Faculty of Computing and Information Technology

School of Computer Science and Software Engineering

# Scalable Access Control

Brian May, Bachelor of Digital Systems

Supervisor: Dr H. R. Wu

Associate Supervisor: Dr B. Qiu

September 18, 2001

**Enquiries:-**

Dr H. R. Wu,

CSSE,

Monash University,

Clayton VIC 3168,

Australia.

# Contents

# List of Figures

# List of Tables

# Abstract

Computer security is an important issue that is often overlooked or underestimated in practice.

Access control is an important area in computer security. A remote service needs to be able to decide if a given user is authorised to access a given resource or not. Access control is an area that has not been sufficiently addressed, leading to limitations that allow security breaches to occur.

Many applications of access control require both authentication and authorisation to work properly. Authentication is required in order to reliably determine the identity of the remote party. Authorisation is the process to determine if the given remote party will be given access to the resource or not.

Although a number of different authentication protocols, such as Kerberos, already exist, most of them fail to produce a secure solution that is scalable to the scope of the Internet. For instance, Kerberos is not scalable because it requires manually distributing private keys between realms.

Similarly, while a number of different authorisation protocols exist, such as SPKI, these also have limitations that prevent it from being used in large wide area networks, such as the Internet. SPKI, for instance was not designed to work with another authentication protocol such as Kerberos.

This thesis proposes a scalable *and* secure authentication method based on Kerberos with PkCross extensions. It does this via two individual proposals: one for securely distributing public keys to all participating comput-

ers, and the other being a proposal to implement non-repudiation into the scheme.

This thesis also proposes a scalable method of authorisation, based on SPKI, that addresses the limitations of current protocols.

The result of having a scalable authentication protocol and a scalable authorisation protocol that can be used together is a scalable access control protocol.

This goal was found to be reasonable, but certain trade-offs had to be made. For instance, the scalable key distribution for very large networks requires bandwidth in order to transfer data between KDCs. Although techniques have demonstrated how this can be minimised, it is still significant, especially for smaller sites.

This thesis is intended for anybody interested in the area of secure and scalable computer access control.

# Declaration

This thesis contains no material which has been accepted for the award of any other degree or diploma in any university or other institution. To the best of my knowledge, this thesis contains no material previously published or written by another person, except where due reference is made in the text of the thesis. This thesis was entirely written by me with corrections from my supervisor.

Consent is given for the distribution of this thesis for photocopying and loan. The author requests that if any copies made be double-sided and where possible to place more than one sheet on each page (be friendly to the environment).

..............................

Brian May

# Acknowledgements

I wish to acknowledge the support from my parents, and my supervisor, Dr H. R. Wu in reviewing my thesis and spotting mistakes. I also wish to acknowledge the assistance I received from the staff at Monash University's libraries in order to find some of the hard to find reference material required. In order to complete this thesis, I received the Monash Graduate Scholarship from Monash University.

# Chapter 1

# Introduction

Modern computers must be kept secure. This means making sure that unauthorised users do not destroy, tamper with, or have access to private information. As almost any application of a computer could use access control, there are many applications. However, only some applications need access control, eg. where access to a restricted resource is required. Examples of this include access to private data, hospital database systems[1], online banking/shopping[2], and secure communications.

In the past, access control could be achieved by locking the computer up and only giving authorised users the key. This is a very primitive form of access control, as only those who are authorised (given a key) may access the resource (files on the computer).

Using physical locks for access control is no longer a satisfactory solution for many applications though. Not only more and more computers are being connected to global networks, such as the Internet, in some way, but the requirements for access control are more complicated as users see the new benefits that global networks offer. For instance, in the past it may have been satisfactory to allow an authorised person access to all entries in a database, while now it is becoming more common to limit access to only a given set of database entries.

## 1.1 Access Control

> “Access control is the process of limiting [...] access to certain resources. Resources can constitute any number of things such as: a particular function, data, or a location. The process of access control involves the three separate processes of identification (recognising or indicating your identity), authentication (reliably establishing your identity), and authorisation (determining what you are allowed to access).”[3].

This definition defines access control in terms of three stages: identification, authentication and authorisation. Authentication is the process of identifying a particular user (ie. giving the user a name), while authorisation is to check if the user is entitled to access the resource or not.

Normally identification is included as a step in the *authentication* process (as how you identify yourself depends on the authentication protocol), so these 2 steps can be merged to form one stage, hence making two stages total.

Actually, there is a third stage of access control: accounting[4]. Users need to be held accountable for their actions in some way. This can be done, for instance, by recording identification, resource, and actions to a disk file.

The above definition would imply that authentication is required for access control, although some forms of access control (eg. a key to open a door) do not require any form of authentication. However, a tradeoff is that no one can be held accountable for their actions, as the server has no record of the user's identity. While some schemes have been developed that change this (eg. anonymous authentication[5]), they are beyond the scope of this thesis.

This definition is an internal view of access control, this contrasts with an external definition of access control, which views the internal stages as a black box.

> "A simple AC access control system [...] is a black box that accepts a query: 'Can user $U$ perform action $A$ on resource $R$' and returns a Yes (Y) or No (N) answer."[1].

However, both these definitions fail to emphasis that access control can be split up into multiple entities, which could range from a simple door lock to being distributed across the globe. The following definitions are used to generalise access control, so that it can be used in any situation.

For the purposes of this thesis, the user ($U$)[1] has direct physical access to the *console* ($D$) (could be keyboard, display, smart-card reader, biometrics scanner, etc) on the *client* ($C$) (eg. a process running on a local computer). $C$ in turn can communicate to the *server* ($S$). The server ($S$) must decide if the user ($U$) is allowed to perform action ($A$)[1] on the resource ($R$)[1]. A resource could mean Central Processing Unit (CPU) cycles, network bandwidth, private information, individual database records, money, and/or anything else depending on the application[6].

For *remote access control*, both $C$ and $S$ could be implemented as separate computers. The user has direct physical access to $C$ but can only access $S$ via $C$.

This is in contrast to *local access control*, where $C$ and $S$ might be separate processes on a single computer, protected by the operating system. Or, in some cases, $C$ and $S$ may both refer to the same process (eg. the initial password prompt when logging into a computer). This is a special case because security of the communications channel between $C$ and $S$ is not an issue (depending on operating system design), because communication is not required between $C$ and $S$. Ideally, access to $R$ needs to be regulated, but in such a way that breaking into the computer does not affect the security of other computers.

An extension of this special case can arise for computers disconnected from a computer network, eg. laptop computers, palm top computers, and private home computers. These systems will be called *disconnected computers* in this thesis. For these cases, all entities are typically stored in the one computer. These systems require special consideration, as it is impossible to contact a remote computer for access control.

In any case, *local computer* refers to the computer system which the user has physical access to.

In the example (Figure 1.1), $U$ needs to be able to prove its identity to $S$ before $S$ can allow $U$ access to $R$. $S$ first needs to authenticate $U$, then it checks that $U$ is authorised to access $R$. This is a good example of access control. Figure 1.2 illustrates the special case of local authentication where $S$ is the same process as $C$.

Note that *all* of the communications links (represented by arrows between the boxes) must be secure, either by secure communications channel, or by some other means, eg. encryption. For instance, somebody could observe a user entering a password ($U$ to $D$ link). Similarly a rogue process could intercept the password before $C$ can access it ($D$ to $C$ link). Regardless of the potential software issues, this thesis assumes that the physical hardware connection (eg. keyboard cable) between $D$ and $C$ is secure. If, however, the password is sent without encryption to $S$, then that could represent a security hole, too ($C$ to $S$ link).

