Distributed Authentication for Resource Control

THESIS

Submitted in fulfilment of the requirements for the degree of MASTER OF SCIENCE of Rhodes University

by

Keith Robert Burdis

January 2000

Abstract

This thesis examines distributed authentication in the process of controlling computing resources. We investigate user sign-on and two of the main authentication technologies that can be used control a resource through authentication and providing additional security services.

The problems with the existing sign-on scenario are that users have too much credential information to manage and are prompted for this information too often. Single Sign-On (SSO) is a viable solution to this problem if physical procedures are introduced to minimise the risks associated with its use.

The Generic Security Services API (GSS-API) provides security services in a manner independent of the environment in which these security services are used, encapsulating security functionality and insulating users from changes in security technology. The underlying security functionality is provided by GSS-API mechanisms. We developed the Secure Remote Password GSS-API Mechanism (SRPGM) to provide a mechanism that has low infrastructure requirements, is password-based and does not require the use of long-term asymmetric keys. We provide implementations of the Java GSS-API bindings and the LIPKEY and SRPGM GSS-API mechanisms.

The Secure Authentication and Security Layer (SASL) provides security to connection-based Internet protocols. After finding deficiencies in existing SASL mechanisms we developed the Secure Remote Password SASL mechanism (SRP-SASL) that provides strong password-based authentication and countermeasures against known attacks, while still being simple and easy to implement. We provide implementations of the Java SASL binding and several SASL mechanisms, including SRP-SASL.

Acknowledgements

I'm extremely grateful for the time and effort that my supervisor, Professor Peter Wentworth, has spent in helping me with this research, as well as for his guidance, encouragement and insightful comments during the write-up process.

I'd like to thank Professor Wentworth, Professor Pat Terry and Austin Poulton for proof-reading drafts of this document, Soteri Panagou for writing tips and Siviwe Kwatsha for helping with the final binding.

This research was aided by some individuals who I have never met, but who nevertheless provided help when I needed it.

- The members of the IETF Common Authentication Technology working group, in particular Theodore Ts'o and Tom Yu for an interesting discussion on ASN.1.
- Charlie Lai of Sun Microsystems for explaining how JAAS works.
- Thomas Wu for developing the Secure Remote Password (SRP) protocol and making it freely available.
- John Myre on the *sci.crypt* newsgroup for pointing out the flaw in my proposed optimisation of SRP.

Thanks to Jeroen van Gelderen, Ian Grigg, Raif Naffah, Paul Waserbrot, Edwin Woudt and the rest of the Cryptix Development Team for allowing me to contribute and making me feel welcome.

The staff and postgraduate students of the Computer Science department are a wonderful group of people, and it is thanks to them that my stay at Rhodes has been an unforgettable and enjoyable experience. Thanks to you all!

The financial support of Telkom SA is also acknowledged.

Finally, this work is dedicated to my family, who have always believed in me.

Contents

List Of Figures **vii**

1 Introduction **1**

1.1 Motivation 1

1.2 Assumptions and Goals 2

1.3 Campus Scenario 3

1.4 Conventions and Terminology used 4

1.5 Document Overview 4

2 Background **6**

2.1 Authentication 6

2.1.1 Entity authentication evidence 7

2.1.2 Message Authentication evidence 10

2.2 Attacks 10

2.2.1 Eavesdropping 11

2.2.2 Tampering 12

2.2.3 Session Hijacking 14

2.2.4 Chosen-plaintext attack 14

2.3 Countermeasures 14

2.3.1 Cryptography 15

2.3.1.1 Symmetric Cryptography 15
2.3.1.2 Asymmetric Cryptography 16
2.3.2 One-way hash functions 17
2.3.3 Nonces 18
2.4 Secure Remote Password protocol 19
2.4.1 Operation 19
2.4.2 Features 21
2.4.3 Optimised SRP 24
2.4.4 Attempted further optimisation 24
2.5 Further reading 26

3 Single Sign-On **27**
3.1 Introduction 27
3.2 Campus Scenario 28
3.3 Single Sign-On 29
3.4 JAAS 30
3.4.1 PAM 31
3.4.2 JAAS Framework 31
3.4.3 Single Sign-on 33
3.4.4 Implementation 34
3.5 Problems with SSO 34
3.6 Conclusion 35

4 GSS-API **36**
4.1 Introduction 36
4.2 Design 37
4.2.1 Goals 37

4.2.2 Encapsulation 38
4.2.3 Resistance to change 38
4.3 Operation 39
4.4 Java GSS-API 42
4.4.1 Description 42
4.4.2 Implementation 44
4.5 Example 44
4.6 Conclusion 45

5 GSS-API Mechanisms 47

5.1 Introduction 47
5.2 Abstract Syntax Notation One 48
5.2.1 Introduction 48
5.2.2 Encoding Rules 49
5.2.3 Using ASN.1 51
5.2.4 Problems with ASN.1 52
5.2.4.1 Abstraction 53
5.2.4.2 Complexity 53
5.2.4.3 Accessibility 54
5.2.4.4 Our experience 55
5.3 Standard Mechanisms 56
5.3.1 SPKM 56
5.3.2 Kerberos 57
5.3.3 Infrastructure Requirements 58
5.4 Low Infrastructure Public Key mechanism 60
5.4.1 Overview 60
5.4.2 Infrastructure Requirements 61

5.4.3 Implementation 61
5.5 Secure Remote Password GSS-API Mechanism 62
5.5.1 Introduction 62
5.5.2 Operation 63
5.5.3 Relationship to SPKM 64
5.5.3.1 Reused features 64
5.5.3.2 Differences 65
5.5.4 Evolution of SRPGM 66
5.5.4.1 Initial Design 66
5.5.4.2 Integrity Protection 67
5.5.4.3 The values n and g 69
5.5.4.4 The value u 70
5.5.4.5 Algorithms 70
5.5.4.6 Attempted optimisation of SRP 71
5.5.4.7 Published Versions 71
5.5.5 Implementation 72
5.6 Conclusion 72

6 SASL **74**

6.1 Introduction 74
6.2 Relationship to the GSS-API 75
6.3 SASL Mechanisms 76
6.3.1 CRAM-MD5 76
6.3.2 PLAIN 78
6.3.3 DIGEST-MD5 78
6.4 SRP-SASL Mechanism 79
6.4.1 Authentication 80

6.4.2 Security Layer 81
6.4.3 Netstrings 82
6.4.4 Evolution of SRP-SASL 83
6.4.4.1 Version 1 83
6.4.4.2 Version 2 85
6.4.4.3 Version 3 86
6.4.4.4 Attempted optimisation of SRP 88
6.5 SASL Profiles 88
6.5.1 SMTP Profile 89
6.5.1.1 UCE 89
6.5.1.2 Email as a critical resource 90
6.5.1.3 Profile 90
6.5.1.4 Example 90
6.5.2 POP3 Profile 92
6.5.2.1 Example 93
6.6 Java SASL Library 93
6.7 Conclusion 94

7 Conclusions and Future Research 95

7.1 Experience and Contributions 95
7.2 Related and future work 97
7.2.1 CORBA Security Service 97
7.2.1.1 Introduction 97
7.2.1.2 Sign-on 98
7.2.1.3 Security Services 98
7.2.1.4 Summary 99
7.2.2 Possible campus scenario 99

7.2.2.1 User sign-on 100
7.2.2.2 Resource control 100
7.2.2.3 Summary 101
7.3 Final Words 101

References **102**

Appendices

A draft-ietf-cat-srpgm-02.txt

B draft-burdis-cat-srp-sasl-02.txt

List of Figures

2.1 Unauthorised access attack tree 11

3.1 JAAS Authentication 32

4.1 Relationships between the main GSS-API classes 40

4.2 GSS-API operation 41

Chapter 1

Introduction

1.1 Motivation

This project grew out of the need to provide a means of controlling the use of scarce computing resources, such as international network bandwidth and network printing. On the Rhodes University campus it is becoming increasingly necessary to have tighter control over access to and use of these computing resources. This is due in part to an increase in the number of users, brought about by an expanding residence networking project and an increase in the number of machines available in public laboratories, and the corresponding increased contention for the use of the available resources.

Many of the computing services available on a network are expensive to provide in terms of the cost of hardware and software, the manpower in setting them up, and the expenses incurred in keeping them running. Resources may need to be controlled because they are scarce and need to be shared (eg. international network bandwidth). A resource may be expensive to use and need to be controlled to justify or recover costs (eg. network printing). A resource may be sensitive and need to have access to it restricted to certain individuals or groups (eg. a database with student fee records). It is therefore necessary to provide a means of controlling access to and use of these resources.

The resource control process can be broken down into essentially 3 phases, namely:

1. Identification and Authentication - the process whereby an entity claims to hold some identity and then provides evidence to prove that this identity is held.

2. Authorisation - this involves consulting the resource control policy to determine how an authenticated identity is allowed to make use of a resource.

3. Auditing - the process of keeping quantitative resource usage information for uses such as charging, tracking, and limiting resource usage.

This work focuses on an important part of the resource control process, namely the Identification and Authentication phase. The other phases are beyond the scope of this dissertation.

1.2 Assumptions and Goals

This research assumes the following scenario:

> There are computing resources available on a network. These computing resources may be standardised network services such as SMTP, POP and IMAP, access to data such as databases and directories, or middleware services such as the CORBA Naming and Trading Services, for example. A resource controller will only allow a user to make use of a resource once it is certain of the identity of the user and the resource control policy entitles the user to do so. There are attackers that attempt to obtain unauthorised access to and use of resources. These attackers are able to monitor all network communications. They attempt to impersonate legitimate users and disrupt communications by tampering with messages sent between a user and a resource controller. Users and resource controllers wish to prevent unauthorised use of resources.

This is a common scenario in everyday computer use. This dissertation investigates some of the main technologies available to make the process of a legitimate user using a resource secure and resistant to attack. In particular we focus on password-based technologies that have low infrastructure requirements. In addition to investigating security technology, we also want to make the technology available to programmers. Current research in our department is in a number of disparate fields from computer security, computer music, video conferencing, electronic commerce, Internet protocols to virtual reality, and more. We provide implementations of existing security technology that provide security functionality

in an abstract fashion that programmers can easily plug into their applications without having to be concerned about the details of how the security works.

Although we make use of our campus as an example environment where this technology can be applied, we are aiming to provide information and implementations that are more generally useful to a wider audience. For this reason we have made our implementations available as part of the Cryptix project [19]:

> "Cryptix is an international volunteer effort to produce robust, open-source cryptographic software libraries. Cryptix products are free, both for commercial and non-commercial use and are being used by developers all over the world. Development is currently focused on Java."

1.3 Campus Scenario

Our campus scenario has a number of interesting security-related features. Much of the information available using campus computing resources is not that valuable in monetary terms (eg. when compared to the information transmitted by an electronic funds transfer system). Issues of access, confidentiality and privacy are more of a concern. However, there are some resources - including student fees, student records, and examination papers - that are much more security-sensitive and need to be protected accordingly.

Some services available to students, such as networked printing, are charged for and there is therefore an incentive to bypass the charging mechanisms. There is also an incentive for students to try to obtain sensitive information, such as copies of examination papers while they are being printed. Staff may have an incentive to access financial records and adjust their salary figures. There is clearly a need to control these resources.

Another factor is that there is a limited number of staff employed to deploy and manage the various available services. Much of their time is spent on the daily activities of interacting with users, monitoring systems and resolving problems. There is no one dedicated to setting up and maintaining a security infrastructure. For this reason it is important that any security architecture employed be resistant to the inevitable changes that will take place in security technology and mechanisms, so that it is easy to maintain. In addition, radical changes in technology, for example moving from a password-based system to a public-key based system, are harder to deploy because staff may not have experience using and

managing these technologies. Currently our campus security infrastructure is password-based, so we have focused on password-based security technology.

1.4 Conventions and Terminology used

There are two main entities involved in the resource control process. The first is the entity that wishes to make use of a controlled resource, and this entity is referred to as the user, the client or the initiator. The second entity is the resource controller that controls access to and use of the resource, verifying the identity of the initiator, and is referred to as the resource controller, the server or the acceptor.

Following the convention used in Applied Cryptography [80, p 23], a seminal work in the field of computer security, and many other security-related publications, entities may also be referred to using common names, as follows:

Name	Description
Alice	Initiator
Bob	Acceptor
Carol	Client
Steve	Server
Trent	Trusted Entity
Mallory	Malicious attacker
Eve	Eavesdropper

1.5 Document Overview

Chapter 2 This chapter provides background information for following chapters. It describes authentication, the various types of evidence used to prove an identity, the types of attacks that are possible and the technologies used to thwart these attacks. It also introduces the Secure Remote Password protocol and describes our unsuccessful attempt to further optimise it.

Chapter 3 Users sign on in order to use resources, providing an identity and credential evidence. This chapter discusses our investigation of the problems with the current

user sign-on scenario, using our campus scenario as an example. We describe how these problems may be solved using single sign-on, and the risks involved in using this solution.

Chapter 4 This chapter introduces and describes the Generic Security Services API (GSS-API) which specifies authentication and security services in an abstract manner. It describes the features of this API, how it works, and the benefits that are gained from using it. We describes our experiences using and implementing the Java GSS-API, which provides a concrete implementation of the abstract GSS-API specification, and conclude with a hypothetical example illustrating its use.

Chapter 5 The underlying security functionality of a GSS-API implementation is provided by GSS-API mechanisms. In this chapter we introduce the Abstract Syntax Notation One (ASN.1) and describe the benefits and problems with its use in specifying message structures for GSS-API mechanisms. Thereafter we describe the standard GSS-API mechanisms, their high infrastructure requirements and the need for low infrastructure mechanisms. This leads to a discussion of our experiences implementing of the Low Infrastructure Public Key mechanism (LIPKEY) and designing and implementing the Secure Remote Password GSS-API Mechanism (SRPGM), a password-based low-infrastructure GSS-API mechanism that we have developed and submitted as an IETF Internet Draft.

Chapter 6 The Simple Authentication and Security Layer (SASL) adds support for security services to connection-based protocols. This chapter explains how SASL works and its relationship to existing Internet protocols. We discuss existing SASL mechanisms, their features and deficiencies, and explain our motivation for developing the Secure Remote Password SASL (SRP-SASL) mechanism that we have submitted as an IETF Internet Draft. We describe the SMTP and POP3 SASL profiles and discuss the need for email security. In addition we discuss the Java SASL binding, which provides a concrete implementation of the SASL functionality, and available SASL mechanisms.

Chapter 7 This chapter summarizes the experience that we have gained working with distributed authentication technology, and the contributions that we have made to this field. We discuss possible applications of our work in developing an implementation of the CORBA Security Service and in improving the security on our campus. We conclude with some final remarks about our research and contributions.

Chapter 2

Background

The chapter provides information on authentication that will serve as a background to the material presented in following chapters. It explains what authentication is and discusses the types of evidence used to prove an identity. Thereafter follows a discussion of the attacks that are possible in a distributed environment, and how these attacks may be countered. We introduce the Secure Remote Password protocol, discuss why we find it an exciting new technology, and our contributions to its use.

2.1 Authentication

> "Authentication is one of the most important of all information security objectives." Menezes et al [58]

The resource control process is aimed at restricting the use of a resource to those entities who are authorised to use the resource according to some policy. It is therefore necessary to be certain at all times of the identity of any entities making use of a resource. An entity may have many identities at any one time. Different identities are used for different purposes. For example, a person may be associated with a student number that they use for borrowing books from a university library, and at the same time an ID number that they use to vote in the general elections. In a computing environment an identity is usually based on a name or a number and is referred to as a username.

However, anyone can claim to hold a particular identity, so in order to use a resource, it is necessary to provide some evidence to prove that this identity is held. *Authentication is*

the process whereby an entity asserts an identity and then provides evidence to prove that this identity is held.

Menezes et al [58] make a distinction between entity authentication and message authentication. Entity authentication is defined as follows:

> "An identification or entity authentication technique assures one party (through acquisition of corroborative evidence) of both the identity of a second party involved, and that the second was active at the time the evidence was created or acquired"

Message authentication is defined as:

> "Data origin authentication or message authentication provide to one party which receives a message assurance (through corroborative evidence) of the identity of the party which originated the message."

The distinction is then whether or not the parties are active at the time authentication takes place. Entity authentication is typically used at the beginning of a communications session between entities when the communicating entities prove their identities to each other. Message authentication is concerned with data integrity mechanisms that prove who originated a message and that it was not altered in transit. Entities exchange some information that enables them to authenticate future exchanged messages.

2.1.1 Entity authentication evidence

This evidence used with entity authentication is in the form of [98]:

- something that the user **knows**
- something that the user **has**
- something that the user **is**

Something that the user *knows* is typically in the form of a remembered password (a short sequence of characters), passphrase (a short sequence of words), or PIN (Personal

Identification Number; a short sequence of digits). This is the most common form of evidence, since it is simple and does not require any external input device other than a keyboard. A problem with this form of evidence is that since users are unable to remember information that has a large amount of entropy[1], it may be possible to guess what this information is. Password cracking, where a program tries passwords based on dictionary words and information about the user, is a popular example of this. In addition the set of possibilities is in some cases small enough that it is viable for an attacker to try every possibility in order to find the correct one [47, Pgs 34, 40-42]. A good example is the four digit PIN number typically used by banks with their Automated Teller Machine (ATM) cards. There are only 9999 possible four digit PINs, which is a small set, so it is may be viable for an attacker to try each of these possibilities. (Fortunately, ATMs have procedural controls that only allow an attacker to enter the incorrect PIN a limited number of times).

Something that a user *has* may involve the use of some physical item such as an id device or a smart card. Id devices contain information that uniquely identifies the device, thereby proving that the user is in possession of the device. (An example of an id device is the Dallas chip system used at Rhodes). A smart card is "a plastic card, the size and shape of a credit card, with an embedded computer chip" [80]. These smart cards may have evidence and cryptographic protocols programmed into them that enable them to be used for authentication. Something that a user has may also involve the use of some non-remembered information, typically a private key which is part of of a public/private keypair used in asymmetric cryptography. The asymmetric nature of a public/private keypair means that the private key can be used to generate evidence such that it can be proved, using the publicly available public key certificate, that the evidence could only have been generated using this private key (See 2.3.1.2). Smart cards and private keys are typically used in conjunction with remembered evidence that "unlocks" the smart card or private key, enabling it to be used. An example of this is the bank cards used with ATMs. A user has a ATM card that they insert into the machine and are able to use by entering their PIN number. This combination of evidence provides better security than each form of evidence used alone - such systems are called multi-factor systems [98].

Something that the *is* user refers to biometric information, which is some recorded physical characteristic of the user. People have unique physical features that may be used to identify them. The most well known example of this is the fingerprint, which is used in

[1] Entropy is the measure of the disorder of a system. This means that user passwords are not random, which makes them easier to guess.

the criminal justice system. Other examples include identifying the unique patterns on the iris and retina, and measuring the physical dimensions of a part of the body, such as the hands. The use of biometric information and smart cards as evidence is not as common as the use of passwords and the like since they require the use of uncommon hardware such as fingerprint readers and retinal scanners. In addition there is some social resistance to the use of biometric techniques due to their association with criminals and fears that this information will be used to invade personal privacy (for example in an Orwellian state, the "Big Brother"). An important point made by Bruce Schneier in his ACM paper on Biometrics [84], is that a biometric is not a secret. Once a reader has obtained the biometric information it is able to store this information for use at a later time. It is therefore important that if biometric information is used as evidence, steps are taken to ensure that the user is actually present at the time the authentication takes place, and the evidence provided is not just a replay of a previous session. (This may be done, for example, by authenticating the biometric reader to the verifier).

Each of these forms of evidence have their strengths and weaknesses. If passwords are used carefully and care is taken not to expose them during the authentication process they are a simple and useful form of evidence. However if the password is exposed, either by writing the password down or during the authentication process, security is compromised since anyone who knows the password can impersonate the user. An id card, smart card or private key is a strong, difficult to guess form of evidence as long as the user keeps possession of it. Anyone who has possession of the id card, smart card or private key can impersonate the user. Biometrics are useful in that they identify unique characteristics of the user. However, if care is not taken to ensure that a trusted reader is used and the user is present at the time of authentication, anyone who has this biometric information can impersonate the user. Clearly the more forms of evidence used by an authentication system the better. Multifactor systems that use combinations of evidence types, such as in the ATM example above, provide greater security since an attacker has to obtain more, varied information. Because the evidence is in different forms, different attacks need to be used to obtain it.

On our campus we make use of an id devices, known as the "Dallas chip" [23], to control access to physical locations such as residences and with the meal booking system. All other computer-based authentication takes place using remembered passwords in different operating environments. Although there is interest in implementing public-key based technology and using smart cards, password-based technology will continue to be used in the

foreseeable future.

2.1.2 Message Authentication evidence

As described above, message authentication is concerned with data integrity mechanisms that prove who originated a message and that it was not altered in transit. (If a message is altered in transit then it is a new message and the origin of the message has changed [58]). In order to facilitate message authentication the entities involved in the communications session make use of cryptography.

When symmetric cryptography is used the evidence is a keyed message authentication code. The entities exchange some information in a process known as key agreement or key exchange and the result is a session key. The sender makes a digest of the message to be sent, incorporating the session key using some agreed upon algorithm (such as HMAC [46]), and sends this digest along with the message. The receiver also computes this keyed digest and then verifies that the digest accompanying the message matches the computed digest.

When asymmetric cryptography is used the evidence is a digital signature. The sender makes a digest of the message to be sent, encrypts this digest with its private key, and sends this digest along with the message. The receiver attempts to decrypt the digest accompanying the message using the sender's public key. If successful the receiver also computes the digest of the message and then verifies that it matches the decrypted digest.

2.2 Attacks

Authentication is concerned with preventing someone from masquerading as someone else and thereby gaining access to and use of resources that they should not have access to and use of. In a distributed environment many more attacks are possible.

