos server simulation. The Kerberos server simulations access the service database and uses the service name (that requested by the end-user) to search for the service data (e.g. service name, domain name and IP address). These service data are merged into a single string (source string). Semicolons are used to separate the individual data. A SHA-2 hash key is created by putting the source string through SHA-2 hash function (Figure 4.13). The hash key is the service ticket; it is then sent to the end-user's user agent.

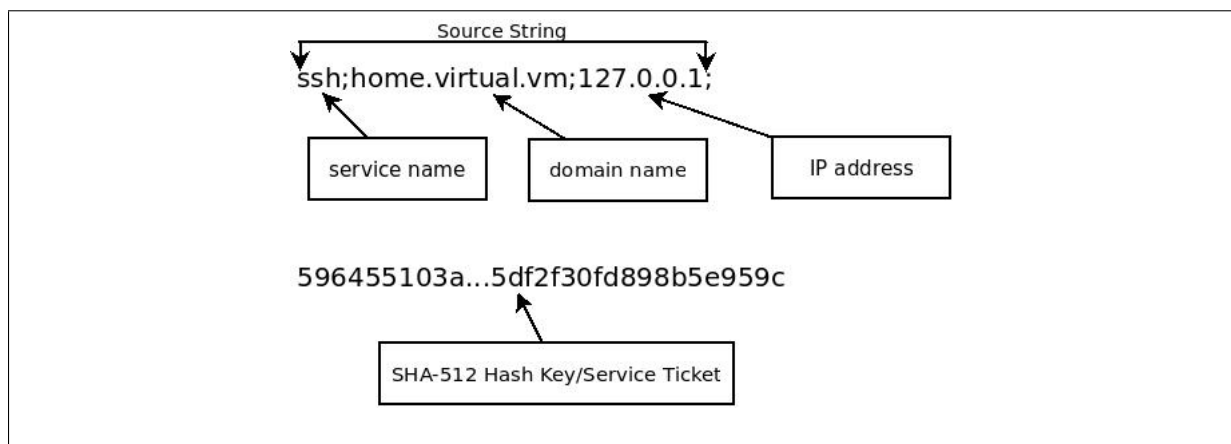


Figure 4.13: Sample Service Ticket

### 4.3.3 Ubuntu Desktop and Secure Shell Simulation

The simulation of Kerberos single sign-on system uses its own set of communication protocol, and it can not interoperate with real world Ubuntu desktop and SSH protocol. The simulations of Ubuntu desktop and Secure Shell protocol (SSH) are developed with the aim to support Kerberos single sign-on system simulation and to demonstrate non-web

based single sign-on.

The simulation of Ubuntu desktop simulations provide basic functionalities (e.g. Linux shell and web browser). It runs on top of real world operating system, and it can access/edit/remove real world files. It can share files with real world applications (e.g. web browsers) that are running on the same real world operating system.

Secure shell protocol simulations provide connection between two Ubuntu desktop simulations. It runs on the simulation of Ubuntu desktop, and it allows end-users to remotely access/use different Ubuntu desktop simulations (Figure 4.14). Relationship 1 represents the exchange of messages between an end-user and a Ubuntu desktop simulation, and relationship 2 represents the exchange of information between a SSH client simulation and a SSH server simulation.

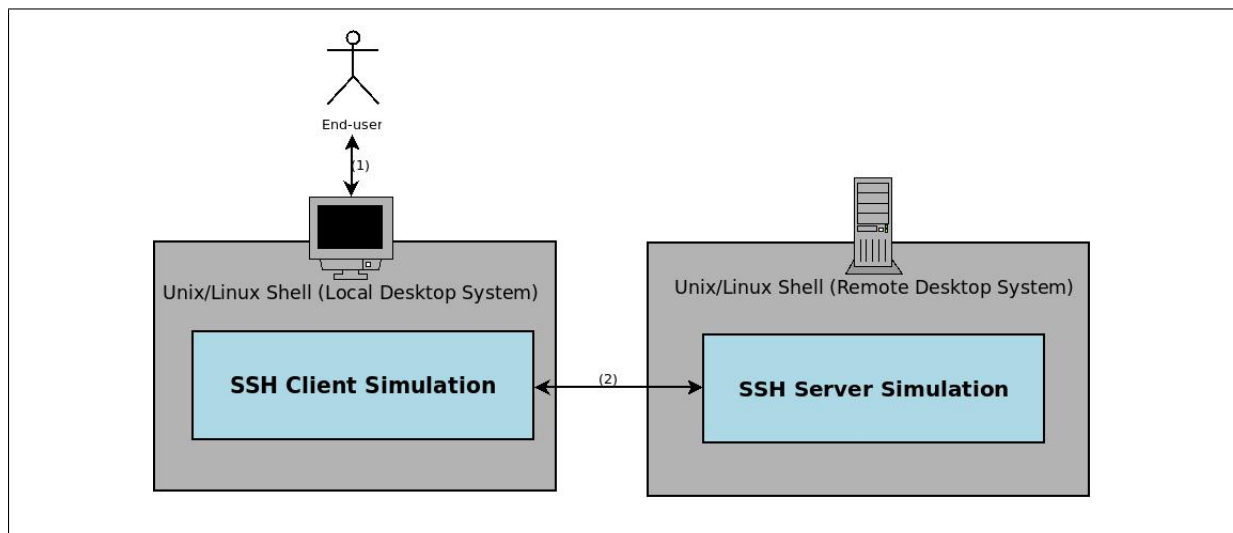


Figure 4.14: Linux Shell and SSH simulation

Both Ubuntu desktop simulation and Secure Shell protocol simulation rely on Kerberos system simulations to provide single sign-on. For example, end-users are authenticated by Kerberos system simulation to use Ubuntu desktop simulations, then they can proceed to access/use remote Ubuntu desktop simulations (via SSH simulations) using the same Kerberos single sign-on sessions (i.e. without requiring re-authentication).

## Ubuntu Desktop Simulation

Ubuntu desktop simulation provides basic functionalities such as Linux shell. The “Shell” is the Linux term for a user interface to the system. It lets end-users to communicate with the computer via the keyboard and the display (Newham & Rosenblatt, 2005). It allows end-users to interact with the Ubuntu desktop system (e.g. creating, modifying, deleting or executing files).

A Ubuntu desktop requires the end-users to be authenticated before accessing the Linux shell. It relies on Kerberos system simulation for authentication. It includes the following functionalities:

- **Connecting to Kerberos Server Simulation**

The Ubuntu desktop simulation uses network socket to connect with the Kerberos server simulation. The connection allows the Ubuntu desktop simulation to send ticket granting ticket requests to the Kerberos server simulation.

- **Present login prompt to the end-user**

When end-users access the Ubuntu desktop simulation for the first time, it presents a login prompt that asks for a username and a password. The login prompt asks for the end-user’s username first, then it asks for the end-user’s password. It will display the username as it been entered; it will not display the password.

- **Requesting and Storing Ticket Granting Ticket**

The simulation sends end-users’ requests for ticket granting tickets to the Kerberos server simulation. Section 4.3.2 describes the verification process for the credentials. It will receive ticket granting ticket from the Kerberos single sign-on system simulations if the end-users’ credentials were successfully verified.

- **Storing Ticket Granting Ticket**

The Ubuntu desktop simulation accepts the end-user’s access and presents it with a Linux shell simulation when it received the ticket granting ticket. The ticket granting ticket (TGT) is stored on the desktop simulation in the form of a text file.

The TGT can be accessed by SSH client simulation.

The Linux shell simulation only supports secure shell protocol. It provide the secure shell command “ssh”, which triggers Secure Shell (SSH) client simulation. End-users can access remote Ubuntu desktop simulations by executing the command “ssh” followed by the domain name of the remote Ubuntu desktop. For example, a remote Ubuntu desktop has the domain name “foreign.virtual.vm.” The SSH client simulation (on the end-user’s desktop) will attempt to communicate with the SSH server simulation on that Ubuntu desktop when the end-user executes “ssh foreign.virtual.vm”. The SSH connections use pre-defined domain names (e.g. home.virtual.vm, foreign.virtual.vm and third.virtual.vm). These are proof of concept systems, and new remote domains need to be manually added into the source code of the SSH simulations.

### **Secure Shell Protocol Simulation**

The research uses a Secure Shell Protocol simulation that allows end-users to access remote Ubuntu desktop simulations from their local Ubuntu desktop simulations. Secure Shell Protocol (SSH) is a protocol for secure remote login and other secure network services over an insecure network (Ylonen & Lonvick, 2006). SSH allows end-users to authenticate with Kerberos authentication system (MIT, 2008; Migeon, 2008).

The goal of a SSH simulation is to support Kerberos server simulation, Ubuntu desktop simulation and AIIP. The SSH simulation needs to support the communication protocol and Kerberos ticket that are used by Kerberos server simulation. It also needs to have two ends, a client and server. The SSH client and server can both run on the same Ubuntu desktop simulation; however, they usually run on different Ubuntu desktops. The SSH clients initiate connections by sending connection requests to the SSH servers. The SSH servers listen and accept connection requests from SSH clients. The SSH sessions are maintained by SSH clients and servers.

### **Secure Shell Client Simulation**

The SSH client simulation supports Kerberos server simulation and SSH server simulation.

It also uses network sockets for connecting to Kerberos single sign-on system simulations and SSH server simulations. It has the following functions:

1. Extracting ticket granting ticket information (TGT) from the TGT text file.

The ticket granting tickets that were created by the Kerberos server simulation are stored on the Ubuntu desktop simulation. The SSH client simulation reads the ticket files and extracts the content into a string. The string is stored in the SSH client simulation's memory.

2. Sending the ticket granting ticket information (with the request for service tickets) to the Kerberos server simulations.

The SSH client simulation connects and sends the ticket granting ticket strings to the Kerberos server simulations.

3. Receiving the service tickets from the Kerberos server simulations.

The SSH client accepts the service tickets that were sent from Kerberos server simulations. The service ticket creation process is described in section 4.3.2.

4. Sending the service ticket to the SSH server simulation.

The SSH client establish connection with SSH server simulation using network sockets. It then sends the service ticket to the SSH server simulation.

5. Receiving the result of service ticket verification.

The SSH client simulation receives the result of service ticket verification. It will Maintain the SSH session if the verification was successful. It will terminate the SSH session if the verification was unsuccessful.

6. Send the end-user's command requests to the SSH server simulation.

The SSH session allows the end-user to execute commands on the remote Ubuntu desktop simulation. The Linux shell simulation receives the commands from the end-user, then the SSH client simulation sends these commands to the SSH server simulation.

### 7. Receiving the result from the SSH server simulation.

The SSH client receives the result of executing the commands, and the Linux shell simulations displays them to the end-user.

## Secure Shell Server Simulation

The SSH server simulations run on Ubuntu desktop simulations. They listen to connection requests, and they can handle multiple connections at the same time. The SSH server simulations have the following functions:

- Requesting shared key from the Kerberos server simulation.

The SSH server simulation needs to be registered in Kerberos single sign-on system's service database in order to obtain the shared key. For example, a SSH server has been manually registered in the service database under the name "ssh" (Figure 4.15). It has the domain name "home.virtual.vm". It has the sample IP address 127.0.0.1.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<services>
  <service>
    <serviceName>ssh</serviceName>
    <domain>home.virtual.vm</domain>
    <ipAddress>127.0.0.1</ipAddress>
  </service>
</services>
```

Figure 4.15: Sample SSH server simulation Entry in the Service Database

The SSH server simulation connects to the Kerberos server simulation, then it sends its service name, domain name and IP address to the Kerberos server simulation. It will acquire a shared key from the Kerberos server simulation if it was registered in the service database. The shared key consists of a service name, a domain name, an IP address (i.e. sample IP address) and a hash key (Figure 4.16).

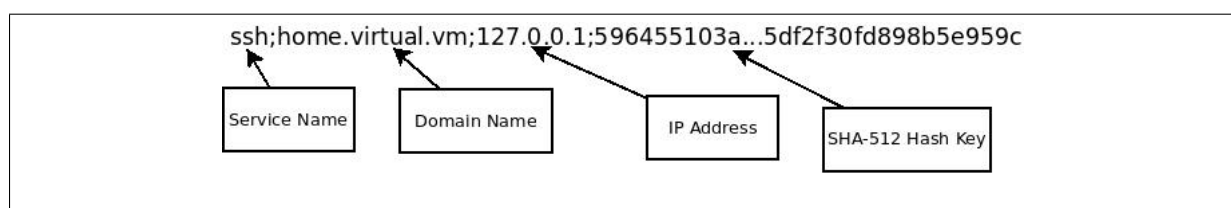


Figure 4.16: Sample Shared Key between SSH Server and Kerberos Server Simulation

- Verifying the end-user's service ticket.

The SSH client simulation sends the service ticket and the end-user's username to the SSH server simulation. The service ticket is appended to the username, and the two strings are separated by a semicolon (Figure 4.17). The SSH server simulation compares the hash key in the shared key with the hash key in the service ticket.

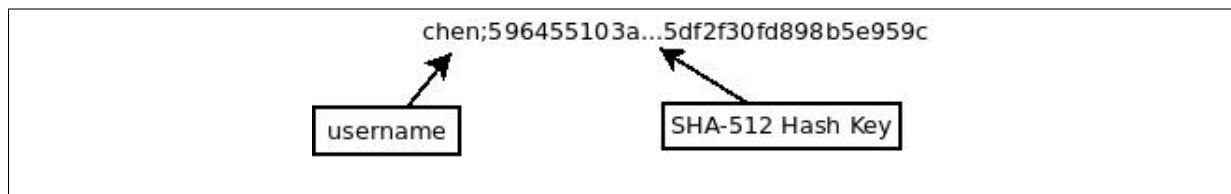


Figure 4.17: Sample Message that includes a Username and Service Ticket

- Maintaining the network socket to maintain the SSH session.

The end-user's service ticket is successfully verified if both hash keys match. The SSH server maintains the network socket that is connecting to the SSH client simulation. It receives the command requests from the SSH client simulation. The remote Ubuntu desktop simulation, which runs the SSH serve simulation, executes these commands. The SSH server simulations returns the results (of the executions) to the SSH client simulation.

- Closing the connection.

The SSH server simulation returns the result and closes the network socket if the service ticket was not verified. Closing the network socket results in the termination of SSH session.

### Exiting the SSH session

SSH allows end-users to access remote Ubuntu desktop simulations. SSH simulation runs on top of the Ubuntu desktop simulation, however, they are isolated sessions. The termination of SSH session will not terminate the (local or remote) Ubuntu desktop session.

Termination of SSH session will not remove the end-user's saved files on the remote Linux desktop simulation. Files and directories that were created in the SSH session are



saved on the remote Ubuntu desktop simulation. End-users can access the same files and directories in the next session.

The SSH client simulation will send a termination message to the remote SSH server simulation. The SSH server confirms the termination, and both sides will terminate the connection by closing their network sockets. The termination of SSH session ends the end-user's access to remote Ubuntu desktop and returns the end-user to its local Ubuntu desktop simulation.

#### **4.3.4 SAML based FSSO System Simulation (Identity Provider)**

SAML based federated single sign-on system simulation contains two components: web-based authentication system simulation (i.e. identity provider end) and web-based service/application simulation (i.e. service provider end). A network domain can have the web-based authentication system simulation, the web-based services/applications or both at the same time. This section presents the simulation of the web-based authentication system. Section 4.3.5 presents the simulation of the web-based services/applications.

The web-based authentication system simulation presents the end-users with a web-based authentication portal. The web portal currently requires end-users to enter their username and password (i.e. authentication). This research will address this problem using IDAWA in section 4.4.2. End-users won't need to enter their username and password for authentication in the new simplified federated single sign-on system.

##### **Create Identity Assertion**

The web-based authentication system simulation provides identity assertions if the end-users have been successfully authenticated. The identity assertions allow end-users to access web-based services/applications in the other domains without requiring re-authentication. The identity assertion message includes the following variables: username, role, domain name and authentication result.

Web cookies are used by real world SAML based federated single sign-on system

to transfer identity assertion message; therefore the simulations also use web cookies. The four variable, username, role, domain name and authentication result, are stored as web cookies. The web cookies are stored on the end-user's web browser. The web-based service/application simulations (i.e. service provider) in different domains can access the end-user's identity assertion variables via the web cookies.

### 4.3.5 SAML based FSSO Simulation (Service Provider)

The simulation of a web-based service/application (i.e. service provider) only presents successful/unsuccessfully federated single sign-on in this research. It verifies end-users' identity assertion messages and grants the access to the end-users if their identity assertion messages indicate that they are from domains within the federation. It then displays the end-users' authentication data (e.g. username, domain name, role).

#### Accessing Identity Assertion

The end-users' identity assertions contain four variables, "username", "domain name", "role" and "authentication result" (presented in section 4.3.4). They are stored as web cookies in the web browsers. They can be accessed by the web-based service/application simulation.

Web-based service/application simulation looks for the variable "authentication result". The "authentication result" is a boolean variable (i.e has the value "true" or "false"). A successfully authenticated end-user will have the "true" value under "authentication result".

## 4.4 New Simplified FSSO System Design

This section presents the design of the new simplified federated single sign-on system. It contains the designs of two major components: the "Authentication Infrastructure Integration Program" (AIIP) and the "Integration of Desktop Authentication and Web-based

Authentication” (IDAWA). The designs of these two components follow the specifications that were presented in the design environment (section 4.2).

AIIP facilitates federated single sign-on by bridging connections between end-users’ desktop systems (in one domain) and Kerberos single sign-on systems (in different network domains). It seamlessly acquires Kerberos tickets from Kerberos single sign-on systems in different network domains and sends these tickets to the end-users’ desktop systems, and end-users can access non-web based services in the different domains using the acquired Kerberos tickets.

IDAWA is an extension to the SAML based federated single sign-on system. It allows the web-based authentication systems (i.e. the identity provider end) to use Kerberos single sign-on tickets as means of authentication (subsection 4.4.2). It allows end-users who obtained Kerberos tickets to access web-based services without requiring re-authentication.

#### 4.4.1 Authentication Infrastructure Integration Program (AIIP)

AIIP contains two ends: the identity provider end ( $AIIP_i$ ) and the service provider end ( $AIIP_s$ ). A network domain can have a  $AIIP_i$ , a  $AIIP_s$  or both at the same time. The  $AIIP_i$  and the  $AIIP_s$  work together to bridge the communications between end-users’ desktop system (in one domain) and Kerberos single sign-on systems (in the other domains). They use the trust that were established between network domains within the federation.

The goal of the  $AIIP_i$  is to verify that end-users have obtained desktop authentication sessions (i.e. the end-users have been authenticated by Kerberos single sign-on systems). It presents itself as a service that supports Kerberos single sign-on, and it requests for end-users’ service tickets. It will create and send identity assertions to the  $AIIP_s$  after it successfully verified end-users’ service tickets.

The goal of the  $AIIP_s$  is to acquire Kerberos service ticket on behalf of the end-users in different domains. It will present itself as an end-user to the Kerberos single

sign-on systems. After it received identity assertions from the  $AIIP_i$ , it will request ticket granting ticket and service ticket using end-users' provisioned credentials. It then sends the service tickets to the  $AIIP_i$ , which will pass these service tickets to the end-users.

### Identity Provider End of AIIP

This section describes the functions of the identity provider end of "Authentication Infrastructure Integration Program" ( $AIIP_i$ ). The  $AIIP_i$  relies on Kerberos server simulations for authentication. It verifies end-users' service tickets (that were created by Kerberos server simulation), and it provides identity assertions for end-users who have valid Kerberos service tickets. It has the following functions:

1. Requesting shared key from Kerberos server simulation.
2. Verifying the end-users' service tickets.
3. Creating and sending the identity assertion messages.
4. Receiving the Kerberos service tickets from domain Beta and passing them to the end-users.

### Requesting Shared Key

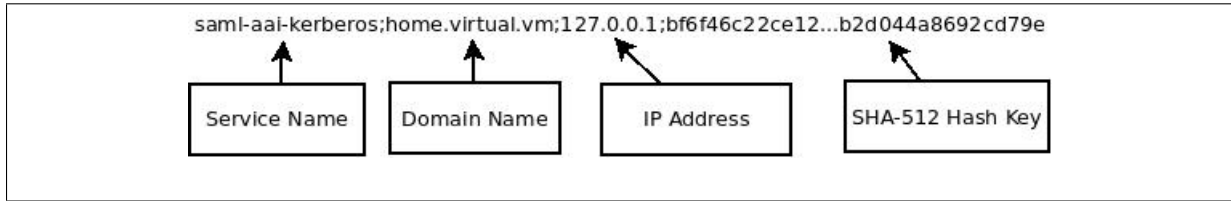
The  $AIIP_i$  presents itself as a service to the Kerberos single sign-on systems in a network domain. Like any services, the  $AIIP_i$  needs to be registered in Kerberos server simulation's service database. The  $AIIP_i$  is stored under "saml-aa-kerberos" in the service database (Figure 4.18). It has the sample domain name of "home.virtual.vm" and the sample IP address of "127.0.0.1". It has different domain names in different domains. For example, the  $AIIP_i$  in domain Beta has the domain name of "foreign.virtual.vm".

The  $AIIP_i$  requests a shared key from the Kerberos server simulation. It will receive a shared key when Kerberos single sign-on system simulation found its entry in the service database. The shared key consists of the  $AIIP_i$ 's service name, domain name and IP address and a hash key (Figure 4.19). The service name, domain name, IP address