Similarly, the entities represent boxes, and should either be trusted or not given private authentication information. Even if the links between entities are secure, it is possible that one of the entities has been compromised, and could send private authentication data to an intruder. A good access control protocol needs to minimize the possibility of this occuring.

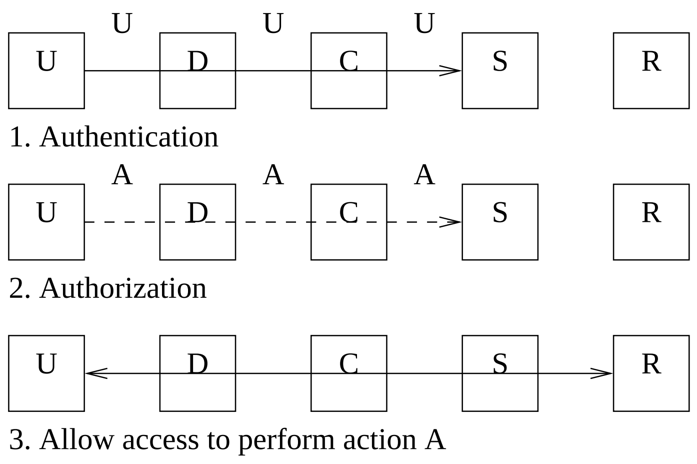


Figure 1.1: Access Control

$U$ =user, $D$ =console, $C$ =client, $S$ =server, $R$ =resource, $A$ =action.

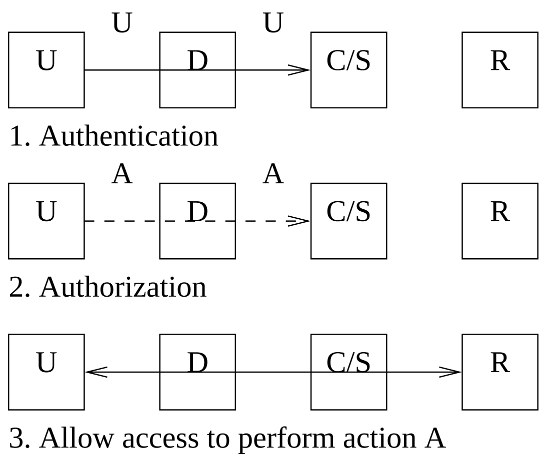


Figure 1.2: Local Access Control (client/server combined)

$U$ =user, $D$ =console, $C$ =client, $S$ =server, $R$ =resource, $A$ =action.

## 1.2 Authentication

In order to determine if one party should have access to a resource, the access control protocol may need to securely identify the remote party. This is called Authentication. Authentication is the process where one party can identify itself to another party, whether by identification (ID) card[3], or some more sophisticated computer protocol.

Authentication and authorisation are two distinct concepts that are often confused.

To quote one simple definition of authentication:

> "The property that the identity of each element of the system may be unambiguously established and validated, ie. that each element of the system is always the one claimed or required."[7].

Another, more precise definition of authentication, which is more specific in its requirements:

> "Whenever one of the parties completes an execution of the authentication protocol, it marks the execution as either accepted (in the case of successful authentication) or rejected. The intention is that executions marked as accepted correspond to runs of the protocol that definitely involved the intended other party."[8].

It is worth noting that neither of these definitions require the identity to be established in a global name space. That means, for instance, that the identity "Bob" could refer to different users depending on who is making the authentication (eg. the Scalable Public Key Infrastructure standard (SPKI)[3]). However, at the same time, it does not exclude a global name space to be used either (eg. Kerberos uses a userid + REALM name space[9]).

The most common means of authentication in computer systems is password authentication. Password based authentication systems, where the password is checked by the server, have the benefit in that they are easy to implement, especially on a large scale. However, this is at the expense of security. For instance password based systems are difficult to use without taking short cuts (eg. writing the password down) that may seriously compromise security[10].

A newer and better method of authentication is the Kerberos authentication protocol[9]. It is secure in that one password can safely be used for many servers. Even if one server is compromised, the security of the password is not compromised. However, Kerberos was never designed to work outside the one organisation. While attempts have been made to improve the scalability, it still suffers from at least three problems.

This thesis addresses three issues that affect scalability with Kerberos. Firstly, it investigates known extensions to Kerberos which already set out to make it more scalable, and attempts to isolate the good ideas from the bad. The use of public key technology with Kerberos will also be explored.

Secondly, it investigates the optimisation of the Kerberos protocol for cross realm authentication. It does this with the Proxy Authentication Ticket (PAT)[11], which enables one realm to issue tickets on behalf of the remote realm without having to contact it for each request.

Thirdly, it investigates the problem of distributing public keys. While public keys do not need to be secret, they must be kept secure at all times. This is an important issue that has not yet been satisfactorily addressed, even with certification authorities or certificate revocation lists[12].

## 1.3 Authorisation

> Authorisation is "the property that all actions in the system are always initiated, executed and terminated in accordance with

requirements and constraints of the security policy."[7]

Or, put in another way, authorisation is the task of determining if a given entity (ie. user or process) should have access to a given resource. This will obviously depend on the security policy of the resource.

Another reference says "[...] the process of authorisation takes place when a person controlling some resource determines who is allowed to access it"[3]. However, this definition is over simplified, because in practise it is a server that makes the decision who is allowed to access the resource, and this decision is based on the security policy being used[1].

Authorisation is closely linked to access control, in that both are intended for the same goal, however, access control includes authentication, and authorisation requires it.

This thesis addresses authorisation issues. Authorisation is the second step of an access control protocol. However, authorisation requires that end-users have access to non-repudiation.

Non-repudiation can be defined in different ways[13], but fundamentally it allows anybody to verify that a digitally signed statement (certificate) was written by the given author. Non-repudiation is important for authorisation, as it makes it possible for a user to have a certificate that grants access to a restricted resource.

Since non-repudiation is important to authorisation, suggestions are made in how this could be achieved in Kerberos, even though Kerberos is largely based on symmetric encryption.

General authorisation issues are investigated, and it is explained why SPKI[14] is a state-of-the-art access control protocol. However, SPKI does have some serious limitations (such as not being able to interoperate with Kerberos) which are addressed in Chapter 8.

## 1.4 Criteria

In order to evaluate the methods of authentication (used for authorisation) and authorisation, a criteria is required. However, as no criterion has been defined in existing references (eg. [3][1][15][16]) the following (idealised) criteria will be used:

- **Security:** The protocol should be secure and not allow intruders to forge their identification. Also, the system should be secure regardless of who is using it, or how much they understand computer security.
- **Scalability:** The protocol should be scalable to the scope of the Internet, without compromising security.
- **Facility of use:** The protocol should not be complicated to use. This means, for instance, that it should not require the user to memorise one password for each different computer. The protocol should be easier to use than to abuse.
- **Robustness:** If a user/administrator makes a mistake (perhaps without realizing it), the security of the protocol should be maintained.

Protocols will be discussed in later chapters that address these issues.

## 1.5 Aim and Scope of Thesis

This thesis is aimed at exploring issues involved in designing a secure and scalable access control system. The first issue involves scalable authentication so that the server can identify who the user is using the system. The second issue involves scalable authorisation, so that the server can tell if the user is allowed to access a remote resource or not.

This thesis explores ways in which these issues can be addressed on a wide scope, without compromising security in any way.

## 1.6 Structure of Thesis

The next chapter introduces the basic concepts behind access control, and how it is made up of authentication and authorisation.