Figure **??** is an attack tree [81] that describes some examples of how an attacker may attempt to gain unauthorised access to a resource. Some of the possible attacks are physical, such as bribing or coercing users into revealing authentication evidence, or stealing physical evidence such as id tokens or smart cards. Although it is important that countermeasures to such attacks be found, we're concerned with attacks of a technical nature that have technical solutions. Examples of such attacks include passive attacks such as eavesdropping on

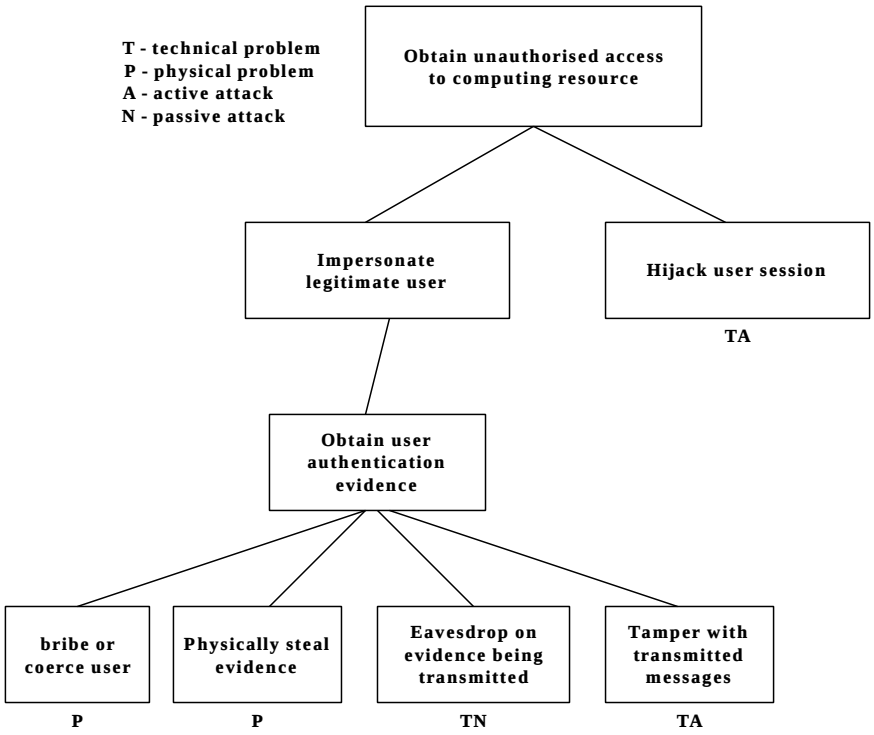

Figure 2.1: Unauthorised access attack tree

transmitted messages, and active attacks such as tampering with transmitted messages, replaying messages, interleaving messages and hijacking user sessions. Passive attacks do not affect the protocol and are therefore difficult to detect, so protocols try to prevent such attacks. Since active attacks affect the protocol in some way, secure protocols must be designed to detect these attacks [80, p 27].

2.2.1 Eavesdropping

Eavesdropping is when an attacker listens to the communication between entities in order to obtain information. In this case the attacker is trying to obtain the secret the user uses for authentication so that it can impersonate the user. In discussing the risks involved in logging in to a service over the network, RFC 2504 [35] notes:

> "All information passing over networks may be eavesdropped on... Information passing over a network may be read not only by the intended audience but can be read by others as well."

It is a basic assumption of most authentication protocols that an attacker is able to eavesdrop on the communications medium ie. monitor all messages exchanged [80, p 5]. It is therefore important that during authentication when the user transmits evidence to prove its identity that this information not be useful to an attacker in trying to obtain the user's secret.

In the proceedings of a workshop aimed at designing a security architecture for the Internet, the Internet Architecture Board had this to say about plaintext passwords [6]:

> "One security mechanism was deemed to be unacceptable: plaintext passwords. That is, no protocol that relies on passwords sent over unencrypted channels is acceptable."

Sending the secret as plaintext makes it directly available to the attacker so challenge-response authentication protocols [58, Ch 10.3] were devised that enable a user to prove knowledge of the secret, without transmitting the secret directly, by correctly answering a challenge based on the secret. Even if the information transmitted cannot be used to directly obtain the secret, an attacker may use the transmitted information to guess the secret. For example if the secret is a password, an attacker may attempt to guess the password and use the information to check whether or not the guess was correct. Such guesses may be made more successful through the use of a dictionary containing commonly used passwords [58][47]. (See also 2.4.2 below). Authentication protocols that do not leak any information that can be used by an attacker (or even the verifier) to obtain the user's authentication secret are known as zero-knowledge authentication protocols [58, Ch 10.4].

The only time that it is safe to use plaintext passwords is when a secure communications session has been set up where all exchanged messages are encrypted. Such a secure communications session may be set up using the Transport Layer Security [24] protocol, for example.

2.2.2 Tampering

Tampering is when an attacker interferes with messages sent between the communicating parties. This may be by modifying, deleting or re-ordering messages, or even creating completely new ones.

An example of such tampering is a man-in-the-middle attack, where an attacker intercepts communications and pretends to be the other communicating party. So, if Alice is communicating with Bob, Mallory (the malicious attacker) pretends to be Bob when communicating with Alice, and pretends to be Alice when communicating with Bob.

Schneier [80, p 48] gives an example of a man-in-the-middle attack where Alice and and Bob wish to set up such a secure session using their public/private keypairs:

1. Alice sends Bob her public key. Mallory intercepts this key and sends Bob his own public key.
2. Bob sends Alice his public key. Mallory intercepts this key and sends Alice his own public key.
3. When Alice sends a message to Bob, encrypted in "Bob's" public key, Mallory intercepts it. Since the message is really encrypted with his own public key, he decrypts it with his private key, re-encrypts it with Bob's public key, and sends it on to Bob.
4. When Bob sends a message to Alice, encrypted in "Alice's" public key, Mallory intercepts it. Since the message is really encrypted with his own public key, he decrypts it with his private key, re-encrypts it with Alice's public key, and sends it on to Alice.

All communications go through the attacker and the communicating entities are unaware that this is taking place. [80] notes that:

> "In general, a man-in-the-middle attack can defeat any protocol that doesn't involve a secret of some kind."

In order to counter such attacks it is necessary to have exchanged secret information, such as a password, in some out-of-band manner. Or, it necessary to make use of a trusted third party such as a key server or a certification authority (CA)[2]. Secret or trusted information can then be used in the authentication process.

Another form of tampering is reflecting messages sent by one entity back to that entity at a later time. To counter this, messages should contain some information that identifies the target of the message or the message type [58]. Malicious re-ordering of messages can be prevented through the use of sequence numbers.

[2] In this case the CA's private key is the secret information.

An attacker can use information that it has eavesdropped to replay authentication message information exchanged during one authentication session during a later or parallel (ie. happening at the same time) authentication session [58, Pgs 417-418]. If the authentication protocol is not designed to detect replay of messages and messages are not linked to a particular authentication session, an attacker may be successful in impersonating a legitimate user. Techniques to prevent such replay include the use of secure timestamps and fresh random numbers embedded in protocol messages.

2.2.3 Session Hijacking

Once a user has undergone the entity authentication process and proved its identity to the resource controller it is able to make use of the resource. One possible attack is to hijack the user session just after the authentication process and indicate to the user that an error has occurred and the session has been terminated. Thereafter the attacker can use the authenticated session to make use of the resource. In order to counter this attack it is necessary to have message authentication in addition to entity authentication.

2.2.4 Chosen-plaintext attack

If an attacker is able to get an entity to encrypt a piece of plaintext, chosen by the attacker, using the entity's private credential evidence, this makes it easier for the attacker to obtain the credential evidence through cryptanalysis [80, p 6]. Such chosen-plaintext attacks can be countered by having the encrypting entity introduce some random data into the message before encrypting it.

2.3 Countermeasures

In addition to assuming that the attacker is able to monitor and manipulate any exchanged messages, as discussed in the previous section, we assume that the attacker has full knowledge of the working and implementation of any algorithms employed [80, p 5].

> "Security is not and cannot be a cookie cutter process. There is no magic pixie dust that can be sprinkled over a protocol to make it secure. Each protocol

> must be analyzed individually to determine what vulnerabilities exist, what risks they may lead to, what palliative measures can be taken, and what the residual risks are." - Bellovin [6]

Authentication mechanisms and protocols need to be designed with attacks in mind, and must provide adequate protection against them. Cryptography, one-way functions and nonces are useful countermeasures.

2.3.1 Cryptography

Cryptography is "the art and science of keeping messages secure" [80, p1]. It may be used during the authentication process as part of generating the evidence that is used to prove an identity and to protect the messages transmitted between the communicating entities. There are two types of cryptography used in computer security: symmetric cryptography where parties share a secret key, and asymmetric cryptography which involves the use of public/private keypairs.

2.3.1.1 Symmetric Cryptography

When using symmetric cryptography the parties involved share knowledge of a secret key. The same key is used in both the encryption and decryption processes. It is imperative that this key remain secret since anyone with access to the key can decrypt messages produced using this key, and encrypt new messages using the key. One of the primary issues involved in using symmetric cryptography is exchanging or negotiating this shared secret key. Many key establishment protocols have been devised to resolve this issue [58, chapter 12] [80, chapter 22]. Key establishment may be broken into two categories: key transport and key agreement.

> "A key transport protocol or mechanism is a key establishment technique where one party creates or otherwise obtains a secret value, and securely transfers it to the other(s)." - Menezes et al [58]
>
> "A key agreement protocol or mechanism is a key establishment technique in which a shared secret is derived by two (or more) parties as a function of information contributed by, or associated with, each of these, (ideally) such that no party can predetermine the resulting value." - Menezes et al [58]

Key establishment may be based on symmetric encryption (eg. Authenticated Key Exchange Protocol 2 [58, p 499]), perhaps using a trusted third party (eg. Kerberos [58, pp 501-503]). It may also be based on asymmetric cryptography (eg. Diffie Hellman [80, pp 513-514]).

Symmetric key encryption and decryption is orders of magnitude faster than asymmetric encryption and decryption. In addition the key sizes necessary to provide the same level of security (ie. difficulty in breaking the key) are much smaller with symmetric cryptography than with asymmetric cryptography.

2.3.1.2 Asymmetric Cryptography

When using asymmetric cryptography (also known as public key cryptography) the user has a keypair that consists of a private key that the user keeps secret and a public key that the user makes available to other entities. The asymmetric nature of the private and public keys means that a message encrypted with the private key can only be decrypted by the corresponding public key, and vice versa. Since the public key is freely available anyone can check whether a message was encrypted using the private key by attempting to decrypt it using the corresponding public key.

An important part of the process of using public key cryptography is ensuring that a public key really is associated with a particular identity. Consider the scenario described in 2.2.2 where Mallory (the malicious attacker) is able to launch a man-in-the-middle attack against Alice and Bob by sending a fake public key to Alice claiming to be Bob, and a fake public key to Bob claiming to be Alice. Alice and Bob think that they are communicating securely with each other when instead Alice and Bob are passing messages through Mallory who has full access to all information that they transmit.

One solution to this problem is though the use of a trusted entity (Trent), called a certification authority, that has a public key that all entities involved know to be valid. Trent verifies the identity of an entity (using some external means) and, if valid, signs the entity's public key using his private key, creating a public key certificate. Other entities that trust Trent can be sure of the validity of the public key certificates by checking the signature on the certificate using Trent's well-known public key. These public key certificates are often stored in a publicly available directory so that the public key certificate associated with an entity can be found easily.

Another solution to the problem involves a decentralised web of trust, popularised by the Pretty Good Privacy (PGP) [102] program by Phil Zimmerman. This model works on the principal that "a friend of yours is a friend of mine". Instead of having a single trusted entity users trust each other to certain degrees. A user might trust his close friends enough to trust any public key certificates that these close friends have signed. This user may then sign these certificates and pass them on to other friends. By trusting entities that they know, and these entities trusting other entities a web of trust is established, where before a user trusts any entity's public key certificate it must have a certification path back to an entity that the user does trust. A scenario where all entities trust a single entity is a degenerate case that works the same as the certification authority described above.

The use of public-key cryptography has some benefits over the use of secret-key cryptography. One advantage is that it supports non-repudiation, which is defined in the CORBA Security Service Specification (CORBASec) [68] as follows:

> "Non-repudiation provides irrefutable evidence of actions such as proof of origin of data to the recipient, or proof of receipt of data to the sender to protect against subsequent attempts to falsely deny the receiving or sending of the data."

Non-repudiation makes entities accountable for their actions by storing evidence that can be used to later resolve disputes over whether or not an action or event took place. This service can be provided using a trusted third party and digital signatures [68]. Section 3.7 of CORBASec [68] describes this process in more detail.

Another advantage is that it provides scalability to large user populations. Using a shared-key system, every entity must have a shared key with every other entity that it wishes to communicate with. This means that the number of keys necessary for communication grows exponentially with the number of entities. Public-key systems do not have this problem due to the fact that each entity has an available public-key that can be used by all other entities to secure communications. (Kerberos, a shared-key system, overcomes this problem using a trusted third party).

2.3.2 One-way hash functions

> "Breaking a plate is a good example of a one-way function. It is easy to smash a plate into a thousand tiny pieces. However, it's not easy to put all of those

> tiny pieces back together into a plate." - Schneier [80, p 29]

A one-way hash function takes an arbitrary length message as input, known as a pre-image, and produces a fixed-length output, known as a hash or digest. Given a hash function h = H(M), where M is the pre-image, H is the hash function, and h is the hash, it has the following properties [80, p 429]:

- Given M, it is easy to compute h
- Given h, it is hard to compute M such that H(M) = h
- Given M, it is hard to find another message, M', such that H(M) = H(M')

One-way hash functions produce a digest of the original message that has security-related applications. The digest may be used as a checksum to ensure that the message was received without errors [58, pp 338-351]. If the digest-creation process includes the use of a key then the output is a message authentication code (MAC). Since the digest provides a "fingerprint" of the original message and the MAC can only be created using the key, it may be used to provide integrity protection or message authentication, which ensures that a message was not tampered with during transport [58, pp 352-367].

One-way hash functions may also be used to prove knowledge of some data to another party without revealing the data to the other party. This has particular application in authentication protocols, such as those classified as challenge-response protocols.

2.3.3 Nonces

> "Nonces are opaque, transient, session-oriented identifiers which may be used to provide demonstrations of freshness. " - Rescorla [76]

Replay attacks, discussed above, involve replaying messages exchanged earlier in a session or during a previous session. Such attacks can be thwarted by providing a way of checking whether or not a message is fresh is not. A fresh message is defined as a message that was recently generated. The inclusion of time-variant data, called nonces, such as secure timestamps [58, pp 399-400], sequence numbers [58, p 399] and fresh random numbers [58, p 398] can be used as an indication of freshness [58].

2.4 Secure Remote Password protocol

> "The lack of a secure authentication mechanism that is also easy to use has been a long-standing problem with the vast majority of Internet protocols currently in use. The problem is two-fold: Users like to use passwords that they can remember, but most password-based authentication systems offer little protection against even passive attackers, especially if weak and easily-guessed passwords are used." - Wu [99]

The Secure Remote Password (SRP) protocol is a zero-knowledge authentication and key exchange protocol developed by Thomas Wu at Stanford University. As described above, zero-knowledge means that it does not leak any information during the authentication process that can enable an attacker to obtain or derive the secret authentication evidence used. It is based on Authenticated Key Exchange (AKE) where key exchange takes place using ephemeral public/private keypairs.

2.4.1 Operation

Rather than attempting to describe SRP in our owns words we shall rather use the protocol description given in the original SRP paper [98]:

C	Client's username
n	A large prime number. All computations are performed modulo n
g	A primitive root modulo n (often called a *generator*)
s	A random string used as the user's *salt*
P	The user's password
x	A private key derived from the password and salt
v	The host's password verifier
u	Random scrambling parameter, publicly revealed
a, b	Ephemeral private keys, generated randomly and not publicly revealed
A, B	Corresponding public keys
$H()$	One-way hash function
m, n	The two quantities (strings) m and n concatenated
K	Session key

	Carol		Steve
1.		$- C \longrightarrow$	(lookup s, v)
2.	$x = H(s, P)$	$\longleftarrow s -$	
3.	$A = g^a$	$- A \longrightarrow$	
4.		$\longleftarrow B, u -$	$B = v + g^b$
5.	$S = (B - g^x)^{a+ux}$		$S = (Av^u)^b$
6.	$K = H(S)$		$K = H(S)$
7.	$M_1 = H(A, B, K)$	$- M_1 \longrightarrow$	(verify M_1)
8.	(verify M_2)	$\longleftarrow M_2 -$	$M_2 = H(A, M_1, K)$

1. Carol sends Steve her username, (eg. carol)

2. Steve looks up Carol's password entry and fetches her password verifier v and salt s. He sends s to Carol. Carol computes her long-term private key x using s and her real password P.

3. Carol generates a random number a, $1 < a < n$, computes her ephemeral public key $A = g^a$, and sends it to Steve.

4. Steve generates his own random number b, $1 < b < n$, computes his ephemeral public key $B = v + g^b$, and sends it back to Carol, along with the randomly generated parameter u.

5. Carol and Steve compute the common exponential value $S = g^{ab+bux}$ using the values available to each of them. If Carol's password P entered in Step 2 matches the one she originally used to generate v, then both values of S will match.

6. Both sides hash the exponential S into a cryptographically strong session key.

7. Carol sends Steve M_1 as evidence that she has the correct session key. Steve computes M_1 himself and verifies that it matches what Carol sent him.

8. Steve sends Carol M_2 as evidence that he also has the correct session key. Carol also verifies M_2 herself, accepting [it] only if it matches Steve's value.

Note that the value of u may be computed as a simple function of B (see 3.2.4 of [98]).

2.4.2 Features

The features of SRP are summed up in the Abstract from the original SRP paper [98]:

> "This paper presents a new password authentication and key-exchange protocol suitable for authenticating users and exchanging keys over an untrusted network. The new protocol resists dictionary attacks by either passive or active network intruders, allowing in principle, even weak passphrases to be used safely. It also offers perfect forward secrecy, which protects past sessions and passwords against future compromises. Finally, user passwords are stored in a form that it not plaintext-equivalent to the password itself, so an attacker who captures the password database cannot use it directly to compromise security and gain immediate access to the host. This new protocol combines techniques of zero-knowledge proofs with asymmetric key exchange protocols and offers significantly improved performance over comparably strong extended methods that resist stolen-verifier attacks such as Augmented EKE or B-SPEKE."

Resists passive and active dictionary attacks

During an the authentication process using passwords the server needs to verify the client's password (P), so it has to store some information about the password in order to do so. In order to avoid storing cleartext passwords, they are often stored as a hash:

$$h = H(P)$$

But this scheme is vulnerable to a passive dictionary attack in which an attacker who knows H (which is usually a well-known hash function) can precompute hashes for a large dictionary of common passwords. By including a random salt value along with the password as input to the hash function, it becomes much harder to precompute dictionaries for use with a dictionary attacks, because each unique salt value gives rise to a unique dictionary.

$$h = H(s, P)$$

(This computation of stored authentication data is commonly used in Unix systems). Nonetheless, a particular password and salt combination can still be attacked offline by testing each entry in a large dictionary of common passwords along with each possible salt value. SRP makes the computation that an attacker has to perform even more difficult by storing a verifier (v) rather than the hash:

$$v = g^h$$

but this does not prevent a dictionary attack from being successful.

Many authentication protocols have the client send data (eg. h) to authenticate itself that is plaintext-equivalent to the authentication data stored on the server (eg. h). This means that if an attacker manages to obtain the authentication data on the server, this data can be used directly to undertake the authentication protocol and impersonate the user. Since the authentication data and not the client's password is used in the authentication exchange, the client's password need not be known. The important feature of SRP's stored verifiers (v) is that they are not plaintext-equivalent to the data sent during the SRP authentication exchange. This means that v is not equivalent to the information sent by the client (A and M_1) to authenticate itself to the server, so an attacker that knows v cannot directly impersonate the client.

Since SRP is a zero-knowledge protocol, it does not leak any information during the authentication exchange that aids an attacker in performing a dictionary attack.

Offers perfect forward secrecy

This means that if a key from a previous session is compromised this does not give an attacker any additional information to aid in obtaining the password or the session key for any future sessions. Conversely, a compromised key from a later session does not provide an attacker with additional information to aid in obtaining the session keys for past sessions.

Does not use a trusted third party

Mechanisms that do not require a trusted third party in order to operate are known as low-infrastructure mechanisms. Such mechanisms are easier to deploy than those that require additional infrastructure such as a key server or a certification authority.

Produces a shared session key

In addition to performing authentication (possibly mutual), a shared session key is negotiated as part of the authentication process. This session key can be used to provide additional security services for the session, such as integrity and confidentiality protection.

Has good performance

SRP has better performance than any of the other mechanisms that are not plaintext equivalent and provide perfect forward secrecy (eg. A-EKE and B-SPEKE) [98, pp 14-15]). Encrypted Key Exchange (EKE) based protocols, such as DH-EKE, SPEKE, A-EKE and B-SPEKE, also provide strong secret-key based authentication. DH-EKE and SPEKE offer better performance than SRP and provide forward secrecy but suffer from plaintext-equivalence. A-EKE is verifier-based, but has poor performance and does not provide forward secrecy. B-SPEKE provides forward secrecy and provides protection against plaintext equivalence but suffers from comparatively poor performance [98, pp 3-4]. SRP's main features, in addition to being zero-knowledge, are that it has good performance, is a verifier-based protocol and also maintains perfect forward secrecy [98].

Menezes et al [58, p 388] describe password-based authentication protocols as follows:

> "Conventional password schemes involve time-invariant passwords, which provide so-called *weak authentication*."

We believe that SRP is a significant advance in password-based authentication technology, since it provides **strong** password-based authentication, and that it has wide application in securing computing resources. We have used it in both the SRPGM GSS-API mechanism described in 5.5 and the SRP-SASL SASL mechanism described in 6.4.

2.4.3 Optimised SRP

Optimised SRP, as described in Wu [98], reduces the number of message rounds it takes to complete the protocol by grouping together pieces of information that do not depend on earlier messages. Mutual authentication is as follows:

Client		Server
	$- C, A \longrightarrow$	
	$\longleftarrow s, B -$	
	$- M_1 \longrightarrow$	
	$\longleftarrow M_2 -$	

where:

- $M_1 = H(A, B, K)$
- $M_2 = H(A, M_1, K)$

2.4.4 Attempted further optimisation

> "Implementing a protocol such as SRP for use in real systems brings practical issues like performance into play. The number of message rounds, the size of the exchanged messages, and the expected execution time of a successful authentication attempt are all important factors in designing concrete protocol specifications. Eliminating even one network message or computational round can significantly improve the utility of an authentication system." - Wu [98]

We thought that we had discovered a simple optimisation for Optimised SRP that would further reduce the number of message exchanges necessary for mutual authentication, but as explained by John Myre on the sci.crypt USENET newsgroup, this optimisation made an offline dictionary attack possible.