```
<?xml version="1.0" encoding="iso-8859-1"?>
<services>
  <service>
    <serviceName>saml-aai-kerberos</serviceName>
    <domain>home.virtual.vm</domain>
    <ipAddress>127.0.0.1</ipAddress>
  </service>
</services>
```

Figure 4.18: Sample  $AIIP_i$  Entry in the Service Database

are merged into a single string (i.e. source string). The hash key is created by putting the source string through SHA-2 hash function.

Figure 4.19: Sample Shared Key of  $AIIP_i$ 

### Verifying Service Ticket

The  $AIIP_i$  needs to verify the end-users' service tickets before creating identity assertions for them. The verification process is the same as SSH server simulation (described in section 4.3.3): an end-user sends its username and service ticket to  $AIIP_i$ , and  $AIIP_i$  compares the hash key in the share key with the hash key in the service ticket.

By supporting the Kerberos single sign-on,  $AIIP_i$  can authenticate end-users using their Kerberos service ticket instead of their credentials. This is the first step of integrating end-users' desktop authentication session with non-web based federated single sign-on sessions. The next stage is creating and send identity assertion to  $AIIP_s$ .

### Creating and Sending Identity Assertion

The  $AIIP_i$  will send a response to the end-user if the service ticket was successfully verified. The response indicates that the end-user's service ticket has been successfully verified. The end-user then sends its username, role and domain name to  $AIIP_i$  upon receiving the response. These data are merged into a single string, and they are separated by semicolons (Figure 4.20). The string is used as the identity assertion message.  $AIIP_i$  then sends the identity assertion message to the  $AIIP_s$  in another domain.

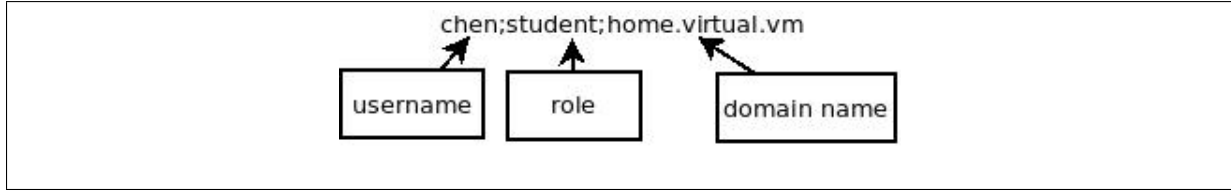


Figure 4.20: Sample Identity Assertion

This step connects the  $AIIP_i$  in one domain and the  $AIIP_s$  in another domain. By sending the identity assertion message, the bridge between two domains is established. The next step involves  $AIIP_s$  acquiring Kerberos service ticket from one domain and sending it to  $AIIP_i$  in another domain.

### Receiving and Passing the service tickets

$AIIP_s$  (in another domain) will reply with a service ticket after it receives the identity assertion message. This service ticket was created by the Kerberos server simulation in another domain. It allows end-users to access non-web based services that are in the same domain as  $AIIP_s$ .  $AIIP_i$  receives the service ticket and sends it to the end-user. The detail processes of acquiring service tickets are described in section 4.4.1.

This step comes after the  $AIIP_s$  acquiring a service ticket on behalf of the end-users. It completes the non-web based federated single sign-on process.  $AIIP_i$  passes the service ticket that was created in another domain to the end-user. The end-user can use the service ticket to access non-web based services in the domain that created the service ticket. The entire federated single sign-on process did not request the end-user to be authenticated.

### Service Provider end of AIIP

This section describes the design of  $AIIP_s$ . The main goals of  $AIIP_s$  are acquiring ticket granting ticket and service ticket on behalf of the end-users (in the other domains).

### Acquiring Ticket Granting Tickets

$AIIP_s$  uses network sockets for network connection. It listens on a specific network port (port 80). The network port can be changed to suite the firewall configuration. It will attempt to acquire ticket granting tickets for the end-users if the identity assertions

indicated that the correspondent end-users have been authenticated by  $AIIP_i$ .

$AIIP_s$  acquires ticket granting tickets on behalf of the end-users. It extracts the username and domain name from the identity assertion messages. It then searches for the correspondent identities in the end-user database. Once it finds the credentials of the end-users in the database, it will request of ticket granting tickets (from the Kerberos single sign-on system simulation in the same domain) using the end-users' credentials. It then proceed to request service tickets after it receives the ticket granting tickets from the Kerberos single sign-on system.

### **Acquiring Service Ticket**

The other function of  $AIIP_s$  is acquiring service tickets on behalf of the end-users. The  $AIIP_s$  sends the ticket granting tickets and requests for service tickets to Kerberos single sign-on system in the same domain. Once it receives the service ticket, it will pass it back to  $AIIP_i$ .

These two functionalities complete the connections between the Kerberos single sign-on sessions in separate domains. The acquired service tickets will be sent back to the  $AIIP_i$  and then passed to the end-users. The acquired service tickets allow the end-users from other domains to access non-web based services/applications (in the same domain that created the service tickets).

### **Modifying SSH clients**

The SSH client selects Kerberos single sign-on systems based on the domain name that were entered in the SSH command. For example, the SSH command “ssh home.virtual.vm” will trigger the SSH client to request service ticket from Kerberos single sign-on system in domain “home.virtual.vm”. This is suitable for Kerberos secure sharing (KSS) since KSS allows SSH clients to connect to Kerberos servers in different domains; however, this is not suitable for the AIIP.

The AIIP bridges the communications between end-users' desktop systems and Kerberos single sign-on systems. Kerberos single sign-on systems are not accessible from

outside their network domains; therefore, the SSH clients can not communicate with Kerberos single sign-on systems in the other domains, and it will not be able to request Kerberos service tickets.

An extension has been added to the SSH client to address this problem. The extension reads the domain names in the SSH command and decides whether they point to local domain or foreign domains. It will allow the SSH client to continue on its request (for service tickets) if the domain names point to local domain. It will direct the request for service tickets to  $AIIP_i$  in the local domain if the domain names point to foreign domains.

The extension currently needs to be added to the source code of SSH clients, and the SSH client needs to be recompiled and redeployed into the network domains. This process requires support from network domains and application creators. This problem can be addressed in the future by separating the extension from the SSH client and into a stand alone program, so that it will not require SSH client to be re-compiled and re-deployed. In addition, the stand along extension can provide the same functionalities for the other clients of non-web based services (e.g. NFS).

Another solution is  $AIIP_i$  can pretend to be Kerberos single sign-on systems from foreign domains. It seamlessly intercepts the ticket requests from SSH clients when end-users attempted to access SSH servers in foreign domains. It can then proceed to acquire Kerberos tickets from the actual Kerberos single sign-on systems in foreign domains and send the tickets back to the SSH clients. This approach does not require SSH clients (or any non-web based services/applications clients) to be modified; therefore, it will further simplify the deployment of the AIIP.

### **AIIP Design Analysis**

The benefits of AIIP (over Kerberos Secure Sharing) is that it does not expose Kerberos single sign-on systems to the outside world, and it does not require any changes to the Kerberos single sign-on systems (Figure 4.21). The  $AIIP_i$  presents itself as a service to



the Kerberos single sign-on systems in one domain, while the  $AIIP_s$  presents itself as an end-user to the Kerberos single sign-on systems in another domain. The Kerberos single sign-on systems will treat them as normal services and normal end-users, and the AIIP seamlessly connects the Kerberos single sign-on sessions in these domains by acquiring/transferring Kerberos service tickets from one domain to another domain. They allow end-users to use Kerberos service tickets (acquired from different domains) to access non-web based services in different domains.

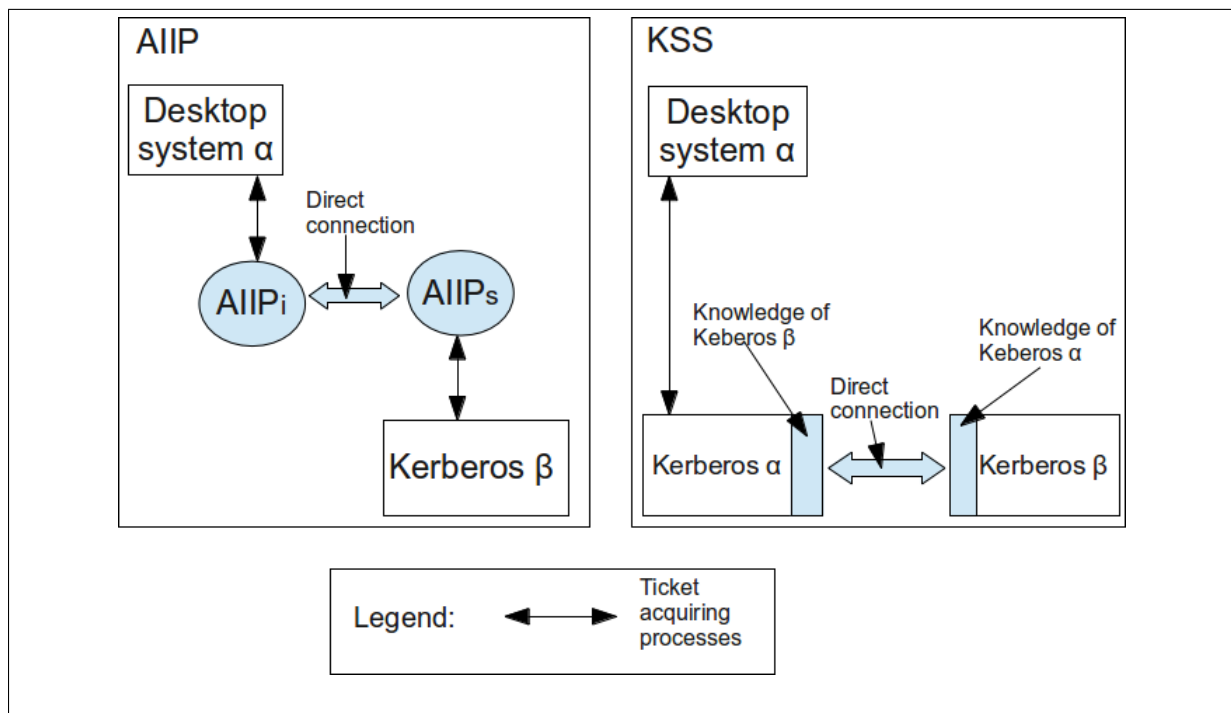


Figure 4.21: Side by side comparison of AIIP and KSS

Although the AIIP does not expose Kerberos single sign-on systems to the outside world, it needs to communicate with the AIIPs in other domains; therefore, AIIPs are exposed to the outside world. They can't be protected by security firewall; therefore, they are open to security problems such as man-in-the-middle attack and single point of failure.

The AIIPs only manage the federated single sign-on processes; therefore, they will not effect the local network authentications. In case of security compromise, network domains can shut down its AIIP to protect the local network. Shutting down the AIIP will stop end-users from accessing non-web based services in the other domains in a single

sign-on fashion; however, end-users can still access/use non-web based services in local network domain in a single sign-on fashion.

The AIIPs can use Transport Layer Security (TLS) (Dierks & Rescorla, 2008) to defend against a man-in-the-middle attack. It uses asymmetric cryptography for key exchange and symmetric encryption for confidentiality, and message authentication codes for message integrity. It can secure the communication between the two end points of AIIP (i.e.  $AIIP_i$  and  $AIIP_s$ ) during communication.

The AIIP handles all of the non-web based federated single sign-on processes. This may cause a single point of failure. Although AIIP does not affect the single sign-on processes in the local domains, end-users will not be able to access non-web based services in the foreign domains in a single sign-on fashion. Each domain should have a primary AIIP system and multiple secondary AIIPs in order to defend against single point of failure. If the primary AIIP has failed, the secondary AIIPs can take over the non-web based federated single sign-on processes.

#### **4.4.2 Integration of Desktop Authentication and Web-based Authentication (IDAWA)**

The Integration of Desktop Authentication and Web-based Authentication (IDAWA) is an extension to the web-based authentication system (i.e. identity provider end of SAML based federated single sign-on system). It aims to allow web-based authentication systems to use end-users' Kerberos tickets as means of authentication instead of end-users' credentials (i.e. username and password). It allows end-users (who have gained access to desktop systems) to access web-based services without requiring re-authentication.

End-users who have been authenticated by Kerberos single sign-on system will acquire Kerberos tickets. Since end-users need to gain access to desktops before accessing a web-browser, it's safe to assume that end-users were already authenticated by Kerberos server (and acquired ticket granting tickets) when they are accessing a web-browser.

##### **Requesting Shared Key**

The IDAWA are added into the source code of web-based authentication system, and it executes as part of the web-based authentication system. It uses shared keys (created by Kerberos single sign-on systems) to verify end-user's service tickets; therefore, it needs to be registered in the Kerberos service database (Figure 4.22). The web-based authentication systems in different domains will have different data. For example, it has domain name "home.virtual.vm" in domain Alpha, and it will have domain name "foreign.virtual.vm" in domain Beta.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<services>
  <service>
    <serviceName>idp</serviceName>
    <domain>home.virtual.vm</domain>
    <ipAddress>127.0.0.1</ipAddress>
  </service>
</services>
```

Figure 4.22: Sample Web-based Authentication System Simulation Entry

The IDAWA connects to the Kerberos server simulation and sends its service name, domain name and IP address to the Kerberos server simulation. It should obtain a shared key from the Kerberos server simulation if it has been registered in the service database. The shared key consists of the service name, domain name, IP address (i.e. sample IP address) and SHA-2 hash key (Figure 4.23).

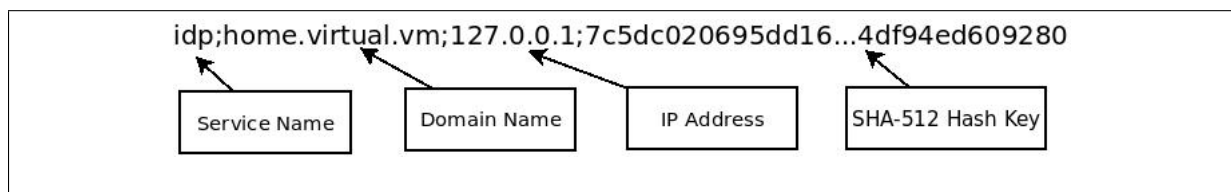


Figure 4.23: Sample Shared Key between Web-based Authentication System Simulation and Kerberos Server Simulation

The IDAWA acquires shared keys from Kerberos single sign-on systems. The shared keys allows the web-based authentication systems to verify end-users' service tickets. This is the first step for web-based authentication systems to support Kerberos tickets. The next step is acquire end-users' service tickets.

### Requesting Service Ticket

The IDAWA creates a file uploading system to accept end-users' TGTs (Figure 4.24). It

passes the contents of TGTs to the Kerberos single sign-on systems once they have been uploaded by the end-users. It will receive the service ticket from the Kerberos server simulation after it verifies TGTs.

The IDAWA verifies the service ticket by comparing the SHA-2 hash key in the service ticket with the SHA-2 hash key in the shared key. It will create identity assertions for the end-users if both keys match.

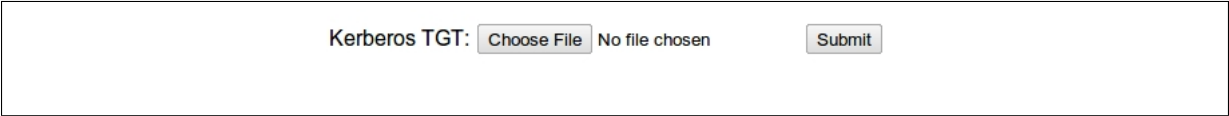


Figure 4.24: Kerberos Ticket Granting Ticket Portal

### 4.4.3 Simplified Federated Single Sign-on System Life-cycles

This section covers four scenarios that may occur to an end-user. Life-cycle 1 shows an end-user (in domain Alpha) attempts to access/use a non-web based service/application in domain Beta. Life-cycle 2 is a continuation of life-cycle 1; it shows the same end-user in domain Alpha attempts to access/use a web-based service/application in domain Gamma after life-cycle 1. Life-cycle 3 shows an end-users in domain Beta attempts to access/use both web-based and non-web based services/applications in domain Alpha.

A federation (i.e. a circle of trust) has been established between there three domains (Alpha, Beta and Gamma). The federation is expressed in the form of security policies (described in section 4.2.2). The policies indicate that end-users in each domain can access both web-based and non-web based services/applications in each other's domains.

The life-cycles assume that encryption methods have been applied during network communications and file storage. For example, the communications between the two ends of AIIP are encrypted using TLS. Communications between Kerberos single sign-on (KSSO) systems and AIIP are encrypted using AES (which is supported by Kerberos). Kerberos tickets (ticket granting tickets and services tickets) are encrypted during transfer and storage (also using AES).

**Life-cycle 1: end-user accesses non-web base application**

Authentication Infrastructure Integration Program (AIIP) includes two ends:  $AIIP_i$  and  $AIIP_s$ . The  $AIIP_i$  resides in domain Alpha, and  $AIIP_s$  resides in domain Beta in life-cycle 1. Both  $AIIP_i$  and  $AIIP_s$  rely on the federation that was established between domain Alpha and Beta.

The  $AIIP_i$  and the  $AIIP_s$  handle the communications between an end-user's desktop system in domain Alpha and the Kerberos single sign-on (KSSO) system in domain Beta. They facilitate federated single sign-on by acquiring Kerberos ticket for end-users (from domain Alpha) in domain Beta (Figure 4.25). The end-user has an input and output relationship with the desktop system. They exchange information such as credentials, authentication results, application requests and applications request results. The full life-cycle can be viewed in appendix C.1; this section only shows the steps that involve the new simplified federated single sign-on system.

9  $SSH_c$  extracts  $ST_\alpha$  from the desktop system and sends it to  $AIIP_i$ .

10  $AIIP_i$  will create an identity assertion message for the end-user and send it to  $AIIP_s$  if  $ST_\alpha$  was successfully verified.

Step 9 and step 10 show the AIIP (in domain alpha) initialises the FSSO processes in the new simplified FSSO system. These steps are the primary difference between Kerberos secure sharing (KSS) and the new simplified FSSO system. The Kerberos single sign-on system (in domain Alpha) only manages local single sign-on and is not aware of the FSSO transactions; therefore, it is not exposed to the outside work in the new simplified FSSO system.

11  $AIIP_s$  accepts the end-user's identity assertion based on the federation's policies.

It creates a database query that searches for end-user's provisioned identity ( $I_\beta$ ) in domain Beta. It sends the query to  $D_\beta$ .

12  $D_\beta$  executes the query and returns  $I_\beta$  to  $AIIP_s$ .

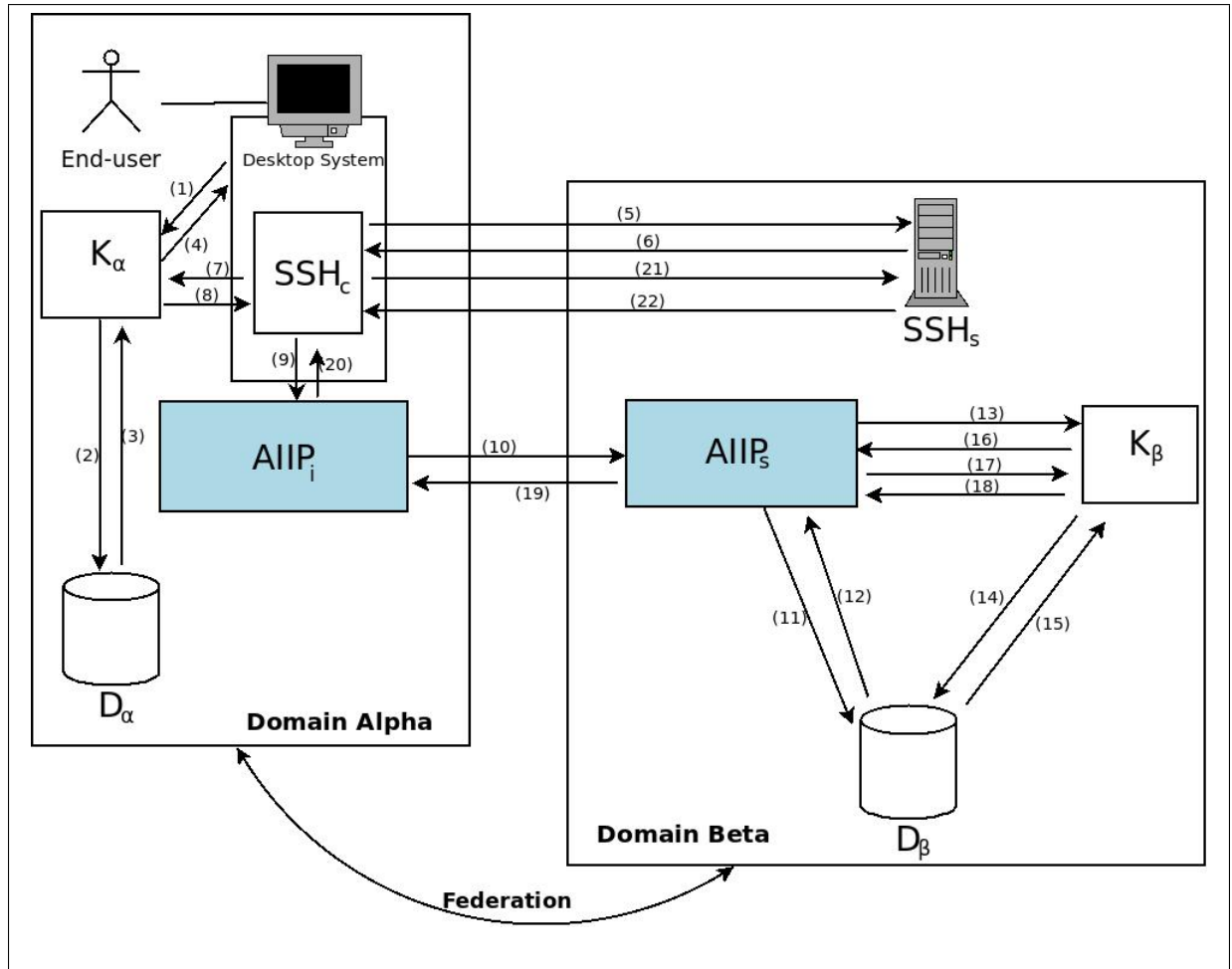


Figure 4.25: A New Simplified Federated Single Sign-on Life-cycle 1

- 13  $AIIP_s$  uses the  $I_\beta$  to authenticate to the KSSO system in domain Beta ( $K_\beta$ ) on behalf of the end-user.
- 14  $K_\beta$  sends a search query to  $D_\beta$ .
- 15  $D_\beta$  finds the credentials and return the result to  $K_\beta$  since  $I_\beta$  exists in  $D_\beta$ .
- 16  $K_\beta$  creates and sends a ticket granting ticket ( $TGT_\beta$ ) to  $AIIP_s$ .  $TGT_\beta$  will be stored in  $AIIP_s$ .
- 17  $AIIP_s$  sends  $TGT_\beta$  to  $K_\beta$  and requests service ticket ( $ST_\beta$ ) for  $SSH_s$  on behalf of the end-user.
- 18  $K_\beta$  verifies  $TGT_\beta$  and sends  $ST_\beta$  to  $AIIP_s$ .

Step 11 to step 18 shows that the AIIP (in domain Beta) accepts the FSSO request

from domain Alpha, and it acquires a Kerberos single sign-on ticket (in domain Beta) for the end-user (in domain Alpha). The ticket is sent from domain Beta to domain Alpha by the AIIP in domain Beta. The Kerberos single sign-on system in domain Beta is also not aware of the FSSO transactions; therefore, it isn't exposed to the outside world as well.

19  $AIIP_s$  sends  $ST_\beta$  to  $AIIP_i$ .

20  $AIIP_i$  sends  $ST_\beta$  to  $SSH_c$ .

Step 19 and step 20 show that the Kerberos ticket (created in domains Beta) is sent to domain Alpha, and it is then passed to the end-user. This is the last stage of the federated single sign-on. The end-user (in domain Alpha) can then use the Kerberos ticket to access the SSH server in domain Beta. The entire FSSO process did not require authentication of the end-user, and they are seamless to the end-user.

The entire non-web based FSSO process didn't expose KSSO systems (in both domains Alpha and Beta) to the outside world. The  $AIIP_i$  presents itself as a service to the KSSO system domain Alpha, while the  $AIIP_s$  presents itself as an end-user to the KSSO system in domain Beta. Neither KSSO system was modified, and neither one aware that they were part of federated single sign-on process.

### **Life-cycle 2: end-user accesses web-based application**

Integration of Desktop Authentication and Web-based Authentication (IDAWA) is another component of the new simplified federated single sign-on system. IDAWA is an extension to web-based authentication system ( $WBAS_\alpha$ ). It allows  $WBAS_\alpha$  to accept and verify Kerberos tickets (Figure 4.26).

Life-cycle 2 is a continuation of life-cycle 1. The end-user already gained access to a desktop system in domain Alpha and a non-web based services in domain Beta. He/she will attempt to access a web-based service/application in domain Gamma ( $WSA_\gamma$ ). The user agent is a web browser ( $WB_\alpha$ ) that runs on the desktop.

The relationship between the end-user and desktop is the same as figure 4.25. They exchange information such as credentials, authentication results, application requests and applications request results. The entire life-cycle can be viewed in appendix C.2. This section only presents the steps that are relate to IDAWA.

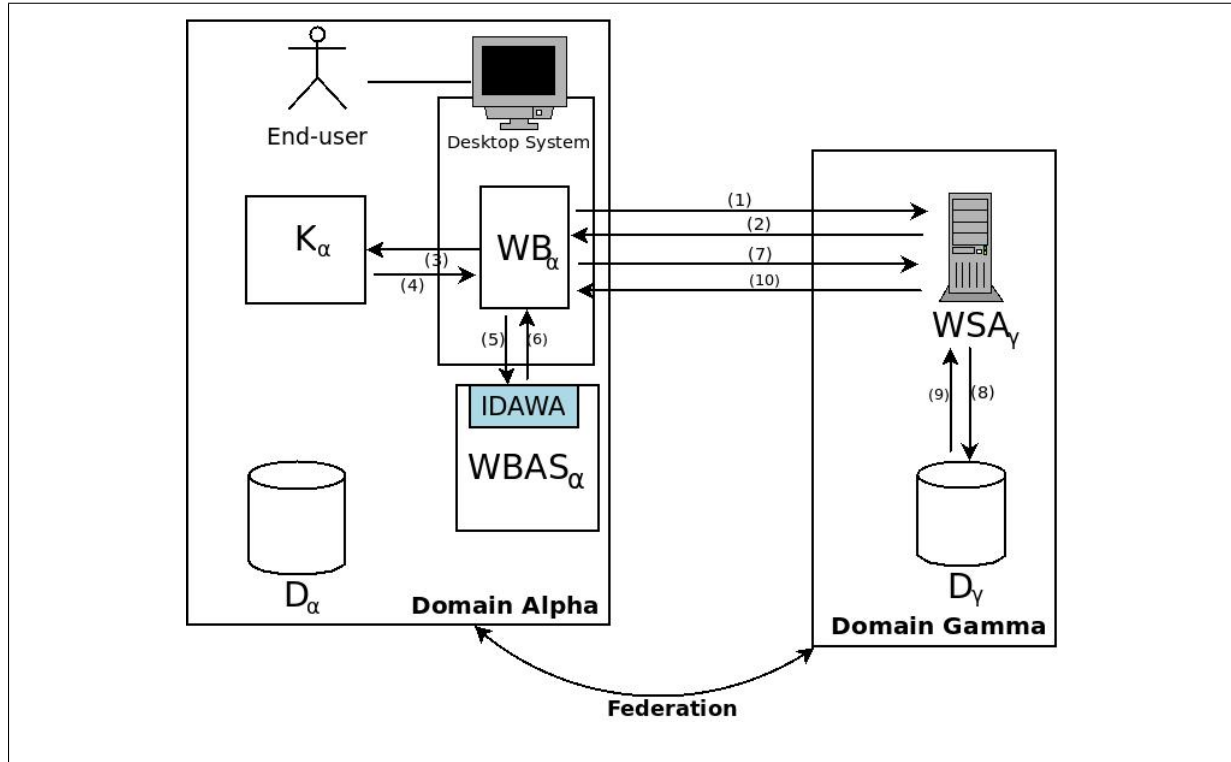


Figure 4.26: A New Simplified Federated Single Sign-on Life-cycle 2

5  $WB_\alpha$  sends the ST to  $WBAS_\alpha$ . IDAWA (in the  $WBAS_\alpha$ ) accepts the ST.

6 After IDAWA successfully verifies the ST,  $WBAS_\alpha$ , it then creates an identity assertion message and passes it back to  $WB_\alpha$ .

Step 5 and step 6 are the most important steps in this life-cycle. They show that the IDAWA uses the Kerberos ticket (from the end-users) as a means for authentication, and the end-user doesn't need to re-enter its credential (e.g. username and password). This enables the web-based authentication system to create identity assertion message without re-authenticating the end-user.

Life-cycle 2 shows that the IDAWA can be used in conjunction with the AIIP. The end-user, who has already gained access to a non-web based service/application



in domain Beta, can gain access to a web-based service/application in domain Gamma without requiring re-authentication. The same Kerberos ticket granting ticket (gained by desktop authentication) was used for both web-based and non-web based FSSO.

Although life-cycle 1 and life-cycle 2 only show three domains (Alpha, Beta and Gamma), each FSSO process only involved two domains (the FSSO process in life-cycle 1 involves domain Alpha and domain Beta, and the FSSO process in life-cycle 2 involves domain Alpha and domain Gamma). The numbers of domains will not affect each FSSO process; therefore, the new simplified FSSO system is not restricted in three domains.

### **Life-cycle 3: an end-user in domain Beta access domain Alpha**

This life-cycle shows a reverse of domains. An end-user in domain Beta attempts to access any computer services/application in domain Alpha in this life-cycle. He/she then attempts to access a web-based service/application ( $WSA_\alpha$ ) using a web browser ( $WB_\beta$ ) on the desktop (Figure 4.27). The same end-user then attempts to access a remote SSH server ( $SSH_s$ ) in domain Alpha using SSH client ( $SSH_c$ ) on the same desktop system. The entire life-cycle can be viewed in appendix C.3.

Steps 1 to 4 show an end-user enter its credentials into the desktop login prompt and gain access to the desktop system in domain Beta. This is the first (and only) authentication process that the end-user has to go through. A Kerberos ticket is created for the end-user and stored on the desktop system.

Steps 5 to 14 show the same end-user attempts to access a web-based service/application in domain Alpha. The end-user uses its Kerberos ticket (that was created during desktop authentication) as mean for web-based authentication. The steps are the same as life-cycle 2: the IDAWA accepts and verifies the end-user's Kerberos ticket in domain Beta, it creates identity assertion message to the web-based service/application in domain Alpha, and the web-based service/application grants access to the end-user.