Chapters 2 to 6 are on authentication and non repudiation. The first of these chapters, Chapter 2, briefly describes known techniques for authentication, including passwords, asymmetric authentication, the Needham Schroeder Protocol, Kerberos, and Sesame. Kerberos is based on the Needham Schroeder Protocol, and Sesame is now based on Kerberos.

One limitation of Kerberos is the requirement to contact a remote realm for every cross realm ticket request. This issue is addressed in Chapter 3. While this is a relatively simple issue, it leads on to more important issues.

One of these more important issues is how to distribute public keys in a scalable way. A protocol is proposed in Chapter 4 that demonstrates how public keys may be securely distributed across a large scale network. This is done by distributing keys across the network in a random manner, discouraging attempts to inject false keys into the system.

Non-repudiation is an important issue that is typically associated with asymmetric encryption methods. Chapter 5 describes how non-repudiation can be achieved using the techniques previously described. While this chapter is not directly related to authentication or authorisation, non-repudiation is important and has been incorporated in Chapters 7 and 8 on authorisation.

An evaluation of these authentication methods is given in Chapter 6.

Authorisation issues are explained in Chapter 7. This includes some existing methods, and their limitations.

In Chapter 8, SPKI is introduced. SPKI is a certificate scheme that does both authentication and authorisation. A number of weak points are addressed in the SPKI protocol, and potential solutions to fix these problems. Chapter 9 covers the conclusions of the thesis, followed by the glossary and references.

## 1.7 Contributions of this Thesis

Throughout the research for this thesis, a number of contributions have been made. In particular, this thesis:

- Provides a comprehensive review of existing authentication protocols, in Chapter 2. This includes simple protocols, such a password based authentication, and works its way up to state-of-the-art protocols such as asymmetric protocols and Kerberos.

- Describes fast cross realm authentication, in Chapter 3. This allows intra-realm information to be cached in order to speed up intra-realm ticket requests. This was published in [17].

- Highlights the problems with scalable public key exchange, and demonstrates a possible solution. Scalable public key exchange is required by any protocol that uses or requires public keys. An example of how this could be achieved is given in Chapter 4. This was published in [11]. This issue is often neglected, as protocols expect users to come up with some external solution.

- Shows that non-repudiation is important to state-of-the-art authorisation protocols. Chapter 5 also demonstrates that non-repudiation can be provided for even in authentication systems that do not use public keys, such as Kerberos. A selection of different solutions are described.

- Provides a comprehensive review of techniques used by authorisation protocols, and the strengths and weaknesses of each protocol (Chapter 6).

- Demonstrates possible enhancements to SPKI, in Chapter 8[18]. SPKI (an access control protocol) has a number of limitations which are addressed here. The proposals here allow SPKI to be used with Kerberos, to use multiple chains of signatures, and to be generalised so that certificates do not need to be specific to each protocol.

## 1.8 Publications by the Author

During the course of research conducted with relation to this thesis, much of the work has been published, or has been submitted for publication, as academic research papers. 95% of the work in these papers have been contributed by the main author, Brian May. The details of these papers are as follows.

**[11]** Brian May and H. R. Wu. Making Kerberos scalable. In Sihan Qing and Jan H. P. Eloff, editors, *IFIP/SEC2000: Information Security*, pages 144–147, Beijing, China, August 2000. IFIP, International Academic Publishers.

**[17]** Brian May and H. R. Wu. Scalable public key distribution. In Håkan Cardfelt, editor, *Papers presented at MADE2000*, Management and Administration of Distributed Environments, Göteborg, Sweden, May 2000. Chalmers University of Technology.

**[18]** Brian May and H. R. Wu. Scalable authorisation with SPKI. Submitted to Journal of Computer Security, March 2001.

# Chapter 2

# Authentication Protocols

This chapter describes known authentication protocols and associated problems. Authentication is the first step in access control, and allows identification of the end user for authorisation.

## 2.1 Password Based Authentication

The most widely used means of authentication as seen on most computer systems, is password based authentication.

Authentication is often mistaken with authorisation, as some access control protocols have strong support for authentication but limited support for support for authorisation, or vice versa. For example, a typical Unix system can have good authentication support, such as Kerberos, but poor authorisation based on file permissions. This contrasts with other systems, such as Windows 98, which uses passwords for authorisation to network resources but without any authentication.

Authentication occurs when the user $U$, sends a shared secret $P$ (the password), to the server $S$, via the local computer $C$ (Figure 2.1). The server knows the identity of $U$, as only the one user is meant to know the value of $P$.

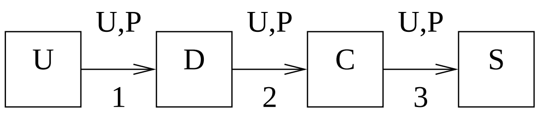


Figure 2.1: Password Based Authentication

There are weak links in the authentication:

1. $U$ to $D$: There is the potential that somebody may observe the user entering the password. Even worse, some systems automatically display the password on the screen as the user is entering it. Fortunately, these systems are reasonable rare.

2. $D$ to $C$: User $U$ manually enters the password $P$, into $C$. Passwords can be remembered by the person, can be readily updated, and (if used correctly) are difficult to steal or guess.

   The user may see a password prompt, think it is the login process, and enter password, when in actual fact a 'Trojan horse' program was running that grabs the users password and stores it for later use by the intruder[19].

   As explained previously, this is a problem when either the $D$ to $C$ link or $C$ can be compromised. This is the case with most authentication schemes, as they usually rely on the user entering a password which could be sent to the wrong program running on the computer. This false program could emulates the original software in every way, except it grabs the users password.

3. $C$ to $S$: The password $P$, may be sent from $C$ to $S$, either decrypted (eg. plain telnet) or encrypted (eg. SSL-telnet or SSH[20] using password based authentication). If the password is sent over the network un-encrypted, it may be stolen by spying on the network data.

   If the remote computer has been compromised (or perhaps it is the wrong computer), then it can grab the password when it is entered. Or, the remote computer would have to store the password somewhere (for verification), and this copy could be stolen. This is especially an issue if a user has the same password on many computers. Even if the remote copy is encrypted, it may be subject to off-line dictionary attack, which is relatively easy, as users cannot always be trusted to pick good passwords[21].

There are a number of different methods that $S$ could use to check the password. Some examples are explained as follows:

**/etc/passwd:** This involves encrypting the password via a one way function with a salt[22] (salt is a random value, its use means that the same password will look different on different computers). The result is checked against a global database in /etc/passwd[23]. Limitations: Anyone can read the password file, and attack it on another computer off-line via dictionary (or similar) attack. While the risk of the attacker cracking passwords can be minimised by enforcing the use of good passwords[10], problems still exist, as the password must be checked by the remote computer (see above section on forwarding authentication).

**/etc/shadow:** Similar to /etc/passwd, but only system processes can read it, hence making it harder to attack via off-line dictionary attack [22][24]. It is not scalable, as it will only work on one computer.

**NIS:** Same as for /etc/passwd, except the password file is shared between computers. Anyone on the network can read the shared password file. Online access to the Network Information Services (NIS) server is required, so it cannot be used for disconnected computer authentication[25].

**NIS+:** The Network Information Service Plus (NIS+) is similar to NIS [26], but communications can be encrypted.

**LDAP:** Similar to NIS+, but the Lightweight Directory Access Protocol (LDAP) is an improved protocol[27] that is more flexible. LDAP is a hierarchy based distributed database protocol that is ideal for authentication, but not restricted to authentication.