After the first message from the client, the server has sufficient information to derive the session key and produce the evidence necessary to prove that it knows the session key. With our optimisation, the acceptor presents evidence that it knows the shared session key along with the data it sends in its first message, rather than doing so after the initiator presents its evidence in round three:

$$\begin{array}{cc} \text{Client} & \text{Server} \\ & -\ C, A \longrightarrow \\ & \longleftarrow\ s, B, M_2\ - \\ & -\ M_1 \longrightarrow \end{array}$$

where:

- $M_2 = H(A, B, K)$
- $M_1 = H(B, M_2, K)$

This means that mutual authentication takes the same number of message rounds as unilateral authentication does, making it practical to use mutual authentication even in situations where the number of message rounds is important.

However, as pointed out by John Myre on the sci.crypt USENET newsgroup, this makes it possible for an attacker pretending to be a legitimate client to undertake an **offline dictionary attack** (see 2.2.1 and 2.4.2).

> "Now the adversary can do an off-line password guessing attack. With the four-message version, the server doesn't provide M2 (which can be used to verify a guess of a password) until it checks M1 (which proves knowledge of the password). In theory, with SRP you can't check a password guess except by trying to log on, as long as you don't break in to the server itself.
>
> (The attack on the 3-message version would be: start the log on, the quit as soon as you get M2. Then guess different passwords until you can compute M2 yourself)." - John Myre, sci.crypt USENET newsgroup, 14 January 2000

This attack is described in more detail as follows:

1. The attacker discovers the client's username (C). This is easy since the username is transmitted in the clear.
2. The attacker generates its ephemeral asymmetric keypair (a and A) as normal and sends C and A to the server.
3. The server sends s, B and M_2 to the attacker.

4. The attacker aborts the connection.

At this point the attacker has knowledge of the following data:

$$n,\ g,\ a,\ A,\ B,\ s,\ H(A,B,K)$$

The following formulae are used by a client to compute the session key:

$$x = H(s,P)$$
$$S = (B - g^x)^{a+ux}$$
$$K = H(S)$$

(It is assumed that u is computed as a simple function of B.)

The attacker has sufficient information to compute a candidate K using a guessed password (P). It is then able to verify whether or not this candidate K is correct or not using the evidence given by the server:

$$H(A,B,K)$$

since it has knowledge of both A and B.

It is therefore important that the client prove its knowledge of the session key before the server does so, because in order to do so the client needs to have knowledge of the real password.

2.5 Further reading

Kwatsha [47] has an excellent discussion on the theoretical and practical aspects of both host and network security. He also discusses attacks and how to counter them, focusing on the mindset that administrators should have when dealing with these issues. Both the *Handbook of Applied Cryptography* [58] and *Applied Cryptography* [80] are well known security books that have extensive coverage of authentication protocols, the issues surrounding them, and computer security in general, and are highly recommended. The author of *Applied Cryptography* [80], Bruce Schneier, maintains an extensive bibliography of security papers available at [85]. Readers are encouraged to read the original SRP paper [98] and the IETF Internet Draft on SRP [99].

Chapter 3

Single Sign-On

3.1 Introduction

Security authors and advocates proclaim that security must be pervasive in an organisation. However security often gets in the way of normal organisational operations and hinders people undertaking their daily work. It is therefore important to take users into consideration when designing and implementing a security infrastructure.

In a computing environment, the usual authentication process is for a user to sign on to a system, providing an identity and credential evidence. If this sign-on procedure is successful then the user is authorised to use certain computing resources. The initial sign-on procedure is termed the primary sign-on. Additional sign-on procedures are termed secondary sign-ons. Typically a user will have undergo a separate sign-on procedure for each specific system or resource. This means that a user usually has to undergo multiple secondary sign-on procedures during a single session, in addition to the primary sign-on. This often requires the memorization of numerous identities and passwords, it may even mean supplying the same information more than once at different times and it may also involve running different sign-on programs depending on the authentication technology being used.

IBM [38] describe several problems associated with the current user scenario:

- Lack of productivity.
- Lack of convenience increases potential for compromised security.

- Increased support costs.
- Devaluation of password as a security mechanism.
- Increased administration expense
- Lack of availability due to expired, forgotten or out of synch passwords.

As mentioned in the previous chapter, passwords are a simple and convenient form of evidence to use in the authentication process as long as they are kept secret. However, experience has shown that when users have too many passwords to remember they tend to write them down or forget them [38]. Writing them down may devalue the password as authentication evidence, since it is no longer something remembered. Also, there are increased support costs involved in resetting passwords that users have forgotten, and users may not be able to access resources that they need to perform their work if they have forgotten or have out-of-date passwords [38]. Making users re-enter a username and password every time they use a resource adversely impacts productivity and inconveniences them.

3.2 Campus Scenario

At Rhodes University, use of many different resources involves a separate sign-on procedure for each resource. For example, the author has access to at least six Unix systems and two Windows systems that have different password authentication databases. This authentication information is also used to control access to other resources such as printing, using the POP3 protocol to access mail, or retrieving files using the FTP protocol. While it is possible to use the same password on all systems, this results in reduced security because a compromise of a password used with one user account leads to an attacker having access to any of the user's accounts. This is especially a problem due to the fact that many authentication exchanges involve the transmission of a plaintext passwords eg. FTP and POP3 exchanges. Even if the same password is used, there is still the inconvenience that the username may be different for different resources.

Some effort has been made to reduce the credentials that a student has to remember in order to make use of resources. Windows NT Primary Domain Controllers (PDCs) are used to provide centralised storage and control of authentication information for a

group of Windows machines. This is a proprietary solution that is incompatible with other available operating system environments. Additionally, Sainsbury [78] notes that NT 4.0 security mechanisms are significantly weakened by having to provide backward compatibility protocols for Windows 95 clients. The Network Information Service (NIS) is used to provide centralised storage and control of authentication information for a group of Unix machines. While this solution is non-proprietary, it is not available for use with Windows machines, and it is vulnerable to passive dictionary attacks. To be fair, these efforts have reduced the amount of username/password combinations that a typical user has to remember to just two or three. Staff members and postgraduate students typically have to remember more authentication information though.

Even though the amount of authentication information has been reduced, there is still the problem that users are prompted for this information whenever they wish to use a resource. Some application programs, such as Netscape's POP3 client, cache this information so that it does not have to be re-entered, but many more do not. For example, a user using the standard available FTP client will have to re-enter username/password information every time they access a particular server, even if they have already visited this same server earlier in the session.

3.3 Single Sign-On

> "Under SSO, you come into work in the morning, plug in your smart-card, enter the PIN that activates it, and for the rest of the day, you don't have to do any more logins. All of that is handled for you by the SSO mechanism. Attractive isn't it? Of course, it's attractive. Authentication is a pain. Anything we can do to avoid it, we'll jump at." - Schneier and Ellison [30]

A solution to this problem is what is known as Single Sign-On. In a Single Sign-On scenario all identification and credential information that may be required to undertake any secondary sign-on procedures during a session is collected from the user during the primary sign-on procedure [73]. In this way, the user is only prompted for credential information once.

One-way to achieve single sign-on is to have a single authentication mechanism in a security domain and ensure that all resources make use of the mechanism. Kerberos, a distributed

authentication system originally from MIT and now an IETF standard, uses this approach. With Kerberos the primary sign-on involves the user authenticating to a Key Distribution Server (KDS) and obtaining a Ticket Granting Ticket (TGT) if successful. This TGT can be used for a limited time period to undertake secondary sign-on, by authenticating to other resources that trust the KDC, without having to supply user credential information. The Kerberos approach works extremely well in a homogeneous Kerberos-only environment. Such centralised stores of authentication information also make management of resource control policies and user account information easier.

Another way to implement single sign-on is to store user-supplied credential information during the primary sign-on and use it to support secondary sign-on, without the need for the user to re-enter any credential information. Secondary sign-on may take place using the user-supplied credentials directly or by using them to obtain other necessary credentials [73]. For example, a user-supplied password may be used to decrypt an asymmetric private key or one password may be mapped to another password. This approach is more suitable to a heterogenous security environment where there are numerous authentication mechanisms, perhaps using different technologies, being used in existing systems. In the Rhodes University scenario such an approach could be used by a typical user to store the username/password pair for use with a Windows PDC, and the username/password pair to use with an NIS server.

The Java Security Authentication and Authorization Service (JAAS), described next, makes use of the second approach to provide single sign-on functionality.

3.4 JAAS

The Java Authentication and Authorization Service (JAAS) is an attempt to add support for entity authentication to the Java Development Kit (JDK). Currently the JDK enforces access controls based on where the code came from and/or who it was signed by. JAAS aims to provide access controls based on who runs the code ie. an authenticated identity [91]. This user-based access control associates permissions with users, authorizing them to perform particular actions, such as accessing files and opening network sockets.

Our interest in JAAS is due to the fact that it can be used to provide single sign-on [92]. During the authentication process, information can be shared between mechanism implementations thereby removing the need for the user to enter the same information

more than once. Authenticated identities and credentials are stored and made available for future authentication procedures.

3.4.1 PAM

There are a large number of authentication technologies available, and traditionally applications have hardcoded support for particular technologies. To upgrade an existing mechanism or use any new authentication technology requires changes to all such programs. For example, many UNIX applications were rewritten to make use of Kerberos. The Open Group (through SUN) developed a solution to this problem by publishing the Pluggable Authentication Modules (PAM) standard. It provides a means for administrators to specify what authentication information is required to use a particular resource and also provides a means for new authentication mechanisms to be seamlessly plugged in. Application programs call methods in the PAM API in order to authenticate before using a particular resource, and callback methods prompt the user for any authentication evidence required. PAM provides a generic API that sign-on programs can use to authenticate users, a Service Provider Interface (SPI) that can be used to plug in new authentication technologies, and configuration that allows the administrator to specify what combination of mechanisms will be used to authenticate a particular application [48]. It is currently being used for authentication on Solaris and some versions of Linux.

JAAS is based on the PAM specification and provides all this functionality. This means that JAAS has a pluggable architecture in that new authentication mechanisms can be introduced and existing ones replaced. It also has a stackable architecture where a user can be authenticated using more than one mechanism at the same time. For example, this allows an administrator to specify a resource control policy where users may use a resource if they are successfully authenticated using one of a number of mechanisms, or using certain combinations of mechanisms [91].

3.4.2 JAAS Framework

In the JAAS framework, the user in the sign-on procedure is represented by an instance of a Subject class. It holds authenticated identities and public and private credentials. For example an identity may be a username on a host computer, a public credential may be

a public key certificate, a private credential may be a password. The LoginContext interface provides an API that sign-on programs can use to authenticate users. The different authentication mechanisms are represented by classes that implement the LoginModule interface. The Configuration object looks up the configuration for a particular application that specifies what LoginModules to use for the application and the order in which to call them.

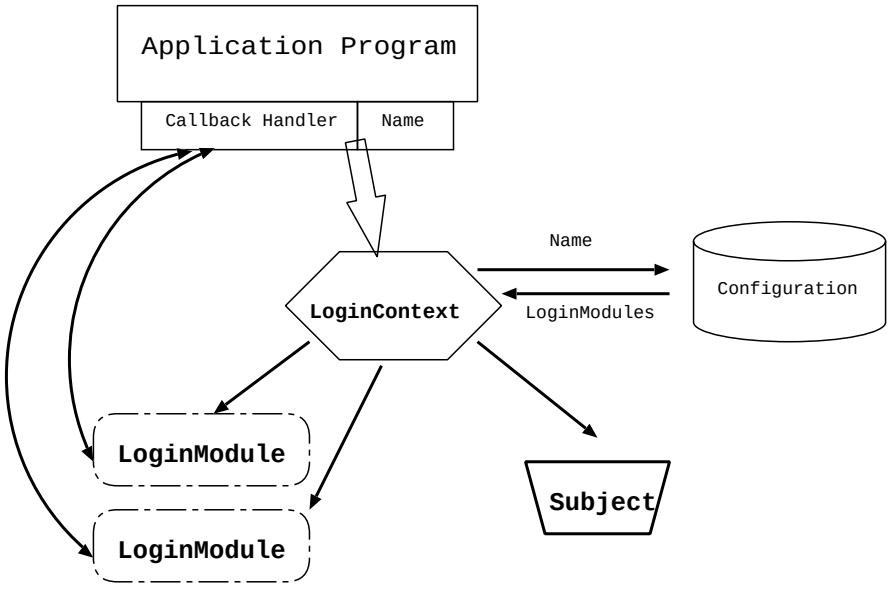

Figure 3.1: JAAS Authentication

An application program using JAAS instantiates a LoginContext passing it the name of the application and a callback handler. Callback handlers are used so that the application itself can decide on the best way to prompt the user for information. This may for example be a simple text prompt, or a pop-up window. The LoginContext looks up the configuration for that application that specifies what LoginModules should be used and how they should interact. The application program then calls the LoginContext login method. This results in the login methods of all the configured LoginModules being called. Each LoginModule attempts to authenticate the user, interacting with the user via the callback handler, and verifying the supplied authentication information in a mechanism-specific manner. (For example, this may be using a password file or a key server). If the authentication procedure meets the requirements specified in the configuration, then the authenticated identities and public and private credential information are added to the Subject. The application can then retrieve the authenticated Subject from the LoginContext [91].

3.4.3 Single Sign-on

If the primary JAAS sign-on operation is configured to obtain from the user all the necessary authentication information that may be required to undertake any further secondary sign-on operations during the session, then the JAAS framework can be used as part of a Single Sign-on implementation.

As an example to illustrate this lets assume that a a username/password and a public/private keypair are all the credentials that are necessary in a particular environment. A user wishes to use a resource that has the following JAAS configuration[1]:

```
auth requisite org.gjt.krb.JAAS.login.PasswordModule
auth required  org.gjt.krb.JAAS.login.PKModule password_stacking=try_first_pass
```

This configuration means that in order to use the resource a user must have both a username/password and a public/private keypair credential. In our implementation we have two LoginModules: the first verifies a username and password, and the second attempts to obtain a public/private keypair from a key store. Obtaining the keypair from the keystore requires the use of a password. The "password_stacking=try_first_pass" indicates to the PKModule that it should attempt to use the password the user entered when interacting with the PasswordModule, thereby eliminating the need for the user to re-enter the password if it is available. In effect, the administrator has implemented a policy to encourage reuse of the same password.

Once the user has undergone the sign-on procedure the password and keypair are stored in the Subject object. Further authentication procedures that need to make use of these credentials will not have to prompt the user but can rather obtain the credentials directly from the Subject object.

Rather than using shared credentials directly, LoginModules can also use these credentials to obtain other credentials. This may for example be by password-mapping, where a given password is used to look up or decrypt another password. Or a user may enter a password that a LoginModule uses to access a LDAP directory and retrieve another password, or perhaps a private key. LoginModules may, with permission, also undertake further sign-on operations on the users behalf, such as connecting to a Kerberos Key Distribution Centre and obtaining a Ticket Granting Ticket, for example.

[1] Interested readers are referred to the JAAS documentation [91] for details on configuration

3.4.4 Implementation

Initially JAAS was only available as an early specification detailing the various components and how they interacted. No implementation was available, although SUN engineers had begun working on one. This specification has evolved over time and at the time of writing SUN have released a beta-quality Early Access version. To validate the specification we have written a partial implementation of an early version of the JAAS API. In particular we have only focused on the authentication part of the framework, omitting the authorisation part. The code works well enough to test the available functionality, but being incomplete and out-of-date should not be otherwise used. The implementation subsequently provided by SUN [91] is more up-to-date.

3.5 Problems with SSO

Entity authentication was defined in the previous chapter as [58]:

> "An identification or entity authentication technique assures one party (through acquisition of corroborative evidence) of both the identity of a second party involved, and that the second was active at the time the evidence was created or acquired"

Schneier and Ellison [30] point out that a serious problem with the whole SSO scenario is that the user may not be active at the time authentication takes place:

> "Unfortunately, the security value of authentication is all but completely defeated by SSO. Authentication is supposed to prove that the user is present at the controlling computer, at the time of the test. Under SSO, when the user has to rush to the washroom, any passing person can walk up to that user's computer and sign on someplace via the SSO mechanism."

It is clear that use of SSO does involve some security risks. Some of these risks can be mitigated by restricting physical access to machines and using tools such as authenticated screensavers to lock a terminal when it is not being used. In addition the time period for which credentials are valid should be restricted to the length of a typical user session,

for example a working day. It is important that such security measures be employed to prevent unauthorised use of a terminal after primary authentication has taken place.

While tools such as password protected screensavers result in additional prompts for credentials, they are a necessary part of a secure system. Since they are only activated after a period of inactivity they should not interfere with normal use of computing resources by users. If such tools are used then a system can be sure that a user has been recently active at the time authentication takes place.

While this is not a perfect solution, SSO seems like a reasonable compromise between applying security measures that are not tiresome for users, while still maintaining an acceptable level of security. In general, there is a direct trade-off between convenience and security. SSO provides convenience to users by reducing the amount of credential information they have to use and remember, and reducing the number of sign-on procedures they have to undertake. It is hoped that this convenience leads to users being more productive and better managing their credential information. Administrators of systems should analyse their particular environments to see whether such a solution is acceptable.

3.6 Conclusion

Much research is focused on authentication protocols, cryptographic techniques and the like. It is important to remember that users are a critical part of the authentication process. Single sign-on aims to reduce the amount of credential information users have to remember and how often they have to supply it, with the goal of making security easier for users and thereby encouraging them to better look after their credentials, while being more productive. Single Sign-on does not guarantee that a user is active at the time secondary authentication takes place, which is an important part of entity authentication. A compromise can be made to allow the system to be sure that the legitimate user whose credentials are being used was recently active on the system. Administrators should decide whether the benefits of single sign-on outweigh the possible risks involved in using it.

Chapter 4

GSS-API

> "It is a generally accepted design principle that abstraction is a key to managing software development. With abstraction, a designer can specify a part of a system without concern for how the part is actually implemented or represented." - Kaliski [42]

4.1 Introduction

The Generic Security Services API (GSS-API), as its name suggests, aims to be a generic API that programmers can use to add security to protocols and applications. It encapsulates security functionality and provides it via a standard abstract API. This shields users from changes in the underlying security technology, localising any changes to the GSS-API implementation and mechanisms rather than all applications that make use of the API. Where APIs such as SASL and the CORBA Security Service (which are discussed in later chapters) are tied to particular application areas and communication environments, the GSS-API tries to be as independent as possible. This means that it is applicable in a wide variety of areas and can be used as the basis for other higher-level or application-specific security architectures.

> "This Generic Security Service Application Program Interface (GSS-API) definition 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." - Linn [53]

Applications and protocols written to use the GSS-API to provide the underlying security will be more resistant to changes in security technology and infrastructure, and will therefore be easier to maintain. Existing GSS-API mechanisms provide strong security using both secret-key and public-key cryptography, and it is easy to introduce new mechanisms as they become available. Because the GSS-API provides the user with an abstraction from the underlying security functionality, this makes it easy for programmers working in other computer-based fields to plug security into their applications without having to be concerned with how the security works, and it shields them from changes in how the security functionality is provided.

4.2 Design

4.2.1 Goals

The GSS-API was designed with certain goals in mind [54]:

mechanism independence - It is independent of the underlying mechanisms used to provide security services. Mechanism implementations can use secret-key or public-key cryptography, for example.

communication protocol independence - It is independent of the communications medium used to carry protocol messages. This means that it can be used on both connection-oriented and connectionless communications environments, for example.

protocol association independence - It is not associated with any particular application protocol (eg. IMAP, SNMP or LDAP), which means that a single GSS-API library can be used by implementations of different application protocols, as well as directly by applications.

placement independence - A GSS-API implementation does not have to operate in a Trusted Computing Base.

This independence makes it particularly resistant to change and applicable for use in a wide number of areas.

4.2.2 Encapsulation

> "One view of object-oriented programming is that it is a discipline that enforces *modularity* and clean interfaces. A second view emphasizes *encapsulation*, the fact that one cannot see, much less design, the inner structure of the pieces. Another view emphasizes *inheritance*, with its concomitant *hierarchical* structure of classes, with virtual functions. Yet another view emphasizes strong *abstract data-typing*, with its assurance that a particular data-type will be manipulated only by operations proper to it." - Brookes [10]

The object-oriented concept of encapsulation is important in security engineering. An object encapsulates some functionality by providing this functionality via a well-defined interface. The actual implementation of the functionality is hidden from the user, and this means that this implementation can be changed without affecting any users, as long as the interface stays the same. It is important for the API to be well-defined so that users know what functionality is available and how to make use of it.

The GSS-API is concerned with encapsulating the functionality of authenticating a user and providing security services over a context. It defines an abstract programming interface that encapsulates this functionality while remaining independent of the way this functionality is implemented.

4.2.3 Resistance to change

A protocol is an agreed upon set of messages that are exchanged between entities in order to achieve some goal. This may for example be the sending of email from one machine to another, establishing a remote shell connection with another machine, or accessing a web page. It is imperative that the protocol is well-defined and all entities involved understand how it works in order for communication using this protocol to be reliable and successful. This means that any changes in the protocol require all parties involved to be aware of these changes and able to support them.

Computer security protocols are designed to achieve the goal of authenticating the identities of the entities involved, and/or protecting the messages exchanged between these entities. With computer security, change is inevitable. This is due in part to advances in cryptology on which much computer security is based, and the ever increasing processing

power available to computer users. In addition, flaws are found in security protocols or mechanisms which cause these protocols or mechanisms to either be revised or discarded. Writing software that uses security in this sea of change is often a challenging and difficult task. It is therefore important that security protocols be designed to be resistant to change.

As an example, Kerberos 4 was (and still is) a popular computer security protocol. Support for this protocol was incorporated into many applications on Unix systems such as telnet, NFS, LDAP etc. Serious flaws were found in the protocol [7], it was revised and Kerberos 5 was released. This version is incompatible with the previous version and changes have to be made to all applications that wish to upgrade to the new version of the protocol. Weaknesses have been found in the Kerberos 5 initial authentication phase [96] and this may also require changes in applications that have hardcoded support for this protocol.