Step 14 to 32 show the same end-user attempts to access a non-web based service/application (i.e. SSH server) in domain Alpha. This steps are the same as life-cycle

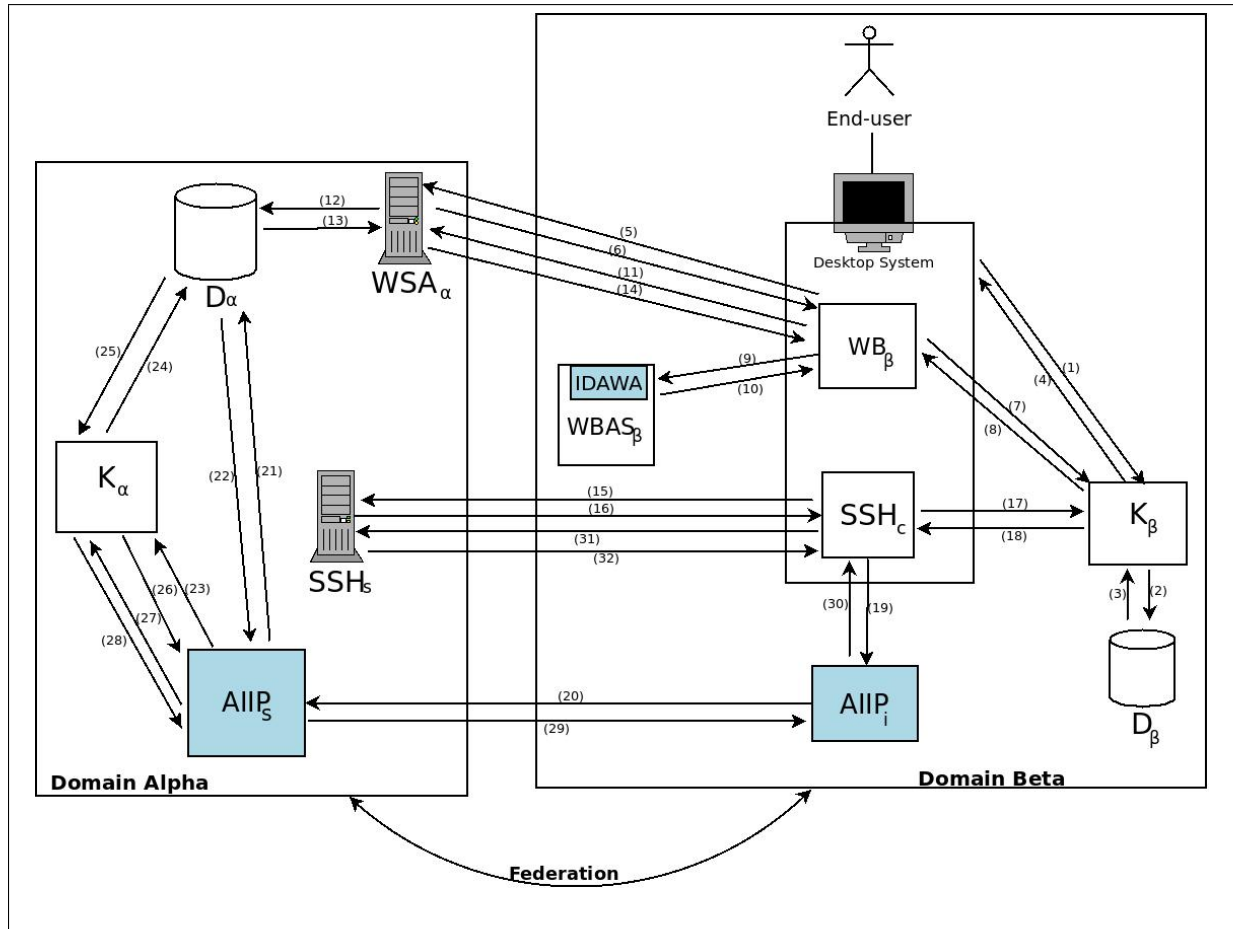


Figure 4.27: Life-cycle 3: an end-user in domain Beta access diverse service/applications domain Alpha

1: the end-user uses the Kerberos ticket (i.e. the same that was created during desktop authentication) as a means for authentication to AIIP in domain Beta. The AIIP (in domain Beta) accepts and verifies the end-user's Kerberos ticket and creates/sends an identity assertion to the AIIP in domain Alpha. The AIIP in domain Alpha replies with a Kerberos ticket (that was created in domain Alpha), and the new Kerberos ticket allows the end-user (in domain Beta) to access the non-web based service/application (in domain Alpha) without requiring re-authentication.

This life-cycle shows an end-user (from domain Beta) accessing both web-based and non-web based services/applications in domain Alpha. Both AIIP and IDAWA share the same desktop session, and they used the same Kerberos ticket for authenticating end-users. The new simplified FSSO system (i.e. the combination of AIIP and IDAWA) allows end-users (from any domain) to access desktop systems, web-based services/applications

and non-web based services/applications (in different domains within the federation) using one authentication process.

## 4.5 Summary

The design of the new simplified federated single sign-on system attempts to address the limitations of existing federated single sign-on systems. By incorporating the authentication systems of desktop systems, web-based services/applications and non-web based services/applications, this new system allows end-users to access desktop systems, web-based services/applications and non-web based services/applications using one authentication process. It achieves it with two major components: the “Authentication Infrastructure Integration Program” (AIIP) and the “Integration of Desktop Authentication and Web-based Authentication” (IDAWA). The AIIP supports desktop systems and non-web based services/applications. Although AIIP provides the same single sign-on as Kerbero secure sharing (KSS), it improves upon KSS by acquiring Kerbeors tickets from Kerberos single sign-on systems (in different network domains) without establishing trust between these Kerberos single sign-on systems. This allows AIIPs to be deployed into network domains without modifying their existing Kerberos single sign-on systems. The IDAWA supports desktop systems and web-based services/applications; it is an extension to the web-based authentication systems. It uses Kerberos tickets as means of web-based authentication instead of end-users’ credentials (e.g. username and password). The new simplified federated single sign-on system uses a combination of AIIP and IDAWA to provide single sign-on.

# Chapter 5

## Proof of Concept Implementation

### 5.1 Introduction

This chapter presents the proof of concept implementation of the new simplified federated single sign-on (FSSO) system. The implementation is divided into two stages. The first stage is the implementation of the design environment. The second stage is the implementation of the new simplified FSSO system.

The design environment includes the implementation of the simulations of Kerberos single sign-on system, SAML based federated single sign-on system, web-based services/applications, Ubuntu desktop and Secure Shell protocol (SSH). The simulations are strip down version of the real world systems. They serve as foundations for implementing the new simplified federated single sign-on system. They provide necessary functionalities such as desktop authentication, web-based authentication, local network single sign-on, web-based federated single sign-on. They assume that encryption methods have been applied during network communication and ticket storage (described in section 4.2).

The new simplified FSSO system includes the implementations of the “Authentication Infrastructure Integration Program” (AIIP) and the “Integration of Desktop Authentication and Web-based Authentication” (IDAWA). AIIP bridges the connections between

end-users' desktop system in one domain and Kerberos single sign-on systems in the other domains; it achieves it by acquiring Kerberos tickets in different domains for end-users. IDAWA is an extension to web-based authentication system, and it uses Kerberos tickets as means of authentication.

A federation (i.e. a circle of trust) has been established between network domains (e.g. domain Alpha, domain Beta and domain Gamma). The federation is expressed in the form of federation policies. The main policy that was introduced in the design environment was: "End-users from any domain can access any services/applications in the other domains." This policy is followed by the SAML based federated single sign-on system, and "Authentication Infrastructure Integration Program" (AIIP).

Section 5.2 presents the technical overview. It includes the development environment (e.g. computer software and programming language) and system configuration (e.g. hardware and software configuration). Section 5.3 presents the implementations of the design environment. Section 5.4 presents the implementations and testing of the new simplified federated single sign-on system. Section 5.6 summaries the chapter.

## 5.2 Technical Overview

### 5.2.1 Development Environment and Programming Languages

The new simplified federated single sign-on system and its design environment were developed on the Ubuntu 10.04.4 LTS Server and Xubuntu 12.04 LTS platform. The programming languages (that are used in the implementations) are Python, PHP and AJAX. Python 2.6 was used to develop AIIP, the simulation of Kerberos single sign-on, the simulation of Ubuntu desktop and the simulation of SSH. PHP 5.3.10 and AJAX were used to develop IDAWA and the simulation of SAML based federated single sign-on system.

AIIP and the simulations of Kerberos single sign-on system, Ubuntu desktop and SSH involve communications between multiple applications. They require the source code

to be clean and easy to debug. Python uses a clean and easy to read code style; therefore, it's suitable for programming and debugging complex applications.

IDAWA and the simulation of SAML based federated single sign-on system are small but dynamic web applications. These reasons promoted the use of PHP and AJAX. PHP is a server-side HTML embedded scripting language. It is suitable to create small scale web applications. AJAX creates asynchronous web applications. It can change contents on a web page without refreshing the web browser.

## 5.3 Implementation of Design Environment

This section presents the implementations of Kerberos single sign-on system simulation, Ubuntu desktop system simulation, SSH simulation and SAML based federated single sign-on system simulation. The implementations follow the design environment described in chapter 4. Kerberos single sign-on system simulations provides authentication for Ubuntu desktop system simulations and SSH simulations. Web-based authentication system (i.e. identity provider end of the SAML based federated single sign-on system) authenticates end-users and provides identity assertions. Web-based services/application (i.e. service provider end of the SAML based federated single sign-on system) only displays successfully federated single sign-on; it does not have any other functions.

The communication protocol that is used by the simulation is implemented using Python sockets. It's a independent communication protocol that ensures the different simulations (e.g. Kerberos single sign-on, Ubuntu desktop, SSH protocol and AIIP) can communicate with each other. It does not interoperate with real world Kerberos single sign-on systems.

### 5.3.1 Kerberos Single Sign-on System

The goal of the Kerberos single sign-on system simulations is to support single sign-on in a single domain. Each system is isolated from the outside world, and it has the following

functionalities that are the same as the real world system: creating system daemon that listing on a specific network port (i.e. port 88) for external connections, receiving requests (e.g. ticket granting ticket requests, service ticket requests and share key requests) and replying to these requests.

The simulation of Kerberos single sign-on system creates human readable Kerberos tickets and shared keys (e.g. ticket granting ticket, service ticket and shared keys). They are stored as text files. They are easier to manage than real world Kerberos tickets since they can be read and edited by text editors instead of real world Kerberos tools.

### Create System Daemon

The socket type of Kerberos single sign-on system simulation is defined using `AF_INET` and `SOCK_STREAM`, and the socket is bind to the machine's IP address (i.e. `LOCAL_IP`) and network port (i.e. `LOCAL_PORT`) (Figure 5.1) . The `AF_INET` defines the socket as an Internet socket, the `SOCK_STREAM` defines the connection as TCP connection, the `LOCAL_IP` is set by the DHCPD server: 169.168.1.59, and the `LOCAL PORT` is set to 88. Port 88 is also used by the real world Kerberos single sign-on system.

```
s = socket.socket (socket.AF_INET, socket.SOCK_STREAM)
s.bind((LOCAL_IP, LOCAL_PORT))
s.listen(5)
```

Figure 5.1: Start Kerberos Server Simulation

This Kerberos system daemon requires `class ClientThread` to manage multiple clients. Each connection (from an end-user) will be managed by a thread and each thread provides Kerberos functions such as creating ticket granting tickets, verifying ticket granting tickets and creating service tickets (Figure 5.2).

The thread determines a client's requests by examine the messages that were sent by the client (using `message = self.conn.recv(1024)`). The messages are long strings that contain multiple short strings. Short strings are separated by semicolons (in the long string). The first short string (in a long string) contains the type of request. The thread determines a client's request by comparing the first short string with three values:

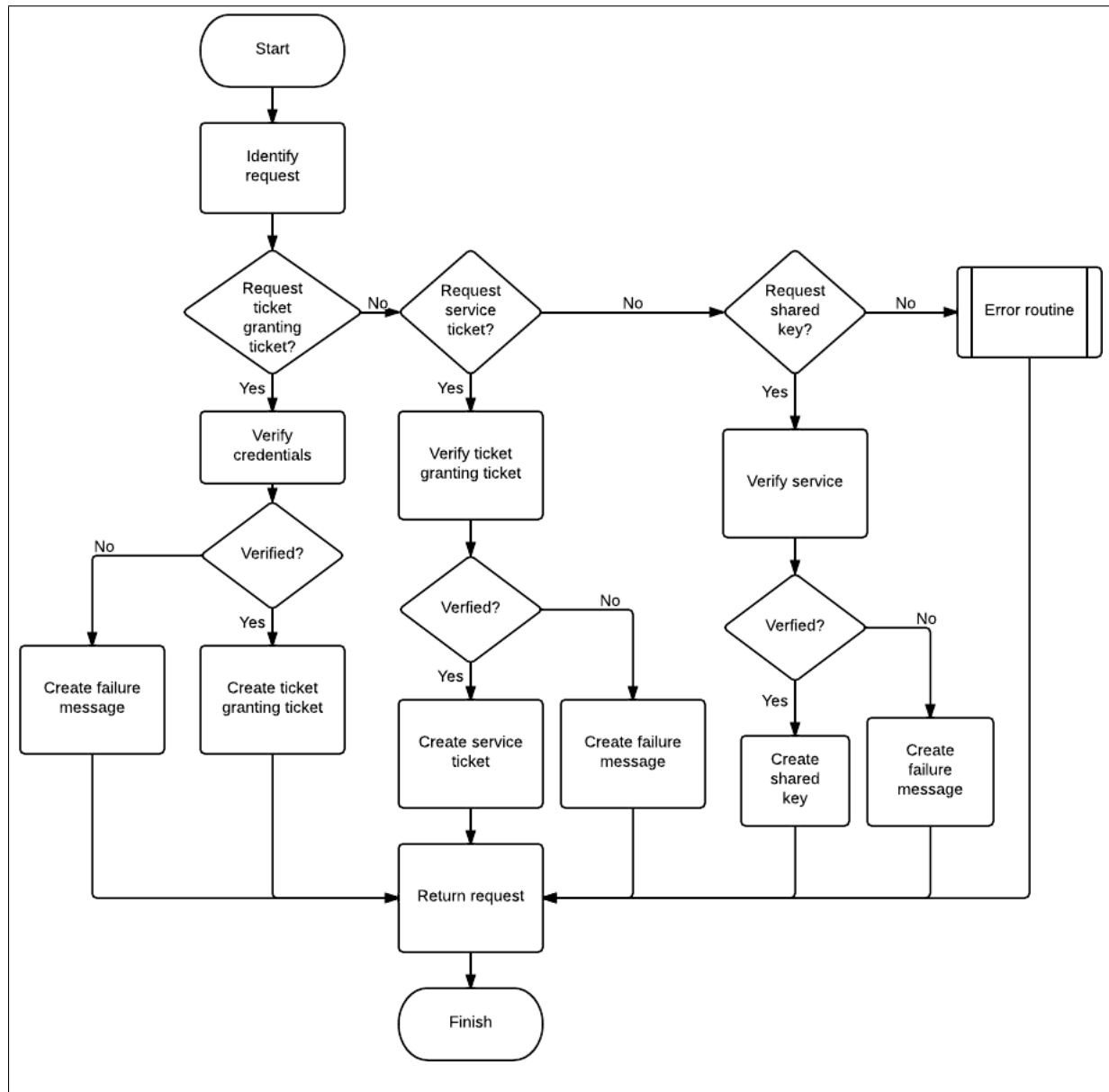


Figure 5.2: Kerberos single sign-on system thread flowchart



`req_tgt`, `req_sk` and `req_st`.

- `req_tgt`: The client is requesting a ticket granting ticket. The message also contains a set of credentials (including username and password).
- `req_sk`: The client obtains a shared key. The message also contains service information (including service name and domain name).
- `req_st`: The client is requesting a service ticket. The message also contains ticket granting ticket (including username, password, role, authentication result, domain name, ticket creation date, expire date and SHA-512 hash key).

### Creating Ticket Granting Ticket

The simulation of Kerberos single sign-on system will execute two functions once it determined a client is requesting a ticket granting ticket:

- `auth_user(self, username, password)`. It authenticates the end-users' identities.
- `create_tgt(self)`. It creates ticket granting tickets for authenticated end-users.

It extracts the username and password and passes them into function `auth_user(self, username, password)`. The function travels from the root of the end-user database (i.e. `userdatabase.xml`) to the child nodes and searches for matching credentials (Figure 5.3). It uses if statements to search for values that match end-user's credentials (e.g. `username` and `password`).

DOM was originally used as interface to XML files (e.g. `userdatabase.xml`). It was later changed to `ElementTree` since `ElementTree` was designed to support Python's coding style. The implementation was changed to use `ElementTree` to provide a easy to read source code.

The Kerberos system simulation uses function, `create_tgt(self)`, to create ticket granting tickets. It uses the end-user's username, password, role, authenticate result,

```

def auth_user(self, username, password):
    tree = xml.parse("userdatabase.xml")    # Parse the user database
    doc = tree.getroot() # Get the root node

    # Check the username and password
    for user in doc:
        if user[0].text == username and user[1].text == password:
            self.user = user
            return True
    return False

```

Figure 5.3: Searching for a Matching Credentials

```

# Create Ticket Granting Ticket
def create_tgt(self):
    ticket_info = ''
    issue_time = datetime.datetime.now()
    expire_time = datetime.datetime.now() + datetime.timedelta(days=1)

    for elements in self.user:
        ticket_info += elements.text
        ticket_info += ';'

    ticket_info += issue_time.strftime("%Y-%m-%d %H:%M")
    ticket_info += ';'
    ticket_info += expire_time.strftime("%Y-%m-%d %H:%M")
    ticket_info += ';'

    hash = hashlib.sha512(ticket_info) # Hash the shared key informations
    hash_hex = hash.hexdigest()

    ticket_info += str(hash_hex)

    f = open(self.user[0].text, 'w') # Keep recodes of authenticated users
    f.write(ticket_info)
    f.close

    return ticket_info

```

Figure 5.4: Creating Ticket Granting Tickets

domain name, ticket creation date and ticket expire date to create a SHA-512 hash string (Figure 5.4).

The hash function is part of the Python's `hashlib` library. The hash string is appended to the end of the presented data. The result is the ticket granting ticket (i.e. `tgt_message`); it's then sent to the client (using `self.conn.send (tgt_message)`). A copy of the ticket granting ticket is also saved on the Kerberos system simulation.

## Creating Shared Key

Kerberos system simulation extracts the two variables (service name and domain name) from the request message. It then passes these two variables into function `auth_service(self, service_name, service_domain_name)` to determine whether or

```

# Authenticate the service
def auth_service(self, service_name, service_domain_name):
    tree = xml.parse("servicedatabase.xml") # Parse the service database
    doc = tree.getroot() # Get the root node

    # Compare the service name and domain name against the database
    for service in doc:
        if service[0].text == service_name and \
            service[1].text == service_domain_name:
            self.service = service
            return True
    return False

```

Figure 5.5: Verifying Services Data

```

# Create Share Key
def create_share_key(self):
    share_key = ''

    for elements in self.service:
        share_key += elements.text
        share_key += ';'

    # Hash the shared key informations
    hash = hashlib.sha512(share_key)
    hash_hex = hash.hexdigest()

    share_key += str(hash_hex)
    return share_key

```

Figure 5.6: Creating Shared Key

not the service has been registered in the service database. It will create a shared key for the service if a matching pair was found in the service database. The function searches through the service database to find a matching pair of service name and domain name (Figure 5.5).

The function `create_share_key(self)` uses a similar method to that used to create ticket granting ticket in creating the shared key (Figure 5.6). The shared key includes the service name, domain name, IP address and hash key. The hash key is a SHA-512 hash key that was created from the service's service name, domain name and IP address. The shared key (i.e. `sk_message`) is then manually input into the service (using `self.conn.send (sk_message)`).

## Creating Service Ticket

The Kerberos system simulation verifies a client's (i.e. end-user's) ticket granting ticket and creates a service ticket for the end-user. It uses function `auth_tgt(tgt_info)` to

```

# Verify Ticket Granting Ticket
def auth_tgt(self, tgt_info):
    f = open(str(tgt_info[0]), 'r')

    if f == None:
        print 'User not found'
        return False
    else:
        key_info = f.readline()
        f.close()

        key_info_list = key_info.split(';', 7)

        if tgt_info[0] == key_info_list[0] and \
            tgt_info[4] == key_info_list[4] and \
            tgt_info[5] < key_info_list[6] and \
            tgt_info[7] == key_info_list[7]:
            return True
        else:
            return False

```

Figure 5.7: Verifying Ticket Granting Ticket

```

# Create Service Ticket
def create_st(self, service_name):
    tree = xml.parse("servicedatabase.xml") # Parse the service database
    doc = tree.getroot() # Get the root node
    st_info = ''

    # Compare the service name and domain name against the database
    for service in doc:
        if service[0].text == service_name:
            self.service = service
            break

    for elements in self.service:
        st_info += elements.text
        st_info += ';'

    hash = hashlib.sha512(st_info) # Hash the shared key informations
    hash_hex = hash.hexdigest()

    return str(hash_hex)

```

Figure 5.8: Creating Service Ticket

verify the end-user's ticket granting ticket (i.e. `tgt_info`).

The function uses a build-in function (i.e. `open("filename", permission)`) of Python to open the saved ticket granting ticket (i.e. `f = open(str(tgt_info[0]), 'r')`). It then compares the username and hash key of the end-user's ticket granting ticket with the saved copy. It also ensures the ticket has not expired (Figure 5.7).

The Kerberos system simulation creates service tickets using the services' service name (e.g. "ssh"), domain name, IP address and a SHA-512 hash key based on these variables (Figure 5.8). It then sends the service tickets to the end-users.

## Kerberos Single Sign-on System Summary

The simulation of Kerberos single sign-on system contains functionalities that can create ticket granting ticket, services tickets and shared keys. It only supports single sign-on in a single domain, and it does not have functionalities to communication with other Kerberos single sign-on systems in different domains. Its communication protocol can be supported by any program that supports a network socket, and it creates human readable ticket/keys. It provides the foundation for implementing the AIIP.

### 5.3.2 Ubuntu Desktop and Secure Shell Protocol

#### Ubuntu Desktop Simulation

The Ubuntu desktop system simulation provides end-users with Linux shell and Secure Shell (SSH) protocol (Figure 5.9). It does not provide any other functionalities.

The Ubuntu desktop simulation relies on Kerberos single sign-on system simulation for authenticating end-users (Figure 5.10). It requests end-users to input their usernames (`(name = raw_input('Username: '))`) and passwords (`(password = getpass.getpass('Password: '))`). The entered credentials are appended to the end of the request strings. For example, `req_tgt;chen;alpha123` is a string that requests ticket granting ticket; it contains the username `chen` and password `alpha123`. It then sends the credentials to the Kerberos single sign-on system simulation.

The Ubuntu desktop simulation will receive the ticket granting ticket if the end-user's credentials were successfully authenticated by Kerberos system simulation. It then grants access to the end-user by presenting them with Ubuntu shell simulations (i.e. `local shell~>`). It also saves the ticket granting ticket (i.e. `ticket`) on the system using function `save_tgt(ticket)`. It saves the ticket granting ticket into a text file (Figure 5.11).

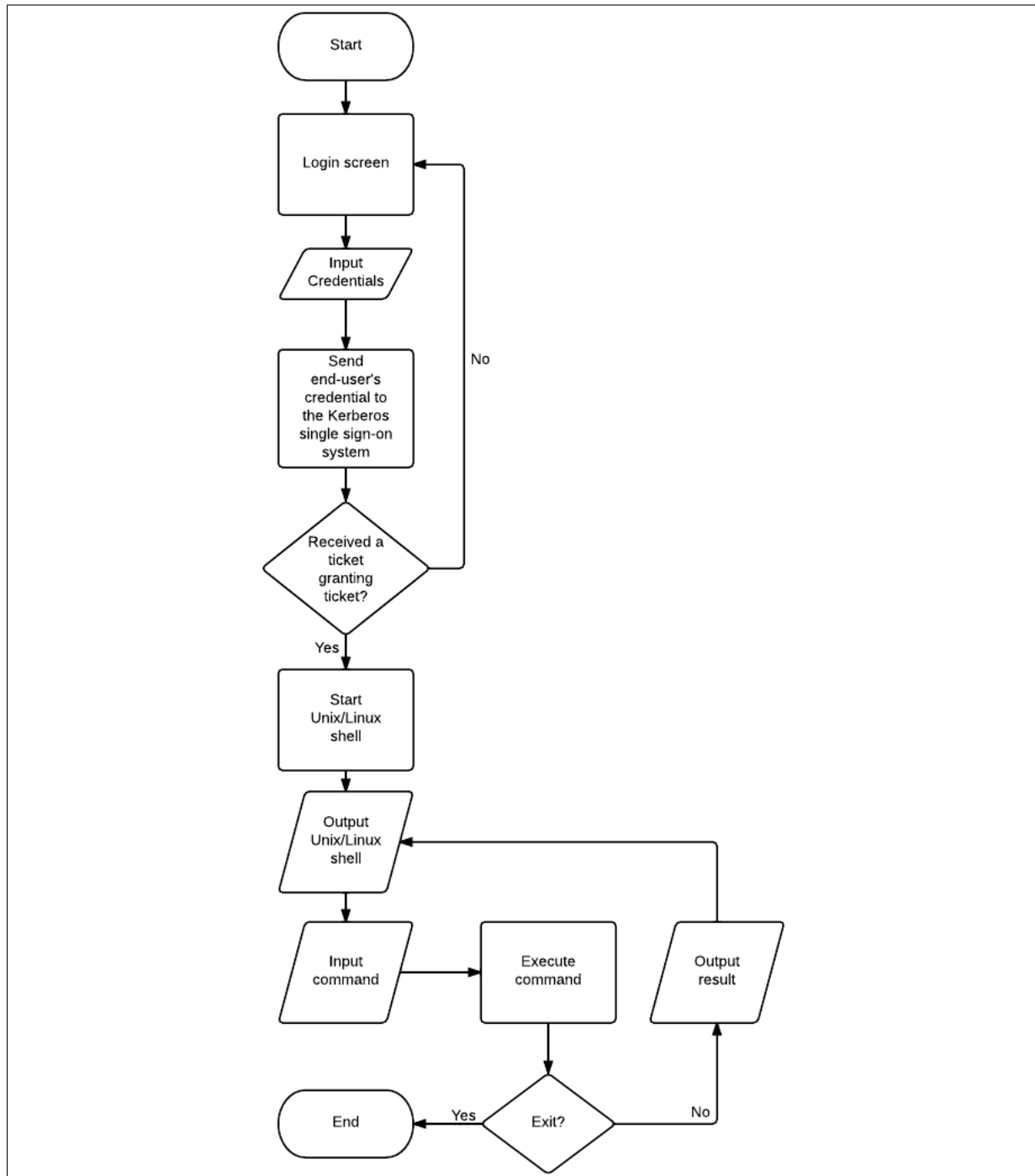


Figure 5.9: Flowchart of Ubuntu desktop simulation

```

# End-user Authentication
def authentication():
    try:
        # Socket for connecting to Kerberos server simulation
        krb_sok = socket.socket (socket.AF_INET, socket.SOCK_STREAM)
        krb_sok.connect((KRB_HOST, KRB_PORT))
        print 'Connecting to', KRB_HOST, '...'

        name = raw_input('Username: ')
        password = getpass.getpass('Password: ')

        krb_sok.send('req_tgt;' + name + ';' + password)

```

Figure 5.10: Desktop Authentication and Requesting Ticket Granting Ticket

```

# Save ticket granting ticket from KDC
def save_tgt(tgt_info):
    # Set ticket granting ticket content in memory
    tgt.set_ticket_info(tgt_info)
    print 'Recieved and saved Ticket Granting Ticket.'

    # Save ticket granting ticket content in file
    f = open('tgt.key', 'w')
    f.write(tgt_info)
    f.close()

```

Figure 5.11: Saving Ticket Granting Ticket

## SSH Client Simulation

The SSH client simulation aims to establish connections to SSH servers (Figure 5.12). It first acquires the service ticket from Kerberos single sign-on system in the same domain using function (`request_service_ticket()`). The process includes sending the end-user's ticket granting tickets to the Kerberos system simulation along with a request for the service ticket (Figure 5.13). It will receives a copy of the service ticket if the Kerberos single sign-on system simulation successfully verified the ticket granting ticket.