Encryption is supported so that spying on the network data is impossible. It is possible to duplicate changes on a disconnected computer for off-line usage, either by generating a slave LDAP server, or generating /etc/passwd and /etc/shadow files from the master LDAP server. However this opens up the system to dictionary attack, as anyone with physical access to the disconnected computer has access to the encrypted passwords.

Each of the three steps is a potential security risk, as it involves an extra stage where the password may be stolen. Furthermore, passwords are easy to abuse[19] and users may not always understand or adhere to common security policies.

For instance, the security of the password has certain requirements. Picking good passwords is extremely important[10]. While the number of possible passwords is extremely high, the number likely to be selected in practice is much smaller[10][21], making it relatively easy to break into accounts via brute guessing.

Even system administrators have difficulties with passwords: eg. should a system administrator recommend that the same password be used on all systems, or should a user have a different password on each system[19]? If one computer is broken into, the advantage of using a different password for each system is that the other computers remain secure. The disadvantage is that users find it harder to remember multiple passwords, and are more likely to write them down somewhere.

## 2.2 Biometrics

Biometrics is another possibility for achieving security. The user's fingerprints, hand geometry, retina, voice, face, writing and/or typing speed may be used in order to authenticate the user to the computer system[28].

Despite research in the area, for example Solms et al[29], the use of biometrics in large networks is limited, as a database is required of given biometric parameters. This has similar problems to keeping a database of non-encrypted passwords. This database must be kept secure, while at the same time must be distributed across the entire network.

While biometrics could be used in disconnected computers, if any unauthorised users gain physical access to the disconnected computer, they could gain access to the digital fingerprints of everyone authorised to access the disconnected computer.

Biometrics is only suitable where all the links (ie. card reader to $C$ and $C$ to $S$) are completely secure. It is not suitable for a general remote authentication system, as there is no way that the remote system can be sure the data is coming from the device, and is not being forged. As such, it resembles a password based authentication system, except that it is difficult to change the password (this may require changing features of the person).

As an example, if an intruder was able to grab a digital copy of $U$'s biometric parameters, then the intruder might be able feed it into the authentication process (between the card reader and $C$ or between $C$ and $S$), fooling $S$ into believing that he/she is really $U$. Not only that, but it would be very difficult to securely authenticate $U$ anymore, as this would involve changing his/her biometric parameters.

The biometrics may be used in conjunction with another mechanism, such as passwords[30]. However, there are serious problems when checking biometric details for large scale authentication. They involve similar problems to the password based authentication protocols.

A better system may be to combine biometrics with smart-cards[31]. This would mean that the user requires the smart card and the user's biometrics need to match before access can be granted. This is further discussed below, in the section on smart cards.

## 2.3 Asymmetric Authentication

Asymmetric authentication (eg. RSA[32] and DSA[33]) uses public key technology in order to prove the identity of $U$ to $S$.

Usually, the user $U$ keeps a copy of his/her secret inverse key (also known as the public asymmetric key) on his/her local computer $C$. Although this key is too complicated to remember as a password, it may be protected by a password.

The server $S$, keeps a copy of the users public key. While this key can be made public, the copy kept on the server must not be altered, and must correspond to the correct secret inverse key, or the wrong user may be granted access.

- $S$ randomly generates a nonce, and sends it.
- $C$ receives a nonce, encrypts (ie. signs) it with the user's private key, and sends it back to $S$.
- $S$ decrypts data with user's public key.
- If the result is the same nonce that was sent, the user is authenticated.

As a direct result of this requirement to keep a copy of the inverse key on the local computer, asymmetric authentication, by itself, is not suitable for local authentication, because in most cases an unauthorised user already has easy access to the secret inverse key. This generally rules out operation on disconnected computers

It can be argued that asymmetric authentication is not a true form of authentication as it must be stored on computer, where it can be used by anyone. The password protecting the public key can also be attacked by off-line dictionary attack.

However, with special hardware, smart cards may solve these issues, see Section 2.5.

A number of other issues remain. For instance, the remote computer needs some way of obtaining the user's public key, in order to verify the received data. While the public key does not need to be kept private, it must still be transmitted securely, or other parties could replace the key while it is being transferred. There are a number of ways this transfer could be done:

- Simply copy the key un-encrypted. As explained above, this is seriously flawed, as somebody else might replace the key with their key while it is in transit, hence the wrong user will be authenticated.

- Set up a CA (certification authority), which is responsible for signing correct keys[19], producing a *certificate*. This is still seriously flawed — how does the CA know when it is signing the correct key? Frequently a phone call is used to verify the identity of the remote user, but how does this prove anything? An intruder could be making the phone call, and the CA administrator would have no way of telling.

- Setup a network of trusted users. A user designates a key as being trusted by signing it. That way, chains of trusted keys can be created. ie. if $S$ trusts $X$ and $X$ trusts $U$, then $S$ automatically trusts $U$. However, this is not scalable, as the longer the chain becomes, the more likely it is that a single link may become compromised and sign a fake key[19].

This is just one issue that is difficult to get right. A total of five potential compliance defects are known[12], when asymmetric authentication systems are used by end-users:

1 Authentication of local user to distant CA, as previously explained, is difficult, and probably can only be done securely via a face to face meeting.

2 When authenticating the CA, it is difficult for end users to know if the CA's signature is valid, as this process requires knowledge of the trusted CA's signature.

3 It is difficult to revoke a public key reliably, in case the associated private key is stolen. It is possible to require servers always check a central key revocation list before using the public key, but this cannot work if the central server is down, or for off-line disconnected computer use.

4 Private keys are generally protected by a pass phrase. This prevents the key from being used by an intruder, if stolen. However, the user either must type in this pass phrase on each use, or store it in memory or disk.

5 There is no way to force a user to pick a good pass phrase, hence if the key was stolen, it could be subject to an off-line dictionary attack[21].

The first two concerns involve distribution of public keys. These concerns are addressed in Chapter 4. The other three issues are not as significant if smart cards are used, see Section 2.5. Another alternative is to use an authentication system such as Kerberos, which does not require public keys for end users.

These concerns have been relayed in an independent study[34], which identified ten security risks in public key infrastructure:

1 A ‘trusted’ CA may not take responsibility for its actions.

2 Protecting the private key from unauthorised use can be difficult (see Items 4 and 5, in the previous discussion).

3 Users may trust forged ‘root certificates’, for CAs (see Item 2, in the previous discussion).

4 Two people may share the same name, making it difficult to distinguish between the two cases.

5 Certificates cannot be used for authorisation, only authentication.

6 Certain implementations of public key protocols, eg. SSL, do not display (by default) the trusted identity of the remote host.

7 Some CAs use a less secure registration authority (RA) structure.

8 It is difficult for the CA to identify the end-user before signing their certificate (see item 1, in the previous discussion).

9 Many vendors of public key infrastructure do not adopt or support secure practices, eg. the use of certificate revocation lists (see item 3, in the previous discussion).

10 Other alternatives exist. Protocols that use Single Sign On (SSO) are suggested[34], for example, Kerberos.

Problem 4 can be solved by using user IDs and organisation names instead of user names. This means the identity can be unique for each user, as long as the organisation preserves the uniqueness of the user ID. This is the approach taken by Kerberos (see Kerberos section, below).

Problem 5 requires better authorisation procedures. Authorisation will be covered in more detail in Chapter 6. Problem 6 and 7 are implementation issues, and not discussed in detail.