By providing a common API that hides the details of the underlying security protocols, the GSS-API insulates applications from changes in these security protocols. For example, applications that use the GSS-API with Kerberos 4 as the underlying protocol (although they may be unaware of this) could at a later stage use the Kerberos 5 (or any other protocol) without any changes. Changes are only necessary in the GSS-API security mechanisms and the GSS-API implementation itself.

RFC 2025, the specification of the Simple Public Key Mechanism (SPKM), a standardised GSS-API mechanism, notes the following [4]:

> "Because it conforms to the interface defined by [RFC 1508], SPKM can be used as a drop-in replacement by any application which makes use of security services through GSS-API calls (for example, any application which already uses the Kerberos GSS-API for security)."

4.3 Operation

Authentication protocols have much in common with each other. They all aim to authenticate users and provide security services and only differ in how they achieve that goal and the additional security features that they provide. Three elements are common to all authentication protocols: a claimed user identity, evidence to prove this identity, and a security context that represents security settings and the security mechanism messages exchanged. It is therefore possible to provide an abstraction that encapsulates this functionality behind a common, unchanging interface.

The GSS-API encapsulates each of these three elements and their functionality individually. An instance of a GSSName class holds the names associated with an entity. An instance of GSSCredential class holds the credentials associated with an entity. An instance of a GSSContext class holds the negotiated context settings (such as security protection required and whether or not mutual authentication should take place) and is used to generate and process messages.

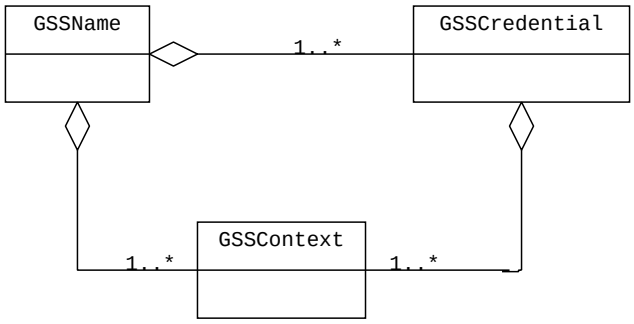

Figure 4.1: Relationships between the main GSS-API classes

In addition various utility classes are specified to handle status information, signal error messages, and manage mechanisms.

In GSS-API terminology the secure session established between the user and the resource controller is called a context and the user is known as the initiator and the resource controller the acceptor. Both the initiator and the acceptor have names and credentials related to security mechanisms that allow them to authenticate themselves.

The GSS-API operational paradigm is described in RFC 2078 [54] as follows:

> "GSS-API operates in the following paradigm. A typical GSS-API caller is itself a communications protocol, calling on GSS-API in order to protect its communications with authentication, integrity, and/or confidentiality security services."

RFC 2025 [4] notes that a GSS-API caller may also be an application program which uses a communication protocol.

A typical context works as follows:

1. The initiator provides to the GSS-API implementation naming and credential information, the security services it requires and the target it wishes to communicate

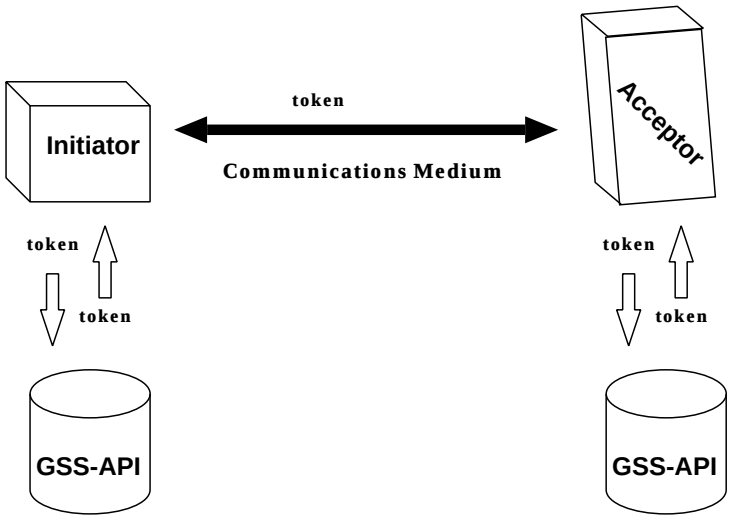

Figure 4.2: GSS-API operation

with. The GSS-API returns an opaque token that the initiator is responsible for transporting to the acceptor.

2. The acceptor provides naming and credential information to its GSS-API implementation, along with the token it received from the initiator. The GSS-API returns an opaque token that the acceptor is responsible for transporting to the initiator.

3. The initiator inputs the received token to its GSS-API implementation. It checks to see whether further tokens need to be sent. If so it transports the token to the acceptor, else the context is established. The acceptor does the same when it receives a token.

4. Once the context is established the initiator and acceptor are able to exchange messages. These messages will be protected with the negotiated security services for the session, such as integrity protection, confidentiality protection and replay detection.

From this description it is clear that the GSS-API is independent of any communication protocol, since it is the caller's responsibility to transport tokens to the other entity involved in the communication. This means that the GSS-API can be used in any communications environment, including a connection-oriented TCP/IP client-server environment, a connectionless UDP/IP client-server environment and a CORBA middleware environment, for example.

When entities request security services from their GSS-API implementations they do so in a generic fashion, specifying the functionality that they require. Although it is possible for a caller to request the use of a particular underlying security mechanism, for maximum portability and to obtain the full benefits of the GSS-API abstraction, callers should merely request the functionality they require. The GSS-API implementation then decides on the appropriate underlying mechanism to use, perhaps after negotiating with the other entity involved in the communication.

The GSS-API specification gives details on naming, credential management, context establishment, status and error reporting, and providing security services. (See section 1.1 "GSS-API constructs" of RFC 2078 [54]). It also specifies function calls in terms of their inputs, their outputs and their expected functionality. These function calls are divided into four groups: credential-management calls, context-level calls, message-level calls, and support calls. (See section 2 "Interface Descriptions" of RFC 2078 [54]).

4.4 Java GSS-API

In order for the GSS-API functionality to be useful in real systems it is necessary to produce an implementation that makes this functionality available to application programmers in a particular programming language. RFC 2078 [54] states that:

> "Concrete language bindings are required for the programming environments where the GSS-API is to be employed"

Such language binding specifications describe how to provide the authentication and security services described at an abstract level by the GSS-API specification in a particular programming language. At present only the binding for the C language has been standardised (RFC 1509 [95]), but work is underway on defining a Java language binding [41].

4.4.1 Description

Java is our language of choice for implementing security-related applications because much security functionality is built into the Java Development Kit (JDK) and related libraries, such as the Java Cryptography Extension (JCE) and the Java Secure Sockets Environment

(JSSE). In addition the fact that Java is object-oriented, type-safe and handles memory management through garbage collection makes it a comfortable language to program in.

The Java language binding has progressed quite quickly during the time that we have undertaken our research and implementation. At the time of writing the IETF Common Authentication Technology (CAT) working group has asked that the Java language binding specification be advanced to Proposed Standard status.

Initially the CAT working group members working on the specification defined concrete classes to implement the GSS-API functionality. Others on the working group felt that such a specification lacked flexibility and made implementations in non-server environments difficult, such as in world-wide-web browsers where applets are heavily restricted. After much discussion it was decided that much of the specification would be defined in terms of interfaces that compliant implementations would implement to provide the necessary functionality. A concrete factory class that is used to instantiate implementations of the interfaces and concrete classes for status and error reporting remain.

Another issue that surfaced during the discussion was the issue of how to handle mechanism implementations. It was thought desirable by some to maximise flexibility by having a provider-based framework, like that used by other Java security services, to allow an implementation to make use of implementations made available by different provider implementations. For example the SUN provider that comes with the SUN JDK provides an implementation of the SHA hash algorithm but not the HAVAL hash algorithm. A user of the JDK can specify that another provider implementing this algorithm should be used in addition to the SUN provider and the HAVAL algorithm will be available in the same way that the SHA algorithm is. Other people keen to use the GSS-API in environments with constraints on space, memory and security, such as embedded environments, did not want to the overheads associated with an additional layer of pluggable providers. It was therefore decided to make the provider-based service provider interface (SPI) a separate specification that would work under the main Java GSS-API (JGSS). Those not wishing to use the SPI are free to implement mechanisms that work directly under the JGSS.

The Java GSS-API specification specifies interfaces whose implementations provide the functionality of the function calls described in the GSS-API. Credential-management calls are provided by the Name and Credential interfaces. Context-level calls, message-level calls and the specification of security services are provided by the GSSContext interface. The MessageProp class is used for status reporting and the GSSException class is used

for error reporting. The Factory class provides a means for applications to instantiate implementations of the JGSS interfaces.

The SPI specification provides a broker layer where GSS-API mechanism providers register the mechanisms that they provide. This broker layer stores naming and credential information for particular mechanisms and allows callers to access the functionality of a particular mechanism implementation.

The API used by application programmers is the JGSS API, which has stabilised and is on the verge of becoming a Proposed Standard. The SPI, which is used by GSS-API mechanism implementors, is under development and still needs to be aligned with the current JGSS API.

4.4.2 Implementation

One of our stated goals of this research was to produce implementations of security functionality that programmers could use to provide security in their applications. To achieve this goal, we have undertaken an implementation of both the main JGSS API and the SPI that works under it. The implementation is functional and has been contributed to the international Cryptix project[1] where it continues to be developed [21]. One problem that we have had with all of our implementations is that the specifications that they are based on have not been finalised and standardised. This has made it difficult to construct an implementation that conforms to the API, because this goal has been a moving target. Our implementations are based on recent versions of the respective specifications, and continue to track new developments.

In order for our Java GSS-API library to provide or test any real functionality it is necessary to have an implementation of a GSS-API security mechanism. GSS-API security mechanisms are discussed in the next chapter.

4.5 Example

As an example of how the GSS-API might be used to secure an application let us assume the following scenario:

[1] http://www.cryptix.org/

We have a resource that makes examination marks available to students. Due to the sensitivity of this information, a student must first authenticate to the resource controller using her student number and memorised password. The resource controller must verify the student's credentials and if successful return the student's examination marks.

This resource is implemented as a client-server application with the server listening on a well-known port. Authentication of the student could be performed by hardcoding support for a particular security technology, such as the Secure Remote Password protocol, into the application. However, if the security technology on the server was at a later stage changed, to Kerberos perhaps, this would require that the client and server be changed to make use of this new technology. To make future maintenance easier, authentication is provided by a GSS-API implementation.

The client and server establish a mutually authenticated security context by making the appropriate GSS-API function calls. They request that the default GSS-API mechanism be used, which makes them unaware of the specific GSS-API mechanism that is being used to provide them with security functionality, and they request that the integrity and confidentiality protection security services be provided over the context. Once the context is established, the server sends the examination marks to the client. Before being sent, this information is wrapped in a GSS-API token, protected using the requested security services. Upon receipt by the client, this information is unwrapped, unprotected and presented to the student in its original form.

The client and server will be unaware and unaffected by any change in the GSS-API or underlying GSS-API mechanism used to provide them with authentication and security services. Any changes will be confined to the GSS-API implementation and GSS-API mechanisms.

4.6 Conclusion

The GSS-API is an abstract API for providing security in a manner independent of the environment in which this security is used. Authentication protocols all aim to authenticate users and provide security services and only differ in the security technology used to do so. The encapsulation of the functionality of authentication protocols behind a common API insulates users of the API from the inevitable changes in the underlying technology.

Implementations of the abstract API in a particular language, such as the Java language binding implementation provided, make the API calls available to users.

Chapter 5

GSS-API Mechanisms

5.1 Introduction

The underlying security functionality of a GSS-API implementation is provided by GSS-API mechanisms.

A GSS-API mechanism specification describes the procedures and messages that the mechanism uses to establish an authenticated context and provide security services over this context. The notation used to specify the message data structures must be unambiguous and flexible enough to express the data used by different mechanisms. The Abstract Syntax Notation One has been chosen to specify the message data structures for GSS-API mechanisms.

The procedures used to populate and interpret mechanism messages depend on the underlying security protocols and technology that are used by the mechanism. Existing mechanism specifications define mechanisms that make use of shared key technologies, asymmetric-key technologies or a combinations of both.

Mechanisms may also be classified according to the infrastructure that they require to operate. Recent interest in the IETF CAT working group has been in mechanisms that have lower infrastructure requirements than the standardised GSS-API mechanisms - SPKM and Kerberos.

5.2 Abstract Syntax Notation One

5.2.1 Introduction

Professor John Larmouth of the University of Salford[1] in his book *ASN.1 Complete* [50] describes the Abstract Syntax Notation One (ASN.1[2]) as follows:

- It is an internationally-standardised, vendor-independent, platform-independent and language-independent notation for specifying data structures at a high level of abstraction.
- It is supported by rules which determine the precise bit-patterns (again platform-independent) to represent values of these data-structures when they have to be transferred over a computer network, using encodings that are not unnecessarily verbose

RSA Laboratories guide to ASN.1 [42] describes the ASN.1 notation:

> "ASN.1 is a flexible notation that allows one to define a variety [of] data types, from simple types such as integers and bit strings to structured types such as sets and sequences, as well as complex types defined in terms of others"

ASN.1 was originally developed for use with the ISO's (International Standards Organisation) OSI (Open Systems Interconnection) architecture, but has gained wide use in the specifications of a number of protocols.

OSS Nokalva [64] describe a number of current uses of ASN.1, a few of which are listed here:

- It is used in the TCAP protocol used in cellular phones
- It is used by telecommunications companies for routing (using the Signalling System 7 OMAP messages) and providing ISDN services
- It is used in aviation flight control systems and ground-to-ground exchanges

[1] http://salford.ac.uk/iti/jl/larmouth.html

[2] The original abbreviation for the Abstract Syntax Notation was ASN1. The dot was added to avoid confusion with ANSI, the American National Standards Institute [50].

- Federal Express uses ASN.1 to track its packages
- It is used in the ISO X.509 standard to specify digital certificates
- Kerberos 5 uses it to specify all of its message structures

5.2.2 Encoding Rules

> "The encoding rules say how to represent with a bit-pattern the abstract values in each basic ASN.1 type, and those in any possible constructed type that can be defined using the ASN.1 notation." - Larmouth [50]

There are several standard encoding rules[3] used to convert the ASN.1 data structures into bitstrings:

- Basic Encoding Rules (BER)
- Distinguished Encoding Rules (DER)
- Canonical Encoding Rules (CER)
- Packed Encoding Rules (PER)

The Basic Encoding Rules (BER) were the first standardised encoding rules and they use the Type-Length-Value encoding. They provide two options for encoding data values depending on whether the length of the data value is known beforehand or not. This gave rise to the Canonical Encoding Rules (CER) and the Distinguished Encoding Rules (DER) which are variants of BER that encode data values each using one the options available in BER [50].

Encoding rules, such as DER and CER, that provide exactly one possible encoding for a particular populated ASN.1 structure are important for security purposes. For example, X.509 public key certificates are specified in ASN.1. The public key of a particular X.509 certificate should only have one possible encoding so that the signature on this public key can be verified correctly. The Distinguished Encoding Rules (DER) are the most popular

[3] ASN.1 Complete [50] provides excellent coverage of the different types of encoding rules. The interested reader is referred there for more information.

with regards to security. They have gained wide use is security services due to their use in the ISO X.509 standard for specifying digital certificates, and their use in specifying message structures in the Kerberos 5 authentication protocol.

The Packed Encoding Rules were designed to provide very compact encodings of ASN.1 data structures. They are used in bandwidth-constrained environments where the protocol messages must be as small as possible. (For example, they are used in aviation to communicate between ground stations and aircraft) [50].

One new set of ASN.1 encoding rules that we find particularly intriguing and hope will gain wider use in future is the **XML Encoding Rules** (XER)[101]. These rules are currently being standardised by a joint ISO/ITU committee. We believe that they will be particularly applicable to IETF protocols, which are typically character-based.

> "Extensible Markup Language (XML) is a simple, very flexible text format derived from SGML. Designed to meet the challenges of large-scale electronic publishing, XML will also play an increasingly important role in the exchange of a wide variety of data on the Web. ... XML is a low-level syntax for representing structured data ... The flexibility of XML makes it ideal for interchange of structured data ." - Connolly [18]

These rules specify a means of converting ASN.1 structures into XML. Since XML is text-based this makes the encodings human-readable, making it easier to manually view and interpret the encoded ASN.1 structure. Also, standard ASN.1 tools can be used to encode and decode the ASN.1 data structures. This is an important feature because one of the biggest criticisms of ASN.1 is the fact that use of the current encoding rules realistically requires the use of specialised tools. One disadvantage of this encoding, however, is that it is much more verbose than any of the other encodings due to the fact that it is character-based.

> "The encoding rules approach enables a degree of information hiding (and flexibility in making future changes to encodings) that is hard to match with other approaches to specifying encodings." - Larmouth [50]

The separation of message syntax from message encoding has some advantages. Separating the encoding rules that convert the abstract ASN.1 structures in bitstrings from the ASN.1

structures themselves means that the encodings can be changed without affecting the ASN.1 structures. This allows different encodings of the same ASN.1 structures do be used in different environments. For example, in a bandwidth-constrained environment the Packed Encoding Rules may be used, and in an environment where it is important that there be only one possible encoding the Distinguished Encoding Rules may be used. It also means that it is possible for a protocol to easily use a new set of encoding rules, without affecting the protocol data structures.

5.2.3 Using ASN.1

ASN.1 provides a protocol designer with a rich set of elements with which to construct the messages used in the protocol. This makes it possible to specify complex message structures in a clear and precise manner, with no ambiguity. It hides final data representation and allows the protocol designer to concentrate on the contents of the protocol message structures rather than how the message structures will be encoded.

Implementing a protocol specified using this notation requires code in a particular programming language to represent, encode and decode the message structures according to a set of encoding rules (eg. DER). Fortunately there are software libraries available that provide implementations of the basic elements that can be used to represent message structures in a particular language. They also provide support for encoding and encoding these message structures using one or more of the standard encoding rules. There are also some tools available that take an ASN.1 specification and automatically generate all the programming language code necessary to represent the message structures and encode and decode messages during protocol interaction.

> "It [ASN.1] is supported by tools available for most platforms and several programming languages that map the ASN.1 notation into data-structure definitions in a computer programming language of choice, and which support the automatic conversion between values of those data structures in memory and the defined bit-patterns for transfer over a communication line ... The implementation task is a simple one: the only code that needs to be written (and debugged and tested) is the code to perform the semantic actions required of the application. There is no need to write and debug complex parsing or encoding code...

> It is the case today that there are good ASN.1 tools (called "ASN'1 compilers") available that will map an ASN.1 type definition to a type definition in (for example), the C, C++, or Java programming languages (see Section I Chapter 6), and will provide run-time support to encode values of these data structures in accordance with the ASN.1 Encoding Rules. Similarly, an incoming bit-stream is decoded by these tools into values of the programming language data-structure." - Larmouth [50]

Attempting to implement an ASN.1-specified protocol without the aid of such automated tools is difficult and error-prone, as we discovered with our LIPKEY implementation (described below). Our first use of ASN.1 involved using the DSTC[4] Java ASN.1 library [27]. This library provides implementations of the basic ASN.1 elements and a means of implementing your own ASN.1 structures using these elements. It also provides a means of encoding and decoding the ASN.1 data structures using BER or DER. Implementing our own ASN.1 structures turned out to be time-consuming and error-prone.

This led to the adoption of the Cryptix ASN.1 compiler [20] which takes an ASN.1 specification and automatically produces Java code that implements the ASN.1 structures. The use of this compiler and its associated library drastically reduces the amount of time necessary to test and use an ASN.1 specification, since many of the time-consuming activities are performed automatically. We also used ARC [32] from Forge Technologies which provides similar functionality.

ASN.1 is used in IETF standards when public key infrastructure is involved, since ISO X.509 digital certificates are used and these are specified using ASN.1 and encoded using DER. Additionally, ASN.1 is used with version 5 of the Kerberos protocol. Hence, it is used to specify the message structures in both the Kerberos and Simple Public Key Mechanism (SPKM) GSS-API mechanisms. Other work-in-progress GSS-API mechanism specifications also use ASN.1 because many of them are based on SPKM.

5.2.4 Problems with ASN.1

ASN.1 is not without its critics, since in practice there are problems related to its abstraction, complexity, accessibility and use.

[4]Distributed Systems Technology Centre at Queensland University of Technology

5.2.4.1 Abstraction

ASN.1 advocates claim that the way in which the ASN.1 structures will eventually be encoded is of no consequence to the designer of these structures and should be ignored.

> "You don't know or care about the electrical or optical signals used to represent bits, so why care about the bit patterns used to represent your abstract values?"
> - Larmouth [50]

Others disagree. Tom Yu from MIT criticises this claim:

> "Another problem with ASN.1 is the outright *lie* that you can write an abstract syntax for the protocol without regards to how it will be encoded. That might be a nice theoretical dream, but actual implementations as well as the design of the encoding rules basically ensure that you *cannot* write an effective abstract syntax for a protocol without careful consideration of how it will be encoded."
>
> "If protocol designers end up designing protocols that they have no clue of the encodings of, it is quite possible that only at the implementation stage will problems with the protocol be discovered. This is a way in which needless abstraction can hurt badly."
>
> – Tom Yu - IETF CAT WG mailing list - 10 December 1999

We believe that designing a protocol without concern for how or where it will be used is not a sensible thing to do. However, there are currently encoding rules available that are suitable for use in most environments that are likely to be encountered at present.