Once the SSH client received the service ticket, it will send it to the SSH server simulation as a means for authentication (Figure 5.14). A connection will be established between the SSH client simulation and SSH server simulation (Figure 5.15) if the service ticket was successfully verified (i.e. if `shell_name != 'fail'`). The connection will not terminate until both sides have confirmed the end request (`while results != 'exit_confirm'`).

## Secure Shell Server

The SSH server simulation is used to test single sign-on (and federated single sign-on at later stage). It does not provide any functionalities other than requesting shared key, verifying end-users' service ticket and accepting/establishing remote connection from SSH clients (Figure 5.16). It uses one socket to listen for requests from SSH client simulations, and it uses another socket to connect to Kerberos servers simulation.

The SSH server simulation connects to Kerberos single sign-on system simulation and requests the shared key (Figure 5.17). The SSH servers simulation will receive a copy

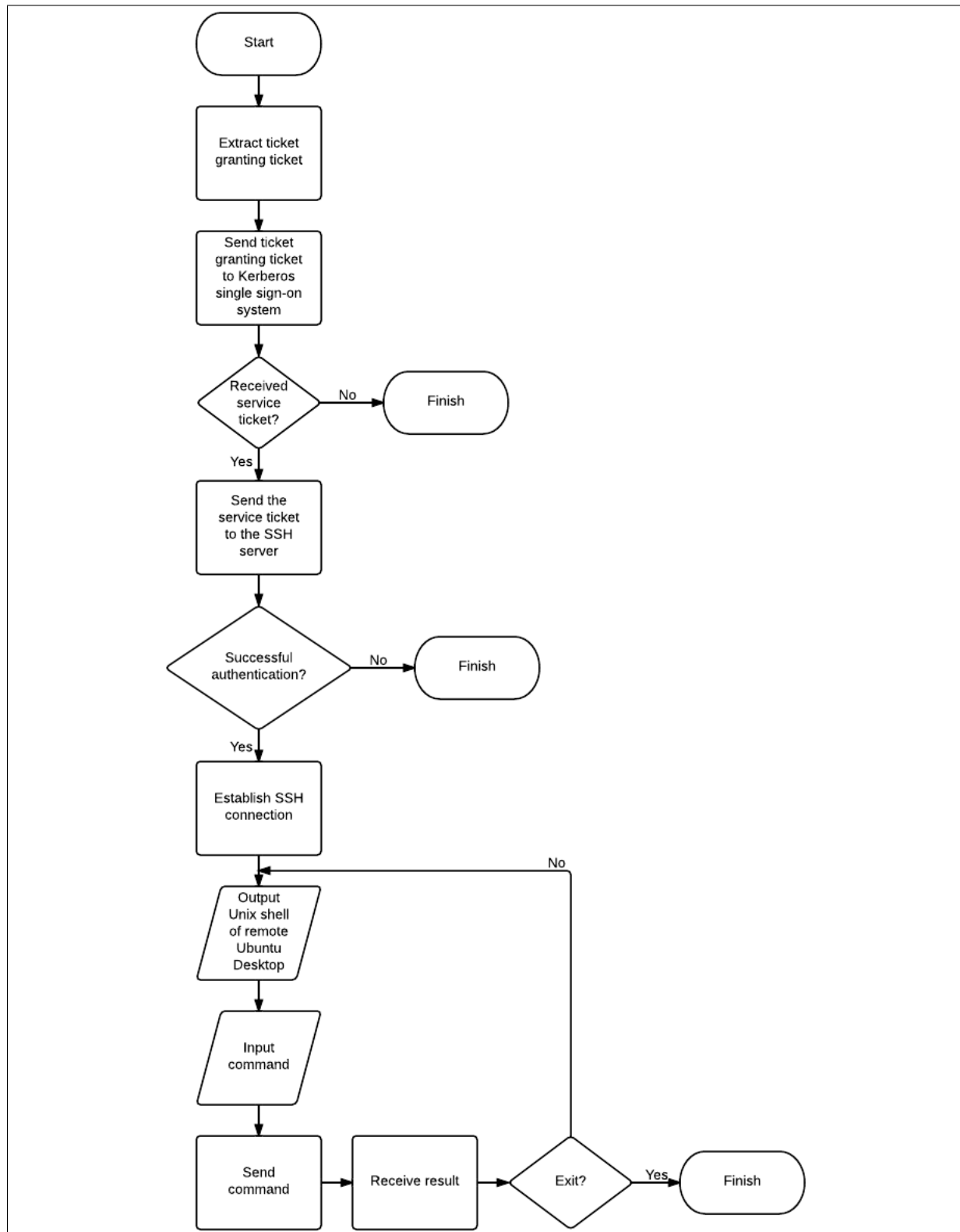


Figure 5.12: Flowchart of SSH client simulation



```
# Get service ticket from KDC to access services
def request_service_ticket():
    krb_sok = socket.socket (socket.AF_INET, socket.SOCK_STREAM) # KRB socket
    f = open('tgt.key', 'r')
    tgt_info = f.readline()
    f.close()
    tgt_info_list = tgt_info.split('; ', 7)

    krb_sok.connect((KRB_HOST, KRB_PORT))
    print 'Connecting to', KRB_HOST, '...'
    krb_sok.send('req_st;' + tgt_info)

    answer = krb_sok.recv(1024)

    if answer == 'fail':
        return 'fail'
    else:
        krb_sok.send('ssh')
        st_info = krb_sok.recv(1024)
        return tgt_info_list[0] + ';' + st_info
```

Figure 5.13: Requesting Service Ticket

```
# SSH client simulation
def ssh_local(st_info):
    ssh = socket.socket (socket.AF_INET, socket.SOCK_STREAM) # SSH socket
    ssh.connect((SSHD_HOST, SSHD_PORT))

    # Send service ticket to ssh server
    ssh.send(st_info)
```

Figure 5.14: Sending Service Ticket

```
# Recieve single sign-on reply
shell_name = ssh.recv(1024)

if shell_name != 'fail':
    results = ''

    while results != 'exit_confirm':
        command = raw_input(shell_name)
        ssh.send(command)
        results = ssh.recv(1024)
        if results != 'exit_confirm':
            print results
```

Figure 5.15: Secure Shell Client Simulation

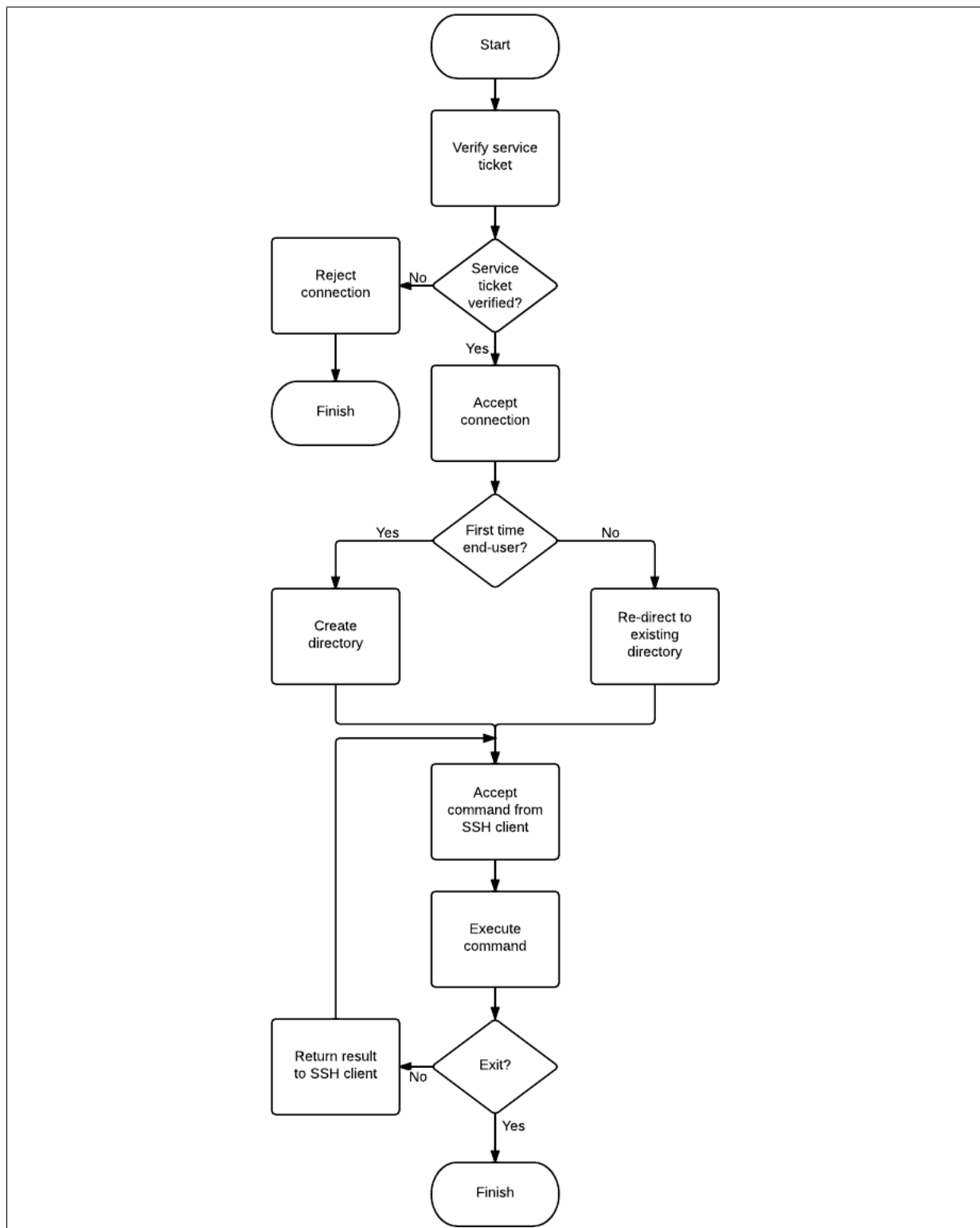


Figure 5.16: Flowchart of a SSH server thread simulation

```

def request_shared_key():
    # Get shared secure ticket from KDC (kdc_sim.py)
    krb_socket = socket.socket (socket.AF_INET, socket.SOCK_STREAM)
    krb_socket.connect((KRB_HOST, KRB_PORT))
    message = ''
    message += 'req_sk;'
    message += (SERVICE_NAME + ';'')
    message += (DOMAIN + ';'')
    message += (IP)

    krb_socket.send(message) # Send service information to KDC
    share_key = krb_socket.recv(1024) # Recieve reply from KDC

    # Check the replied message
    if share_key != 'fail':
        f = open(SERVICE_NAME + '.key', 'w') # Save the shared key
        f.write(share_key)
        f.close
        krb_socket.close
        return True
    else:
        krb_socket.close
        return False

```

Figure 5.17: SSH Server Simulation Requesting Shared Key

```

f = open('ssh.key', 'r') # open ssh.key to read the key
key_info = f.readline()
key_info_list = key_info.split('; ', 3)

```

Figure 5.18: SSH Server Simulation Opens the Shared Key

of the shared key if its service name (e.g. `ssh`), domain name (e.g. `home.virtual.vm`) and IP address (e.g. `192.168.1.59`) were found in Kerberos's service database.

The SSH server simulation will not listen for connection requests until it obtained a shared key from the Kerberos system simulation. This helps to debug the system during implementation. If the service was not registered in the Kerberos service database, the simulation will not start.

Secure shell (SSH) server simulation uses thread (`ClientThread`) to manage connections from multiple clients. It receives the end-user's service ticket (`st_info = self.conn.recv(1024)`). It extracts the username and SHA-512 hash key from the service ticket (`username, st = st_info.split('; ', 1)`). It then extracts the SHA-512 hash key from the shared key that was issued by the Kerberos server simulation (Figure 5.18).

The SSH server simulation will accept the SSH connection if the two SHA-512 hash keys match. It then directs the end-users to their dedicated directories and will create

```
if key_info_list[3] == st:
    # Create home directory for the user if the directory doesn't exist
    if os.path.exists(username):
        os.chdir(username)
    else:
        os.makedirs(username)
        os.chdir(username)

print "Authenticated the service ticket"
self.conn.send('home remote shell~>')    # Send the SSH server's shell name to the SSH client.
```

Figure 5.19: Create Dedicated Directories

dedicated directories for first time to the end-users (Figure 5.19).

## Ubuntu Desktop and SSH Protocol Summary

The goals of the simulations using the Ubuntu desktop and SSH protocol are to demonstrate local network single sign-on and federated single sign-on. The Ubuntu desktop relies on Kerberos single sign-on for providing authentication and ticket granting ticket. The SSH client and server runs on top of Ubuntu desktop simulation. They communicate with each other using network sockets. The SSH server can authenticate the end-user by verifying their service tickets instead of verifying their credentials (i.e. usernames and passwords). Currently the end-user can access remote Ubuntu desktops (in the local domain) via SSH protocol in a single sign-on fashion.

### 5.3.3 SAML based Federated Single Sign-on System Simulation

#### Web-based Authentication System

The simulation of web-based authentication system provides the foundation for implementing IDAWA. It's the identity provider end of the SAML based federated single sign-on system. The other end is the simulation of the web-based service/application.

The authentication system currently authenticates end-users by verifying their usernames and passwords, and it creates identity assertion for end-users that have been successfully authenticated. It provides the foundation for implementing IDAWA, which will accept Kerberos service tickets as means of authentication.

The identity assertion message is a strip down version of the real world assertion message. It includes the end-users' username, password, domain name, role and the result of authentication (e.g. true or false). The value "true" means the end-user has been successfully authenticated; the value "false" means the end-user hasn't been authenticated. It does not include any SAML tags (e.g. <SAML>) or SAML attributes.

The web-based authentication system uses session variables to transfer variables between web pages. Session variables are named as following: `$_SESSION["username"]`, `$_SESSION["idp_domain"]`, `$_SESSION["role"]` and `$_SESSION["authenticated"]`. They are created and maintained by placing `session_start()`; at the start of the PHP script.

Session variables can not be accessed by web pages in different domains. For example, the session variables in `home.virtual.vm` can not be accessed by web pages in `foreign.virtual.vm`. My implementation addresses this problem using web browser cookies.

Web cookies are stored on end-users' web browsers. They are accessible by web pages. An end-user's identity assertion message can be stored as cookies on the web browsers. These cookies can be accessed by web-based service/application simulation when the end-users are accessing the web-based service/applications (using web browsers).

The use of web cookie is based on the assumption that the web-browsers support the use of web cookies. The web-browsers can save and read the web cookies. If they do not support web cookies, then the implementation will not work.

PHP script, `setcookie("user", $value, time()+3600, "/", "virtual.vm")`, is used to create cookies. Variable "user" is the name of the cookie. `\$value` contains the value of the cookie; it contains the session variables such as username, role, domain name and authentication result (Figure 5.20). `time()+3600` is the life time of the cookie. "/" indicates the cookie is place at the root of the domain and accessible by sub domains. `virtual.vm` is the root of the domain. This allows domain "foreign.virtual.vm" to access the identity assertion messages.

```
$value = $_SESSION["username"].';'. $_SESSION["role"]. \
';'. $_SESSION["idp_domain"].';'. $_SESSION["authenticated"];
```

Figure 5.20: Creating Cookie Value

```
$user = $_COOKIE["user"];
$user_list = explode(";", $user);

$username = $user_list[0]; //$_SESSION["username"];
$role = $user_list[1]; //$_SESSION["role"];
$domain = $user_list[2]; //$_SESSION["idp_domain"];
```

Figure 5.21: Extracting Cookie Value

## Web-based Service/Application

The goal of the simulation of web-based service/application is to display successfully federated single sign-on. It displays that end-users have been successfully authenticated by the simulation of web-based authentication systems, and it displays the roles of the end-users. It does not provide any real world functionalities (e.g. web-based document editor).

The simulation of web-based service/application extracts the variables (e.g. username, role, domain) from the cookies. These variables compose an end-user's identity assertion message (Figure 5.21).

The simulation of web-based service/application reads an end-user's identity assertion message (i.e. user "chen"). It determines the permission of the end-user according to the federation's policy (described in section 4.2.2). It then displays the end-user's data (Figure 5.22).

## SAML based FSSO Simulation Summary

The SAML based federated single sign-on system provides the foundation for implementing the IDAWA. The identity provider side (i.e. web-based authentication system) does not support Kerberos single sign-on system, and it authenticates end-users by verifying their usernames and passwords. It provides identity assertions, which are strip down version of real world assertions for end-users that have been successfully authenticated. The service provider side (i.e. web-based services/applications) displays the end-users' data

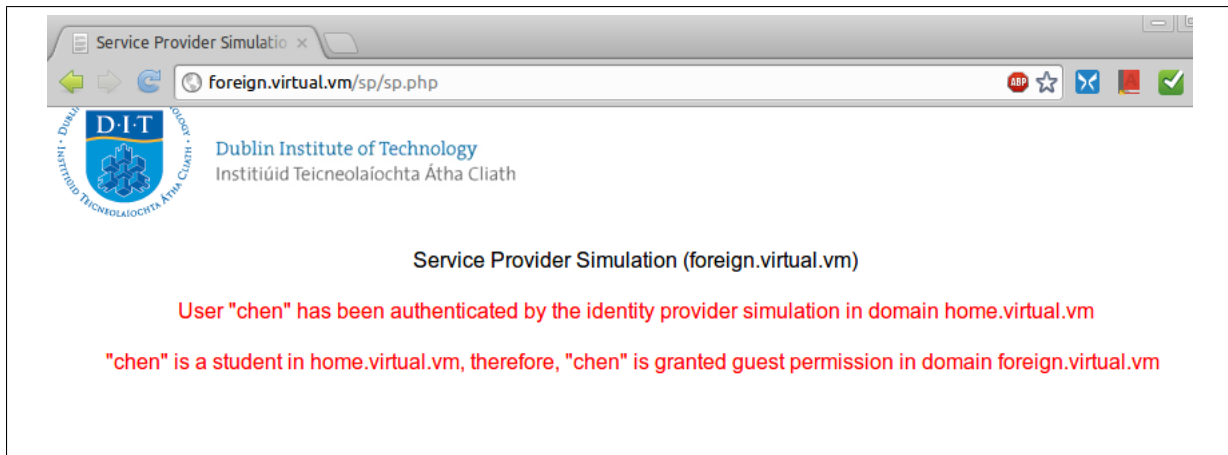


Figure 5.22: Web-based Service/Application Simulation in Domain `foreign.virtual.vm`

(e.g. username, domain name, role); it does not provide any real world functionalities.

## 5.4 Implementation of the New Simplified FSSO System

This section presents the implementation of the new simplified federated single sign-on (FSSO) system. The implementation incorporates the simulations of Kerberos single sign-on system and SAML based federated single sign-on system, and it includes two major components: Authentication Infrastructure Integration Program (AIIP) and Integration of Desktop Authentication and Web-based Authentication (IDAWA).

This section also presents the modifications to the SSH client simulations. These modifications allow the SSH clients to request Kerberos tickets (from foreign domains) through AIIP instead of requesting ticket from foreign domains' Kerberos single sign-on systems.

### 5.4.1 Implementation of AIIP

The "Authentication Infrastructure Integration Program" (AIIP) follows the federation's policies, which state, "End-users from any domain can access any services/applications in the other domains." It includes the implementation of two ends: the identity provider

end ( $AIIP_i$ ) and service provider end ( $AIIP_s$ ). A network domain may have both ends; however, a single federated single sign-on process involves two ends that are in separate domains.

Both ends work together to extend end-users' desktop authentication sessions (i.e. Kerberos authentication sessions) from one domain to other domains. The  $AIIP_i$  uses Kerberos system simulation for authentication, and it creates/sends identity assertions to the service provider. The  $AIIP_s$  receives and verifies the identity assertions; it acquires Kerberos tickets in the same domains and sends them to the  $AIIP_i$ .

The  $AIIP_i$  chooses  $AIIP_s$  based on end-users' requests. For example, an end-user entered the command "ssh foreign.virtual.vm", then the  $AIIP_i$  will communicate with the  $AIIP_s$  in the domain that has the domain name "foreign.virtual.vm".

### **AIIP (Identity Provider End)**

The identity provider end ( $AIIP_i$ ) of the AIIP has two main functions: authenticating end-users using Kerberos server simulation and creating/sending identity assertions for the authenticated end-users. It authenticates end-users by verifying their Kerberos service tickets, and it creates identity assertions and sends them to the service provider end of the AIIP (Figure 5.23).

The  $AIIP_i$  uses a network socket to listen for federated single sign-on requests from the end-users' non-web based user agents (e.g. SSH client). It requires two more sockets for connecting to Kerberos server simulation and the service provider end of AIIP ( $AIIP_s$ ).

In order to support Kerberos single sign-on, the  $AIIP_i$  requests a shared key from the Kerberos server simulation (Figure 5.24). It receives and stores the shared key. The shared key is later used to verify end-users' service tickets.

The  $AIIP_i$  listens for requests from end-users. It accepts connections from end-users' SSH clients and accepts their service tickets (Figure 5.25). It also reads the contents in the shared key (`saml-aii-kerberos.key`). It compares the SHA-512 hash key (i.e.



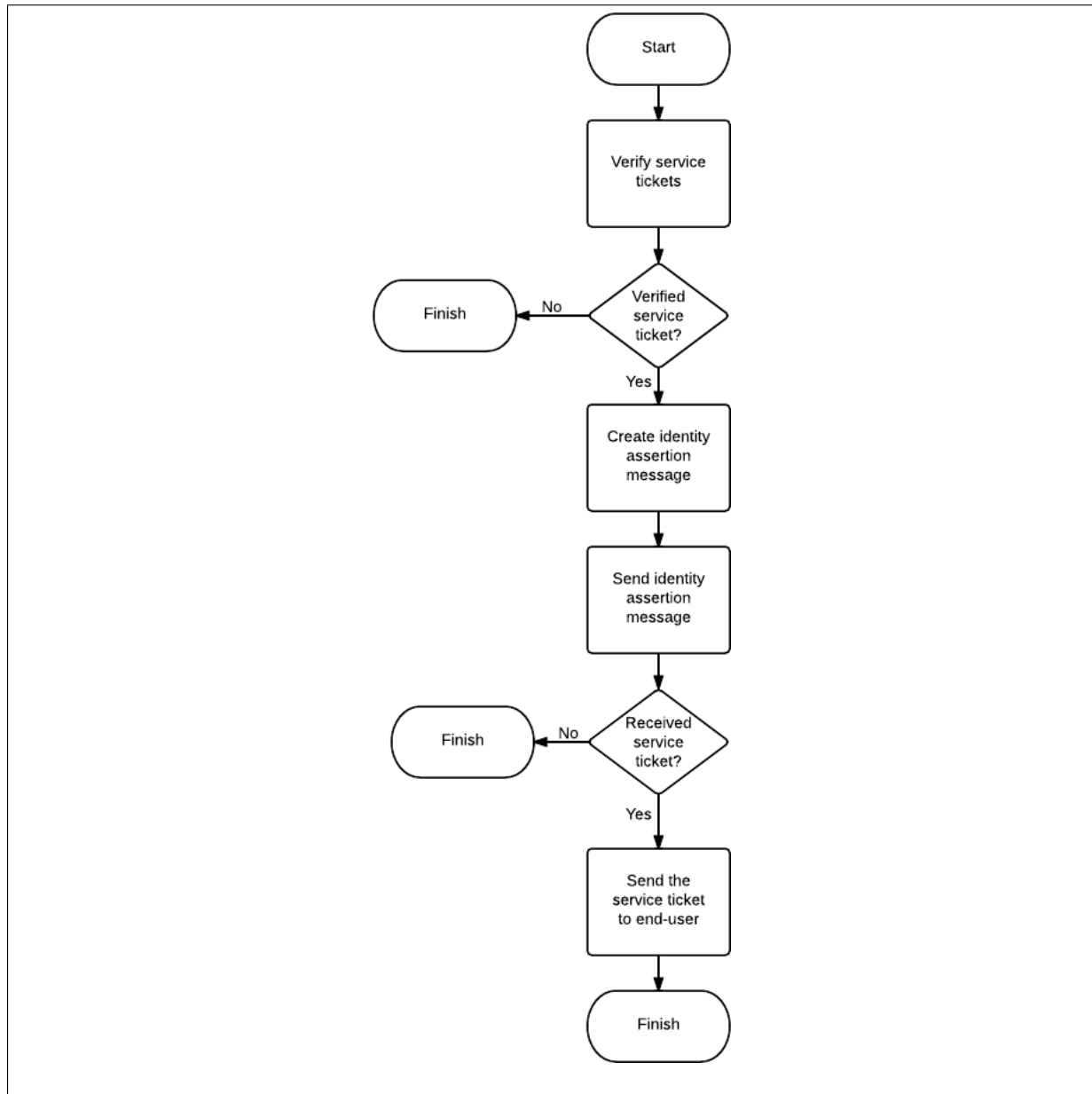


Figure 5.23: Flowchart of identity provider end of AIIP

```

def request_shared_key():
    # Get shared secure ticket from Kerberos system simulation
    krb_socket = socket.socket (socket.AF_INET, socket.SOCK_STREAM)
    krb_socket.connect((KRB_HOST, KRB_PORT))
    message = ''
    message += 'req_sk;'
    message += (SERVICE_NAME + ';')
    message += (DOMAIN + ';')
    message += (IP)

    krb_socket.send(message)    # Send service information to Kerberos system
    share_key = krb_socket.recv(1024)    # Recieve reply from Kerberos system

    # Check the replied message
    if share_key != 'fail':
        f = open(SERVICE_NAME + '.key', 'w')    # Save the shared key
        f.write(share_key)
        f.close
        krb_socket.close
        return True
    else:
        krb_socket.close
        return False

```

Figure 5.24: Requesting Shared Key

```

f = open('saml-aai-kerberos.key', 'r')    # open ssh.key to read the key
key_info = f.readline()
key_info_list = key_info.split('; ', 3)

st_info = self.conn.recv(1024)
username, st = st_info.split('; ', 1)

```

Figure 5.25: Accepting Service Tickets

`key_info_list[3]`) in the shared key with the hash key (`st`) in the service ticket. The end-users are authenticated if the two SHA-512 hash keys match.