Since all the problems unique to the second list have been addressed, the rest of this thesis will concentrate on the issues in the first list. It has been suggested that because of these limitations, public key systems are best suited for system administrators who can realistically be expected to meet the full demands of public key distribution[12]. In fact, this is what the PkCross[35] proposal aims for.

There are well known protocols for asymmetric authentication: Secure Shell[20] (SSH) (replacement for telnet) and Secure Sockets Layer (SSL). A later versions of SSL is called Transport Layer Security[36] (TLS). Both protocols are used by secure versions of Hypertext Transfer Protocol[37] (HTTP), Lightweight Directory Access Protocol[27] (LDAP), Post Office Protocol[38] (POP), Internet Message Access Protocol[38] (IMAP), and other communication protocols.

## 2.4 Symmetric Authentication

One computer (either local or remote) encrypts data with the user's password, and the other computer decrypts it. The remote computer must be trusted with the copy of the user's password. When the local computer decrypts and/or encrypts the data with the user's password, this proves the user's identity. Instead of the local computer doing the encryption/decryption, it could be the user's smart-card (as applicable).

Basic symmetric systems offer no additional security over password based systems with a secure channel between $C$ and $S$. This is because both systems need to know the user's password. However, better protocols, such as Kerberos, to be discussed later, overcome this limitation by using a trusted third party.

Symmetric authentication methods are a trade-off:

- symmetric systems generally require a strong password to remain secure. This can, and should be memorised.
- asymmetric systems require the private key to remain secure. This cannot be memorised, but must remain on disk. If it is stolen, an off-line dictionary attack is possible. Use of smart cards reduces the risk of the private key being stolen without notice.

While this simple protocol has problems, the ideas lead up to better protocols.

## 2.5 Smart-Cards

The user's secret inverse key (asymmetric protocols) or private key (symmetric protocols) may be stored on smart-card. Ideally this would be done in such a manner to make it impossible for anyone to read or alter the private data. This has many benefits, for instance asymmetric authentication protocols can now be used for local authentication. It also makes the traditional trojan horse attack impossible, as $C$ never needs to see any private keys or private passwords.

However, it could be argued that this is not a form of authentication, as anyone with the card can pretend to be the original user. To solve this, methods have been developed to authenticate the user to the smart-card first, eg. via a secret, a fingerprint scanner[31], or a portable secondary device[39]. This authentication would be carried out in a specialised smart card reader, so the secret can not be stolen by the computer.

Some applications may require a fingerprint scanner. For instance, if the user's smart card contains a drug prescription, then it must not be used by any other users in case the prescription is used illegally. Perhaps the original user wants to sell the prescription on the black market. While there is nothing you can do about this (the user could always illegally sell the drugs after obtaining them through legal means via a doctors prescription), using a smart card helps, as only one person can have the private key at a time. Also if a fingerprint scanner was required it would be harder for the user to allow illegal access to the smart card.

Computers with smart card readers are not yet common, nor are there any standards to make different smart card readers interact with wide scale authentication protocols. Also, current smart cards are not tamper resistant[15], hence there is always the risk of security being breached. Hence, smart card readers are considered as a distinct and separate solution for this thesis.

## 2.6 Needham Schroeder Protocol

The Needham Schroeder Protocol[40], first published in 1978, gives the framework for newer authentication protocols[19]. There are several versions of this protocol, however all of them rely on the same principles.

For the operation of this protocol, a trusted third party is required. This third party, now called the Key Distribution Centre (KDC), is trusted to authenticate one user to a computer (Figure 2.2). Essentially, the user $U$ authenticates to the KDC first, which is trusted by $S$. For implementation issues, the KDC does not communicate directly to the server, but instead sends an encrypted ticket back to $C$. As this ticket is encrypted in such a way that only $S$ can decrypt it, the end affect is that it is passed directly from the KDC to $S$.

When the client $C$ (acting on behalf the the user $U$) wishes to communicate to the server $S$, it securely obtains a ticket from the KDC. This ticket is encrypted in such a way that only $S$ can read it. When $S$ decrypts the ticket, it knows that it is from the KDC. The client, $C$ forwards this ticket to $S$, and this proves $U$'s authentication to $S$. This ticket contains a

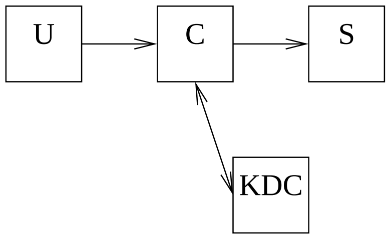


Figure 2.2: Trusted Third Party

shared secret key for use between $U$ and $S$ which can be used for subsequent communications.

However the different variants of the Needham Schroeder Protocol have different variants of the replay attack[41][42][19]. In one case, an intruder might steal a copy of the ticket that the client $C$ used to communicate to $S$, perhaps several years later. While $U$ no longer needs or is using the ticket, it remains valid indefinitely.

A number of solutions exist to this security flaw. One of them is the use of time stamps, as done in Kerberos.

## 2.7 Kerberos

Kerberos is an authentication protocol[9], based on the Needham Schroeder Protocol, created at the Massachusetts Institute of Technology (MIT). Kerberos, is similar to the Needham Schroeder Protocol in that it relies on a trusted KDC to issue tickets to clients. The clients in turn, pass the ticket on to the server for authentication.

The use of time stamps completely eliminates the security problems in the Needham Schroeder Protocol, however, it could be argued this introduces a new problems. For instance, some have argued that it is a security risk if the clocks aren't properly synchronised[43]. This issue has been satisfactorily addressed in a proposal[42] to synchronise the clocks on initial contact.

In Kerberos[9], each administrated domain is called a *realm*. Each user and computer within that realm is called a principle. The principle is typically the user's login name with the realm appended at the end. Initially, the client obtains a ticket for the ticket granting service (TGS), called the ticket granting ticket (TGT). This ticket allows the client to obtain tickets for other services without the user being prompted for his/her password each time.

In order to keep the notation consistent, this thesis will continue use the current notation for user ($U$), local computer ($C$), and remote server ($S$).

In the following, $X \longrightarrow Y : M$ means the message $M$ is being sent from $X$ to $Y$, $\{D\}_K$ means $D$ has been encrypted with the key $k$, and $X : \{D\}_K, K \longrightarrow D$ indicates the process where $\{X\}_K$ is decrypted with the key $K$ to form $X$ by $X$. For simplicity, only the fields listed in [44] are listed here. I have not included the fields not listed in [44].

Kerberos authentication[4] involves several steps[1]:

1. The local user asks the local computer to obtain a ticket.

2. The local computer requests the ticket from the KDC, given the principle of the local user.

   $$C \longrightarrow KDC : (C \text{ wants } T_{C,TGS})$$

   where:

   $$\begin{aligned} T_{C,TGS} &= \{K_{C,TGS}\}K_{TGS} \\ &= \text{Ticket for } C \text{ to use the TGS} \\ &= TGT \\ K_{TGS} &= \text{private key only known to KDC and TGS} \\ K_{C,TGS} &= \text{private TGS session key} \end{aligned}$$

[1]this example assumes that pre-authentication is not used[9]

3. The KDC returns a Ticket Granting Ticket (TGT), which contains the TGS session key ($K_{C,TGS}$) encrypted with the private key known only to the KDC and TGS ($K_{TGS}$) so that only the KDC can read it. It also returns the TGS session key ($K_{C,TGS}$) to the user, encrypted with $K_C$ (the user's password). This TGT contains a time restriction, and cannot be used after it expires.