5.2.4.2 Complexity

The currently standardised ASN.1 encoding rules are highly optimised and complex. This complexity means that it is necessary to employ the use of a software tool in order to perform these encodings, as described above, since it is extremely difficult to perform the encodings by hand. This complexity and the need for such specialised tools has hampered ASN.1's adoption for use with Internet standards. Even in IETF standards where ASN.1 has been adopted, problems have been experienced, as noted by one of the Kerberos developers:

> "Certainly our experience with Kerberos is that there were a lot of interoperability problems caused by the use of people misunderstanding ASN.1, and if the protocol data units are simple enough, it's not clear that ASN.1 adds sufficient value to be worth its cost in implementation complexity. If we were doing Kerberos V5 all over again, we would *not* have used ASN.1. It has caused us far more problems than it has ever saved us, and it is continuing to cause us problems even today as we discuss RFC 1510bis."
>
> – Theodore Y. Ts'o - IETF CAT WG [40] mailing list - 5 March 1999

In a later message he adds:

> "ASN.1 has a lot of complexity to it, and so it gives you plenty of rope with which to hang yourself (and other hapless users and implementors of your protocol) with. The flameage of whether or not ASN.1 is a good thing or not basically revolves around quantifying the costs and the benefits. Advocates of ASN.1 think the benefits are great, and downplay the costs, saying that "someone who knows what they're doing won't have problems with ASN.1" and "that's what ASN.1 compilers are for". People who don't like ASN.1 counter that the costs are a lot higher"
>
> – Theodore Y. Ts'o - IETF CAT WG [40] mailing list - 10 December 1999

We believe that the complexity of the encoding rules and the difficulty of using them in practice without specialised tools is one of the biggest problems with the use ASN.1. It is our hope that the XML Encoding Rules (XER) described above will alleviate this problem somewhat due to the fact that generic XML tools can be used to perform the encoding and decoding.

5.2.4.3 Accessibility

An additional factor is that the ASN.1 specification (X.208) is an ISO document and thus only available for a fee. Although it is possible to download documents such as RSA Laboratory's *A Layman's Guide to a Subset of ASN.1, BER and DER* [42] and, recently and more importantly, Professor John Larmouth's *ASN.1 Complete* [50] for free, these are no substitute for the actual standards documents, as expressed by Tom Yu from MIT on the CAT WG mailing list:

"Anyway, the problems I see with the ASN.1 standards in terms of someone not connected to the ISO/ITU-T standards process are largely due to the inaccessibility of authoritative copies of the specification, including:

* The ASN.1 specification is a joint ITU-T/ISO effort, thus multiplying by two the number of documents out there.

* There have been multiple revisions of the documents, some of which are not completely compatible with each other, but these incompatibilities largely revolve around the deprecation of the macro notation in 1994 or so and replacing it with the "information object" specification.

* The ISO and ITU-T versions of the same documents have different "publication" dates due to procedural differences between the ISO and ITU-T.

* Not the least, both the ISO and the ITU-T want serious $$$ for copies of the spec."

– Tom Yu - IETF CAT WG [40] mailing list - 9 December 1999

5.2.4.4 Our experience

In our implementation of the LIPKEY GSS-API mechanism and the development of our GSS-API mechanism (SRPGM), ASN.1 presented us with the greatest difficulties.

It was initially difficult for us to find information on ASN.1 and when we did it was incomplete. (The recent availability of *ASN.1 Complete* [50] as a free download is extremely welcome in this regard). The initial version of the SRPGM specification was based primarily on the SPKM specification without a thorough understanding of ASN.1. We have improved on the specification as we have gained more experience working with ASN.1.

In our experience, the use of an ASN.1 compiler can make the use of ASN.1 in a protocol implementation fairly painless if the compiler output is well documented and bug-free. However, existing Java compilers that we have found are either expensive, have bugs or are not yet complete. We hope that this will change in the near future.

5.3 Standard Mechanisms

The currently standardised GSS-API mechanisms are Kerberos [45], which is uses secret-key technology, and the Simple Public Key Mechanism (SPKM) [4] that uses public-key technology. Recently GSS-API mechanisms have been proposed that have lower infrastructure requirements than these standard mechanisms. These include the Low Infrastructure Public Key (LIPKEY) mechanism and the Secure Remote Password GSS-API Mechanism (SRPGM).

5.3.1 SPKM

The Simple Public Key Mechanism (SPKM), as its name suggests, provides authentication and security services under the GSS-API based on public-key cryptography (see 2.3.1.2). RFC 2025 [4] (the SPKM specification) describes it as follows:

> "This mechanism provides authentication, key establishment, data integrity, and data confidentiality in an on-line distributed application environment using a public-key infrastructure."

RFC 2025 defines the protocol procedures and message structures for negotiating the algorithms to be used over the context and the security services to be provided, for undertaking unilateral or mutual authentication and for exchanging messages after context establishment is complete. It describes two GSS-API mechanisms that have minor differences. SPKM-2 requires the use of secure timestamps for replay detection during context establishment, which requires that the entities involved have access to secure time. To avoid the requirement of secure time, SPKM-1 uses random numbers for replay detection during the context establishment phase, but this involves an extra message exchange in the case of unilateral authentication.

SPKM makes use of public key cryptography to perform authentication. All context-establishment tokens are integrity protected which allows the receiver to check if they have been tampered with. The two parties involved negotiate the sets of one-way function, integrity, and confidentiality algorithms that will be available for use over the context. They also negotiate the security services, such as integrity protection, confidentiality protection sequencing and replay detection, that will be used to protect messages after context establishment.

5.3.2 Kerberos

Kerberos is a network authentication service based on shared secret keys and the use of a trusted third parties.

RFC 1510 [45] describes the Kerberos authentication process:

> "The authentication process proceeds as follows: A client sends a request to the authentication server (AS) requesting "credentials" for a given server. The AS responds with these credentials, encrypted in the client's key. The credentials consist of 1) a "ticket" for the server and 2) a temporary encryption key (often called a "session key"). The client transmits the ticket (which contains the client's identity and a copy of the session key, all encrypted in the server's key) to the server. The session key (now shared by the client and server) is used to authenticate the client, and may optionally be used to authenticate the server. It may also be used to encrypt further communication between the two parties or to exchange a separate sub-session key to be used to encrypt further communication."

The above description describes two different sub-protocols. The first is the protocol involving message exchanges between the client and the Authentication Server (AS) - a trusted third party with which the client shares a secret - where the client obtains a ticket. The second in the protocol involving message exchanges between the client and the server where the client uses the ticket to authenticate itself to the server.

Kerberos also has the notion of a special type of server called a Ticket-Granting Server (TGS) that issues tickets for other servers. In order to use the TGS, the client first obtains a special ticket from the Authentication Server called the Ticket Granting Ticket (TGT). The client then presents this TGT to the Ticket-Granting Server and requests a ticket for an application server [45]. (The use of the TGT allows Kerberos to provide Single Sign-On - see 3).

The Kerberos GSS-API mechanism, described in RFC 1964 [56], encapsulates the Kerberos client/server authentication protocol that takes place between the client and the application server. It assumes that the client has already authenticated itself to the Authentication Server (and perhaps Ticket-Granting Server) and obtained a ticket for the application server. RFC 1964 describes how a secure context is established between the client and the

application server, authenticating the client and server to each other, and negotiating the security services to be employed and the algorithms to be used.

5.3.3 Infrastructure Requirements

SPKM requires some additional infrastructure in order to operate. As discussed in 2.3.1.2, when public-key technology is used it is necessary to have a Certification Authority that all entities in a security domain trust to sign and certify public keys, generating public key certificates. All entities involved in using SPKM need to have a private key and a public key certificate signed by a trusted Certification Authority. This means that procedures need to be put in place to control the generation of keypairs and the signing of public keys. Secure storage needs to be provided to store the private keys, and procedures need to be implemented to restrict access to these keys only to legitimate users.

Also RFC 2025 [4] says:

> "In order to accomplish context establishment, it may be necessary that both the initiator and the target have access to the other party's public-key certificate(s)".

This is certainly the case with mutual authentication, which RFC 2025 states is expected to be the common case [4]:

> "It is envisioned that typical use of SPKM-1 or SPKM-2 will involve mutual authentication. Although unilateral authentication is available for both mechanisms, its use is not generally recommended." - Adams [4]

This means that it is necessary for the initiator and acceptor to exchange public-key certificates during context establishment phase, or obtain the required certificates from an external source, which is usually a centralised directory where entities can look up the required certificates using the name of the entity associated with the certificate.

Kerberos also has additional infrastructure requirements. In order for entities to communicate using the Kerberos protocol they have to make use of a trusted third party - an authentication server with which they have established a shared key - in addition to the infrastructure required on each client and application server to manage and use tickets [45]:

> "The implementation consists of one or more authentication servers running on physically secure hosts. The authentication servers maintain a database of principals (i.e., users and servers) and their secret keys." - Kohl [45]

These infrastructure requirements make it difficult to deploy these mechanisms in some environments. Due to their complexity, the client and server architectures need to be installed and maintained by experienced administrators who have the necessary skills to do so. The use of public-key infrastructure on the client-side, as opposed to a simple remembered password, may require the use of additional technology such as smart cards and the training of users in the management and use of this infrastructure and technology.

The need for GSS-API mechanisms with lower infrastructure requirements was acknowledged by the community making up the Common Authentication Technology (CAT) working group (WG) [40] of the Internet Engineering Task Force (IETF) that defines and manages the GSS-API and related security technologies. John Linn, chairman of the CAT working group, in an email message to the CAT mailing list:

> "At several points during the Orlando IETF meeting, culminating at the EAP BOF[5] on Friday morning, I heard interest expressed in the availability of "low-infrastructure" mechanisms under GSS-API. Most of the mechanisms which have been discussed within CAT require fairly substantial infrastructures to be deployed in order to operate. Simpler technologies (e.g., challenge-response mechanisms, some of which have been designed for use at the SASL layer), have been considered easier to apply but may not offer the same range of security services or resist the same range of attacks. If such mechanisms were to be implemented under GSS, it could become possible to port GSS caller applications to a broader range of environments."

Several low-infrastructure mechanisms have been proposed. The Low Infrastructure Public Key mechanism (LIPKEY) has gained the greatest support and is soon to become a proposed standard. Other mechanisms include the SPKM with Shared Secret Keys Mechanism (SSKM), the GSSEasy mechanism, and the Secure Remote Password GSS-API Mechanism (SRPGM), which we developed.

[5] Birds of a Feather(BOF). A meeting of people with a common goal.

5.4 Low Infrastructure Public Key mechanism

We wanted to implement a GSS-API mechanism to make security functionality available with our implementation of the Java GSS-API bindings. After searching for existing Java implementations of these mechanisms, we found that a Java implementation of Kerberos had been undertaken by both the Open Group [72] and DSTC [26], but that there was no available Java implementation of SPKM.

However, SPKM requires the deployment of public key infrastructure on both clients and servers in order to operate. Instead, our interest in mechanisms with lower infrastructure requirements led us to implement LIPKEY, a close relative, that only requires such supporting infrastructure on the server side.

5.4.1 Overview

LIPKEY is a low-infrastructure GSS-API mechanism that performs password-based authentication using a variant of SPKM. It aims to operate in the same paradigm as the Transport Layer Security protocol (formerly known as the Secure Sockets Layer or SSL protocol), which is as follows [29]:

- initiator obtains the acceptor's certificate,
- verifies that it was signed by a trusted Certification Authority (CA),
- generates a random symmetric session key,
- encrypts the session key with the server's public key,
- sends the encrypted session key to the server
- sends additional data protected using the session key

With SPKM-1 and SPKM-2 both the client and the server are required to have private keys and public key certificates. LIPKEY only requires that the server have a private key and public key certificate, thereby reducing the infrastructure requirements on the client-side. The LIPKEY specification defines a new SPKM mechanism, called SPKM-3, that is similar to unilateral authentication using SPKM-1. It makes some changes to

the mandatory integrity, confidentiality and one-way-function algorithms, and makes some requirements on the contents of tokens, but does not substantially change the operation of SPKM. After SPKM-3 authentication is complete, LIPKEY makes use of the GSS-Wrap and GSS-Unwrap GSS-API methods to encrypt the username and password sent from the client to the server [29]. As with all other GSS-API mechanisms, message structures are defined using Abstract Syntax Notation One (ASN.1) and messages are encoded using the Distinguished Encoding Rules (DER).

5.4.2 Infrastructure Requirements

The client is required to have a memorized password and a list of trusted Certification Authorities (CAs) and their public keys. The server is required to have a password database, a private key, and a public key certificate signed by one of the trusted CAs. Many systems already have a password database, so the private key and public key certificate on the server, and the public keys of trusted CAs on the client are the only additional infrastructure requirements [29]. Hence the reason why LIPKEY is known as a low-infrastructure mechanism.

5.4.3 Implementation

Our implementation of LIPKEY was based on an early version of the Java GSS-API specification. We implemented it directly under this API, rather than through a service provider layer, since no such layer had been defined at this time. We used the ASN.1 library from DSTC [27], which required us to manually implement the LIPKEY ASN.1 structures using the ASN.1 elements provided. Fortunately the source code for the library was provided which gave us some idea of how to go about implementing the ASN.1 structures as Java classes. This implementation of the ASN.1 classes turned out to be time-consuming and error-prone though, and led to slow development of the mechanism implementation. The implementation of the context-establishment functionality was completed, but there were still some minor ASN.1-related bugs in the code. We have recently started moving our LIPKEY implementation over to using the Cryptix ASN.1 compiler and related classes, but this has not yet been completed.

5.5 Secure Remote Password GSS-API Mechanism

5.5.1 Introduction

Our initial work with the GSS-API involved understanding SPKM, and then implementing LIPKEY (SPKM-3). During the course of this implementation we learnt a lot about how the GSS-API functions, how the Abstract Syntax Notation One (ASN.1) and its Distinguished Encoding Rules (DER) work, and what measures could be employed to counter attacks.

We noted that there was no available password-based GSS-API mechanism that provided strong authentication and security services, while not requiring the use of long-term asymmetric keys or significant infrastructure to operate. The only standardised password-based GSS-API mechanism was Kerberos, and it needed to make use of trusted third party infrastructure - a Key Distribution Server (KDS) - in order to operate. LIPKEY (described above) is password-based and has low infrastructure requirements, but it requires that each server use and manage long-term asymmetric keypairs in addition to maintaining a password-database used to authenticate users.

The Secure Remote Password protocol (SRP), which is described in more detail in 2.4, provides strong authentication and the participants negotiate a session key which can be used to provide security services. It requires very little infrastructure to operate:

> "Trusted key servers and certificate infrastructures are not required, and clients are not required to store and manage any long-term keys." - Wu [99]

The initiator requires a remembered password, and the acceptor a verifier database. This means that it has even fewer infrastructure requirements that LIPKEY does, while still providing the same level of security. We realised that we could use SRP and the knowledge that we had gained working with SPKM to implement our own GSS-API mechanism that filled this niche.

Our aim was to take the SRP protocol and make it available as a GSS-API mechanism, providing additional security services and countermeasures. We decided to base our mechanism on SPKM with which we were familiar.

5.5.2 Operation

We used optimised SRP, as described in [98], since this reduces the total number of messages exchanged by grouping together pieces of information that do not depend on earlier messages. The SRP data (refer to 2.4) and the tokens used to carry this data are illustrated below:

	SRP data		**GSS-API Token**		**SRP data**	
	C, A, n, g	–	SRPGM-REQ	$\longrightarrow$		
Initiator		$\longleftarrow$	SRPGM-REP	–	s, B	**Acceptor**
	M_1	–	SRPGM-EVIDENCE	$\longrightarrow$		
		$\longleftarrow$	SRPGM-EVIDENCE	–	M_2	

The *SRPGM-REQ* and *SRPGM-REP* tokens also contain information used to negotiate the sets of integrity, confidentiality, and one-way function algorithms that will be used over the context, the security services that will be active after context-establishment is complete, and the time period that the context will be valid for.

This assumes that the values for n and g that the initiator used were the same as those being used by the acceptor. If initiator uses different values or doesn't know what values to use, an extra message needs to be sent by the acceptor to inform the initiator of the correct values:

	SRP data		**GSS-API Token**		**SRP data**	
	C, A, n, g	–	SRPGM-REQ	$\longrightarrow$		
		$\longleftarrow$	SRPGM-PARAMETERS	–		
Initiator	C, A, n, g	–	SRPGM-REQ	$\longrightarrow$		**Acceptor**
		$\longleftarrow$	SRPGM-REP	–	s, B	
	M_1	–	SRPGM-EVIDENCE	$\longrightarrow$		
		$\longleftarrow$	SRPGM-EVIDENCE	–	M_2	

We recommend that the initiator cache the values for n and g used by a particular acceptor to avoid this extra message exchange in future contexts.

5.5.3 Relationship to SPKM

SRPGM context establishment tokens correspond to SPKM context establishment tokens as follows:

SPKM	SRPGM
SPKM-REQ	SRPGM-REQ
SPKM-REP-TI	SRPGM-REP
SPKM-REP-IT	SRPGM-EVIDENCE
SPKM-ERROR	SRPGM-ERROR

5.5.3.1 Reused features

We were able to reuse many of the procedures, structures and countermeasures that SPKM employs:

- The method of negotiating algorithms to be used over the context

Algorithm negotiation takes place as follows: The initiator proposes sets of integrity, confidentiality and one-way functions that it wishes to use over the context and sends this information to the acceptor in the *SRPGM-REQ* token. For interoperability, these sets must include those algorithms for which mandatory support is specified in the SRPGM specification. The acceptor selects out of these sets those algorithms that it is able to support over the context, and sends this information to the initiator in the *SRPGM-REP* token. If the initiator is not happy with the negotiated algorithms it aborts the context, otherwise these sets of algorithms become those that are available for use over the context. See section 4.5 of SRPGM [14].

- The process of deriving subkeys from the negotiated session key

Once the sets of integrity and confidentiality algorithms that are to be used over the session have been negotiated, and the shared session key has been established, subkeys are generated for each of these algorithms. Having a separate key for each algorithm improves security, because it means that there are more keys that an attacker has to crack in order to completely breach security, and if an attacker manages to obtain a particular subkey

due to a weakness in the algorithm, this does not compromise the keys used with other algorithms. It also means that the negotiated session key is not used directly, which means that it is not available for an attacker to crack.

- The use of sequence numbers to avoid malicious loss, replay or re-ordering of tokens after context establishment

As discussed in 2.3.3, sequence numbers can be used to detect tampering by making each integrity protected token in a session unique. They are only really useful when used in conjunction with integrity protection, because this allows modification of the sequence numbers to be detected. Also, since a different session key is used for each session, tokens from other sessions (past or parallel) can be detected because the integrity checksums will not match.

- Quality of Protection values, and support functions

Quality of Protection values are a way for callers of GSS-API message protection functions to select amongst the available integrity and confidentiality algorithms using high level qualifiers, such as strong, medium and weak.

- ASN.1 token formats and procedures for per-message and context deletion tokens

After context establishment is complete, SRPGM operates in the same way that SPKM does, using the same token formats and procedures.

5.5.3.2 Differences

- Use of public-key technology

SRPGM is not based on public-key technology. Public-key specific data (such as digital certificates) have been replaced with SRP-specific data. Public-key specific options (such as requesting the other party to send a public key certificate) have been removed. In addition, algorithms that use asymmetric keys (eg. md5WithRSAEncryption) have been replaced with corresponding algorithms that use symmetric keys (eg. HMAC-MD5) .

- Key establishment

The entities using SPKM negotiate the use of a separate key establishment algorithm to establish a session key for use over the session. For key establishment, RFC 2025 specifies that mandatory support for the use of the RSA Encryption algorithm must be provided and recommends that support for the use of the Diffie-Hellman key agreement protocol be provided. With SRP, key establishment is an inherent part of the protocol.

- Replay detection during the context establishment phase

SPKM-1 uses random numbers for replay detection during the context establishment phase. The initiator generates a fresh random number (ie. a number with a high probability of not having being used before) and sends it to the acceptor with the *SPKM-REQ* token. The acceptor generates a fresh random number and sends it along with the initiator's random number to the initiator. Subsequent tokens include both of these random numbers.

With SRP, the initiator and acceptor generate fresh ephemeral asymmetric keypairs at the start of each protocol session. The generated public keys are included in the *SRPGM-REQ* and *SRPGM-REP* tokens respectively. Since these public keys are freshly generated they perform the same function as the random numbers in SPKM. In addition, each *SRPGM-EVIDENCE* token is different because it contains evidence to prove knowledge of the session key, which is different for each session.

5.5.4 Evolution of SRPGM

This mechanism has evolved over time and below we discuss the various versions of the mechanism, the problems and issues we encountered and how they were resolved.

5.5.4.1 Initial Design

As mentioned above, we decided to base the mechanism on SPKM-1. So, we started with the SPKM ASN.1 tokens and began removing the public-key specific data, since SRP is secret-key based. Thereafter we added the SRP data elements to the appropriate tokens.

	SRP data		GSS-API Token		SRP data	
	C, A	–	SRPGM-REQ	$\longrightarrow$		
Initiator		$\longleftarrow$	SRPGM-REP	–	s, B, u, n, g	**Acceptor**
	M_1	–	SRPGM-EVIDENCE	$\longrightarrow$		
		$\longleftarrow$	SRPGM-EVIDENCE	–	M_2	

Although the contents and structure of the tokens has changed as the mechanism evolved, this basic framework has remained.

5.5.4.2 Integrity Protection

With SPKM-1 both the initiator and the acceptor have public keys that they can use to integrity protect tokens that they send. All SPKM-1 tokens sent during the context-establishment phase are integrity protected, using a checksum of the token data signed using the sender's public key, which means that any tampering with these tokens can be detected.

Since SRP is based on shared keys, integrity protection can only be provided once both parties know the shared key. The shared context key is only available to both parties after the acceptor has received the *SRPGM-REQ* token and the initiator has received the *SRPGM-REP* token. The contents of the *SRPGM-REQ* and *SRPGM-REP* tokens were not integrity-protected and could be modified by an attacker (to negotiate the use of the weakest possible encryption algorithm, for example). For this reason, information negotiated using these tokens (eg. algorithms, sequence numbers and security settings to be used over the context) was included in the *SRPGM-EVIDENCE* token, which was integrity protected using a checksum of the token data, the HMAC-SHA1 algorithm and the negotiated session key. The initiator and acceptor were required to check that the values in the *SRPGM-EVIDENCE* token were what they expected them to be.

Using digests

Initially we included all the negotiated information the was sent in both the *SRPGM-REQ* and *SRPGM-REP* tokens in the *SRPGM-EVIDENCE* token. This resulted in much data being transmitted twice, so we decided to include a digest of the data sent in the

SRPGM-REQ token and a digest of the data sent in the *SRPGM-REP* token instead. So, a digest of the encoded *SRPGM-REQ* token sent by the initiator would be included in the *SRPGM-EVIDENCE* token sent by the acceptor in order for the initiator to be sure that the *SRPGM-REQ* token was received unmodified. Similarly for the *SRPGM-REP* token sent by the acceptor and *SRPGM-EVIDENCE* token sent by the initiator.