The  $AIIP_i$  requests authenticated end-users' data (e.g. username, role and domain name). It receives the data and appends them to the end of FSSO request message (`'req_st_foreign;'`) to create identity assertion messages. It then sends the identity assertion messages to the service provider end of AIIP (Figure 5.26). The data (in the identity assertion messages) are necessary for the other domains to determine the level of access of the end-users to their services/applications.

The firewalls between the two domains have been set to support the web-based services; therefore, port 80 was opened. The AIIPs can use any network port for communication. They can be reconfigured to suit the network domains' firewall rules. In this case, the AIIPs chose port 80 as the communication port so that network domains (that already opened port 80) don't need to re-configure their firewall rules. In addition, AIIP's

```

def send_assertion(self, assertion):
    foreign_sak_socket = socket.socket (socket.AF_INET, socket.SOCK_STREAM)
    foreign_sak_socket.connect((self.FOREIGN_MIDDLE_HOST, self.FOREIGN_MIDDLE_PORT))

    message = 'req_st_foreign;'
    message += assertion    # Create identity assertion message
    print message
    foreign_sak_socket.send(message)    # Send identity assertion message

    # Wait for the reply from the service provider end of AIIP
    foreign_st_info = foreign_sak_socket.recv(1024)

    foreign_sak_socket.close()

    return foreign_st_info

```

Figure 5.26: Creating/Sending Identity Assertion to the Service Provider End of AIIP

sockets are in the Transport Layer of the OSI stack. It will not interfere with HTTP since HTTP is in Application Layer of the Internet model.

The  $AIIP_i$  will receive the Kerberos service tickets (created in the other domains) from the  $AIIP_s$  if the identity assertion messages were accepted. It then passes these service tickets to the end-users' user agents (e.g. SSH clients). The end-users can then use the service tickets to access non-web based services in the other domains without requiring re-authentication.

### AIIP (Service Provider)

The service provider end of AIIP ( $AIIP_s$ ) listens for new connections from the  $AIIP_i$ s, and it manages clients using thread (`ClientThread`). Each thread manages each request from the identity provider end of AIIP (Figure 5.27). It verifies the identity assertions (`req_st_foreign`) and extracts the end-users' data (e.g. username, role/permission and domain name) from the identity assertion messages (Figure 5.28).

The  $AIIP_s$  maps an end-user's identity to an identity in the  $AIIP_s$ 's domain (Figure 5.29). It reads the end-users database (in the same domain) and looks for a matching username (`if user[0].text == username`). It reads the rest of the credentials (username, password, domain name, permission) if it found a matching username. The provisioned credentials will be used for acquiring Kerberos tickets.

The  $AIIP_s$  requests the Kerberos ticket granting ticket on behalf of the end-

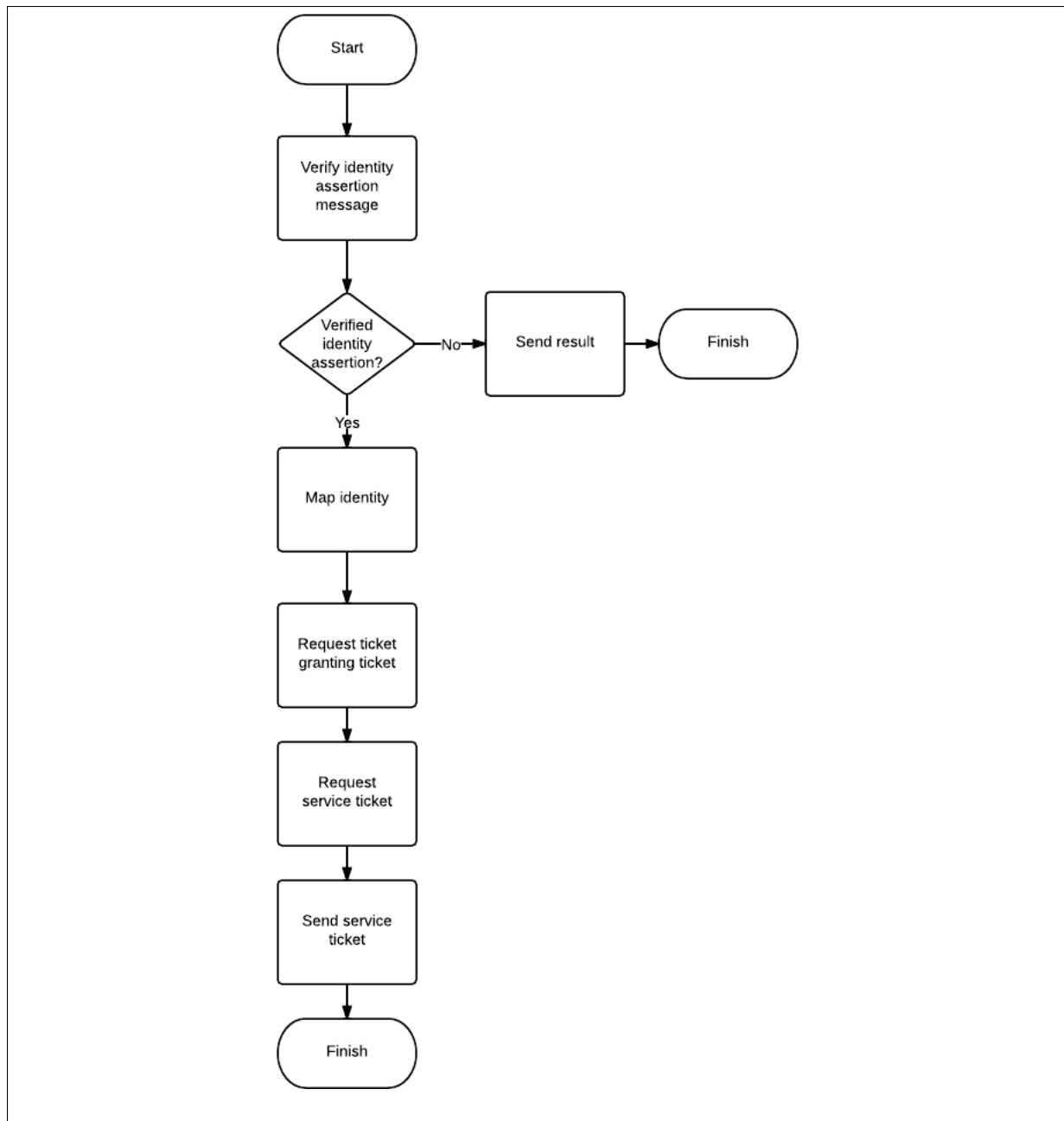


Figure 5.27: Flowchart of a thread of service provider end of AIIP

```

foreign_user_info = self.conn.recv(1024)    # Receive identity assertion message

foreign_user_info_list = foreign_user_info.split(';', 1)    # Extract request message

if foreign_user_info_list[0] == 'req_st_foreign':
    detailed_info = foreign_user_info_list[1].split(';', 2)    # Extract end-user data
    username = detailed_info[0]
    domain = detailed_info[1]
    permission = detailed_info[2]

```

Figure 5.28: Extracting End-user's Identity Information from Identity Assertion Message

```

def exist_user(self, username, domain, permission):
    try:
        tree = xml.parse("foreignuserdatabase.xml") # Parse the user database
        doc = tree.getroot() # Get the root node

        # Check the username and password
        for user in doc:
            if user[0].text == username:
                self.user = user
                provisioned_user_info = ''
                provisioned_user_info += user[0].text
                provisioned_user_info += ';'
                provisioned_user_info += user[1].text
                provisioned_user_info += ';'
                provisioned_user_info += user[2].text
                provisioned_user_info += ';'
                provisioned_user_info += user[3].text

        return provisioned_user_info

```

Figure 5.29: Mapping End-user's Credentials

```

# Requesting TGT on behalf of the end-user
def s4u2selfuser(self, provisioned_user_info):
    detailed_info = provisioned_user_info.split(';')
    username = detailed_info[0]
    password = detailed_info[1]

    krb_socket = socket.socket (socket.AF_INET, socket.SOCK_STREAM) # Kerberos socket
    krb_socket.connect((KRB_HOST, KRB_PORT))

    message = 'req_tgt;'
    message += username
    message += ';'
    message += password

    # Request TGT
    krb_socket.send(message)
    tgt_info = krb_socket.recv(1024)
    krb_socket.close()
    return tgt_info

```

Figure 5.30: Requesting Ticket Granting Ticket on behalf of End-user

user similar to SAML-AAI/Kerberos (Figure 5.30). It acquires ticket granting ticket (`req_tgt`) by sending the provisioned credential (i.e. username and password) to the Kerberos server (in the same domain as the  $AIIP_s$ ). It then stores the ticket granting ticket in the programme's memory.

The  $AIIP_s$  will request the Kerberos service ticket on behalf of the end-user after acquiring the ticket granting ticket (Figure 5.31). It sends the request for service ticket (`req_st`) and the ticket granting ticket (`tgt_info`) to the Kerberos server simulation. It then sends the name of the service (e.g. '`ssh`') to the Kerberos server simulation if the ticket granting ticket was successfully verified. It will send the acquired service ticket to the  $AIIP_i$  in the other domains.

```

def s4u2selfproxy(self, tgt_info):
    krb_socket = socket.socket (socket.AF_INET, socket.SOCK_STREAM)    # Kerberos socket
    krb_socket.connect((KRB_HOST, KRB_PORT))    # Create connection

    message = 'req_st;'
    message += tgt_info

    krb_socket.send(message)

    answer = krb_socket.recv(1024)

    if answer == 'fail':
        krb_socket.close()
        return 'fail'
    else:
        krb_socket.send('ssh')
        st_info = krb_socket.recv(1024)
        krb_socket.close()
        return st_info

```

Figure 5.31: Requesting Service Ticket Ticket on behalf of End-user

## Modifying SSH Client

The SSH client simulation requires extra functions to use the AIIP for federated single sign-on. Instead of accessing Kerberos single sign-on systems in the foreign domains, the SSH client requests a service ticket from the  $AIIP_i$  when an end-user issued a command to access a SSH server in a separate domain. This is achieved using the following functions: requesting service tickets to access the identity provider end of AIIP ( $AIIP_i$ ), authenticating (to the  $AIIP_i$ ) using the service ticket, accepting Kerberos tickets that were created in the other domains, and using the service ticket (created in the other domains) to access SSH servers in the other domains.

The SSH client will receive a service ticket for accessing a SSH server in a separate domain. It then can use the service ticket to access the SSH server in the separate domains (Figure 5.32). It sends the service ticket (**st\_info**) to the SSH server (in the other domain). The SSH server should successfully verify the service ticket since it was created by the Kerberos server in the same domain (as the SSH server). A connection will be established between the client and server.

```

def ssh_foreign(st_info):
    ssh_foreign_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM) # SSH socket
    ssh_foreign_socket.connect((FOREIGN_SSHD_HOST, FOREIGN_SSHD_PORT))
    # Send service ticket to ssh server
    ssh_foreign_socket.send(st_info)

    # Recieve single sign-on reply
    shell_name = ssh_foreign_socket.recv(1024)

    if shell_name != 'fail':
        results = ''

        while results != 'exit_confirm':
            command = raw_input(shell_name)
            ssh_foreign_socket.send(command)
            results = ssh_foreign_socket.recv(1024)
            if results != 'exit_confirm':
                print results

        else:
            pass

    else:
        print 'Service ticket failed to authenticate'

    ssh_foreign_socket.close()
    print 'exiting...'

```

Figure 5.32: Accessing SSH Servers in Separate Domains

### Implementation of AIIP Summary

The AIIP has two ends, identity provider end and service provider end. Both end follows the federated single sign-on policies that were established between network domains. A non-web based federated single sign-on involves an identity provider end in one domain and a service provider end in another domain. The identity provider end authenticates end-users using their service tickets, and it creates/sends identity assertions to the service provider end. The service provider (in another domain) acquires Kerberos tickets in the same domain and passes them back to the identity provider end, which will proceed to pass them to the end-users. End-users can use these service tickets to access non-web based services in the foreign domains. The two end of AIIP work together to bridge the Kerberos single sign-on sessions in different domains. The SSH clients were added new functions in order to request a service ticket from AIIP when end-users attempted to access SSH server in foreign domains.

### 5.4.2 Implementation of IDAWA

IDAWA is an extension to the web-based authentication system simulation. It allows end-users to use their Kerberos ticket for authentication (Figure 5.33). It integrates the desktop authentication session with web-based authentication session; therefore, end-users do not need to be authenticated again to access web-based services/applications.

IDAWA uses sockets to connect to Kerberos server simulation. It manually obtain the shared key from the Kerberos server simulation, and it saves the shared key into session variable `$_SESSION["idp_shared_key"]` (Figure 5.34).

IDAWA adds the support of Kerberos tickets onto the web-based authentication system. It allows end-users to upload their Kerberos ticket granting tickets (Figure 5.35). The ticket granting ticket is named `tgt` and saved as a file variable (`$_FILE["tgt"]`).

The IDAWA gets the content of the Kerberos ticket granting ticket using (`$tgt = file_get_contents($_FILES["tgt"]["tmp_name"]);`). It assigns the value of the ticket granting ticket to variable `$tgt`. It then sends the request of service ticket to the Kerberos server simulation on behalf of the end-user (Figure 5.36).

The IDAWA will continue send the name of the service (e.g. `‘‘idp’’`) to the Kerberos server simulation if `$answer` showed the authentication was successful. It then compares the hash key in the service ticket (`$service_ticket`) with the hash in the shared key (`$idp_shared_key`). It will create identity assertion for the end-user if the two hash keys are a match (Figure 5.37). It allows end-users to continue to access web-based services/applications using the identity assertion (i.e. without requiring re-authentication).

## 5.5 Demonstration of the New Simplified FSSO System

An end-user (in domain Alpha) attempts to access a remote SSH server and the web-based service/application in domain Beta. The demonstration attempts to show that



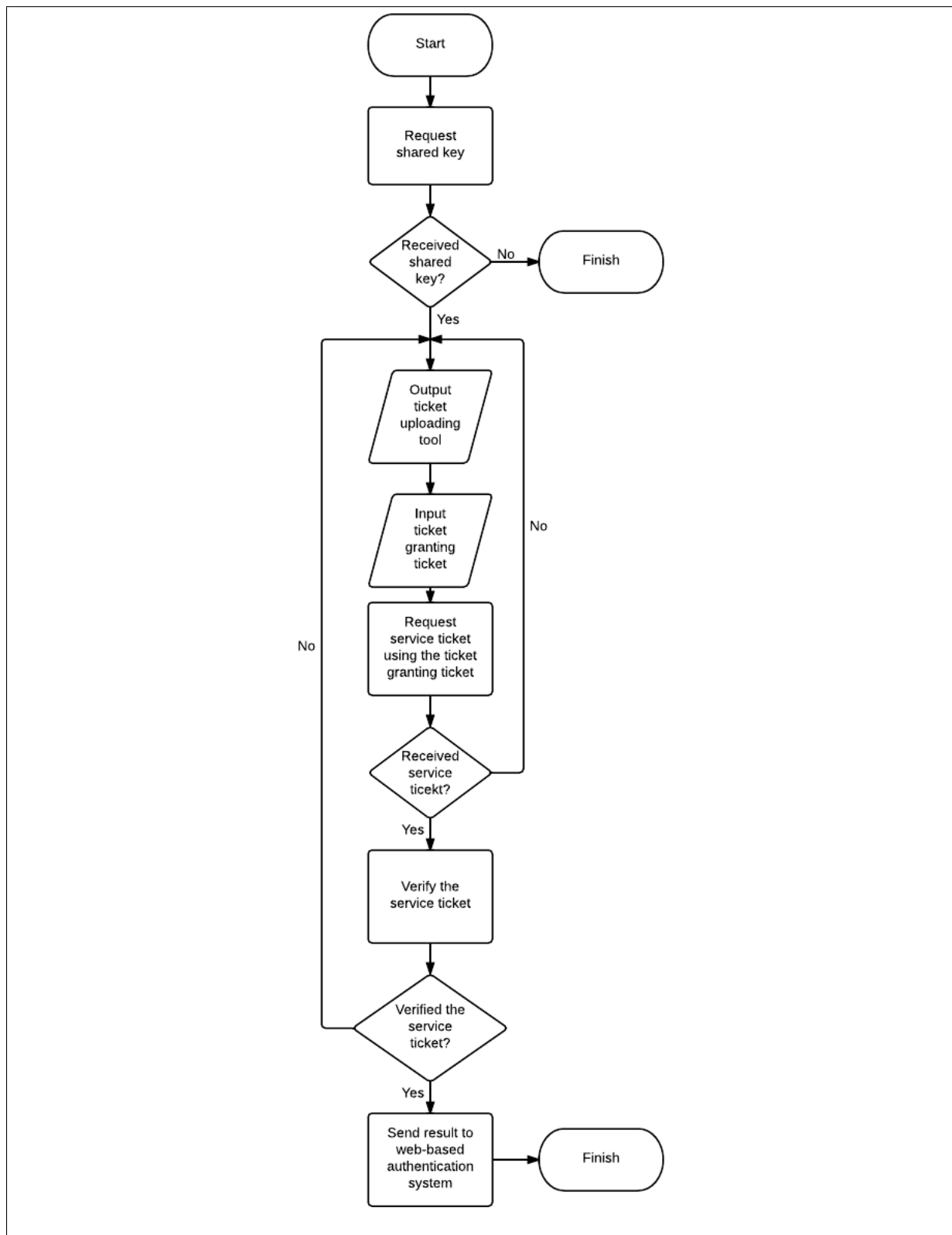


Figure 5.33: Flowchart of IDAWA

```

$kerberos_port = 88;    // Get the port for the Kerbero service.
$kerberos_ip = "192.168.1.59";    // Get the IP address for the target host.

/* Create a TCP/IP socket. */
$socket = socket_create(AF_INET, SOCK_STREAM, SOL_TCP);
$connect = socket_connect($socket, $kerberos_ip, $kerberos_port);

$request = "req_sk;";
$request .= "idp".',';
$request .= "home.virtual.vm".',';
$request .= "192.168.1.59".',';

socket_write($socket, $request, strlen($request));

$shared_key = socket_read($socket, 1024);

$_SESSION["idp_shared_key"] = $shared_key;

```

Figure 5.34: IDAWA Requesting Shared Key

```

<form action="auth_key.php" method="post" enctype="multipart/form-data">
  <label for="key">Kerberos TGT:</label>
  <input type="file" name="tgt" id="tgt" />
  <input type="submit" name="submit" value="Submit" />
</form>

```

(a) Source Code

Kerberos TGT:  No file chosen

(b) Web page

Figure 5.35: Uploading Kerberos Ticket Granting Ticket onto IDAWA

```

$request = "req_st;";
$request .= $tgt;

$reply = '';

// Sending service ticket request.
socket_write($socket, $request, strlen($request));

// Recieving response.
$answer = socket_read($socket, 1024);

```

Figure 5.36: Web-based Authentication System Requests Service Ticket

```

// request service ticket
socket_write($socket, "idp", strlen("idp"));    //request service ticket for idp
$service_ticket = socket_read($socket, 1024);
socket_close($socket);

$idp_shared_key = $_SESSION["idp_shared_key"];

$idp_shared_key_info = explode(";", $idp_shared_key);

if ($idp_shared_key_info[3] == $service_ticket)    //compare service ticket and shared key
{
    $user_info = explode(";", $tgt);
    $_SESSION["username"] = $user_info[0];
    $_SESSION["role"] = $user_info[2];
    $_SESSION["idp_domain"] = $user_info[4];

    // $_SESSION["authenticated"] asserts that the end-user is authenticated
    $_SESSION["authenticated"] = true;
}

```

Figure 5.37: Web-based Authentication System Verifies the Service Ticket

AIIP integrates the Kerberos single sign-on sessions in domain Alpha to domain Beta and that IDAWA allows the end-user to use the Kerberos ticket (created during desktop authentication) for web-based authentication.

### 5.5.1 Demonstration of AIIP

#### Access the Local Ubuntu Desktop Simulation in Domain Alpha

The end-user attempts to access a desktop system in domain Alpha by entering its username and password at the login prompt (Figure 5.38). Upon successfully verification of the credentials, the desktop simulation accepts the end-user and present a Linux shell (i.e. `local shell~>`). It will also receive a copy of the ticket granting ticket from the Kerberos single sign-on system in domain Alpha.

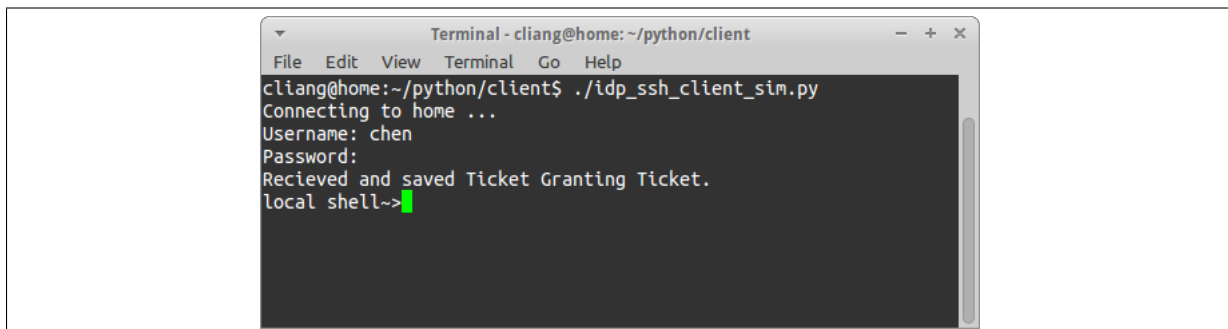
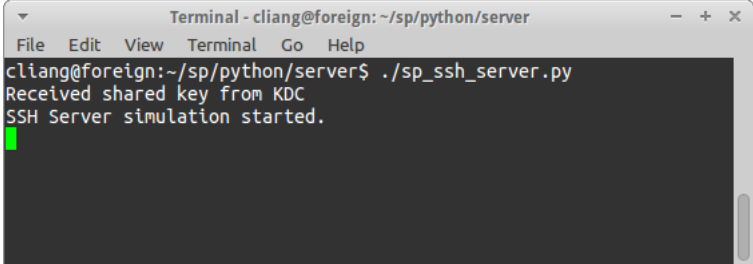


Figure 5.38: End-user login the Local Ubuntu desktop simulation in home.virtual.vm (i.e. domain Alpha)

### Initiate the Remote SSH Server Simulation in Domain Beta

The remote SSH server simulation manually obtains a shared key from the Kerberos single sign-on system when it's started (Figure 5.39). It then waits for a connection request (from SSH clients), and it will remain in this state until receiving a connection request.



```

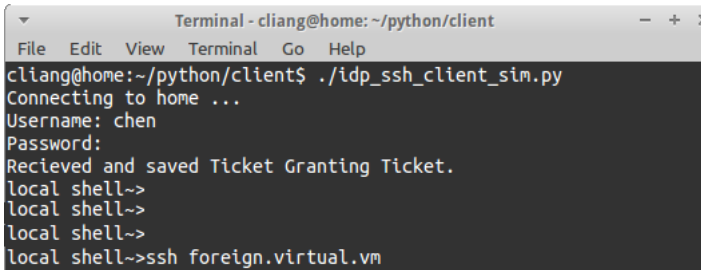
Terminal - cliang@foreign: ~/sp/python/server
File Edit View Terminal Go Help
cliang@foreign:~/sp/python/server$ ./sp_ssh_server.py
Received shared key from KDC
SSH Server simulation started.

```

Figure 5.39: Start the remote SSH server simulation in domain foreign.virtual.vm (i.e. domain Beta)

### Non-web based Federated Single Sign-on using AIIP

The end-user executes the command “ssh foreign.virtual.vm” in the local Ubuntu desktop simulation (Figure 5.40). The remote SSH server simulation requests authentication. This triggers the local Ubuntu desktop simulation to request a service ticket from the identity provider end of AIIP in domain Alpha.



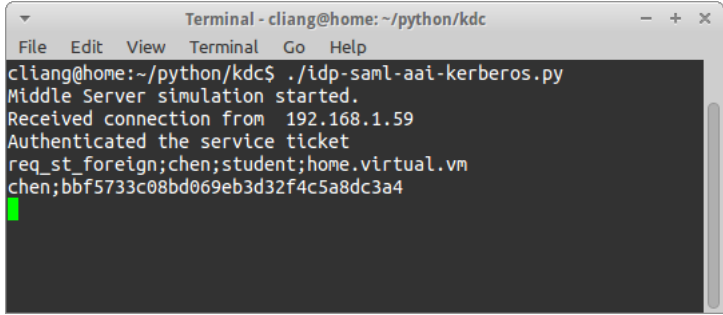
```

Terminal - cliang@home: ~/python/client
File Edit View Terminal Go Help
cliang@home:~/python/client$ ./idp_ssh_client_sim.py
Connecting to home ...
Username: chen
Password:
Recieved and saved Ticket Granting Ticket.
local shell~>
local shell~>
local shell~>
local shell~>ssh foreign.virtual.vm

```

Figure 5.40: End-user attempts to access the remote SSH server in foreign.virtual.vm (i.e. domain Beta)

$AIIP_i$  first authenticates the end-user by verifying its service ticket. It then creates/sends an identity assertion to the service provider end of AIIP ( $AIIP_s$ ) in domain Beta (Figure 5.41). If  $AIIP_s$  replies with a service ticket (that was created in domain Beta),  $AIIP_i$  will pass the acquired Kerberos ticket to the end-user's SSH client simulation in domain Alpha.