   $$KDC \longrightarrow C : T_{C,TGS}, \{K_{C,TGS}\}K_C$$

   where:

   $$K_C = \text{user's password}$$

4. The local computer asks the user for their password and decrypts the session key. This in affect is local authentication. If the session key can be decrypted, it proves that the user is $U$. However, care must be taken not to get tricked into decrypting the wrong session key.

   $$C : \{K_{C,TGS}\}K_C, K_C \longrightarrow K_{C,TGS}$$

When the user wishes to access a new remote server (ie. remote authentication):

1. The local computer sends request for the ticket to the TGS. This request contains the principle for the remote computer, as well as the TGT. The authenticator is the clients principle encrypted with $K_{C,TGS}$, and is used to prove that the request did come from $C$.

   $$C \longrightarrow TGS : (C \text{ wants } T_{C,S}), T_{C,TGS}, A_{C,TGS}$$

   where:

   $$\begin{aligned} A_{C,TGS} &= \{C\}K_{C,TGS} \\ &= \text{Kerberos authenticator} \end{aligned}$$

2. The TGS returns a ticket, $T_{C,S}$ that can only be decrypted by the remote computer, as well as a copy of the new session key, $K_{C,S}$ that is encrypted with $K_{C,TGS}$, so it can be decrypted by $C$.

   $$TGS \longrightarrow C : T_{C,S}, \{K_{C,S}\}K_{C,TGS}$$

   where: $T_{C,S} = \{K_{C,S}\}K_S$

3. The local computer decrypts the session key, $K_{C,S}$, using the TGS session key, $K_{C,TGS}$. It forwards the ticket, $T_{C,S}$ to the remote computer. The remote computer knows the identity of the local user.

   $$C \longrightarrow S : T_{C,S}, A_{C,S}$$

   Where: $A_{C,S} = \{C\}K_{C,S}$

4. It can encrypt something with the session key and send it back to the local user. The local user decrypts the data and knows it is communicating to the correct remote computer.

   $$S \longrightarrow C : \{S\}_{K_{C,S}}$$

Full details can be found in [45] and [9].

### 2.7.1 Security of Kerberos

Unlike other competing techniques (e.g. Pretty Good Privacy[46] (PGP), SSH with asymmetric based authentication[47][20] and SSL[48]), Kerberos is based entirely on symmetric encryption protocols. This means that there is no need for end-users to maintain and distribute public keys and there is no risk of an attacker stealing a copy of the private key from a computer[12]. This makes Kerberos one of the most secure protocols for the mainstream public.

Kerberos has been exposed to intense international scrutiny. It has been proven numerous times, by completely different methods, to be entirely secure[41][49]. A number of limitations were found in Version 4 of the protocol[43]. While most of these issues have been addressed in Version 5[9][42][50], at least two problems still exist.

One of these is the risk of dictionary attack on the key which has been encrypted with the user's password. It has been demonstrated that this is easy if the users do not pick good passwords[21]. A number of solutions have been proposed, from improved protocols[21], using smart-cards[51], to using asymmetric keys[52]. The other problem is scalability.

### 2.7.2 Scalability of Kerberos

Kerberos has always been designed around the concept of a single realm. However, with the increasing growth of the Internet, more and more people want to be able to access sites from other realms (eg other organisations). MIT has tried to address these issues by introducing Kerberos Version 4 which allows cross realm authentication[45]. With this standard, authentication can now occur between realm boundaries. It is extremely limited though, as a pair of secret keys have to be setup for each pair of realms in advance. This requirement prevents it from being used in large scale environments.

Kerberos Version 5 tried to improve on this limitation[9]. It allows the configuration of tree like hierarchies, so no two realms need to directly know each other. Instead, one realm can contact another realm via several intermediate realms. However, this still has limitations in large scale networks: only one path may be used, and if any single host along this path is subject to a denial of service attack, the entire path becomes subject to denial of service. Even worse, if any single host is not trustworthy, it can forge any user's identity on the remote system.

Kerberos has a limitation that the user cannot authenticate without an active network connection. This is because there is no way to contact the KDC, a requirement of Kerberos. While it would be possible to copy the KDC onto the disconnected computer, this is dangerously insecure as anyone with access to the disconnected computer could break into every computer in the entire realm. Solving this limitation is not possible though without major changes to Kerberos (eg. make it more like an asymmetric based system).

## 2.8 Kerberos Extensions

As Kerberos is gaining in popularity, a number of different extensions have already been proposed to address some of its limitations. The remainder of this chapter investigates these extensions in detail, and discusses the pros and cons of some of the proposals.

### 2.8.1 PkInit

PkInit adds asymmetric authentication capabilities to Kerberos for end-users[52]. While this is an interesting idea, and is required for the PkCross proposal, it suffers from all of the problems (see Section 2.3, page 18) that asymmetric authentication schemes suffer, without adding any significant benefits.

### 2.8.2 PkCross

PkCross is a proposed extension to PkInit and Kerberos that allows it to use asymmetric authentication for cross realm authentication while continuing to use symmetric authentication for end users[35]. As end users do not need to worry about administration of the public keys, the compliance defects[12] (see Section 2.3, page 18) are no longer relevant.

However, the problem of securely distributing public keys still remains. This is one of the issues addressed in this thesis.

### 2.8.3 Sesame/DCE

Another technique, similar to Kerberos is Sesame (a Secure European System for Applications in a Multi-vendor Environment[53]). Sesame, can use asymmetric keys for authentication, similar to the proposed PkInit [52] and PkCross[35] extensions for Kerberos.

Sesame also implements an authorisation, using a privilege certification. As explained in Chapter 6, this privilege certificate allows listing the groups which the user belongs to.

However, Sesame does not add any significant features to the Kerberos protocol for authentication, and distribution is seriously restricted:

- Unlike some other implementations of Kerberos (eg. MIT [44] and Heimdal[54]), Sesame is not free software[55]. Instead, it is proprietary software because the the license requires you ask for permission before using it for non-experimental purposes. Commercial use requires payment for a commercial license. From my experience, free software encourages use and is more likely to comply with known defacto standards.

- Some functions (eg. Certification Authority) require the use of a proprietary operating system [56] (alternatives exist but are discouraged[56]).

The Distributed Computing Environment (DCE[57]) is similar to Sesame in that it adds authorisation to Kerberos[15], using a Privilege Authentication Certification (PAC). DCE is now fully open source. As such, it can be freely distributed and modified without restriction.

However, neither Sesame nor DCE changes the Kerberos authentication protocol, except to add similar capabilities for authentication to those of Kerberos with PkInit and PkCross extensions.

### 2.8.4 Smart Cards

A better approach for authentication is to use smart cards[51]. With a smart card system, it is easy to determine if a smart card has been lost or stolen. If the card is not lost or stolen, unauthorised use is not possible.

Authentication is very similar to standard Kerberos (Figure 2.3), except that the 4th step (highlighted) has changed:

1. The user, $U$, asks the local computer to obtain a ticket.

2. The client, $C$ requests the ticket from the KDC, given the principle for $U$.

   $$C \longrightarrow KDC : (C \text{ Wants } T_{C,TGS})$$

3. The KDC returns a TGT (ticket granting ticket), $T_{C,TGS}$, which is encrypted so that only the KDC can read it. It also returns a session key to the user, encrypted with $K_C$. This TGT contains a time restriction, and cannot be used after it expires.

   $$KDC \longrightarrow C : T_{C,TGS}, \{K_{C,TGS}\}K_C$$

   Where: $T_{C,TGS} = \{K_{C,TGS}\}K_{TGS}$

4. **The local computer sends the encrypted session key to the smart card reader ($D$). The smart card reader will decrypt the session key with the stored value of $K_C$, but only if the user has been authenticated to the smart card first. $K_C$ could be a random number, and does not have to be based on an easy to remember text password.**

$$C \longrightarrow D : \{K_{C,TGS}\}K_C$$
$$D : \{K_{C,TGS}\}K_C, K_C \longrightarrow K_{C,TGS}$$
$$D \longrightarrow C : K_{C,TGS}$$

After the ticket is obtained, authentication to $S$ occurs exactly as per normal Kerberos.