In order to reduce the number of times the digest was computed, we decided to use the following process: The initiator computes the digest of the *SRPGM-REQ* token, stores this digest and sends it along with the *SRPGM-REQ* token. The acceptor can then include the digest directly into the *SRPGM-EVIDENCE* token without having to compute it. Upon receipt of the *SRPGM-EVIDENCE* token the initiator can verify the digest against its stored value. In this way, the digest value only gets computed once. Similarly for the *SRPGM-REP* token sent by the acceptor and *SRPGM-EVIDENCE* token sent by the initiator.

Using the context key earlier

The acceptor has sufficient knowledge to compute the shared context key once it has received the *SRPGM-REQ* token. We realised that this means that the acceptor can use the context key to integrity protect the *SRPGM-REP* token using a keyed checksum. The initiator can only derive the context key once it has information that is contained in the *SRPGM-REP* token, but once it has this information it can derive the context key and verify that the information in the *SRPGM-REP* token is correct by computing the keyed checksum itself and checking that it matches the checksum included with the *SRPGM-REP* token.

This means that the digest of the *SRPGM-REQ* token can now be included in the *SRPGM-REP* token, rather than the *SRPGM-EVIDENCE* token, since it is now also integrity protected. This allows the initiator to determine earlier that the *SRPGM-REQ* token has been tampered with, and no digests need to be included in the *SRPGM-EVIDENCE* token.

5.5.4.3 The values n and g

SRP requires that both the initiator and the acceptor use the same values of n and g. Wu [98] states that either the values can either be determined in some external means (such as by standardising on particular values for a security domain or agreeing on them in some out-of-band manner) or they can be transmitted from the acceptor to the initiator during the protocol exchange. With GSS-API the initiator initiates the context establishment process, so the acceptor sending the values for n and g before the initiator creates A requires an additional protocol round, since for security reasons the acceptor can only transmit B once it has received A[6](See 3.2.4 of [98]). For performance reasons we wanted to avoid this extra round if possible.

We decided that the creation of the *SRPGM-REQ* token would initially assume that the values for n and g had been predetermined and generate the ephemeral asymmetric keypair using these values. Upon receipt of the *SRPGM-REQ* token the acceptor would assume that the initiator had used the correct values. It would then generate its own asymmetric keypair using its own values for n and g, but include these values in the *SRPGM-REP* token sent to the initiator. Upon receipt of the *SRPGM-REP* token, the initiator would then check that the values for n and g used by the acceptor were the same values that it had used in its calculations. If the values were the same it would continue with context establishment by sending the *SRPGM-EVIDENCE* token, and if not it would restart context establishment, recreating the *SRPGM-REQ* token using these new values.

This achieved the goal of not requiring extra message exchanges if the values for n and g had been predetermined, but it had performance problems in the case where n and g had not been predetermined. The generation of the ephemeral asymmetric keypairs is a computationally-intensive time-consuming process. In the case where the values for n and g had not been predetermined, both the initiator and the acceptor would have to undertake this generation process twice. Our goal was now to avoid both extra message exchanges and unnecessary generation of ephemeral keypairs.

The solution that we came up with is to have the initiator include the values for n and g that it is using in the *SRPGM-REQ* token. Upon receipt of the *SRPGM-REQ* token, the acceptor checks that these values are the same as the values it is using. If they are it continues context-establishment by generating an *SRPGM-REP* token, and if they are not

[6]The reason for this is that the value u is usually a simple function of B and u must not be revealed before A is received (see section 3.2.4 of [98]).

it sends the correct values to the initiator using an *SRPGM-PARAMETERS* token.

If the initiator has no idea what values the acceptor is using, it can set the values of n, g and A in the *SRPGM-REQ* token to zero. These values for n and g are guaranteed to be incorrect and the acceptor will send the correct values using an *SRPGM-PARAMETERS* token. This way the initiator does not have to waste time generating an ephemeral keypair using possibly incorrect values, and extra message exchanges are avoided if possible. It is recommended that the initiator cache the values for n and g used by a particular acceptor to avoid the extra message exchanges in future.

5.5.4.4 The value u

Initially we included the value u as a separate data element in the *SRPGM-REP* token.. However, Wu notes in section 3.2.4 of [98] that the value of u (see 2.4) can be computed as a simple function of B. We decided to make u be the result of using B as input to the SHA-1 one-way-function, instead of transporting it as a separate value in the *SRPGM-REP* token.

5.5.4.5 Algorithms

We decided to change the mandatory integrity algorithm from HMAC-SHA1 to the HMAC-MD5 for performance reasons. HMAC-MD5 has superior performance to both HMAC-SHA1 and MAC algorithms based on secret key encryption algorithms [29]. Our initial concerns about using MD5 with HMAC, due to weaknesses found in MD5, turned out to be unfounded[25].

Due to the fact that some countries have restrictions on the use of cryptography we decided to follow the convention used in other GSS-API mechanisms of specifying a confidentiality algorithm as recommended rather than as mandatory. The recommended confidentiality algorithm was initially DES-CBC, because this is what SPKM recommends, but was changed to Blowfish-CBC, because there are concerns about the security of DES [80, pp 300-301]. We chose the Blowfish cipher because [82]:

- it is free from intellectual property constraints
- it is fast

- it supports variable key lengths from 32 bits to 448 bits
- it has withstood cryptanalysis since 1993
- a number of implementations in various programming languages are freely available

5.5.4.6 Attempted optimisation of SRP

As discussed in 2.4.4, we thought that it would be possible to reduce the number of message exchanges necessary to undertake mutual authentication with SRP by having the acceptor send its evidence proving its knowledge of the session key before the initiator does so. SRPGM could then be optimised as follows:

	SRP data		**GSS-API Token**		**SRP data**	
	C, A, n, g	–	SRPGM-REQ	$\longrightarrow$		
Initiator		$\longleftarrow$	SRPGM-REP	–	s, B, M_2	**Acceptor**
	M_1	–	SRPGM-EVIDENCE	$\longrightarrow$		

However, it has been established that this optimisation adversely affects the security of SRP by making an offline dictionary attack possible.

5.5.4.7 Published Versions

The initial specification for this mechanism was written by taking RFC 2025 (the SPKM specification), removing inappropriate sections and adding in sections motivating for the use of SRP, specifying what algorithms were to be used, and describing the ASN.1 token structures. It was published as an IETF Internet Draft *draft-ietf-cat-srpgm-00.txt*.

This document was updated two months later with another Internet Draft - *draft-ietf-cat-srpgm-01.txt*. It contained more details on functionality of SPKM that was reused in SRPGM. It also removed whole sections that had been copied verbatim from RFC 2025, providing references to the appropriate sections in RFC 2025 instead. The mandatory integrity algorithm was changed from HMAC-SHA1 to HMAC-MD5, and the recommended confidentiality algorithm was changed from DES-CBC to Blowfish-CBC.

We then began to work on many of the issues discussed in 5.5.4 above. This resulted in *draft-ietf-cat-srpgm-02.txt*, a refined version of the previous document, being published.

This document provides clarifications in a number of areas and more detail to the ASN.1 token definitions, as well as incorporating our new ideas regarding the integrity protection of context establishment tokens and establishing the values of n, g and u.

5.5.5 Implementation

As described in 4.4 above the Java GSS-API (JGSS) Service Provider Interface (SPI) is a provider-based broker layer that allows mechanism implementors to easily plug their mechanism implementations into the JGSS framework.

We have implemented SRPGM under the JGSS SPI and have made it available, along with the rest of the JGSS implementation, as part of the Cryptix project. Due to the fact that the JGSS API and SPI specifications have not been finalised and that SRPGM has undergone many changes, this implementation is still work-in-progress.

5.6 Conclusion

Available GSS-API mechanisms are based on both shared-key and public-key technology and their message structures are specified using the Abstract Syntax Notation One (ASN.1). ASN.1 separates the specification of the message syntax from message encoding, which allows the messages to be encoded using the encoding rules best suited to the environment were the messages are being used. It is widely used in many different areas, although there are problems related to its abstraction, complexity, accessibility and use.

An important criterion for the use of a particular GSS-API mechanism is the infrastructure required in order to deploy it. The standardised GSS-API mechanisms, SPKM and Kerberos, have relatively high infrastructure requirements and there has been recent interest in the development of so-called low-infrastructure GSS-API mechanisms that have lower infrastructure requirements. Such low-infrastructure mechanisms include the Low Infrastructure Public Key (LIPKEY) mechanism and the Secure Remote Password GSS-API Mechanism (SRPGM), both of which we have implemented.

SRPGM was designed by us based on the Secure Remote Password (SRP) protocol and the existing Simple Public Key Mechanism (SPKM). It has evolved over time into a specification that has been published as an IETF Internet Draft, and is an important contribution

due to the fact that it is password-based, has low infrastructure requirements and does not require the use of long-term asymmetric keys.

Chapter 6

SASL

"Security's worst enemy is complexity" - Bruce Schneier

6.1 Introduction

Some computing resources available on campus are in the form of services provided using standard Internet protocols. Examples include access to email via POP3[1] and IMAP[2], and sending email via SMTP[3]. As currently employed on our campus, these services offer little or no security, since at best they use plaintext passwords or weak challenge-response mechanisms. The Simple Authentication and Security Layer provides a means of adding stronger security to these protocols.

The Simple Authentication and Security Layer (SASL) is a Internet Engineering Task Force (IETF) standard specified in RFC 2222 [61]. It describes a means of using authentication with connection-based protocols. Each such protocol must have an SASL profile that specifies how the available mechanisms are listed, how a mechanism is selected, and how the exchange of mechanism data takes place. Profiles have already been written for existing connection-based Internet protocols, such as SMTP, POP3, IMAP and LDAP[4]. An authentication mechanism specifies the data that must be transmitted between the initiator and the verifier in order to authenticate the identity of one or both parties, and

[1] Post Office Protocol version 3
[2] Internet Message Access Protocol
[3] Simple Mail Transfer Protocol
[4] Lightweight Directory Access Protocol

in some cases to set up a security layer. This security layer may provide features such as integrity and confidentiality protection, and protection against other network attacks by providing features such as sequencing and replay detection.

The SASL specification in RFC 2222 [61] describes how SASL works in general terms. We have implemented the API described in the Java-SASL specification [94] which describes an API for providing and using SASL functionality using the Java programming language. In addition we have implemented some of the standard SASL mechanisms, as well as some work-in-progress mechanisms, including one developed by the author.

6.2 Relationship to the GSS-API

SASL and the GSS-API complement each other nicely. The GSS-API specifies that it is the application's responsibility to transport tokens between the client and the initiator. SASL profiles specify how authentication mechanism messages are transmitted between the communicating entities. When the authentication mechanism is the SASL GSS-API [62] mechanism then these messages are GSS-API tokens. Thus, the GSS-API SASL mechanism allows GSS-API functionality to be used by the connection-based protocols that support it by calling on an underlying GSS-API library and providing a means of transporting the returned GSS-API tokens.

However, currently standardised GSS-API mechanisms (Kerberos and SPKM) require significant infrastructure to be available in order to operate. Also, most GSS-API mechanisms make use of Abstract Syntax Notation One (ASN.1) with the Basic Encoding Rules (BER) or Distinguished Encoding Rules (DER), that require a compiler to encode and decode message structures. These high infrastructure requirements and the complexity of using ASN.1 based encodings are reasons why there is resistance to the use of the GSS-API in the environments where SASL mechanisms are typically employed. For this reason much of the current activity in the CAT working group has been on specifying low-infrastructure mechanisms, such as LIPKEY and SRPGM.

SASL mechanisms on the other hand are much simpler. They do not require a significant amount of infrastructure in order to operate [43], typically just a remembered password on the client side and a verifier database on the server side. No special encoding of messages is done either. One reason for this simplicity is that many of them are only used for authentication - entity authentication and sometimes message authentication - and don't

provide additional security services, such as sequencing, replay-detection and confidentiality protection. Consequently, while being more complex, GSS-API mechanisms offer more security functionality than existing SASL mechanisms. The SRP-SASL mechanism that we have developed aims to provide all the functionality that GSS-API mechanisms provide, while being simpler and easier to implement, because it requires little infrastructure and does not use a complex encoding scheme.

6.3 SASL Mechanisms

A number of SASL mechanisms have been proposed. Some of these have been standardised and are available as IETF RFCs. Others are specified in working documents as IETF Internet Drafts. We have implemented some of these mechanisms and they are available as part of the Cryptix SASL library, which we also developed[5].

Mechanism	Specification	Implemented
ANONYMOUS	RFC 2245	yes
CRAM-MD5	RFC 2195	yes
PLAIN	RFC 2595	yes
OTP	RFC 2444	no
KERBEROS_V4	RFC 2222	no
GSS-API	RFC 2222	partially
DIGEST-MD5	draft-leach-digest-sasl-04.txt	no
SRP-SASL	draft-burdis-cat-srp-sasl-02.txt	yes

(At the time of writing the SASL GSS-API mechanism implementation does not support a security layer, although this will be added in future).

As an introduction to why we developed the SRP-SASL mechanism, we describe some existing SASL mechanisms that provide authentication using reusable passwords.

6.3.1 CRAM-MD5

The Challenge Response Authentication Mechanism (CRAM-MD5 [43]) was specified for use with the POP3 and IMAP protocols but may be used with other protocols that have

[5]This library is still under development and is available from http://www.cryptix.org/products/sasl

SASL profiles. It is a challenge-response protocol that makes use of a keyed message authentication code (produced using the HMAC-MD5 algorithm [46]) to protect the evidence being transmitted. A message-id that changes with every authentication session provides protection against replay attacks.

Carol		Steve
<—-	m	—
—	$H_P(m)$	—->

where:

- m is a message-id eg. *<5HeScLXP2qs.945642602947@146.231.31.104>*
- P is the shared secret
- $H()$ is the HMAC-MD5 algorithm
- $H_P()$ is the HMAC-MD5 algorithm using the shared secret

This message-id is a nonce (see 2.3.3) and is constructed as per RFC 1939 [60]:

> "For example, on a UNIX implementation in which a separate UNIX process is used for each instance of a POP3 server, the syntax of the timestamp might be:
>
> <process-ID.clock@hostname>
>
> where 'process-ID' is the decimal value of the process's PID, clock is the decimal value of the system clock, and hostname is the fully-qualified domain-name corresponding to the host where the POP3 server is running." - Myers et al [60]

As noted by the authors, this mechanism is susceptible to certain attacks. Sufficient information is made available to enable an attacker to undertake an offline dictionary attack in order to guess the authentication evidence. (An attacker knows the algorithm, the message-id, and the keyed MAC). The mechanism only authenticates the client to the server at the start of the session and provides no message authentication during the session, so protocols using this mechanism are also susceptible to active attacks such as session hijacking and message tampering.

6.3.2 PLAIN

Using the PLAIN mechanism [67] the client sends a plaintext username and password to the server in order to authenticate itself.

Carol Steve
— U, P —->

where:

- U is the username
- P is the password

This mechanism is useful where existing clients and servers only support plaintext passwords. However, the Internet Architecture Board Security Workshop [6] made it unacceptable to use plaintext password mechanisms with Internet protocols over unencrypted channels:

> "One security mechanism was deemed to be unacceptable: plaintext passwords. That is, no protocol that relies on passwords sent over unencrypted channels is acceptable."

Therefore the PLAIN mechanism may only be used when an encrypted channel is in place. Newman [67] explicitly states that it should not even be advertised if an encrypted security layer is not in place. Such an encrypted channel is typically provided using the Transport Layer Security (TLS) protocol described in [24]. As with SASL protocols each connection-based protocol that uses TLS must have a profile that specifies how the encrypted channel is invoked. Such TLS profiles are specified for the POP, IMAP and ACAP protocols in [67] and the SMTP protocol in [36], for example.

6.3.3 DIGEST-MD5

This challenge-response mechanism introduces the HTTP Digest Authentication mechanism [33] as an SASL mechanism. It is intended to be more secure than the CRAM-MD5 mechanism and provides functionality that counters many the attacks possible against

CRAM-MD5. Like CRAM-MD5 it makes use of a nonce to counter replay attacks. However, it also provides protection against chosen-plaintext attacks and provides a security layer with optional integrity and/or confidentiality protection. As the author of the mechanism discusses in [51] it only provides weak authentication, since it still susceptible to some active attacks, such as an online dictionary attack and a man-in-the middle attack. He summarizes the mechanism as follows [51]:

> "By modern cryptographic standards Digest Authentication is weak, compared to (say) public key based mechanisms. But for a large range of purposes it is valuable as a replacement for plaintext passwords. Its strength may vary depending on the implementation."

Although this mechanism achieves its desired goal of being more secure than existing SASL mechanisms such as CRAM-MD5, we believe the fact that it provides weak authentication makes it less useful. A password-based mechanism that provides strong authentication and protection against all passive and active attacks is necessary.

6.4 SRP-SASL Mechanism

While the SASL mechanisms discussed above provide authentication, many of them are susceptible to active attacks during the authentication process, and provide no protection for messages exchanged after authentication. Out of all the currently specified SASL mechanisms, both standardised and work-in-progress, only KERBEROS_V4, GSS-API, Digest and SRP-SASL provide support for a security layer. Problems have been found with the Kerberos version 4 protocol [7], so it is not be advisable to use the KERBEROS_V4 SASL mechanism. As discussed in 6.2 employing GSS-API based mechanisms may be difficult due to infrastructure requirements and/or the difficulty in implementing these mechanisms due to the use of ASN.1 (see 5.2.4). As discussed in 6.3.3 Digest authentication is weak.

As discussed in 2.4, the Secure Remote Password (SRP) protocol provides strong password-based authentication and the negotiation of a session key between the communicating parties. This makes it an excellent basis for developing an SASL mechanism that provides more comprehensive security than existing mechanisms.

The SRP-SASL mechanism aims to provide strong authentication and a security layer that employs countermeasures for known attacks, while being relatively simple in design and easy to implement. It makes use of the SRP-SHA1 protocol described in [99] and optimized SRP as described in [98], since this reduces the total number of messages exchanged by grouping together pieces of information that do not depend on earlier messages. The mechanism describes how the SRP-SHA1 data is encoded for transmission between the client and server, and it adds extra control information to enable the client to request additional security services to be provided by a security layer. Messages are encoded using netstrings [8] because they are very easy to generate and parse, unlike more complex encoding schemes such as ASN.1 and its associated encoding rules.

6.4.1 Authentication

The mechanism data exchanges are shown below (refer to 2.4):

Client		**Server**
$\longleftarrow$	n, g, Z	–
–	C, A, o	$\longrightarrow$
$\longleftarrow$	s, B	–
–	M_1	$\longrightarrow$
$\longleftarrow$	M_2	–

where:

- $M_1 = H(H(n) \oplus H(g) \,|\, H(C) \,|\, s \,|\, Z \,|\, A \,|\, B \,|\, K)$
- $M_2 = H(C \,|\, A \,|\, o \,|\, M_1 \,|\, K)$

and:

- H is a one-way function (in this case SHA-1)
- $|$ is the concatenation operator
- $\oplus$ is the XOR operator
- Z is the options byte indicating which of the security services the server can provide

- o is the options byte indicating requested security services
- M_1 is the client's evidence that the shared key is known
- M_2 is the client's evidence that the shared key is known
- n, g, C, A, s, B are as on page 19 in 2.4.1.

6.4.2 Security Layer

SRP provides authentication and the negotiation of a shared session key between the client and server. Additional security services can be provided over the session using the negotiated session key. Support for security layer functionality is advertised by the server using an 8-bit bitstring (Z) sent by the server, and such functionality is requested by the client using an 8-bit bitstring (o) sent by the client during the authentication exchange. (Unassigned bits are reserved for future use). The client and server verify that the other party's settings were received unmodified by including them as part of the evidence (M_1 and M_2) that they use to prove to the other party that they have knowledge of the shared session key.

Bit number	Meaning if set
0	Integrity protection using HMAC-MD5
1	Replay detection using sequence numbers (Bit 0 must also be set)
2	Confidentiality protection using Blowfish in CBC mode

Integrity protection, provided using the HMAC-MD5 algorithm [46] and the negotiated session key to produce a message checksum, provides message authentication for messages exchanged during the session and prevents active attacks such as session hijacking and message tampering. Replay detection may be provided in conjunction with integrity protection by using sequence numbers to ensure that every message exchanged during a session will be different, which means that the checksum on messages with identical data will be different. The receiver of the message must check that the sequence number of a received message matches the sequence number that it expected to receive and discard the message if it does not. After each message is received the expected sequence number is incremented. Since session keys are different for each session, interleaving messages from currently active sessions and reusing messages from past sessions will also be detected, because the

integrity checksums will not match. Confidentiality protection, using the Blowfish cipher [82] in CBC mode and the negotiated session key, provides a means to ensure the privacy of exchanged messages.

The reason that security layer functionality is made optional is due to the fact that the client or server may not want the functionality enabled or may not be able to use it due to legal restrictions. For example, a server may not be allowed to use confidentiality protection due to legal restrictions on the use of cryptography, or a client may not wish to use replay detection due to the extra overhead required in maintaining sequence numbers.

6.4.3 Netstrings

This mechanism makes extensive use of netstrings[6], which are described in more detail in [8]:

> "A netstring is a self-delimiting encoding of a string. Netstrings are very easy to generate and to parse. Any string may be encoded as a netstring; there are no restrictions on length or on allowed bytes. Another virtue of a netstring is that it declares the string size up front. Thus an application can check in advance whether it has enough space to store the entire string."

Since the length of the data is declared up front it is possible for the receiver of the message to allocate buffer space in advance and know exactly how many bytes to read in order to obtain all the data. Extracting the different fields out of a sequence of bytes is also easy since netstrings clearly delimit where one field ends and another field starts. The fact that you can nest netstrings means that you can have an arbitrary number of subfields inside any field, so you can represent complex data structures if necessary.

> "Any string of 8-bit bytes may be encoded as {len}":"{string}",". Here {string} is the string and {len} is a nonempty sequence of ASCII digits giving the length of {string} in decimal. The ASCII digits are <30> for 0, <31> for 1, and so on up through <39> for 9. Extra zeros at the front of {len} are prohibited: {len} begins with <30> exactly when {string} is empty.

[6]Professor Daniel Bernstein at the University of Illinois at Chicago (UIC) originated the netstring idea.