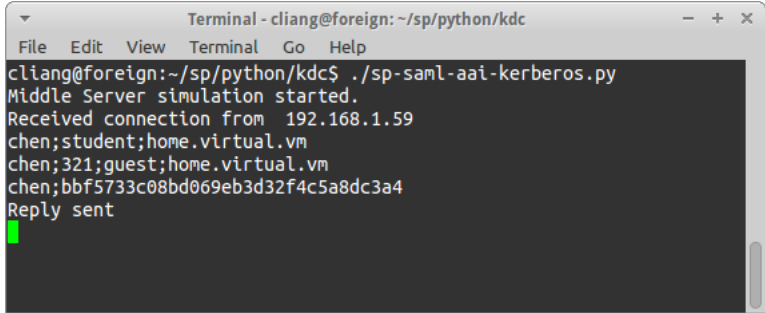
```

Terminal - cliang@home: ~/python/kdc
File Edit View Terminal Go Help
cliang@home:~/python/kdc$ ./idp-saml-aai-kerberos.py
Middle Server simulation started.
Received connection from 192.168.1.59
Authenticated the service ticket
req_st_foreign;chen;student;home.virtual.vm
chen;bbf5733c08bd069eb3d32f4c5a8dc3a4

```

Figure 5.41:  $AIIP_i$  verifies the end-user's identity and sends an identity assertion to  $AIIP_s$

$AIIP_s$  verifies the identity assertion message and acquires a ticket granting ticket and service ticket on behalf of the end-user (Figure 5.42). It then passes the service ticket to  $AIIP_i$ .

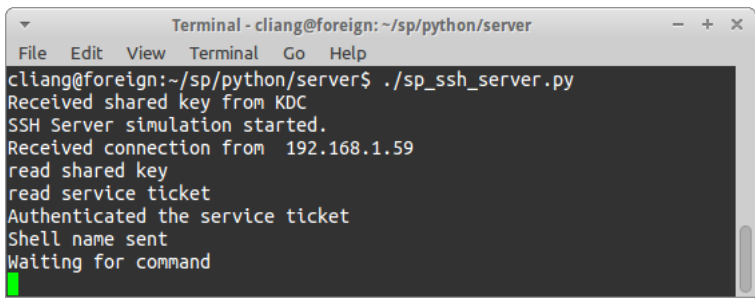


```

Terminal - cliang@foreign: ~/sp/python/kdc
File Edit View Terminal Go Help
cliang@foreign:~/sp/python/kdc$ ./sp-saml-aai-kerberos.py
Middle Server simulation started.
Received connection from 192.168.1.59
chen;student;home.virtual.vm
chen;321;guest;home.virtual.vm
chen;bbf5733c08bd069eb3d32f4c5a8dc3a4
Reply sent

```

Figure 5.42:  $AIIP_s$  accepts the end-user's identity assertion and requests ticket granting ticket and service ticket on behalf of the end-user in domain foreign.virtual.vm



```

Terminal - cliang@foreign: ~/sp/python/server
File Edit View Terminal Go Help
cliang@foreign:~/sp/python/server$ ./sp_ssh_server.py
Received shared key from KDC
SSH Server simulation started.
Received connection from 192.168.1.59
read shared key
read service ticket
Authenticated the service ticket
Shell name sent
Waiting for command

```

Figure 5.43: The remote Ubuntu desktop simulation accepts the request from the end-user

The end-user uses the service ticket (created in domain Beta) to access the remote SSH server simulation in domain Beta. The remote SSH server simulation verifies the end-user's service ticket, and it accepts the service request (Figure 5.43). The end-user (in domain Alpha) is granted access to the remote SSH server simulation in domain Beta

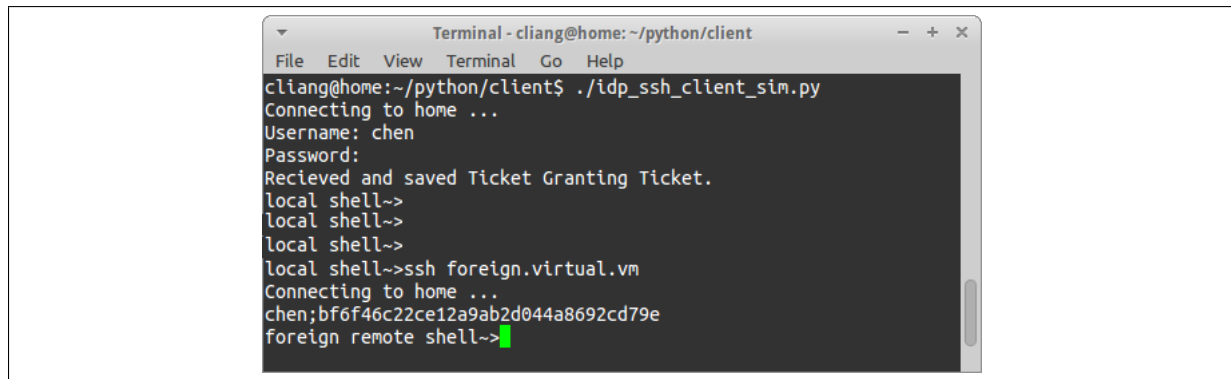


Figure 5.44: The end-user is granted access the to remote SSH server in domain foreign.virtual.vm

(Figure 5.44).

## Demonstration of IDAWA

IDAWA is an extension to web-based authentication systems, and it allows the end-users to use their Kerberos ticket for as means for authentication. The end-users need to upload their ticket granting ticket onto the IDAWA during the first web-based federated single sign-on process (Figure 5.45).

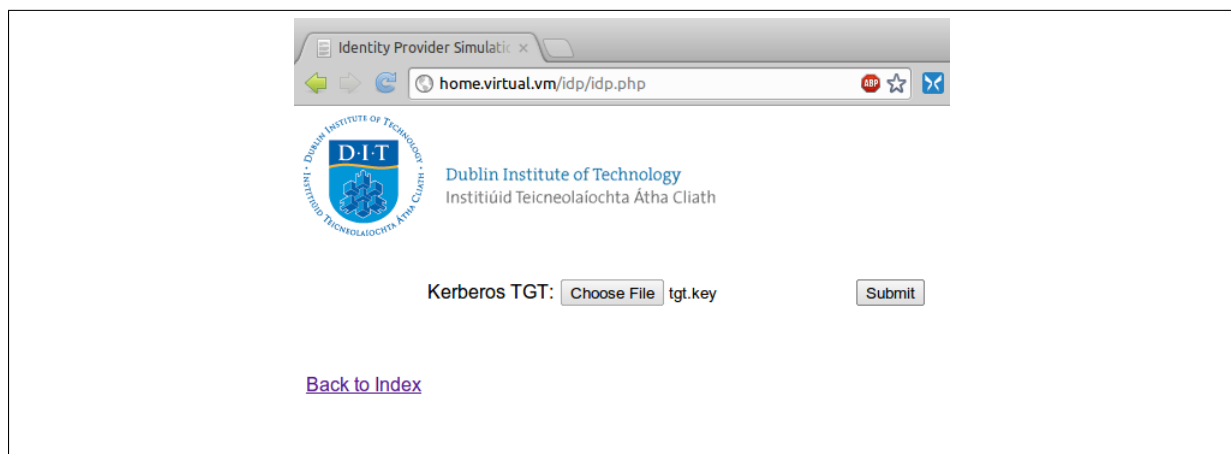


Figure 5.45: Use Kerberos Authentication Method in domain home.virtual.vm (i.e. domain Alpha)

The web-based authentication system creates an identity assertion message for the end-user after IDAWA verified the end-user's identity (using Kerberos single sign-on). The identity assertion message includes the end-user's username, the issue date of the message, the end-user's domain and the end-user's role.

The end-user accesses the web-based service/application (in domain Beta) with the identity assertion message. The web-based service/application verifies the identity assertion and displays the result. The result shows the end-user has been granted access to the web-based service/application (Figure 5.46).

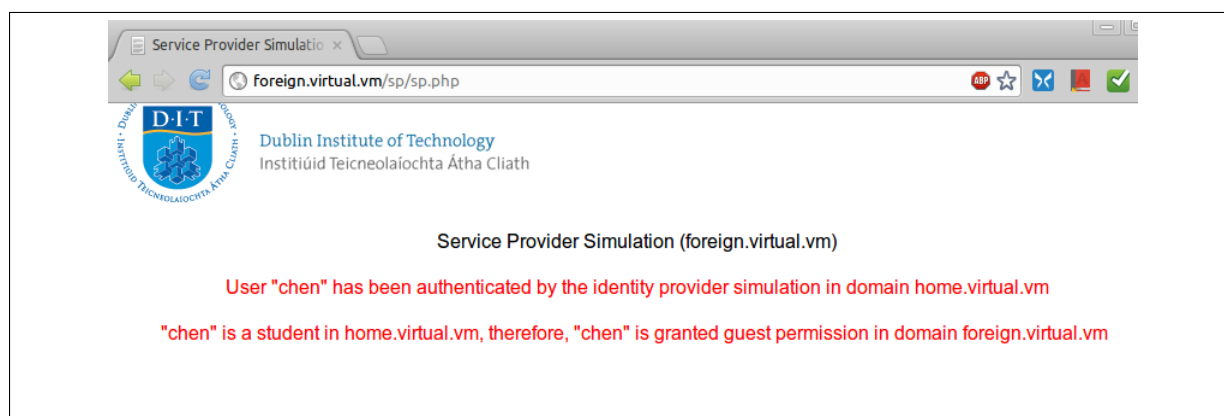


Figure 5.46: Successful access to web-based service/application in domain foreign.virtual.vm

## 5.6 Summary

This chapter presented the proof of concept implementation of the new simplified federated single sign-on system. It includes the proof of concept implementations of these two major components: Authentication Infrastructure Integration Program (AIIP) and the Integration of Desktop Authentication and Web-based Authentication (IDAWA). The AIIPs do not require any modifications to network domains' Kerberos single sign-on systems and their firewall rules during deployment, and they don't expose Kerberos single sign-on systems to the outside world. They facilitate federated single sign-on by acquiring Kerberos service tickets in different network domains and send them to the end-users; end-users can authenticate using these Kerberos service tickets in different domains. The IDAWA, which is added to the web-based authentication systems, allows end-users to upload their Kerberos ticket. End-users are authenticated using their Kerberos ticket instead of their credentials. The demonstrations showed that the end-users can acquire Kerberos tickets after desktop authentication, and the combination of AIIP and IDAWA allows the end-users to use these Kerberos tickets to access both web-based and non-web based ser-

vices/applications instead of using their credentials. Therefore, the end-users access desktop systems, web-based services/applications and non-web based services/applications using one authentication process.

A good implementation change is changing from using DOM as the interface to XML files to using an ElementTree instead. ElementTree was designed to support Python's coding style; therefore, the source code is easier to read and debug than using DOM. Another good practice during the implementation stage is inserting debug messages into source code in order to track any coding problems (e.g. tracking positions in loops).



# Chapter 6

## Research Evaluation

### 6.1 Introduction

This chapter presents the evaluation of the research, in which we use five criteria for evaluating the new simplified federated single sign-on systems. The first, second and third criteria were developed as part of this research. The fourth and fifth criteria were adopted from Painless Security (2010).

The evaluation assumes that a federation has been established between network domains. It also assumes that network domains employ their own Kerberos single sign-on systems (i.e. non-web based authentication infrastructures) for managing desktop authentications, and network domains use SAML based federated single sign-on systems for web-based federated single sign-on.

The new simplified federated single sign-on (FSSO) includes two major components: “Authentication Infrastructure Integration Program” (AIIP) and the “Integration of Desktop Authentication and Web-based Authentication” (IDAWA). The AIIP bridges the communications between end-users’ desktop systems (in one domain) and Kerberos single sign-on systems (in the other domains), and it acquires Kerberos service tickets in different domains without setting up trust between these Kerberos single sign-on systems. IDAWA is an extension to web-based authentication systems (i.e. identity provider end of

SAML based federated single sign-on system), and it supports the use of Kerberos tickets as means for authentication. The evaluation compares the new simplified federated single sign-on system (i.e. the combination of AIIP and IDAWA) with Project Moonshot, Kerberos Secure Sharing, SASL-SAML and SAML-AAI/Kerberos.

Section 6.2 presents the development of criteria. Section 6.3 evaluates Project Moonshot, SASL-SAML, Kerberos Secure Sharing, SAML-AAI/Kerberos and the new simplified federated single sign-on system using the developed criteria. Section 6.5 summarizes this chapter.

## 6.2 Criteria Development

Although this research specified security measures such as TLS and master/slave setup to address security risks (e.g. man-in-the-middle attack and single point of failure) in section 4.4.1, the security methods are not the focus of the evaluation. The goals of the new simplified federated single sign-on system is to provide single sign-on; therefore, the new criteria focuses on evaluating the usability of the systems. The five criteria are:

1. How many authentication processes are required for an end-user to access a desktop system (in the same domain as the end-user), a web-based service/application (in a separate domain) and a non-web based service/application (in a separate domain)?
2. Does the federated single sign-on system modify network domains' existing non-web based authentication infrastructures (i.e. Kerberos), or does it deploy a new authentication infrastructure?
3. Does the system expose network domains' existing authentication infrastructures to the outside world?
4. Whether the client needs to talk directly to the authentication infrastructure or whether the conversation with the authentication infrastructure is tunnelled?
5. How much new development is required?

## 6.3 Evaluation using the Developed Criteria

### 6.3.1 First Criterion

The first criterion examines the number of authentication processes that are required for an end-user to access a desktop system, web-based service/application and a non-web based services/application (Table 6.1). This criterion considers the authentication processes start at desktop authentication (i.e. an end-user walks up to a desktop system and enters his/her username and password into the login prompt). The end-user and the desktop system reside in one domain, and the web-based service/application and the non-web based service/application reside in a separate domain.

This criterion is the key to identify whether a federated single sign-on system can deliver single sign-on. This is a new criterion that considers desktop authentication as part of the federated single sign-on process, and it starts from desktop authentication and counts the number of times that an end-user needs to enter his/her username and password at a login prompt. An end-user attempts to access both web-based and non-web based services/application in both local and foreign domains.

The evaluation (using the new criterion) showed that Kerberos Secure Sharing, Project Moonshot, SASL-SAML and SAML-AAI/Kerberos require more than one authentication process for end-users to access desktop systems, web-based services/applications and non-web based services/applications. The new simplified FSSO system addresses the limitation of the existing system by providing true single sign-on for end-users. It allows end-users to access desktop systems, web-based services/applications and non-web based services/applications using one authentication process.

### 6.3.2 Second Criterion

The second criterion examines whether the federated single sign-on system modifies network domains' existing non-web based authentication infrastructures (i.e. Kerberos single sign-on systems) or deploy a new authentication infrastructure (Table 6.2).

This criterion determines the cost of deploying the federated single sign-on system. The most ideal scenario is the federated single sign-on system can use the existing non-web based authentication infrastructures without modifying them. In this way, network domains do not need to spend resources on modifying existing authentication infrastructures or deploying new authentication infrastructures.

A federated single sign-on process involves two ends: an identity provider and a service provider. The identity provider manages identities and authentications. The service provider provides computer services/applications. A network domain can be an identity provider or a service provider in a single federated single sign-on process.

The comparison showed that the simplified federated single sign-on system has advantages over the existing approach (e.g. KSS, Project Moonshot, SASL-SAML and SAML-AAI/Kerberos): it uses network domains' existing authentication infrastructures (i.e. Kerberos single sign-on systems) for authentication, and it does not modify these existing authentication infrastructures (i.e. minimum cost of modification). In addition, it only facilitates federated single sign-on and does not affect the other authentication processes (e.g. local domain single sign-on).

Project name	Comparison using the first criterion
<b>Kerberos Secure Sharing</b>	Kerberos Secure Sharing allows an end-user to access a desktop system and non-web based services/applications using one authentication process. It does not support web-based federated single sign-on; therefore, the same end-user needs to be authenticated again (i.e. second authenticate process) to access web-based services/applications.
<b>Project Moonshot</b>	Project Moonshot does not include the desktop authentication as part of federated single sign-on process, and it does not support web-based federated single sign-on. An end-user needs to be authenticated three times to access a desktop system, non-web based services/applications, and web-based services/applications.
<b>SASL-SAML</b>	SASL-SAML does not include the desktop authentication as part of federated single sign-on process. It supports both web-based and non-web based federated single sign-on. An end-user requires two authentication processes: once for desktop authentication, then once for both web-based and non-web based services/applications.
<b>SAML-AAI/Kerberos</b>	SAML-AAI/Kerberos also does not include desktop authentication as part of the federated single sign-on process. It supports both web-based and non-web based federated single sign-on. An end-user also requires two authentication processes: once for desktop authentication, then once for both web-based and non-web based services/applications.
<b>The New Simplified FSSO System</b>	The new simplified FSSO system includes desktop authentication as part of federated single sign-on process. It supports both web-based and non-web based federated single sign-on. An end-user requires one authentication process to access the desktop system, the web-based service/application and the non-web based service/application.

Table 6.1: Comparing approaches using the first criterion

### 6.3.3 Third Criterion

The third criterion examines whether the system exposes the network domains' existing authentication infrastructures (i.e. Kerberos single sign-on systems) to the outside world (i.e. Internet) (Table 6.3). Authentication infrastructures manage all of the authentication processes in network domains; therefore, network domains are reluctant to use any system that will expose their authentication infrastructures to the outside world. The most ideal scenario is the federated single sign-on system doesn't expose network domains to the outside world.

Although the new simplified federated single sign-on system relies on network domains' Kerberos single sign-on systems (similar to KSS), it does not expose them to the outside world (similar to Project Moonshot, SASL-SAML and SAML-AAI/Kerberos). It is more ideal than KSS since it only exposes AIIPs to the outside world instead of Kerberos single sign-on systems; therefore, it protects the Kerberos single sign-on systems from been accessed from the outside world.

Project name	Comparison using the second criterion
<b>Kerberos Secure Sharing</b>	Kerberos secure sharing (KSS) requires details (e.g. name, domain and network location) of Kerberos single sign-on systems to be registered in each other's Kerberos single sign-on system records, so that they can directly communicate with each other.
<b>Project Moonshot</b>	Project Moonshot does not modify network domains' existing authentication infrastructures. It deploys a separate authentication infrastructure (at both identity provider end and service provider end) for non-web based federated single sign-on.
<b>SASL-SAML</b>	SASL-SAML does not modify network domains' existing non-web based authentication infrastructures. It deploys a separate authentication infrastructure (at the identity provider end and the service provider end) for non-web based federated single sign-on.
<b>SAML-AAI/Kerberos</b>	SAML-AAI/Kerberos does not modify network domains' authentication infrastructures. It incorporates the network domains' Kerberos single sign-on system at the service provider end. It deploys a separate authentication infrastructure at the identity provider end.
<b>A new Simplified FSSO System</b>	The new simplified FSSO system doesn't request trust to be established between the Kerberos single sign-on systems; therefore, it does not require any modification to network domains' existing Kerberos single sign-on systems. It also does not deploy new authentication infrastructure.

Table 6.2: Comparing approaches using the second criterion

Project name	Comparison using the third criterion
<b>Kerberos Secure Sharing</b>	Kerberos secure sharing requires network domains to share information about their Kerberos key distribution centres. It exposes Kerberos single sign-on systems to the outside world, so that Kerberos single sign-on systems (in different domains) can directly communicate with each other.
<b>Project Moonshot</b>	Project Moonshot is separate from the existing authentication infrastructures. It does not expose the existing authentication infrastructure to the outside world.
<b>SASL-SAML</b>	SASL-SAML is separate from the existing authentication infrastructures similar to Project Moonshot. It does not expose the existing authentication infrastructure to the outside world.
<b>SAML-AAI/Kerberos</b>	SAML-AAI/Kerberos uses web-based software to communicate with Kerberos single sign-on systems, and the processes do not go beyond network domains' boundaries. It does not expose network domains' authentication infrastructures to the outside world.
<b>A new Simplified FSSO System</b>	The new simplified FSSO system uses methods that are similar to SAML-AAI/Kerberos. It doesn't require the Kerberos single sign-on systems (in different domains) to communicate with each other. It does not expose Kerberos single sign-on systems to the outside world.

Table 6.3: Comparing approaches using the third criterion

### 6.3.4 Fourth Criterion

The fourth criterion is whether the client needs to talk directly to the authentication infrastructure or the conversation with the authentication infrastructure is tunnelled (Table 6.4). Network domains usually configure their firewall (i.e. open network port 80) to support web services. They need to add new firewall rules to support new direct communications (between different domains). Tunnelling communication can change the ports



that are used by the communications. For example, tunnelling communication can change different communication protocols to use port 80 for communication. Communications through opened firewall ports are more desirable to network domains since it does not require additional firewall configurations.

Although AIIPs directly communicate with each others, they can be configured to use any network port (e.g. port 80). If firewalls open port 80 to support web-based services/applications, the new federated single sign-on system can communicate without adding new rules to the firewalls (similar to Project Moonshot, SASL-SAML and SAML-AAI/Kerberos). KSS requires network domains to open port 88 on the firewalls in order for Kerberos single sign-on systems (in different domains) to communicate with the others.

The new simplified FSSO system uses network sockets, which are in the Transport Layer of the OSI stack. A web-based service uses HTTP, which is in the Application Layer of the OSI stack. The communication of the new simplified FSSO system will not interfere with the communication of web-based services/applications.

### 6.3.5 Fifth Criterion

The fifth criterion for comparison examines the level of new developments that are required and how well the mechanism will fit into existing applications (Table 6.5). Less modification to existing applications means easier to deploy to network domains.

Although the new simplified federated single sign-on system improves upon KSS (i.e. it facilitates FSSO without modifying and exposing Kerberos single sign-on systems), the improvements require modifications to clients of non-web based services/applications. Similar to Project Moonshot and SASL-SAML, the new simplified federated single sign-on system added extra source code into the clients, so the clients will request Kerberos ticket from AIIP instead of Kerberos single sign-on systems when end-users attempt to access non-web based services/applications in other domains. The adoption of the new simplified federated single sign-on systems requires application providers to re-compile their software, and it requires network domains to re-deploy these clients.

Project name	Comparison using the fourth criterion
<b>Kerberos Secure Sharing</b>	Kerberos secure sharing requires all the Kerberos single sign-on systems to directly communicate with each other. It requires the additional port (e.g. port 88) to be opened on network domains' firewalls.
<b>Project Moonshot</b>	Project Moonshot uses EAP for federated single sign-on. EAP uses tunnel communication between clients and authentication infrastructures. It does not require additional configurations to network domains' firewalls.
<b>SASL-SAML</b>	SASL-SAML uses SAML based federated single sign-on system. SAML based federated single sign-on systems use web communication protocols and ports (e.g. port 80). It does not require any additional configurations to the firewalls since the firewalls already support web-based services/applications.
<b>SAML-AAI/Kerberos</b>	SAML-AAI/Kerberos uses web-based communication protocol for communication. It uses SAML based federated single sign-on system (i.e. web communication protocol) for federated single sign-on. It does not require any additional configurations to the firewalls similar to SASL-SAML.
<b>A new Simplified FSSO System</b>	The new simplified federated single sign-on system uses Authentication Infrastructure Integration Program (AIIP) to manage non-web based federated single sign-on. AIIP use direct communication, and it can be configured to use any network port (e.g. port 80). It does not require any additional configurations to firewalls.

Table 6.4: Comparing approaches with the fourth criterion

Project name	Comparison using the fifth criterion
<b>Kerberos Secure Sharing</b>	Kerberos secure sharing uses Kerberos as it's primary authentication system. Kerberos has been established as the predominant single sign-on system (Kerberos support has been built into Microsoft Windows systems, GNU/Linux and Mac OS X). Existing applications don't need to be modified to use KSS.
<b>Project Moonshot</b>	Project Moonshot proposes to modify the existing user agents to use EAP for federated single sign-on. This means all non-web based applications creators needs to modify their existing user agents.
<b>SASL-SAML</b>	SASL-SAML is similar to Project Moonshot. It modifies the existing user agents to communicate with a web-based identity provider through SASL.
<b>SAML-AAI/Kerberos</b>	SAML-AAI/Kerberos requires the creation of new web-based applications to access non-web based service applications.
<b>A new Simplified FSSO System</b>	The new simplified federated single sign-on system added extra functionalities to existing clients of non-web based services/applications, so the clients can request Kerberos tickets from AIIP (instead of Kerberos single sign-on systems) when end-users attempt to access non-web based services/applications in other domains.

Table 6.5: Comparing approaches with the fifth criterion

## 6.4 Comparison Overview

This section presents the comparison overview using the same five evaluation criteria (Table 6.6). It compares new simplified federated single sign-on system with Kerberos secure sharing (KSS), Project Moonshot, SASL-SAML and SAML-AAI/Kerberos.

The new simplified federated single sign-on (FSSO) system is the only system that supports desktop systems, web-based services/applications and non-web based services/applications (according to the evaluation). This system can deliver single sign-on

for end-users to access desktop systems, web-based services/applications and non-web based services/applications according to the evaluation. The other approaches (e.g. KSS, Project Moonshot, SASL-SAML and SAML-AAI/Kerbers) require more than one authentication processes for an end-users to access diverse computer services/applications.

Although the new simplified single sign-on system uses network domains' existing Kerberos single sign-on systems similar to KSS, it improves upon KSS by requesting Kerberos tickets from Kerberos single sign-on systems (in different network domains) without setting up trust between these Kerberos single sign-on systems. The trust is established between the AIIPs; therefore, the existing Kerberos single sign-on systems are not modified and not exposed to the outside world. The system also does not require new modifications to network domains' firewall rules.

The new simplified single sign-on system, however, requires new modifications to user agents (i.e. clients) of non-web based services/applications. An extra model was added to the user agents, and it allows the user agents to acquire Kerberos tickets (that were created in foreign network domains) via the new simplified federated single sign-on system when end-users requested non-web based services/applications in foreign network domains. This system requires active support from application creators and network domains (similar to Project Moonshot, SASL-SAML and SAML-AAI/Kerberos). This is discussed in section 7.4.