A possible improvement to this scheme would be to store the ticket on the smart-card, too, so it cannot be stolen after the card has been removed from the card reader.

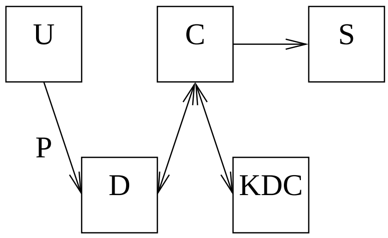


Figure 2.3: Trusted Third Party

While these techniques could considerably improve security, care still needs to be taken if the smart-card is stolen, as a tampered smart card reader could read the secrets being used to access the card. In this case, the smart-card can be disabled by changing the value of $K_c$.

However, the biggest problem with wide scale use of smart cards is that a single standard needs to be developed that can be used by everyone. This is a very important issue that is beyond the scope of this thesis.

### 2.8.5 Kerberos Configuration

Another important issue is locating the KDC. Traditionally, this information has been hard-coded into every computer in a file called `krb5.conf`. This is a serious limitation, as this file would have to be modified on every computer whenever any changes occur to any KDC on the Internet.

A good solution exists, where this information is now stored in the domain name service (DNS) [58]. The DNS was designed from the beginning to be scalable across the scope of the Internet. While a potential problem with the DNS is that information is not secure, Kerberos does not make any assumptions on the accuracy of this data anyway.

This extension has already been implemented in at least two implementations of Kerberos: MIT Kerberos and Heimdal.

## 2.9 Summary

Authentication is an important part of access control, as it identifies the user requiring access to a resource.

This chapter has shown that while a number of different protocols exist for authentication, they all have different tradeoffs. For instance, password based systems are the easiest to implement on a large scale, but are the least secure because users need to remember a different password for each system. Biometrics (eg. fingerprint scanner), on the surface, would appear to overcome this limitation, but biometrics require secure connections and a secure global database. If somebody's fingerprint is stolen, it is difficult to revoke it more than 10 times, as most people only have 10 fingers.

The asymmetric authentication methods remove the need to transfer a private key. Instead these require transferring a public key securely, currently there is no satisfactory way of doing this. They also require that end users maintain their own private keys in a secure manner. While smart cards would make the latter step easier, no single standard of smart card authentication system has yet been widely deployed..

Symmetric authentication methods (eg. Kerberos) remove the need for the user to maintain a private key on computer or the need to memorise many passwords, as the single password can allow access anywhere. However, Kerberos has other limitations, such as the design makes authentication between organisations difficult to administrate. While there are proposals to allow Kerberos to use asymmetric keys for cross realm authentication, no method has been designed for securely transferring the public keys required for cross realm authentication.

The next chapter addresses the major limitation in the Kerberos protocol in that cross realm authentication is not scalable to large internetworks.

# Chapter 3

# Fast Cross-Realm Authentication

The standard Kerberos protocol requires the local KDC to contact the remote KDC for every cross realm ticket request. This is slow, and more likely to fail (eg if the remote KDC is not contactable for some reason). The proposal in this chapter improves Kerberos performance between multiple realms, by allowing the local KDC to produce tickets on behalf of the remote KDC.

## 3.1 Introduction

If a number of local users wish to access a remote realm, then normally the local KDC would have to contact the remote realm once for each request. The proposal in this chapter allows the local KDC to issue tickets on behalf of the remote KDC with minimal contact with the remote KDC[1].

[1]The Proxy Authentication Ticket (PAT) proposal described here is not related to proxiable tickets as defined in the Kerberos standard. This proposal aims to allow the local KDC to perform authentication on behalf of the remote KDC, while the proxiable ticket standard in Kerberos allows a user to give his/her ticket to third party services, as a primitive form of authorisation (see Chapter 6).

## 3.2 Notation

Note the following use of terminology. All other notations are consistent with those in [45], the original Kerberos 4 proposal.

| | |
|---|---|
| Realm | Administrative domain. |
| $U$ | Any user who may be a client. |
| $C$ | A computer that provides access a server. |
| $S$ | A computer that provides a service to a client. |
| KDC | *Key Distribution Centre*, i.e. a trusted computer. |
| *LKDC* | *Local KDC* is the KDC in the same realm as the client. |
| *RKDC* | *Remote KDC* is the KDC in the same realm as the server. |
| PAT | *Proxy Authentication Ticket* allows a KDC to act as a proxy for the KDC at the remote realm and issue tickets on its behalf, not to be confused with proxy tickets, which can be given to a server to allow it to act on the user's behalf. |

## 3.3 Current Technology

The problem is that it is not currently possible for one KDC to hand over the responsibility to another KDC to issue tickets in a secure manner.

In order to access a server in a remote realm, the local client needs the ticket for the remote server. The client puts the request through to the local KDC for the ticket on the remote server. Although the ticket must be encrypted with the private key of the server, the local KDC does not know the private key. The local KDC could be given a copy of this private key, but that raises other issues, such as keeping this key private. Hence, it is not scalable.

Current versions of Kerberos get around this problem by forwarding the request to the next KDC up the tree or the remote KDC server. This means that if any KDC is compromised, then the results of the authentication can not be trusted.

## 3.4 Solution

However, the above techniques can be combined to produce a secure system. The local KDC can get the ticket from the remote KDC, for the Proxy Authentication Service (PAS). In order to simplify matters, the ticket is called a PAT. This ticket would allow the local KDC to produce tickets on behalf of the remote KDC. The full details are given below:

1. The client $C$ asks the local KDC for its service ticket, in clear-text:

   $$C \longrightarrow LKDC : (C \text{ wants } T_{C,S})$$

2. The local KDC asks the remote KDC for a PAT[2]:

   $$LKDC \longrightarrow RKDC : (LKDC \text{ wants } T_{LKDC,PAS})$$

   Note: If the local KDC was configured for PkCross[35] operation, the PAT would be obtained via PkInit[52] rather than standard Kerberos. PkCross would not be used, as the remote KDC sees the local KDC as the end client, not as an intermediate KDC.

3. The remote KDC returns the PAT as if the local KDC were a normal client.

   $$RKDC \longrightarrow LKDC : T_{LKDC,PAS}, \{K_{LKDC,PAS}\}K_{LKDC}$$

   where: $T_{LKDC,PAS} = \{K_{LKDC,PAS}\}K_{PAS}$ (assuming standard Kerberos is used).

4. The local KDC will create a ticket for the local user, as per the normal procedure, with one difference: This ticket contains the local KDC's

[2] This step and the next step can be skipped if the local KDC already has a PAT that will last for the lifetime for the client's ticket.