> For example, the string "hello world!" is encoded as <31 32 3a 68 65 6c 6c 6f 20 77 6f 72 6c 64 21 2c>, i.e., "12:hello world!,". The empty string is encoded as "0:,".
>
> {len}":"{string}"," is called a netstring. {string} is called the interpretation of the netstring." - Bernstein [8]

One thing that this representation lacks in comparison to a more complex representation format such as ASN.1 is any specification of the names and types of the data fields that it contains. However, such specification introduces complexity which we aim to avoid. Since the protocol is simple, the data fields are well-defined, and there is no optional data, it is not necessary for type information to be transmitted in order for the protocol to operate. The use of netstrings makes protocol messages easy to construct on the sender side and easy to parse on the receiver side. This is simple enough that it may even be done by hand if necessary.

6.4.4 Evolution of SRP-SASL

This mechanism has evolved over time, adding support for more security services, adding countermeasures against attacks and optimising the message exchanges. Below we discuss the various versions of the mechanism, the problems and issues we encountered and how they were resolved.

6.4.4.1 Version 1

The initial version was published as an Internet Draft - *draft-burdis-cat-srp-sasl-00.txt*. A minor update, that contained fixes for typos and clarification on certain issues thanks to feedback from readers, was published as *draft-burdis-cat-srp-sasl-01.txt*. The abstract described it as follows:

> "This document describes an SASL mechanism based on the Secure Remote Password protocol. This mechanism allows a client to be authenticated to a server, and optionally the server authenticated to the client. Additionally a security layer providing integrity protection can be provided."

We based the mechanism on the SRP-SHA1 protocol [99], with some additions:

Client		**Server**
$\longleftarrow$	n, g	$-$
$-$	C, A	$\longrightarrow$
$\longleftarrow$	s, B	$-$
$-$	M_1, o, m	$\longrightarrow$
$\longleftarrow$	M_2	$-$

where:

- o is the options byte indicating requested security services
- m is the integrity checksum of the options byte

We added the options byte (o) to provide a means of selecting whether or not mutual authentication should take place, and whether or not a security layer providing integrity protection should be provided.

Bit number	Meaning if set
0	Mutual authentication is requested.
1	Integrity protection using HMAC-SHA1 is requested.

Integrity protection was provided using the HMAC-SHA1 algorithm. We did not use the more efficient HMAC-MD5 algorithm, due to that fact that weaknesses had recently been found in the MD5 algorithm. The options byte could be modified by an attacker before reaching the server. For this reason a keyed MAC (m), generated using the HMAC-SHA1 algorithm and the shared context key, was used so that the server could verify its integrity.

We decided not to support confidentiality protection with the following reasoning:

> "It was decided not to provide support for confidentiality protection. Such support usually requires the negotiation of a suitable cryptographic algorithm and it is felt that such negotiation would make the protocol unnecessarily complex. It is suggested that those who need a mechanism with this functionality use one of the GSS-API based mechanisms. Avoiding confidentiality protection also has benefits in that it may allow use of this mechanism in countries that have strict

> controls on the use of cryptography. It is felt that a simple SASL mechanism that provides authentication and integrity protection will be useful."

At the time of writing this draft we had just finished the initial version of our SRPGM specification (see 5.5), which provided may additional security services including integrity protection, confidentiality protection and replay detection. We felt that adding support for additional security services to SRP-SASL would go against our goal of making a mechanism that was simple to implement, and that those who wished to have confidentiality protection could use SRPGM under the SASL GSS-API mechanism.

6.4.4.2 Version 2

After more research into the possible attacks that can be undertaken in a distributed environment (see 2.2) we noticed that this mechanism did not provide any protection against replay attacks. This could be remedied by making use of integrity protected sequence numbers to make each message exchanged during a session unique, and since session keys are different for each session, interleaving messages from other sessions can be detected because the integrity checksums won't match. So we decided to add an option for replay detection to the security layer.

We also came up with a way of including support for confidentiality protection without having to undertake a complex negotiation procedure. The HMAC-SHA1 integrity algorithm was mandatory and all implementations had to have support for it. By making support for a particular set of confidentiality algorithms mandatory we could ensure that an algorithm selected by the client would be supported by the server. So we decided to add support for confidentiality protection using the Blowfish cipher.

We chose the Blowfish cipher because [82]:

- it is free from intellectual property constraints
- it is fast
- it supports variable key lengths from 32 bits to 448 bits
- it has withstood cryptanalysis since 1993
- a number of implementations in various programming languages are freely available

At this time we also decided to switch the mandatory integrity algorithm to HMAC-MD5, since it offers better performance than both HMAC-SHA1 and MAC algorithms based on secret key encryption algorithms [29]. Our concerns about using MD5 with HMAC turned out to be unfounded [25].

Bit number	Meaning if set
0	Mutual authentication is requested.
1	Integrity protection using HMAC-MD5 is requested.
2	Replay detection using sequence numbers (Bit 1 must also be set)
3	Confidentiality protection using Blowfish in CBC mode

This new specification was written up as *draft-burdis-cat-srp-sasl-02.txt* and was due to be submitted. However, we discovered a possible optimisation to the authentication exchange described in 2.4.4 which caused us to delay publication in order to verify whether or not the optimisation compromised the security of SRP in any way. In the meantime we made further changes.

6.4.4.3 Version 3

One concern we had was with using an integrity checksum (MAC) on a byte of data, as used to protect the client's options byte (o) during the authentication exchange, since the input is so small. It transpires that it was not necessary to make use of a MAC to protect the client's options byte (o). Section 3.1 of [99] describes how the client and server generate the evidence necessary to prove to the other party that they have knowledge of the shared session key. In order to verify that elements of data being transmitted (eg. n and g) were received unmodified, these data elements are included as part of the evidence generation process. By including the options byte (o) as part of the evidence generated by the server we can ensure that the server received the options byte unmodified, since the evidence will not verify correctly if it did not. This is a much simpler way of performing the verification, which meets one of our stated goals. The message exchanges were then as follows:

Client		**Server**
$\longleftarrow$	n, g	–
–	C, A, o	$\longrightarrow$
$\longleftarrow$	s, B	–
–	M_1	$\longrightarrow$
$\longleftarrow$	M_2	–

As noted in 6.4.4.1 we were concerned that adding support for confidentiality protection would preclude use of SRP-SASL in countries that have restrictive policies on the use of cryptography. However, we had not provided a means for a server to specify whether or not it was able to provide confidentiality protection. The use of sequence numbers for replay detection requires that each party maintain a running count of the number of messages that have been received. This may be undesirable in certain situations, so we decided it was also necessary to provide a means for a server to indicate that it was not willing to provide replay detection. For these reasons, we decided that the server should advertise the security services that it was willing to provide using an options byte (Z), the same as that used by the client to select options, and that the client would then select services from those advertised. However, we decided that it would be mandatory for a server to provide integrity protection, since this counters a wide variety of attacks, does not require maintenance of state, and does not use cryptography. The message exchanges were then as follows:

Client		**Server**
$\longleftarrow$	n, g, Z	–
–	C, A, o	$\longrightarrow$
$\longleftarrow$	s, B	–
–	M_1	$\longrightarrow$
$\longleftarrow$	M_2	–

The unmodified transmission of the server's options byte (Z) needs to be verified in the same way that the client's options byte (o) is verified, so the client includes it as part of the evidence that is uses to prove to the server that it knows the shared session key, so that that server can verify that it was received correctly. This means that mutual authentication is now a necessity in order for proper verification to take place, so it is now mandatory and is no longer a security service option. The options byte is now specified as follows:

Bit number	Meaning if set
0	Integrity protection using HMAC-MD5 is requested.
1	Replay detection using sequence numbers (Bit 0 must also be set)
2	Confidentiality protection using Blowfish in CBC mode

6.4.4.4 Attempted optimisation of SRP

As discussed in 2.4.4, we thought that it would be possible to reduce the number of message exchanges necessary to undertake mutual authentication with SRP by having the acceptor send its evidence proving its knowledge of the session key before the initiator does so. SRP-SASL could then be optimised as follows:

Client		Server
$\longleftarrow$	n, g, Z	–
–	C, A, o	$\longrightarrow$
$\longleftarrow$	s, B, M_2	–
–	M_1	$\longrightarrow$

where:

- $M_1 = H(H(n) \oplus H(g) \,|\, H(C) \,|\, s \,|\, Z \,|\, B \,|\, M_2 \,|\, K)$
- $M_2 = H(C \,|\, A \,|\, o \,|\, B \,|\, K)$

However, it has been established (see 2.4.3) that this optimisation adversely affects the security of SRP by making an offline dictionary attack possible.

6.5 SASL Profiles

Each protocol that wishes to use SASL for authentication and security services must have an SASL profile that specifies how the available mechanisms are listed, how a mechanism is selected, and how the exchange of mechanism data takes place. Profiles have already been written for existing connection-based Internet protocols, such as SMTP, POP3, IMAP, ACAP and LDAP. Below we describe the SMTP and POP profiles, motivating for their use and giving examples from our implementation. Profiles for other protocols are similar to the examples given.

6.5.1 SMTP Profile

The SMTP SASL profile [63] specifies how SASL functionality is made available with the SMTP protocol. Two motivations for the use of this profile are given below. Thereafter follows an example of the profile in action.

6.5.1.1 UCE

Unsolicited Commercial Email (UCE), more commonly known as spam, is a big problem for mail system administrators. Not only does UCE make undesired use of computing resources, such as processor time and queue space, it also irritates some users. For these reasons, senders of UCE have been blocked by many mail administrators. When the source of the email is known such blocking can be done effectively, even on a per-user basis if necessary. However, in order to get around these restrictions and to avoid being identified as the origin of these unsolicited messages, some spammers have taken to using third-party relays to send their messages.

In the early days of the Internet, when it was primarily used by military and educational institutions, mail hosts would accept messages from anyone and happily resend messages not destined for them on to the destination host. The message volumes were not large and this was the friendly thing to do. However, in the current unfriendly Internet such open relays are targets for spammers who wish to obscure the origin of UCE messages. Current recommended practice is to only accept messages destined for the mail host, and to only relay messages originated by users known to the mail host. Initiatives such as the Real-time Blocking List (RBL) [57] aim to enforce this practice by maintaining an active list of known open relays. Many mail system administrators block all mail from mail hosts in the RBL list in order to cut down on the amount of UCE received. The maintainers of the RBL actively advise and encourage the administrators of those hosts on the list to restrict their relaying and thereby not make their mail systems susceptible to abuse by spammers. The fact that many mail systems will refuse to accept mail from these hosts is an added incentive.

Relaying is a necessary practice, since there is rarely an SMTP server on every user's machine directly connected to the Internet. Usually there is a central mail server (or pool of mail servers) for a department or organisation that is used to send outgoing mail. It is therefore necessary to provide a means of identifying those who should be allowed to relay

messages. One such method is to use the source IP address of the user. This method is commonly used in practice but is susceptible to DNS spoofing [15]. Until Secure DNS [28] is in wide use, reliance on DNS is not secure. Another better method is to authenticate the user to the mail server.

6.5.1.2 Email as a critical resource

Email is a critical part of many organisations today, where it is fast becoming the primary means of disseminating information. It is therefore important that user's have some means of ensuring that the system that they are submitting their message to is really their mail system, and not some attacker's machine. This assurance can be provided by having the mail server authenticate to the user, in addition to the user authenticating to the mail server - a process known as mutual authentication. Submission of messages may also need to be protected from attack. For example, an attacker may hijack a session after the user has authenticated and then use the mail server to send messages (possibly UCE) as the user, or an attacker may replay previously entered commands used to send a message, resulting in multiple copies of a message being sent to the recipient. (Such a denial-of-service attack is known as a mail-bomb.)

6.5.1.3 Profile

In order to address these problems, SASL can be used for authentication and the provision of a security layer that has countermeasures against various attacks. The SASL SMTP profile [63] specifies how SASL mechanisms are listed and chosen, how mechanism messages are exchanged, and how error messages and warnings are indicated. It introduces a new command keyword - AUTH. When the client supplies this command without any parameters, the server lists the available mechanisms. In order to specify a particular mechanism the client issues the AUTH command with the mechanism name as an argument. Exchanges of mechanism messages, which are base64 encoded, continue until the server indicates that either authentication was successful or failed.

6.5.1.4 Example

Below is an example exchange using the SASL SMTP profile and our SRP-SASL mechanism. Lines beginning with "C:" indicate messages sent by the client, and lines beginning

with "S:" indicate messages sent by the server. These indicators and the fixed line wrapping are for illustrative purposes only and are not part of the protocol exchange. (Note that with the SMTP profile, messages are Base64 encoded).

- Server sends list of supported SASL mechanisms

```
S: 250 AUTH SRP-SASL CRAM-MD5 ANONYMOUS PLAIN
```

- Client selects a mechanism

```
C: AUTH SRP-SASL
```

- Server sends n, g and Z

```
S: CZOrEZ8rDZgiQzj1CagQc/5ctbuJYLWlhtAsPHc7xWVyCPAKFRLWKADpASkqe9d
   jWPFWTNTdeJtL8nAhImCn3Sr/IAdQ1FrGw0WvQUstPx3FO9KNcXOwisOQ1VlL.g
   heAHYfbYyBaxXL.NcJx9TUwgWDTOhRzFzqSrdGGTN3FgSTA1v4QnHtEygNj3eZ.
   uOMThqWUaDiP87nqha7XnT66bkTCkQ8.7T8L4KZjIImrNrUftedTTBi.WCi.zlr
   BxDuOMOda0JbUkQlXqvpOyvJAPpC11nxmmZOAbQOywZGmu9nhZNuwTlxjfIroOF
   OdthaDTuZRL9VL7MRPUDo/DQEyW.d4H.UIlzpB34w0Ymi
```

- Client sends C, A, o

```
C: 38tCperEcjbQNHeB38rDZfx1sE4pEQoyeD2nbzmtUMWwSSTCp6TGtb7vcap8qV/
   bbVcn.IeqomkF22YSV4Ugjx.xUScNgrxKAmh78Mp84kWw1Hyuy1BZSAtb12Hfq7
   OhLHlfMKTG2XoHkJdUpVdPBmcVxPGfjgHj1ZbP4BK/DYVMWJry8Jav12Hi0d2XP
   CByMhnOVPSLxIxB16Py.rDmLmuxvOVf6KgxNa3aviAn7f5XjxLhKAQfTdGP3Cwy
   PHadLYOpnl6ItlpujK48ODaO6uonK/v6nzm4xPRf3TKtJ2K3ew2ILj9l7erh9bs
   ljQf7YhkOwwTAtp2HRk4xSy/X/NGm7F2qxJjXwwn38cDBH8EB34w0omi
```

- Server sends s, B

```
S: oDpKwCJOwypqEXUPzRC1jHomoDJOw5xCIXiCv.qIiwAYp4JfHS8tzT/ldIJebtD
   V.OuOKl6lzvfRchVHVVfFMb3IQ8Rx7KTj/X/4KNMAOULuTCdvfHr69nhyUyqeYz
   HDhAgG5jgGRprEXgFw/wYfWpkpN2.vBPA28l7WhFRgpYUI157g7sA1/3fOCF1cc
   oX9OwuQadWWjlKOW9/gCTn6QzQ5eXvGOxPw718iOeLsCrWmj.RSIuAA1GGsSSqY
   .nkO8R1Vufybf91ro9bO3iLA5R9ujU.xGa46ePak2ebmiQk//wjBic2VkOOvoph
   AC3uQuAfGCZspHZpN2MvX8SKuwx9rtYxOLGkbXZ6ybvK5NJI8hq5paa2mi
```

- Client sends M_1

```
C: CZOwxM16xDTUotqOlHkBGmAy6ZA/1Qyi
```

- Server sends M_2

```
S: CZOweNVacSqwztVeRR3SJ79.Yk/TSO8i
```

- Client sends null string so that the server can send its success or failure notice

```
C:
```

- Server sends success notice

```
S: 235 SRP-SASL authentication successful
```

Most existing SMTP servers do not have support for authentication using SASL. For this reason we developed an SMTP proxy that understands the SASL SMTP profile commands and undertakes authentication on behalf of the SMTP server. Once authentication is complete all other commands are passed on to the SMTP server. Since the proxy listens on the standard SMTP port (port 25), the real SMTP server should be configured to listen on a different port. The server should also only accept local connections to avoid remote users bypassing the proxy and accessing the SMTP server directly. Note that clients are not required to have SASL support, but those that do have support can now be accommodated. Since the proxy listens directly for client connections it must be configured with information, either based on DNS or preferably an authenticated user, to determine which clients are allowed to relay. Unfortunately this proxy is not very useful at present since few, if any, mail user agents support SMTP authentication. We expect this to change in future.

6.5.2 POP3 Profile

The Post Office Protocol (POP) enables a user to fetch their mail from a central mail store for local reading. Mailbox manipulation features are also provided. For example, after reading a message the user may choose to leave it on the server or delete the message. As

discussed in section 6.5.1.2 email is becoming a critical resource, so it is important that retrieval and management of email messages be secure. Version 3 of the POP protocol (known as POP3) is specified in IETF RFC 1939. This RFC also specifies two types of authentication. The first (USER/PASS) is a simple plaintext username/password mechanism which is unacceptable for use over unencrypted channels. The second (APOP) sends a digest of the password instead of the password itself, but is susceptible to a wide range of passive and active attacks, and therefore provides very little security. The POP3 SASL profile provides a much better means of protecting the protocol. It introduces a new command keyword - AUTH. When the client supplies this command without any parameters, the server lists the available mechanisms. In order to specify a particular mechanism the client issues the AUTH command with the mechanism name as an argument. Exchanges of mechanism messages, which are base64 encoded, continue until the server indicates that either authentication was successful or failed.

6.5.2.1 Example

Below is an example exchange using the SASL POP profile and the CRAM-MD5 SASL mechanism. Lines beginning with "C:" indicate messages sent by the client, and lines beginning with "S:" indicate messages sent by the server. These indicators are for illustrative purposes only and are not part of the protocol exchange. (Note that with the POP profile, messages are Base64 encoded).

```
S: <5HeScLXP2qq.945642019150@146.231.31.104>
C: AUTH CRAM-MD5
S: + 3mrI6LJOqnOK39nSIuvD3KsD38mCJaoDpPOCJGsBZ8pCIupCIunC3G.
C: 1hPMbqQ22b0VnvC9GBbC1biWYili3t
S: +OK CRAM-MD5 authentication successful
C: QUIT
S: +OK
```

Refer to 6.3.1 for details of the CRAM-MD5 message exchanges.

6.6 Java SASL Library

The Java SASL API [94] provides a concrete implementation of the concepts and protocols described and specified in RFC 2222 [61]. This API provides functionality via well-defined

abstract interfaces that hide the complexity of the underlying mechanisms. A means is provided for callers to query the available mechanisms, but all mechanisms are accessed through the same common API. Mechanism implementations are also written to a standard API, which enables changes to these implementations to remain localised. It is also easy to add and remove mechanism implementations. In addition to providing authentication services it also provides a security layer that can be used to provide security services, such as message authentication, replay detection and confidentiality protection, after authentication has taken place.

The abstract nature of the API means that a single Java SASL library can be used with multiple protocols. Our Java SASL implementation, which is available as part of the Cryptix project, was used with both the SMTP and POP3 SASL profile implementations described above.

6.7 Conclusion

SASL is used to provide support for authentication and security services to connections-based protocols, such as the SMTP and POP3 protocols that are employed on our campus. SASL profiles for these protocols specify how the available mechanisms are listed, how a mechanism is selected, and how the exchange of mechanism data takes place. SASL mechanisms provide authentication and security service functionality, but existing SASL mechanisms provide weak authentication and do not counter all known attacks. While SASL can be used in conjunction with GSS-API mechanisms by being a transport for GSS-API tokens, there are concerns due infrastructure requirements and message encoding complexity of these mechanisms. The SRP-SASL mechanism that we have developed aims to provide strong authentication and a security layer that employs countermeasures for known attacks, while being relatively simple in design and easy to implement. The Cryptix SASL Project provides a concrete implementation of the Java SASL API and some SASL mechanisms, which enables implementors of connection-based protocol clients and servers to make use of SASL functionality in their applications.

Chapter 7

Conclusions and Future Research

7.1 Experience and Contributions

During the course of our research we have gained much experience with distributed authentication and its use as part of a resource control architecture, and have made some useful contributions to this field:

- Our research into the authentication process, the attacks that are possible in a distributed environment and how they can be countered, gave us sufficient background knowledge to understand how to protect the mechanisms that we designed from known attacks.
- Much of our work is based on the Secure Remote Password protocol, which we consider to be a significant advance in password-based security technology. We made an unsuccessful attempt to further optimise this protocol, and we give the details of this attempt so that others can learn from and not repeat our experience.
- We investigated the problems with the current user sign-on scenario, with particular reference to our campus environment, and discovered that it is common for users to have too much credential information to manage and to be prompted for it too often.. Research indicated that these problems could be solved using single sign-on, and we determined, through a partial implementation, that the Java Authentication and Authorisation Service (JAAS) could be used for this purpose. We provide pragmatic physical solutions to the security concerns associated with single sign-on.

- The GSS-API insulates callers from changes in the underlying security technology by encapsulating security functionality under an abstract API. By making applications independent of the inevitable changes in security implementations, the GSS-API reduces maintenance requirements by localising changes to the GSS-API implementation and GSS-API mechanisms.

- In our work with GSS-API mechanisms we gained experience with the use of the Abstract Syntax Notation One (ASN.1). We found that ASN.1 is very flexible and powerful, but that the complexity associated with the encoding rules used to convert the ASN.1 specification into bitstrings means that in practice the use of automated tools is recommended.

- Existing standardised GSS-API mechanisms require significant additional infrastructure to operate. Much current work in the IETF CAT working group is centered on developing GSS-API mechanisms with lower infrastructure requirements. We developed SRPGM because we noted that there was no available GSS-API mechanism that was password-based, had low infrastructure requirements and did not require the use of long-term asymmetric keys.

- SASL can be used to provide authentication and security functionality to existing Internet protocols, many of which currently have weak or non-existent security. These protocols are being used to provide standard services to many users, so improving the security of these protocols will have the effect of significantly upgrading the standard of security available at many sites.

- SRP-SASL is a simple, easy-to-implement SASL mechanism that provides strong password-based authentication and a security layer with integrity protection, replay detection and confidentiality protection. We developed SRP-SASL because existing SASL mechanisms provide weak security and are susceptible to various active and passive attacks. SRP-SASL avoids the complexity of encoding protocol messages using ASN.1 by using netstrings which are simple to create and parse.