Security attacks such as replaying attack, eaves dropping and man in the middle attacks can be used to compromise the communications between AIIPs and federations. These security attacks can also compromises the the digital signatures (i.e. hash strings) that were used for authenticating end-users in the single sign-on processes. Replay attack allows the attackers to use the authentication information that were previously used for authenticating end-users, eaves dropping can steal the end-users' identity assertion information when they are transferred between federations, and man-in-the-middle attack allows the attackers to inject arbitrary messages between two end points (e.g. two AIIPs) or steal authentication messages.

The new simplified FSSO system requires additional security protocols (e.g. timestamping, TLS protocol and digital certificate) before the new simplified FSSO system can be deployed into universities/institutions (e.g. DIT, UCD and Trinity college). Timestamping limits the life span of Kerberos tickets and identity assertion messages; therefore, it limits the risk of replay attacks. TLS protocol use asymmetric encryption and symmetric encryption to encrypt the transferred messages. The attackers can not read the encrypted messages (e.g. authentication messages and identity assertions); therefore, TLS reduces the risks of eaves dropping. Digital certificates (e.g. X.509) can be used to prevent spoofing attacks (e.g man-in-the-middle) between the AIIPs (AIIP-AIIP and AIIP to end-users) and communication between federations.

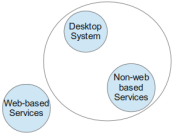
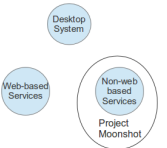
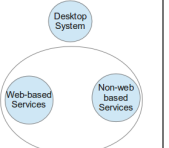
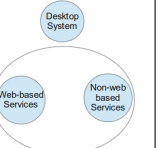
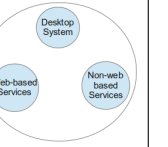
Evaluation Criteria	Keberos Secure Sharing (KSS)	Project Moonshot	SASL-SAML	SAML-AAI/Kerberos	New Simplified FSSO System
Support of desktop system, web-based services/applications and non-web based services/applications					
Numbers of authentication processes (including desktop authentication) for accessing two (web-based and non-web based) applications	Two	Three	Two	Two	One
Modifications to existing non-web based authentication infrastructure or deploying new authentication infrastructure	Yes	Yes	Yes	Yes	No
Exposing existing authentication infrastructure to the outside world	Yes	No	No	No	No
Modifications to firewall rules	Yes	No	No	No	No
Modifications to existing computer services/applications	No	Yes	Yes	Yes	Yes

Table 6.6: Comparison of the New Simplified FSSO System with Existing Approaches

## 6.5 Summary

This chapter used five criteria to evaluate the new simplified federated single sign-on (FSSO) system (i.e. the combination of AIIP and IDAWA). The evaluation shows that the new simplified FSSO system addresses the limitation of existing FSSO systems. The end-users, who use the new simplified FSSO system, can access desktop systems, web-based services/applications and non-web based services/applications using one authentication process. In addition, the new system is simpler to deploy than other existing approaches: it can incorporate network domains' existing non-web based authentication infrastructures (i.e. Kerberos single sign-on systems) without requiring additional modifications to them, and it does not require modifications to network domains' firewall rules.



# Chapter 7

## Discussion and Conclusion

### 7.1 Introduction

This research found that current federated single sign-on (FSSO) systems (e.g. Project Moonshot, SASL-SAML, Kerberos Secure Sharing and SAML-AAI/Kerberos) do not allow end-users to access desktop systems, web-based services/applications and non-web based services/applications (in multiple domains) using one authentication process.

The current FSSO systems partially incorporates the authentication systems of desktop systems, web-based services/applications and non-web based services/applications (Figure 7.1): Project Moonshot supports non-web based services/applications. Keberos secure sharing (KSS) supports desktop systems and non-web based services/applications. SASL-SAML and SAML-AAI/Kerberos support both web-based and non-web based services/applications. The limitations of existing FSSO systems mean end-users need to go through multiple authentication systems in order to access these services.

This research developed a new simplified FSSO system that incorporates the authentication systems of desktop systems, web-based services/applications and non-web based services/applications. The system includes the developments of two major components: an “Authentication Infrastructure Integration Program” (AIIP) and an “Integration of Desktop Authentication and Web-based Authentication” (IDAWA).

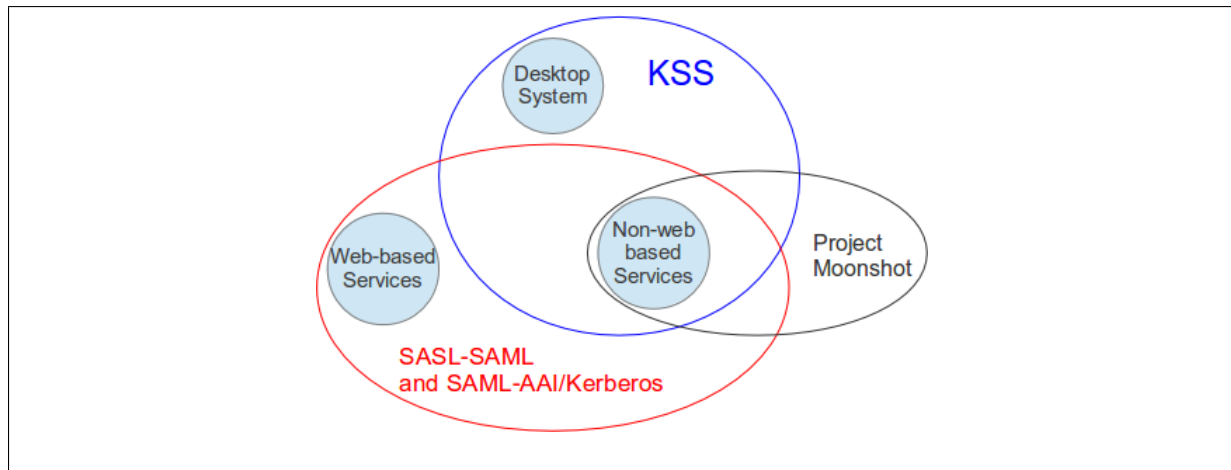


Figure 7.1: Existing FSSO systems overview

The Authentication Infrastructure Integration Program (AIIP) incorporates the authentication systems of desktop systems and non-web based services/applications. The AIIPs (in different network domains) bridge communications between end-users' desktop systems in one domain and Kerberos single sign-on systems in the other domains. They facilitate non-web based FSSO by seamlessly acquiring Kerberos tickets in different domains for the end-users (that have been authenticated by Kerberos single sign-on systems) in one domain, so that the end-users can proceed to access non-web based services/applications in different domains using these tickets (instead of requiring re-authentication).

The Integration of Desktop Authentication and Web-based Authentication (IDAWA) incorporates the authentication systems of desktop systems and web-based services/applications. The IDAWA is an extension to the web-based authentication system (of SAML based federated single sign-on system). It allows end-users to use their Kerberos ticket as means of authentication (instead of their credentials); therefore, the end-users (who had gained access to desktop systems) can access web-based services/applications (in multiple domains) without requiring re-authentication.

The new simplified federated single sign-on system (i.e. the combination of AIIP and IDAWA) allows end-users to access desktop systems, web-based services/applications and non-web based services/applications using one authentication process; however, it's open for security attacks such as man-in-the-middle attack and single point of failure. To address the security risks, this research has specified the following security measures:

- The AIIPs use Transport Layer Security (TLS) (Dierks & Rescorla, 2008) to defend against man-in-the-middle attack. TLS uses asymmetric cryptography for key exchange and symmetric encryption for confidentiality, and message authentication codes for message integrity. It can secure the communication between the AIIPs in different domains.
- Each domain should have a primary AIIP system and multiple secondary AIIPs in order to defend against a single point of failure. If the primary AIIP has failed, the secondary AIIPs can take over the non-web based federated single sign-on processes.

## 7.2 Contributions

### Contribution 1

The major contribution of this research is the development of the new simplified FSSO system. The new system addresses the limitation of exiting FSSO systems – they do not incorporate the authentication systems of desktop systems, web-based services/applications and non-web based services/applications. The two major components (of the new simplified FSSO system) are the “Authentication Infrastructure Integration Program” (AIIP) and the “Integration of Desktop Authentication and Web-based Authentication” (IDAWA). By combining AIIP and IDAWA, the new simplified FSSO system allows the end-users to access desktop systems, web-based services/applications and non-web based services/applications using one authentication process.

- The AIIP acquires Kerberos tickets (for end-users who have been authenticated by a Kerberos single sign-on system in one domain) from Kerberos single sign-on systems in different network domains without establishing trust between these Kerberos single sign-on systems.
- The IDAWA is an extension to the web-based authentication system (i.e. web portal), and it authenticates end-users by verifying their Kerberos tickets.

## Contribution 2

The research has developed three new evaluation criteria (in addition to the two existing criteria) to compare the new simplified FSSO system with known FSSO systems (e.g. Project Moonshot, Kerberos secure sharing, SASL-SAML and SAML-AAI/Kerberos). These criteria determine which FSSO system can deliver true single sign-on (i.e. allowing end-users to access desktop systems, web-based services/applications and non-web based services/applications using one authentication process). They also determine which FSSO system requires the least active support and commitment from network domains and has lower risk to be compromised by attackers from the outside world (i.e. the Internet).

## 7.3 Critical Review

The work (i.e. the new simplified FSSO system) presented in this thesis is a step towards achieving true single sign-on. The primary advantage of the new simplified FSSO system is that it allows end-users to access a desktop system, web-based services/applications and non-web based services/applications using one authentication process. For example, the end-users (who have gained accesses to desktop systems in DIT) can proceed to access IEEE Xplore Digital Library and remote Linux systems in Trinity college without requiring re-authentication. Any universities and institutions can use the new simplified FSSO system to provide true single sign-on for their students and staffs.

Although the design of AIIP (i.e. a component of the new simplified FSSO system) draws extensively from Kerberos secure sharing (KSS) and SAML-AAI/Kerberos, the AIIP has the following advantages over KSS and SAML-AAI/Kerberos. The AIIP can be deployed into universities/institutions (e.g. DIT, UCD and Trinity college) without requiring modifications to their existing Kerberos single sign-on systems and firewall rules. Therefore, the universities/institutions can provide single sign-on without exposing their Kerberos single sign-on systems to each other, without setting up trust between their Kerberos single sign-on systems, and without modifying their firewall rules.

The new simplified FSSO system currently requires modifications to the user agents of non-web based services/applications unlike KSS. This is because the user agents request Kerberos tickets from Kerberos single sign-on systems in the foreign domains (instead of the AIIPs); therefore, an extra model was added to the user agents. The extra model allows the user agents to request Kerberos tickets from the AIIPs when end-users are requesting non-web based services/applications in foreign network domains. Additional functionalities that allow the AIIPs to masquerade as Kerberos single sign-on systems (in the foreign network domains) are required to address this limitation. By masquerading as Kerberos single sign-on systems (in foreign network domains), the user agents will assume that they are requesting Kerberos tickets from Kerberos single sign-on systems, but they are in fact requesting them from the AIIPs.

As described in the research evaluation (section 6.4), the new simplified FSSO system requires additional security protocols (e.g. timestamping, TLS protocol and digital certificate) before it can be deployed into universities/institutions (e.g. DIT, UCD and Trinity college). Timestamping limits the life span of Kerberos tickets and identity assertion messages; therefore, it limits the risk of replay attacks. TLS protocol use asymmetric encryption and symmetric encryption to reduces man-in-the-middle attacks, while digital certificates (e.g. X.509) can be used to prevent spoofing attacks between the AIIPs (AIIP-AIIP and AIIP to end-users) and communication between federations.

## 7.4 Future Work

Future work includes adding security protocols (e.g. timestamping, TLS protocol) to the new simplified FSSO system in order to deploy it into the real world (e.g. DIT, UCD). Time-stamping can limit the valid period of the FSSO requests (e.g. identity assertion messages) and replies (e.g. Kerberos tickets); therefore, it can reduce the threat of replay attacks on the AIIPs. The TLS protocol provides symmetric encryption, asymmetric encryption and message authentication code (Freier et al., 2011). The symmetric encryption technology ensures the privacy of the communications between the AIIPs by encryption

the messages with a shared key. The asymmetric encryption technology ensures the security of the symmetric encryption technology's key exchange process. The message authentication code ensures the integrity and authenticity of the messages (Krawczyk et al., 1997). The use of time-stamping and TLS protocol can increase the security of the message exchanges between the AIIPs.

Additional functionalities that allow the AIIPs (in the home network domain) to masquerade as Kerberos single sign-on systems (in the foreign network domains) may be included in the future work. These functionalities can reduce the modifications to the user agents by allowing the AIIP (in the same domain as the user agents) to pretend to be Kerberos single sign-on systems in the foreign network domains, so that user agents will send requests (e.g. requesting ticket granting tickets and service tickets) to the AIIP instead of the Kerberos single sign-on systems.

Microsoft Windows operating systems use authentication systems (i.e. Active Directory) that were developed based on the Kerberos single sign-on systems; therefore, Windows supports Kerberos single sign-on. Although the proof of concept implementation of the new simplified federated single sign-on system was developed using the Linux based environment, the concept should be able to support Window-based operating systems. One of the future work topics in this area will involve experiments of the concept of the new simplified federated single sign-on system in a Windows environment, and determine the requirements for allowing the AIIP to support the single sign-on protocols of Active Directory.

User-centric federated single sign-on (UFed SSO) was proposed by Suriadi et al. (2009), and it allows end-users to manage their own identity information instead of network domains' identity providers. This approach allows end-users to have an effective control over the use and management of their private identification information. One of the future work topics in this area will involve incorporating the concept of UFed SSO into the new simplified FSSO system. By allowing the end-users to provide identity assertions for themselves, the new simplified FSSO system can be used by end-users that do not belong to specified network domains.

# References

- Local and Metropolitan Area Networks: Port-Based Network Access Control, IEEE Standard 802.1X.
- Standard for Local and Metropolitan Area Networks: Overview and Architecture, IEEE Standard 802, 1990.
- Unapproved Draft Supplement to Standard for Telecommunications and Information Exchange Between Systems - LAN/MAN Specific Requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications: Specification for Enhanced Security, IEEE Draft 802.11i (work in progress).
- Aboba, B., Blunk, L., Vollbrecht, J., Carlson, J., and Levkowetz, E. H., 2004. Extensible Authentication Protocol (EAP). *Request for Comments: 3748*, .
- Ahituv, N., Lapid, Y., and Neumann, S., 1987. Verifying the Authentication of an Information System User. *Computers & Security*, **6**(2):152 – 157.
- Ahn, G.-J. and Ko, M., 2007. User-centric Privacy Management for Federated Identity Management. In *Collaborative Computing: Networking, Applications and Worksharing, 2007. CollaborateCom 2007. International Conference on*, pages 187 –195.
- Anderson, J., 1973. Information Security in a Multi-User Computer Environment. *Advances in Computers*, **12**:1–35.
- Balasubramaniam, S., Lewis, G., Morris, E., Simanta, S., and Smith, D., 2009. Identity Management and Its Impact on Federation in a System-of-systems Context. In *Systems Conference, 2009 3rd Annual IEEE*, pages 179 –182.

- Benantar, M., 2005. *Access Control Systems: Security, Identity Management and Trust Models*. Springer.
- Bertino, E. and Takahashi, K., 2010. *Identity Management: Concepts, Technologies, and Systems (Information Security & Privacy)*. Artech House Publishers.
- Bhargav-Spantzel, A., Camenisch, J., Gross, T., and Sommer, D., 2007. User centrality: A Taxonomy and Open Issues. *Journal of Computer Security*, **15**.
- Bishop, M., 2003. What is computer security? *Security Privacy, IEEE*, **1**(1):67 – 69.
- Bishop, M., 2005. *Introduction to Computer Security*. Addison-Wesley.
- Bleazard, D. J. and Marceau, J., 2002. One user, One Password: Integrating Unix Accounts and Active Directory. :5–8.
- Bray, T., Paoli, J., Sperberg-McQueen, C. M., and E. Maler, F. Y., 2008. *Extensible Markup Language (XML) 1.0*. W3C Recommendation, <http://www.w3.org/TR/xml/>., fifth edition edition.
- Buecker, A., Ashley, P., and Readshaw, N., 2008. Federated Identity and Trust Management. *IBM Red Paper*, .
- Carter, G., 2003. *LDAP System Administration*. O'Reilly.
- Chappell, D., 2006. Introducing Windows CardSpace. *Retrieved from <http://msdn.microsoft.com/en-us/library/Aa480189> in 2012*, .
- Chokhani, S. and Ford, W., 1999. Internet X.509 Public Key Infrastructure Certificate Policy and Certification Practices Framework. *Request for Comments: 2527*, Retrieved from <http://www.ietf.org/rfc/rfc2527.txt> in 2012, .
- Dierks, T. and Rescorla, E., 2008. The Transport Layer Security (TLS) Protocol Version 1.2. *RFC 5246*, .
- Don and Smith, 2008. The Challenge of Federated Identity Management. *Network Security*, **2008**(4):7 – 9.



- Dunleavy, P., Margetts, H., Bastow, S., and Tinkler, J., 2006. Digital Era Governance. *Oxford University Press*, :251.
- El-Hadidi, M., Hegazi, N., and Aslan, H., 1997. Performance analysis of the Kerberos protocol in a distributed environment. In *Computers and Communications, 1997. Proceedings., Second IEEE Symposium on*, pages 235 –239.
- El Maliki, T. and Seigneur, J.-M., 2007. A Survey of User-centric Identity Management Technologies. In *Emerging Security Information, Systems, and Technologies, 2007. SecureWare 2007. The International Conference on*, pages 12 –17.
- Ellison, C. and Schneier, B., 2000. Ten Risks of PKI: What You’re not Being Told about Public Key Infrastructure. *COMPUTER SECURITY JOURNAL*, **16**:1–8.
- Freier, A., Karlton, P., and Kocher, P., 2011. The Secure Sockets Layer (SSL) Protocol Version 3.0. *Request for Comments: 6101*, .
- Glasser, U. and Vajihollahi, M., 2008. Identity Management Architecture. In *Intelligence and Security Informatics, 2008. ISI 2008. IEEE International Conference on*, pages 137 –144.
- Godik, S. and Moses, T., 2003. eXtensible Access Control Markup Language (XACML) Version 1.1. Technical report, Organization for the Advancement of Structured Information Standards.
- Google Inc., 2012. SAML Single Sign-On (SSO) Service for Google Apps - [https://developers.google.com/google-apps/sso/saml\\_reference\\_implementation](https://developers.google.com/google-apps/sso/saml_reference_implementation). Technical report.
- Gridwise Tech, 2010. Kerberos: Secure Sharing of Distributed Resources. Technical report.
- Haller, N. and Metz, C., 1996. A One-Time Password System. *RFC 1938*, .
- Halperin, R. and Backhouse, J., 2008. A Roadmap for Research on Identity in the Information Society. *Identity in the Information Society*, **1**:71–87. 10.1007/s12394-008-0004-0.

- HEAnet, 2008. Board Approves New 5-year Strategic Plan. Technical report.
- HEAnet, 2012. HEAnet, Ireland’s National Education & Research Network. *Retrieved from <http://www.heanet.ie> in 2012*, .
- Howes, T. and Smith, M., 1997. *LDAP: Programming Directory-enabled Applications with Lightweight Directory Access Protocol*. Sams Publishing.
- Howes, T., Smith, M., and Good, G. S., 2003. *Understanding and Deploying LDAP Directory Services*. Addison-Wesley, 2nd edition.
- Howlett, J., 2010. Project Moonshot. *Briefing paper for IEFT, Maastricht 2010*, .
- Howlett, J., 2011. Project Moonshot. *Retrieved from <http://www.project-moonshot.org/>*, .
- Hunter, L. E., 2006. *Active Directory Cookbook*. O’Reilly, 2nd edition.
- Internet2, 2012. Shibboleth. *Retrieved from <http://shibboleth.net/>*, .
- Jain, A., Bolle, R., and Pankanti, S., 1999. *Biometrics: personal identification in networked society*. The Kluwer international series in engineering and computer science. Kluwer.
- JANET, 2012. JANET, the UK’s education and research network. *Retrieved from <http://www.ja.net/> in 2012*, .
- Jason and Goode, 2012. The Importance of Identity Security. *Computer Fraud & Security*, **2012**(1):5 – 7.
- Jøsang, A. and Pope, S., 2005. User Centric Identity Management. *AusCERT Conference*, .
- Jøsang, A., Zomai, M. A., and Suriadi, S., 2007. Usability and Privacy in Identity Management Architectures. In *Proceedings of the fifth Australasian symposium on ACSW frontiers - Volume 68*, ACSW ’07, pages 143–152, Darlinghurst, Australia, Australia. Australian Computer Society, Inc.

- Kohl, J. T., Neuman, B. C., and Ts'o, T. Y., 1991. The Evolution of the Kerberos Authentication Service. pages 78–94. IEEE Computer Society Press.
- Krawczyk, H., Bellare, M., and Canetti, R., 1997. HMAC: Keyed-Hashing for Message Authentication. *Request for Comments: 2104*, .
- Lee, H., Jeun, I., and Jung, H., 2009. Criteria for Evaluating the Privacy Protection Level of Identity Management Services. In *Emerging Security Information, Systems and Technologies, 2009. SECURWARE '09. Third International Conference on*, pages 155 –160.
- Linn, J., 2000. Generic Security Service Application Program Interface Version 2, Update 1. Retrieved from <http://tools.ietf.org/html/rfc2743> in 2012, .
- Madsen, P., Maler, E., Wisniewski, T., Nadalin, T., Cantor, S., Hodges, J., and Mishra, P., 2005. SAML V2.0 Executive Overview. .
- Maler, E., Mishra, P., and Philpot, R., 2003. Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML) V1.1. *OASIS Standard*, .
- Maler, E. and Reed, D., 2008. The Venn of Identity: Options and Issues in Federated Identity Management. *Security Privacy, IEEE*, **6**(2):16 –23.
- Mather, T., Kumaraswamy, S., and Latif, S., 2009. *Cloud Security and Privacy: An Enterprise Perspective on Risks and Compliance*. O'Reilly Series. O'Reilly Media.
- McRae, M., 2009. Approval of WS-Federation v1.2 as an OASIS Standard. Technical report, OASIS.
- Melnikov, A. and Zeilenga, K., 2006. Simple Authentication and Security Layer. *Request for Comments: 4422*, .
- Microsoft, 2012a. Kerberos Protocol Extensions: Service for User and Constrained Delegation Protocol Specification. Technical report, Retrieved at <http://msdn.microsoft.com/en-us/library/cc246071>

- Microsoft, 2012b. WS-Federated Authentication Module Overview. *Retrieved from <http://msdn.microsoft.com/en-us/library/ee517293.aspx>, .*
- Migeon, J.-Y., 2008. The MIT Kerberos Administrators How-to Guide. *MIT Kerberos Consortium*, :62.
- Miller, S. P., Neuman, B. C., Schiller, J. I., and Saltzer, J. H., 1988. Kerberos Authentication and Authorization System. In *In Project Athena Technical Plan*.
- MIT, 2008. Best Practices for Integrating Kerberos into Your Application. *MIT Kerberos Consortium*, .
- Miyata, T., Koga, Y., Madsen, P., ichi Adachi, S., Tsuchiya, Y., Sakamoto, Y., and Takahashi, K., 2006. A Survey on Identity Management Protocols and Standards. *IEICE TRANSACTIONS on Information and Systems*, **E89-D No.1**:112–123.
- Mont, M. C., Bramhall, P., and Pato, J., 2003. On Adaptive Identity Management: The Next Generation of Identity Management Technologies. .
- Needham, R. M. and Schroeder, M. D., 1978. Using Encryption for Authentication in Large Networks of Computers. *Commun. ACM*, **21**:993–999.
- Neuman, B. and Ts'o, T., 1994. Kerberos: an authentication service for computer networks. *Communications Magazine, IEEE*, **32**(9):33 –38.
- Newham, C. and Rosenblatt, B., 2005. *Learning the bash shell, third edition*. O'Reilly Media, Inc., 3 edition.
- NIST, 2004. NIST Brief Comments on Recent Cryptanalytic Attacks on Secure Hashing Functions and the Continued Security Provided by SHA-1. .
- OASIS, 2010. SAML OASIS Standard. *Retrieved from <http://saml.xml.org>, .*
- Painless Security, 2010. Project Moonshot: Feasibility Analysis. .
- Papez, R., 2009. SAML-AAI/Kerberos integration.

- Patterson, P., Thiyagarajan, P. V., and Sum, M., 2004. Federated Identity: Single Sign-On Among Enterprises. :1–12.
- Paul and Madsen, 2004. Federated Identity and Web Services. *Information Security Technical Report*, **9**(3):56 – 65.
- Pfleeger, C. P. and Pfleeger, S. L., 2006. *Security in Computing*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 4th edition.
- Project, L. A., 2003. Introduction to the Liberty Alliance Identity Architecture. .
- Ragouzis, N., Hughes, J., Philpott, R., Maler, E., Madsen, P., and Scavo, T., 2008. Security Assertion Markup Language (SAML) V2.0 Technical Overview. *Committee Draft 02*, .
- Rashid, F. Y., Accessed on Jan 18th 2012. Zappos Breach Illustrate the Need for Stronger Password Rules. <http://www.eweek.com/c/a/Security/Zapps-Breach-Illustrate-the-Need-for-Stronger-Password-Rules-672979/?kc=rss>, .
- Richards, J., Allen, R., and Lowe-Norris, A. G., 2006. *Active Directory*. O’Reilly, 3rd edition.
- Salomon, D., 2005. *Foundations of Computer Security*. Springer; 1st Edition.
- Saltzer, J. and Schroeder, M., 1975. The Protection of Information in Computer Systems. *Proceedings of the IEEE*, **63**(9):1278 – 1308.
- Sato, H. and Nishimura, T., 2011. Federated Authentication in a Hierarchy of IdPs by Using Shibboleth. In *Applications and the Internet (SAINT), 2011 IEEE/IPSJ 11th International Symposium on*, pages 327 –332.
- Schneier, B., 1998. Cryptographic Design Vulnerabilities. *Computer*, **31**(9):29 – 33.
- Schuman, E., 2006. Consumers Resist Retail Biometrics. *eWeek*, .
- Seigneur, J.-M., 2005. Trust, security and privacy in global computing Ph.D. thesis. *Dublin: Trinity College*, .