- Our work on SRPGM and SRP-SASL culminated in the publication of specifications for these two mechanisms as IETF Internet Drafts. SRPGM was published under the auspices of the IETF Common Authentication Technology (CAT) working group, and SRP-SASL was an individual submission.

- One of our stated goals was to make security technology available to programmers. We have achieved this goal by implementing both the Java GSS-API and Java SASL bindings, in addition to the LIPKEY and SRPGM GSS-API mechanisms and various SASL mechanisms, including SRP-SASL. All of this work is freely available as part of the international Cryptix project.

7.2 Related and future work

7.2.1 CORBA Security Service

7.2.1.1 Introduction

The Common Object Request Broker Architecture (CORBA) is the communications component of the Object Management Architecture, a distributed middleware architecture produced by the Object Management Group [69]. It makes use of object request brokers (ORBs) that allow for communication between remote objects, providing the means to perform remote method calls on objects. It provides several benefits for distributed programming, such as location transparency, programming language independence, a clear separation of object interfaces from object implementations and a number of standardised services[71].

The CORBA Security Service [68] provides security services in the CORBA middleware environment. It works by intercepting all method requests to be invoked on CORBA objects and determining whether or not they are allowed according to the security policy.

> "Conceptually, the object invocation access control service is implemented by having it intercept every object invocation (ie. Request), possibly on both the Client and Target sides. Having intercepted the object invocation, the Credentials object is consulted to obtain the privileges with which the Client is operating. This set of privileges are then used as a parameter to the access_allowed operation on the AccessDecision object, which either grants or denies permission to continue the object invocation." - Chizmadia [17]

7.2.1.2 Sign-on

The Credentials object contains the client's credentials, such as a username/password or public/private keypair, that enable it to authenticate itself and thereby succeed in invoking a method on a particular object if allowed to do so by the security policy.

> "A principal must establish its credentials before it can invoke an object securely. For many clients, there are default credentials, created when the user logs on. This may be performed prior to using any object system client. These default credentials are automatically used on object invocation without the client having to take specific action." - Chizmadia [17]

These default credentials could be obtained in the traditional way by prompting the client for information, but a more user-friendly way would be to use a Single Sign-on setup as described in chapter 3. A Java implementation of the CORBA Security Service could use the Java Authentication and Authorisation Service (JAAS) to provide this functionality.

7.2.1.3 Security Services

> "clients and objects in a CORBA system deal exclusively with operation requests and responses back to those requests; however "under the covers" the ORB maintains a more persistent connection between the client and the object. The CORBA Security Service uses this persistent connection as the basis for secure associations, which provide: identification and authentication of the client to the object and the object to the client, protected transfer of credentials between the client and the object, and a way to negotiate the minimum amount of transport message ... protection that is acceptable to both the client and the object" - Chizmadia [17]

The Generic Security Services API (GSS-API) provides authentication, a secure context and security services over this context via a generic API. It is possible for a CORBA Security Service implementation to have the required security functionality provided by an underlying GSS-API implementation. Such reuse of existing technology implementations is recommended by the CORBA Security Service specification:

> "The use of standard, generic APIs for interactions with external security services not only allows interchangeability of security mechanisms, but also enables exploitation of existing, proven implementations of such mechanisms." - OMG [68]

The GSS-API was one of the generic APIs that the writers of this specification had in mind:

> "several interfaces in Section 15.4, Security Architecture, have been designed to allow easy mapping to GSS-API functions, and the Credentials and Security Context objects are consistent with the GSS-API credentials and contexts." - OMG [68]

As discussed in 4.1 and 4.2 the GSS-API insulates users of the API from changes in the underlying security technology, confining the affect of any changes to the GSS-API implementation and mechanisms.

7.2.1.4 Summary

A CORBA Security Service implementation built on top of the GSS-API would make it substantially independent of the security technology used to provide the security functionality, and the architecture of this service lends itself to such an implementation. In addition, making use a Single Sign-On architecture, such as that provided by JAAS, to provide client credentials would make such an implementation easier and more transparent for clients to use.

A possible application of our work would be to implement the CORBA Security Service in Java using the JAAS implementation from SUN and GSS-API implementation that we have produced.

7.2.2 Possible campus scenario

Below we describe a possible new campus scenario that could be implemented using the technology that we have investigated, developed and partially implemented.

7.2.2.1 User sign-on

The user's primary sign-on for a session requires the entering of a memorised username/password combination for each security domain of which it is a member. For example, there may be a student security domain and an administration security domain. This may require an undergraduate student to enter a username/password for the student security domain that allows her to access resources available to students, such a public laboratory machines and networked printers, and it may require that a staff member enter an additional username/password combination for the administration security domain that gives him access to student record information. This credential information is stored and made available for secondary sign-on procedures.

Now the user is able to use resources that require sign-on without having to re-enter credential information. Such secondary sign-on procedures take place using the stored credential information. Physical security measures are employed, such as password-protected screen savers, to prevent unauthorised users from undertaking secondary sign-on operations after primary authentication has taken place.

Due to the security risks involved in undertaking sign-on using stored credential information, where the authenticated user may not be present, it is recommended that access to sensitive resources, such as financial records for example, be protected using a separate username/password combination that is not cached by the single sign-on architecture. However, most resources available on campus are not sensitive, so the risks involved in using single sign-on with these resources is acceptable.

7.2.2.2 Resource control

Where possible, each server that makes resources available to users has an SRP authentication database that it uses to verify the identity of users attempting to make use of a resource. This database of verifiers may be stored in a distributed directory - such one accessible using the Lightweight Directory Access Protocol (LDAP) - that is replicated so that the same authentication database can be used by a number of application servers.

Connection-based Internet services are secured using the Simple Authentication and Security Layer (SASL). Security functionality is provided either by the SRP-SASL SASL mechanism or by the GSS-API SASL mechanism with SRPGM as the underlying GSS-API mechanism. This means that protocols such as FTP (remote file access), and IMAP

(remote access to electronic mail), now use strong password-based authentication instead of plaintext passwords, and provide a connection that is resistant to many forms of attack.

Connections to remote UNIX hosts are provided using the TELNET protocol and secured using the SRP TELNET security extension [100], using applications such as TeraTERM and the Java Telnet Applet [97] that provide support for this. Other computer resource implementations, such as virtual reality and music applications for example, are secured using the GSS-API with SRPGM as the underlying GSS-API mechanism.

7.2.2.3 Summary

The use of SRP, SASL and GSS-API can be used to improve the level of security on our campus.

7.3 Final Words

Distributed authentication is an important part of the process of controlling access to and use of computing resources. We have investigated and explored two of the primary authentication APIs available and have made contributions towards their use. In addition to researching and developing technology to thwart attackers, we have focused on making security more user-friendly by taking the user into consideration when developing a security infrastructure. We hope that our contributions are tools that others can use to provide better security for people.

References

[1] Adams, Carlisle, "IDUP and SPKM: Developing Public-Key-Based APIs and Mechanisms for Communication Security Services", IEEE, 1996

[2] Adams, Carlisle; "The CAST-128 Encryption Algorithm", IETF RFC 2144, May 1997

[3] Adams, Carlisle; "The CAST-256 Encryption Algorithm", IETF RFC 2612, June 1999

[4] Adams, Carlisle; "The Simple Public-Key GSSAPI Mechanism (SPKM)", IETF RFC 2025, October 1996

[5] AES, "Advanced Encryption Standard (AES) Development Effort", URL: *http://www.nist.gov/aes, 2000*

[6] Bellovin, A; "Report of the IAB Security Architecture Workshop", IETF RFC 2316, April 1998

[7] Bellovin, S.M; Merrit, M; "Limitations of the Kerberos Authentication System", Proceedings of the 1991 USENIX Conference, 1991

[8] Bernstein, Daniel J; "Netstrings", URL: *ftp://koobera.math.uic.edu/www/proto/netstrings.txt*, February 1997

[9] Boeyen S.; Howes T.; Richard P.; "Internet X.509 Public Key Infrastructure Operational Protocols - LDAPv2", IETF RFC 2559, April 1999

[10] Brookes, Frederick P; "The Mythical Man-month", Anniversary edition, ISBN: 0-201-83595-9, July 1995

[11] Brose, Gerald; "JacORB - a free Java ORB", URL: *http://www.inf.fu-berlin.de/~brose/jacorb/jacorb.html*, January 2000

[12] Bryan, Martin; "Introduction to XML", *http://www.personal.u-net.com/~sgml/xmlintro.htm*, 1997

[13] Burdis, Keith; "Secure Remote Password SASL Mechanism", IETF Internet Draft draft-burdis-cat-srp-sasl-02.txt, January 2000

[14] Burdis, Keith; "The Secure Remote Password GSS-API Mechanism", IETF Internet Draft draft-ietf-cat-srpgm-02.txt, January 2000

[15] CERT, "IP Spoofing Attacks and Hijacked Terminal Connections", 23 January 1995, CERT Advisory CA-95:01, URL: *ftp://ftp,cert.org/pub/cert_ advisories/CA-95:01.IP.spoofing*

[16] Cheng, P; Glenn R; "Test Cases for HMAC-MD5 and HMAC-SHA-1", IETF RFC 2202, September 1997

[17] Chizmadia, David; "A Quick Tour Of the CORBA Security Service", Object Management Group Information Security Bulletin, September 1998

[18] Connolly, Dan; "W3C Extensible Markup Language (XML) Activity", *http://www.w3.org/XML/Activity.html*, January 2000

[19] Cryptix, "The Cryptix Project", URL: *http://www.cryptix.org/*, January 2000

[20] Cryptix, "Cryptix - Cryptix ASN.1 Developers' Kit", URL: *http://za.cryptix.org/products/asn1/index.html*, November 1999

[21] Cryptix, "Cryptix - Cryptix Java GSS-API Library", URL: *http://za.cryptix.org/products/jgss/index.html*, January 2000

[22] Cryptix, "Cryptix - Cryptix SASL Library", URL: *http://za.cryptix.org/products/sasl/index.html*, January 2000

[23] Dallas Semiconductor; "Automatic Information Overview", URL: *http://www.dalsemi.com/Prod_ info/AutoID/index.html*, January 2000

[24] Dierks, T; Allen, C; "The TLS Protocol Version 1.0", IETF RFC 2246, January 1999

[25] Dobbertin, H; "The Status of Md5 After a Recent Attack", RSA Laboratories' CryptoBytes, Volume 2, Number 2, URL: *ftp://ftp.rsasecurity.com/pub/cryptobytes/crypto2n2.pdf*, Summer 1996

[26] DSTC; "GSSAPI Java Binding for Kerberos", URL: *http://security.dstc.qut.edu.au/projects/java/gssapi/*, November 1999

[27] DSTC; "JCSI - Java crypto and security implementation", URL: *http://security.dstc.edu.au/projects/java/jcsi.html*, 1999

[28] Eastlake, Donald E; "Domain Name System Security Extensions", IETF RFC 2535, March 1999

[29] Eisler, Mike; "LIPKEY - A Low Infrastructure Public Key Mechanism Using SPKM", IETF Internet Draft draft-ietf-cat-lipkey-02.txt, December 1999

[30] Ellison, Carl; Schneier, Bruce; "PKI Risks", Computer Security Journal, v16, n1, 2000, pp. 1-7, URL: *http://www.counterpane.com/pki-risks.html*

[31] FIPS, "Secure Hash Standard" (SHA-1), National Institute of Standards and Technology (NIST), URL: *http://www.itl.nist.gov/fipspubs/fip180-1.htm*, 1995

[32] Forge; "arc - An ASN.1 Compiler", URL: *http://www.forge.com.au/products/arc/index.html*, September 1999

[33] Franks, J; Hallam-Baker, P; Hostetler, J; Leach, P; Luotonen, A; Sink, E; Stewart, L; "An extension to HTTP: Digest Access Authentication", IETF RFC 2069, January 1997

[34] Gong, Li; "Inside Java 2 Platform Security - Architecture, API Design, and Implementation", Addison-Wesley, ISBN: 0-201-31000-7, June 1999

[35] Guttman, P; Leong, L; Malkin G; "Users' Security Handbook", IETF RFC 2504, February 1999

[36] Hoffman, P; "SMTP Service Extension for Secure SMTP over TLS", IETF RFC 2487, January 1999

[37] Housley R; Hoffman P; "Internet X.509 Public Key Infrastructure Operational Protocols: FTP and HTTP", IETF RFC 2585, May 1999

[38] IBM, "Secure Single-Entry Access to Your Computing Resources", IBM eNetwork Software White Paper, URL: http://www.software.ibm.com/network/globalsignon, 1998

[39] IETF, Public Key Infrastructure working group, URL: *http://www.ietf.org/html.charters/pkix-charter.html*

[40] IETF, Common Authentication Technology working group, URL: *http://www.ietf.org/html.charters/cat-charter.html*, September 1999

[41] Kabat, Jack; Upadhyay, Mayank; "Generic Security Service API Version 2 : Java bindings", IETF Internet Draft draft-ietf-cat-gssv2-javabind-03.txt, October 1999

[42] Kaliski, Burton J jr; "A Layman's Guide to a Subset of ASN.1, BER, and DER", RSA Data Security, Public-Key Cryptography Standards (PKCS), URL: *http://www.rsa.com/rsalabs/pubs/PKCS/*, 1 November 1993

[43] Klensin, J; Catoe, R; Krumviede, P; "IMAP/POP AUTHorize Extension for Simple Challenge Response", IETF RFC 2195, September 1997

[44] Knudsen, Jonathan; "Java Cryptography", O'Reilly & Associates, ISBN: 1-56592-402-9, May 1998

[45] Kohl, J; Newman, BC; "The Kerberos Network Authentication Service *(V5)*", IETF RFC 1510, September 1993

[46] Krawczyk, H; Bellare, M; Canetti, R; "HMAC: Keyed-Hashing for Message Authentication", IETF RFC 2104, February 1997

[47] Kwatsha, Siviwe; "SysHackAdminer: Overcoming host and network security problems though a paradigm shift", Rhodes University, Bsc (Honours) thesis, 1998

[48] Lai, Charlie; Samar, Vipin; "Making Login Services Independent of Authentication Technologies", 3rd ACM Conference on Computer and Communications Security, URL: *http://java.sun.com/security/jaas/pam/pam.html*, March 1996

[49] Lai, Charlie; Gong Li; Koved, Larry; Nadalin, Anthony; Schemers, Roland; "User Authentication and Authorization in the Java Platform", Proceedings of the 15th Annual Computer Security Applications Conference, December 1999

[50] Larmouth, John; "ASN.1 Complete", Morgan Kaufmann Publishers, ISBN: 0-12233-435-3, URL: *http://www.nokalva.com/asn1/larmouth.html*, October 1999

[51] Leech, Paul; "Using Digest Authentication as a SASL Mechanism", IETF Internet Draft draft-leech-digest-sasl-05.txt, 21 October 1999

[52] Levin, Roy; Redell, David; "How (and How Not) to Write a Good Systems Paper", An Evaluation of the Ninth SOSP Submissions, July 1983

[53] Linn, John; "Generic Security Service Application Program Interface", IETF RFC 1508, September 1993

[54] Linn, John; "Generic Security Service Application Program Interface, Version 2", IETF RFC 2078, January 1997

[55] Linn, John; "Generic Security Service Application Program Interface, Version 2, Update 1", IETF Internet Draft draft-ietf-cat-rfc2078bis-08.txt, 16 December 1998

[56] Linn, John; "The Kerberos Version 5 GSS-API Mechanism", IETF RFC 1964, June 1996

[57] MAPS; "MAPS Realtime Blackhole List", Mail Abuse Prevention System, URL: *http://mail-abuse.org/rbl/*, December 1999

[58] Menezes, A; van Oorschot P; Vanstone S; "Handbook of Applied Cryptography", CRC Press, ISBN: 0-8493-8523-7, October 1996

[59] Myers, John; "POP3 AUTHentication command", IETF RFC 1734, December 1994

[60] Myers, John; Rose, M; "Post Office Protocol - Version 3", IETF RFC 1939, May 1996

[61] Myers, John; "Simple Authentication and Security Layer", IETF RFC 2222, October 1997

[62] Myers, John; "SASL GSSAPI mechanisms", IETF Internet Draft draft-ietf-cat-sasl-gssapi-00.txt, March 1999

[63] Myers, John; "SMTP Service Extension for Authentication", IETF RFC 2554, March 1999

[64] Nokalva; "OSS - ASN.1 Reference", URL: *http://www.nokalva.com/asn1/index.html*, July 1999

[65] Newman, C; "Anonymous SASL Mechanism", IETF RFC 2245, November 1997

[66] Newman, C; "The One-Time-Password SASL Mechanism", IETF RFC 2444, October 1998

[67] Newman, C; "Using TLS with IMAP, POP3 and ACAP", IETF RFC 2595, June 1999

[68] Object Management Group, "The CORBA Security Service Specification (Revision 1.2)", URL: *ftp://ftp.omg.org/pub/docs/ptc/98-01-02.pdf*, January 1998

[69] Object Management Group, "Object Management Group Home Page", URL: *http://www.omg.org/*, 2000

[70] Object Management Group, "CORBA 2.2/IIOP Specification", URL: *http://www.omg.org/corba/c2indx.htm*, 1 December 1998

[71] Object Management Group, "CORBA Services Specification", URL: *http://www.omg.org/library/csindx.html*, July 1998

[72] Open Group, "Java Kerberos", URL: *http://www.opengroup.org/RI/www/jkrb/*, 1998

[73] Open Group, "X/Open Single Sign-on Service (XSSO) - Pluggable Authentication Modules", URL: *http://www.opengroup.org/onlinepubs/008329799/toc.htm*, 15 June 1997

[74] Parker, T; Pinkas, D; "SESAME V4 OVERVIEW", URL: *http://www.esat.kuleuven.ac.be/cosic/sesame/*, November 1999

[75] Postel, Jon; Reynolds, Joyce; "Instructions to RFC Authors", IETF RFC 2223, October 1997

[76] Rescorla, E; "The Secure Hyphertext Transfer Protocol", IETF RFC 2660, August 1999

[77] Rivest, R; "The MD5 Message-Digest Algorithm", IETF RFC 132, April 1992

[78] Sainsbury, Paul; "NT 4.0 Security", Honours thesis, Rhodes University, 1999

[79] Samar, Vipin; Lai, Charlie, "Making Login Services Independent of Authentication Technologies", 3rd ACM Conference on Computer and Communications Security, March 1996

[80] Schneier, Bruce; "Applied Cryptography - Protocols, Algorithms and Source Code in C", Second edition, John Wiley & Sons, ISBN: 0-471-12845-7, October 1995

[81] Schneier, Bruce; "Attack Trees", Dr. Dobbs Journal, December 1999

[82] Schneier, Bruce; "The Blowfish Encryption Algorithm", URL: *http://www.counterpane.com/blowfish.html*, December 1998

[83] Schneier, Bruce; "Risks of Relying on Cryptography", Inside Risks 112, Communications of the ACM, vol 42, n 10, URL: *http://www.counterpane.com/insiderisks3.html*, October 1999

[84] Schneier, Bruce; "Biometrics: Uses and Abuses", Inside Risks 100, Communications of the ACM, vol 42, n 8, URL: *http://www.counterpane.com/insiderisks1.html*, August 1999,

[85] Schneier, Bruce; "Index of Cryptography Papers Available Online", URL: *http://www.counterpane.com/biblio/*, 1998

[86] Schneier, Bruce; "Security Pitfalls in Cryptography", URL: *http://www.counterpane.com/pitfalls.html*, 1998

[87] Schneier, Bruce; "The Trojan Horse Race", Inside Risks 111, Communications of the ACM, vol 42, n 9, URL: *http://www.counterpane.com/insiderisks2.html*, September 1999

[88] Schneier, Bruce; "Why Cryptography is Harder than it Looks", Counterpane Systems, URL: *http://www.counterpane.com/whycrypto.html*, 1997

[89] Smith, Michael; "A Service Provider API for GSS mechanisms in Java", IETF Internet Draft draft-ietf-gssv2-javabind-api-02.txt, October 1999

[90] Smith, Michael; "A Java GSS binding, Part I: Interfaces", IETF Internet Draft draft-ietf-gssv2-javabind-ifs-00.txt, April 1999

[91] Sun Microsystems; "Java Authentication and Authorization Service", URL: *http://java.sun.com/products/jaas/*, January 2000

[92] Sun Microsystems; "Java Authentication and Authorization Service Frequently Asked Questions", URL: *http://java.sun.com/security/jaas/faq.html*, January 2000

[93] Sun Microsystems; "Java Naming and Directory Interface", URL: *http://java.sun.com/products/jndi/*, November 1999

[94] Weltman, Rob; Lee, Rosanna; Earhart, Rob; "The Java SASL Application Program Interface", IETF Internet Draft draft-weltman-java-sasl-02.txt, 4 June 1999

[95] Wray, J; "Generic Security Services API : C-bindings", IETF RFC 1509, September 1993

[96] Wu, Thomas; "A Real-World Analysis of Kerberos Password Security", Internet Society Network and Distributed System Security (NDSS) symposium, URL: *http://www.isoc.org/ndss99/proceedings/*, February 1999

[97] Wu, Thomas; "SRP: For Further Reference", SRP on the Net, URL: *http://berlin.arcot.com/srp/references.html*, April 1998

[98] Wu, Thomas; "The Secure Remote Password Protocol", 1998 Internet Society Symposium on Network and Distributed System Security (NDSS) symposium, URL: *http://srp.stanford.edu/srp/doc.html#papers*, February 1999

[99] Wu, Thomas; "The SRP Authentication and Key Exchange System", IETF Internet Draft draft-wu-srp-auth-03.txt, July 1999

[100] Wu, Thomas; "Telnet Authentication: SRP", IETF Internet Draft, draft-wu-telnet-auth-srp-04.txt, August 1999

[101] XER; "XER (XML Encoding Rules)", URL: *http://asf.gils.net/xer/*, August 1999

[102] Zimmerman, Philip; "PGP Documentation", URL: *http://www.pgpi.com/doc/*, December 1999

Appendices

Appendix A

draft-ietf-cat-srpgm-02.txt

Appendix B

draft-burdis-cat-srp-sasl-02.txt