- Shim, S., Bhalla, G., and Pendyala, V., 2005. Federated Identity Management. *Computer*, **38**(12):120 – 122.
- Simpson, W., 1994. The Point-to-Point Protocol (PPP). *Request for Comments: 1661*, .
- Smith, R. E., 2001. *Authentication: From Passwords to Public Keys*. Addison-Wesley Professional, 1st edition edition.
- Solomon, D., 1998. The Windows NT Kernel Architecture. *Computer*, **31**(10):40–47.
- Sullivan, R. K., 2005. The Case for Federated Identity. *Network Security*, **2005**(9):15 – 19.
- Sun Microsystems, 2004. Securing your Business Performance Management (BPM) Environment with Identity Solutions from Sun Microsystems. *Retrieved from <http://developers.sun.com/identity/reference/whitepapers/hyperion.pdf> in 2012*, .
- Suriadi, S., Foo, E., and Jøsang, A., 2009. A User-centric Federated Single Sign-on System. *Journal of Network and Computer Applications*, **32**(2):388 – 401.
- SWITCHaai, 2012. *Shibboleth Demo*. Swiss Education & Research Network.
- Takaaki, K., Hiroaki, S., Noritoshi, D., and Ken, M., 2011. Design and Implementation of Web Forward Proxy with Shibboleth Authentication. In *Applications and the Internet (SAINT), 2011 IEEE/IPSJ 11th International Symposium on*, pages 321 –326.
- Thomas, I. and Meinel, C., 2009. Enhancing Claim-Based Identity Management by Adding a Credibility Level to the Notion of Claims. In *Services Computing, 2009. SCC '09. IEEE International Conference on*, pages 243 –250.
- Vacca, J. R., Seigneur, J.-M., and Maliki, T. E., 2010. *Managing Information Security*. Syngress Media.
- Voglmaier, R. E., 2003. *The ABCs of LDAP: How to Install, Run, and Administer LDAP Services*. CRC Press.
- Walla, M., 2000. Kerberos Explained. *Windows 2000 Advantage*, .

- Wang, X., Schulzrinne, H., Kandlur, D., and Verma, D., 2008. Measurement and Analysis of LDAP Performance. *IEEE/ACM Trans. Netw.*, **16**(1):232–243.
- Wason, T., Cantor, S., Hodges, J., Kemp, J., and Thompson, P., 2003. Liberty ID-FF Architecture Overview. Version: 1.2-errata-v1.0. Retrieved from <http://projectliberty.org/liberty/content/download/318/2366/file/draft-liberty-idff-arch-overview-1.2-errata-v1.0.pdf> in 2012, .
- Wierenga, K. and Lear, E., 2012. A SASL and GSS-API Mechanism for SAML. *Internet Engineering Task Force (IETF)*, .
- Ying, W., 2010. Research on Multi-level Security of Shibboleth Authentication Mechanism. In *Information Processing (ISIP), 2010 Third International Symposium on*, pages 450 –453.
- Ylonen, T. and Lonvick, C., 2006. The Secure Shell (SSH) Authentication Protocol. *Internet Official Protocol Standards*, .
- Zhang, J., Lang, B., and Duan, Y., 2011. An XML Data Placement Strategy for Distributed XML Storage and Parallel Query. In *Parallel and Distributed Computing, Applications and Technologies (PDCAT), 2011 12th International Conference on*, pages 433 –439.

# Appendix A

## System Specifications

### A.1 Hardware Specifications

The new simplified federated single sign-on system was developed on a Sun Ultra 40M2 Workstation and HP-ProBook 6550b (Table A.1). The hardware specifications are as following:

In addition to the hardware that were used for developing the system, the virtual machines' control centre and the XenCenter 6.0, was installed on a Dell Optiplex GX260. Hardware includes 2.40 GHz Intel(R) Pentium(R) 4 CPU, 512 MB RAM, Intel(R) PRO/1000MT Network Connection and Intel(R) 82845G Graphic Controller. A Linksys Wireless-N Gigabit Router (WRT350N) is used to create network connections between applications and machines. The router issues IP addresses to all of the machines (physical and virtual). The router has the IP address 192.168.1.1. The IP address range for all the machines is from 192.168.1.10 to 192.168.1.254.

### A.2 Software Specifications



	<b>Sun Ultra 40M2 Workstation</b>	<b>HP-ProBook 6550b</b>
Processor	2 x Dual-Core AMD Opteron (tm) Processor 2222	4 x Intel(R) Core(TM) i3 CPU M 370
Memory	4096MB	2048MB
Hard disk drive (HDD)	128GB	300GB
Networking	2 x Bridge: nVidia Corporation MCP55 Ethernet	<ul style="list-style-type: none"> <li>• Intel Corporation 82577LC Gigabit Network Connection</li> <li>• Broadcom Corporation BCM4313 802.11b/g/n Wireless LAN Controller</li> </ul>
Video	nVidia Corporation G92 [Quadro FX 3700]	Intel Corporation Core Processor Integrated Graphics Controller

Table A.1: IBM eServer Hardware Overview.

	<b>DNS server</b>	<b>Beta Virtual Server</b>
Processor	Single CPU	Single CPU
Memory	512MB	1024MB
Hard disk drive (HDD)	8GB	8GB
Networking	Bridge Connection	Bridge Connection
Operating System	Ubuntu Server 10.04.4 LTS 32bit	Ubuntu Server 10.04.4 LTS 32bit
Key Software	<ul style="list-style-type: none"> <li>• bind9</li> <li>• openssh-server</li> </ul>	<ul style="list-style-type: none"> <li>• apache2</li> <li>• php5</li> <li>• openssh-server</li> <li>• python2.6</li> </ul>
IP Address	192.168.1.24	192.168.1.17
Domain Name	hostmaster.virtual.vm	foreign.virtual.vm
MAC	CA:78:6B:9B:9B:5D	B2:CD:11:E2:F5:87

Table A.2: Sun Ultra 40M2 Workstation Configuration.

Operating System	<b>DNS server</b> Xubuntu Desktop 12.04 LTS 64bit
Key Software	<ul style="list-style-type: none"> <li>• apache2</li> <li>• php5</li> <li>• python2.7</li> </ul>
IP Address	192.168.1.59
Domain Name	home.virtual.vm
MAC	AC:81:12:43:58:7C

Table A.3: HP Probook 6550b Software Configuration.

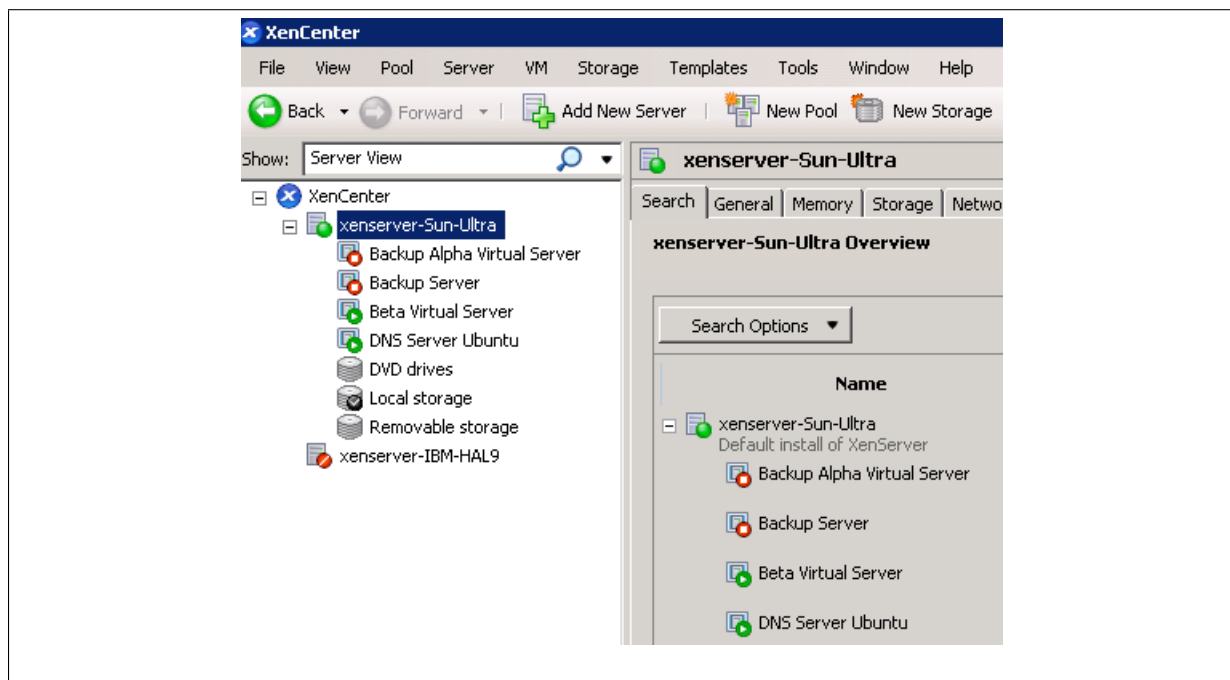


Figure A.1: Dell Optiplex GX260 configuration.

# Appendix B

## List of Security Technologies

### B.1 The Simple Authentication and Security Layer (SASL)

The Simple Authentication and Security Layer (SASL)(Melnikov & Zeilenga, 2006) is a framework for providing authentication and data security services in connection-oriented protocols via replaceable mechanisms. SASL provides a structured interface between protocols and mechanisms. SASL also provides a protocol for securing subsequent protocol exchanges within a data security layer. The data security layer can provide data integrity, data confidentiality, and other services.

SASL's design is intended to allow new protocols to reuse existing mechanisms without requiring redesign of the mechanisms and allows existing protocols to make use of new mechanisms without redesign of protocols.

SASL is conceptually a framework that provides an abstraction layer between protocols and mechanisms as illustrated in figure B.1.

It is through the interfaces of this abstraction layer that the framework allows any protocol to utilize any mechanism. While this layer does generally hide the particulars of protocols from mechanisms and the particulars of mechanisms from protocols, this layer does not generally hide the particulars of mechanisms from protocol implementations.

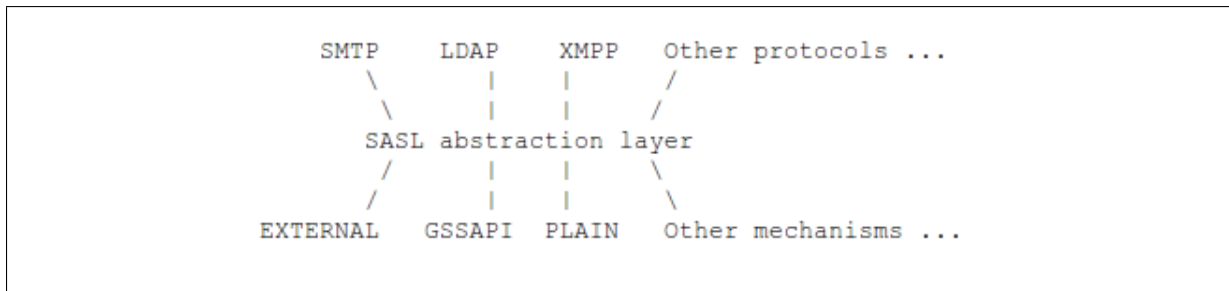


Figure B.1: SASL is conceptually a framework that provides an abstraction layer between protocols and mechanisms(Melnikov & Zeilenga, 2006)

For example, different mechanisms require different information to operate, some of them use password-based authentication, some of them require realm information, others make use of Kerberos tickets, certificates, etc. Also, in order to perform authorization, server implementations generally have to implement identity mapping between authentication identities, whose form is mechanism specific, and authorization identities, whose form is application protocol specific.

## B.2 Generic Security Service Application Program Interface (GSS-API)

Generic Security Service Application Program Interface (GSS-API)(Linn, 2000) Version 2 provides security services to callers in a generic fashion, supportable with a range of underlying mechanisms and technologies and hence allowing source-level portability of applications to different environments. This specification defines GSS-API services and primitives at a level independent of underlying mechanism and programming language environment, and is to be complemented by other, related specifications:

- documents defining specific parameter bindings for particular language environments.
- documents defining token formats, protocols, and procedures to be implemented in order to realize GSS-API services atop particular security mechanisms.

### B.3 Extensible Authentication Protocol (EAP)

Extensible Authentication Protocol (EAP)(Aboba et al., 2004) is an authentication framework which supports multiple authentication methods. EAP typically runs directly over data link layers such as Point-to-Point Protocol (PPP) or IEEE 802, without requiring IP. EAP provides its own support for duplicate elimination and retransmission, but is reliant on lower layer ordering guarantees. Fragmentation is not supported within EAP itself; however, individual EAP methods may support this.

EAP may be used on dedicated links, as well as switched circuits, and wired as well as wireless links. To date, EAP has been implemented with hosts and routers that connect via switched circuits or dial-up lines using PPP(Simpson, 1994). It has also been implemented with switches and access points using IEEE 802(IEE, b). EAP encapsulation on IEEE 802 wired media is described in IEE (a), and encapsulation on IEEE wireless LANs in IEE (c).

One of the advantages of the EAP architecture is its flexibility. EAP is used to select a specific authentication mechanism, typically after the authenticator requests more information in order to determine the specific authentication method to be used. Rather than requiring the authenticator to be updated to support each new authentication method, EAP permits the use of a backend authentication server, which may implement some or all authentication methods, with the authenticator acting as a pass-through for some or all methods and peers(Aboba et al., 2004).

### B.4 Lightweight Directory Access Protocol (LDAP)

Lightweight Directory Access Protocol (LDAP for short) is a directory service for managing the user accounts. It's an open Internet standard, produced by the Internet Engineering Task Force (IETF), the same body that gave the world TCP/IP as an alternative to the heavy weight DAP.(Howes & Smith, 1997) LDAP is a directory service, it is a simplified database, and should not be confused with traditional database. Typically,

it does not have the database mechanisms to support transactions (Wang et al., 2008). Directories are flexible, secure, and can be personalized. Another defining characteristic of a directory is that it supports Information Extensibility and Data Distribution (Howes et al., 2003).

A computer system requires the compatibility to locate certain types of information easily, efficiently, and quickly. As a directory service, LDAP is reasonably simple, but provides a wealth of features. It consolidate existing services into a single directory that can be accessed by it's clients from various vendors, such as browsers, email clients, and authentication protocols (Carter, 2003; Howes & Smith, 1997).

In the 1980s, two separate standard bodies, International Telegraph and Telephone Consultative Committee, now know as International Telecommunication Union and International Organization for Standardization (ISO) started work on directory services for their own purpose. Eventually, the two independent directory specification efforts merged into one effort, and X.500 standard was approved in late 1988 then published in 1990. Like other open standards, LDAP have many implementations, from property software, to open source software such as openLDAP (Howes & Smith (1997); Voglmaier (2003)).

## B.5 ActiveDirectory

“Active Directory (AD) is a Microsoft network operating system (NOS) directory, built on top of Windows 2000 and Windows Server 2003” (Richards et al., 2006; Hunter, 2006). In 1997 Microsoft release Active Directory beta as the basis of their domain management function (Richards et al., 2006). The initial version was released with Windows 2000 and Windows Server 2003 (Hunter, 2006). Active Directory proved to be one of the most important new features in Windows NT 5.0. It greatly simplify the tasks involved in administering and managing large Windows NT networks, and it improves the user's interaction with networked resources.

The Active Directory is also the underpinning key that enables the improvements

in distributed system security. All resources are stored and managed on the network, it's easy for developers, administrators, and users to find and use. It provides a single, consistent, open set of interfaces for performing common administrative tasks, such as adding new users, managing printers, and locating resources throughout the distributed computing environment (Solomon, 1998).

The Active Directory data model has many concepts similar to X.500. It includes many other useful features such as group policies and delegation of authority that are useful in a diverse and distributed environment such as the university (Blezard & Marceau, 2002). The directory holds objects that represent various resources, which are described by attributes. The universe of objects that can be stored in the directory is defined in the schema. For each object class, the schema defines what attributes an instance of the class must have, what additional attributes it can have, and what object class can be a parent of the current object class. This directory structure has the following key features:

- Flexible hierarchical structure.
- Efficient multimaster replication.
- Granular security delegation.
- Extensible storage of new classes of objects and properties.
- Interoperability through Lightweight Directory Access Protocol (LDAP) version 3 support.
- Scalability to millions of objects per store.
- Integrated dynamic Domain Name System (DNS) server.
- A programmable class store.

Programmability and extensibility are significant capabilities of the Active Directory. Developers and administrators deal with a single set of directory service interfaces, regardless of the installed directory service(s). The programming interface, called the

Active Directory Service Interfaces (ADSI), is accessible by any language. You can also access the directory using the LDAP API. The LDAP C API, defined in RFC 1823, is a lower-level interface available to C programmers.

To establish if the functionality offered by Active Directory above NT domains is actually required we need to examine the functionality required in our environment for the support of Windows clients. Some simple questions need to be asked. These questions include the following:

- Is there anything that Microsoft Active Directory domain offers that is not readily available from Samba based Windows NT or (if supported in Samba) Active Directory domains?
- Is there anything that we need that Samba cannot provide to Windows clients?

One requirement that is clear is the availability of simple tools for use by systems administrators for the management of existing user accounts and resources. From experience, it is the opinion of the author that the majority of regular sysadmin tasks should be performed by scripting rather than by hand. This is the best way to ensure the consistent application of settings, options and parameters to user and resource accounts. It is true that in certain cases, individual settings may need to be verified and for this reason a graphical tool is desirable. However, in many cases, the assertion or re-assertion of such settings using a pre-defined script tool is an equally sure guarantee that the required settings have been applied. The different opinions on this matter are somewhat analogous to the difference between a stateful and a stateless approach to system management. While one system administrator might prefer to check the settings applied to a user or resource and to do this with a graphical tool, another might ignore the current state and simply apply a pre-defined state using a configuration script.



# Appendix C

## Life-cycles of the New Simplified FSSO System

### C.1 Life-cycle 1: end-user accesses non-web base application

1. An end-user in domain Alpha logs onto a desktop system by entering his/her credentials. The credentials are then sent to the KSSO system ( $K_\alpha$ ).
2.  $K_\alpha$  sends a search query to the end-user database ( $D_\alpha$ ) in domain Alpha.
3.  $D_\alpha$  returns the results to  $K_\alpha$ .
4.  $K_\alpha$  creates and sends a ticket granting ticket ( $TGT_\alpha$ ) to the desktop system if the credentials were found in  $D_\alpha$ . The  $TGT_\alpha$  will be stored on the end-user's desktop system.
5. The end-user requests access to SSH server ( $SSH_s$ ) in domain Beta. The end-user accesses  $SSH_s$  using SSH client ( $SSH_c$ ).
6.  $SSH_s$  requests the end-user to be authenticated.

7.  $SSH_c$  extracts  $TGT_\alpha$  from the desktop system and presents it to  $K_\alpha$ . It then requests a service ticket ( $ST_\alpha$ ) from  $K_\alpha$  to access  $AIIP_i$ .
8.  $K_\alpha$  will create and send  $ST_\alpha$  to  $SSH_c$  if  $TGT_\alpha$  was successfully verified.  $ST_\alpha$  will be stored on end-user's desktop system as well.
9.  $SSH_c$  extracts  $ST_\alpha$  from the desktop system and sends it to  $AIIP_i$ .
10.  $AIIP_i$  will create an identity assertion message for the end-user and send it to  $AIIP_s$  if  $ST_\alpha$  was successfully verified.
11.  $AIIP_s$  accepts the end-user's identity assertion based on the federation's policies. It creates a database query that searches for end-user's provisioned identity ( $I_\beta$ ) in domain Beta. It sends the query to  $D_\beta$ .
12.  $D_\beta$  executes the query and returns  $I_\beta$  to  $AIIP_s$ .
13.  $AIIP_s$  uses the  $I_\beta$  to authenticate to the KSSO system in domain Beta ( $K_\beta$ ) on behalf of the end-user.
14.  $K_\beta$  sends a search query to  $D_\beta$ .
15.  $D_\beta$  finds the credentials and return the result to  $K_\beta$  since  $I_\beta$  exists in  $D_\beta$ .
16.  $K_\beta$  creates and sends a ticket granting ticket ( $TGT_\beta$ ) to  $AIIP_s$ .  $TGT_\beta$  will be stored in  $AIIP_s$ .
17.  $AIIP_s$  sends  $TGT_\beta$  to  $K_\beta$  and requests service ticket ( $ST_\beta$ ) for  $SSH_s$  on behalf of the end-user.
18.  $K_\beta$  verifies  $TGT_\beta$  and sends  $ST_\beta$  to  $AIIP_s$ .
19.  $AIIP_s$  sends  $ST_\beta$  to  $AIIP_i$ .
20.  $AIIP_i$  sends  $ST_\beta$  to  $SSH_c$ .
21.  $SSH_c$  presents  $ST_\beta$  to  $SSH_s$ .
22.  $SSH_s$  verifies  $ST_\beta$  and accepts the request from  $SSH_c$ .

## C.2 Life-cycle 2: end-user accesses web-based application

1. The end-user uses a web browser ( $WB_\alpha$ ) to request a web-based service ( $WSA_\gamma$ ) in domain Beta.
2.  $WSA_\gamma$  requests  $WB_\alpha$  to identify the domain that the end-user belongs to.  $WSA_\gamma$  then re-directs  $WB_\alpha$  back the web-based authentication system ( $WBAS_\alpha$ ) in domain Alpha.
3.  $WB_\alpha$  will retrieve the TGT from the desktop and send it to  $K_\alpha$ .
4.  $K_\alpha$  verifies the TGT and sends the service ticket ST to  $WB_\alpha$ .
5.  $WB_\alpha$  sends the ST to  $WBAS_\alpha$ . IDAWA (in the  $WBAS_\alpha$ ) accepts the ST.
6. After IDAWA successfully verified the ST,  $WBAS_\alpha$  then creates an identity assertion message and passes it back to  $WB_\alpha$ .
7.  $WB_\alpha$  sends the identity assertion message to  $WSA_\gamma$ .
8.  $WSA_\gamma$  sends a search query to the end-user database in domain Beta ( $D_\gamma$ ).
9.  $D_\gamma$  returns the result to  $WSA_\gamma$ .
10.  $WSA_\gamma$  grants the request from  $WB_\alpha$ .

## C.3 Life-cycles 3: an end-user in domain Beta access domain Alpha

1. An end-user in domain Beta logs onto a desktop system by entering his/her credentials. The credentials will be sent to the Kerberos server ( $K_\beta$ ) in domain Beta.
2.  $K_\beta$  sends a search query to the end-user database ( $D_\beta$ ) in domain Alpha.

3.  $D_\beta$  returns the results to  $K_\beta$ .
4.  $K_\beta$  will create and send a ticket granting ticket ( $TGT_\beta$ ) to the desktop system if the credentials were found in  $D_\beta$ . The  $TGT_\beta$  will be stored on the end-user's desktop system, and it's accessible by  $WB_\beta$  and  $SSH_c$ .
5. The end-user uses  $WB_\beta$  to request  $WSA_\alpha$  in domain Alpha.
6.  $WSA_\alpha$  requests  $WB_\beta$  to identify the domain that the end-user belongs to.  $WSA_\alpha$  then re-directs  $WB_\beta$  back the web-based authentication system ( $WBAS_\beta$ ) in domain Beta.
7.  $WB_\beta$  will retrieve the TGT from the desktop and send it to  $K_\beta$ .
8.  $K_\beta$  verifies the TGT and sends the service ticket ST to  $WB_\beta$ .
9.  $WB_\beta$  sends the ST to  $WBAS_\beta$ . IDAWA (in the  $WBAS_\beta$ ) accepts the ST.
10. After IDAWA successfully verified the ST,  $WBAS_\beta$  then creates an identity assertion message and passes it back to  $WB_\beta$ .
11.  $WB_\beta$  sends the identity assertion message to  $WSA_\alpha$ .
12.  $WSA_\alpha$  sends a search query to the end-user database in domain Beta ( $D_\alpha$ ).
13.  $D_\alpha$  returns the result to  $WBAS_\beta$ .
14.  $WSA_\alpha$  grants the request from  $WB_\beta$ .
15. The end-user proceeds to request access to SSH server ( $SSH_s$ ) in domain Alpha. The end-user accesses  $SSH_s$  with SSH client ( $SSH_c$ ).
16.  $SSH_s$  sends a authentication request to  $SSH_c$ .
17.  $SSH_c$  extras  $TGT_\beta$  from the desktop system and presents it to  $K_\beta$ . It then requests another service ticket ( $ST_\beta$ ) from  $K_\beta$  to access  $AIIP_i$ .
18.  $K_\beta$  will create and send  $ST_\beta$  to  $SSH_c$  if  $TGT_\beta$  was successfully verified.  $ST_\beta$  will be stored on end-user's desktop system as well.

19.  $SSH_c$  extracts  $ST_\beta$  from the desktop system and sends it to  $AIIP_i$ .
20.  $AIIP_i$  will create an identity assertion message for the end-user and send it to  $AIIP_s$  if  $ST_\beta$  was successfully verified.
21.  $AIIP_s$  accepts the end-user's identity assertion based on the federation's policies. It creates a database query that searches for end-user's provisioned identity ( $I_\alpha$ ) in domain Alpha. It sends the query to  $D_\alpha$ .
22.  $D_\alpha$  executes the query and returns  $I_\alpha$  to  $AIIP_s$ .
23.  $AIIP_s$  uses the  $I_\alpha$  to authenticate to the KDC in domain Beta ( $K_\alpha$ ) on behalf of the end-user.
24.  $K_\alpha$  sends a search query to  $D_\alpha$ .
25.  $D_\alpha$  finds the credentials and return the result to  $K_\alpha$  since  $I_\alpha$  exists in  $D_\alpha$ .
26.  $K_\alpha$  creates and sends a ticket granting ticket ( $TGT_\alpha$ ) to  $AIIP_s$ .  $TGT_\alpha$  will be stored in  $AIIP_s$ .
27.  $AIIP_s$  sends  $TGT_\alpha$  to  $K_\alpha$  and requests service ticket ( $ST_\alpha$ ) for  $SSH_s$  on behalf of the end-user.
28.  $K_\alpha$  verifies  $TGT_\alpha$  and sends  $ST_\alpha$  to  $AIIP_s$ .
29.  $AIIP_s$  sends  $ST_\alpha$  to  $AIIP_i$ .
30.  $AIIP_i$  sends  $ST_\alpha$  to  $SSH_c$ .
31.  $SSH_c$  presents  $ST_\alpha$  to  $SSH_s$ .
32.  $SSH_s$  supports  $K_\alpha$ . It verifies  $ST_\alpha$  and accepts the request from  $SSH_c$ .