PAT in the TicketExtensions field. The rest of the ticket is encrypted with the $LKDC$'s session key. The PAT must not expire before the ticket issued to the user expires.

$$LKDC \longrightarrow C : T_{C,S}, \{K_{C,S}\}K_C$$

where: $T_{C,S} = \{K_{C,S}\}K_{LKDC,PAS}, T_{LKDC,PAS}$

5. The client forwards the ticket to the server along with its authenticator to the server in the normal manner.

$$C \longrightarrow S : T_{C,S}, A_{C,S}$$

where: $A_{C,S} = \{C\}K_{C,S}$

6. The server will create a secure channel to the PAS, using any secure protocol, such as Kerberos[9] (The PAS can be a machine independent of the KDC, but must be as secure as the KDC). It will provide the PAS a copy of the ticket. The PAS will decrypt the PAT, hence obtaining $K_{lkdc,PAS}$. This enables it to decrypt the contents of the key to obtain $K_{c,s}$.

   The PAS checks that the ticket is valid (it must be for a user in the same realm as the PAT was issued to), and returns it to the server.

$$S \longrightarrow PAS : T_{C,S}$$
$$PAS \longrightarrow S : K_{C,S} \text{ (encrypted)}$$

7. The server continues authentication as normal, but instead of trying to decrypt the packet itself, it trusts the decrypted results obtained from the PAS. The only task left is to decrypt $A_{c,s}$ using $K_{c,s}$ to obtain $C$. This verifies that the ticket did come from the client.

Previously, the client (either directly/indirectly) had to contact the remote KDC to obtain a ticket. Now, it is the server's responsibility to contact the PAS. This is better than the previous situation, as it is highly likely that the server would be on the same network as the PAS, and the PAS only needs to keep track of one secret, the PAS key. In addition, the PAS could be split up, as required, so different PAS are responsible for different remote realms.

## 3.5 Proof

Proof of security using Autlog[59], an improved form of BAN[41] logic. It uses the following definitions:

| | |
|---|---|
| $X \lhd Y$ | $X$ sees the data $Y$. |
| $X \models Y$ | $X$ believes the data $Y$ to be correct. |
| $X \mid\approx Y$ | $X$ recently sent $Y$. |
| $\#(Y)$ | $Y$ has never been sent before. |
| $\rho(Y)$ | $Y$ is recognised. |
| $X \overset{K}{\longleftrightarrow} Y$ | $K$ is a private key known only to $X$ and $Y$. |

The proof follows:

1. The PAS checks the PAT (contained within client's ticket):

$$PAS \lhd \{K_{LKDC,PAS}\}_{K_{PAS}} \tag{3.1}$$

$$PAS \models RKDC \overset{K_{PAS}}{\longleftrightarrow} PAS \tag{3.2}$$

$$PAS \models \rho(K_{LKDC,PAS}) \tag{3.3}$$

$$PAS \models \#(K_{LKDC,PAS}) \tag{3.4}$$

$$PAS \models RKDC \mid\approx K_{LKDC,PAS} \tag{3.5}$$

The first step indictates that the PAS, $PAS$, sees the key $K_{LKDC,PAS}$, encrypted by the server's private key, $K_{PAS}$. The next three steps assume that $K_{PAS}$ is for private use between $RKDC$ and $PAS$, and that the encrypted key can be decrypted and is determined to be fresh (never sent before). The last step is derived using rule K1[59]. It is the conclusion that *PAS* believes that *RKDC* recently sent the key that was received.

2. The PAS decrypts the client's ticket, and transmits it to the server. This step does not introduce any security holes as long as the transfer is encrypted. It does not even matter if the replay attacks occur, as the PAS does not check for replays, the server does when it receives the decrypted ticket:

$$S \quad \lhd \quad \{K_{C,S}\}_{K_{LKDC,PAS}} \tag{3.6}$$

$$S \quad \models \quad LKDC \overset{K_{LKDC,PAS}}{\longleftrightarrow} PAS \tag{3.7}$$

$$S \quad \models \quad \rho(K_{C,S}) \tag{3.8}$$

$$S \quad \models \quad \#(K_{C,S}) \tag{3.9}$$

$$S \quad \models \quad PAS \mid\approx \{K_{C,S}\}_{K_{LKDC,PAS}} \tag{3.10}$$

$$S \quad \models \quad C \mid\approx K_{C,S} \tag{3.11}$$

3. The server checks client's authenticator (as per standard Kerberos):

$$S \quad \lhd \quad \{C\}_{K_{C,S}} \tag{3.12}$$

$$S \quad \models \quad C \overset{K_{C,S}}{\longleftrightarrow} S \tag{3.13}$$

$$S \quad \models \quad \rho(C) \tag{3.14}$$

$$S \quad \models \quad \#(C) \tag{3.15}$$

$$S \quad \models \quad C \mid\approx \{C\}_{K_{C,S}} \tag{3.16}$$

Now $S$ believes that the client, $C$, recently said $C$ (ie its name).

This can illustrated as a search tree[60], as in Figure 3.1. For an intruder to tamper with the authentication process, he/she would ultimately have to alter $C$ (at the top of the tree). While $C$, in encrypted form is available (right hand side), the key required is not available (hence the required state, or req for short). The key, $K_{C,S}$ is only ever sent in encrypted form (next layer down). It can only be decrypted if $K_{LKDC,PAS}$ was available. However finding $K_{LKDC,PAS}$ requires knowledge of $K_{PAS}$ which is only known by $PAS$ and $RKDC$.

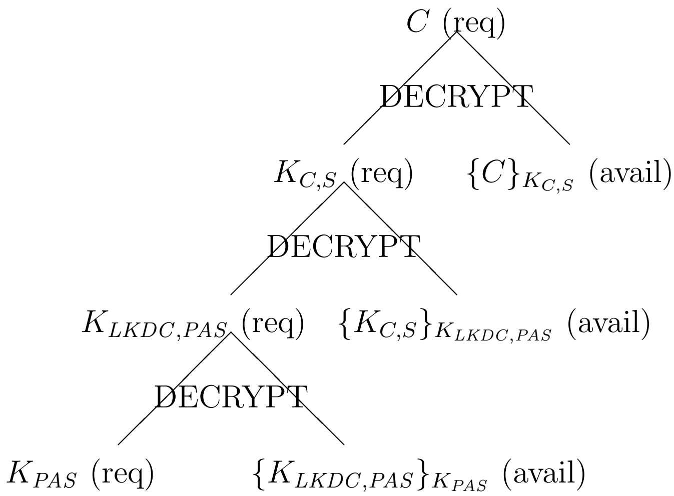


Figure 3.1: Search Tree

## 3.6 Implementation

This proposal only requires minimal modifications. No changes are required to the client software, as it just passes the ticket on without modification. The only changes required are to the local KDC (needs to request and use the PAT), the server (needs to contact the PAS and ask it to decrypt the ticket), and the PAS (needs to decrypt the incoming ticket).

## 3.7 Scalability

Greater scalability is achieved as any KDC can produce an indefinite number of tickets in a limited time span (see next section on security) for any other realm with just the initial contact. This results in less overhead traffic between the two realms. Also this means that the local KDC can issue tickets for the remote realm, even if the remote KDC is off-line, as long as it has a valid PAT (it may not be possible to use it though unless the PAS is online).

## 3.8 Security

This proposal is a tradeoff between efficiency and security:

- The efficiency of not having to contact the remote realm for every request.
- The potential security risk of someone stealing the PAT. This would allow the intruder to authenticate to the remote realm as anybody in the local realm. This risk would expire when the PAT expires.

Since the PAT must be kept as secure as the local KDC, and at the same time loss of the PAT will only affect users of the local KDC, the expiry time can be reasonable large. This will maximise the time allowed before the next PAT must be issued.

If somebody steals the PAT, this would typically involve breaking into the local KDC first. Normally, if the local KDC is broken into, the remote KDC can be told to deny access to future ticket requests (ie. the value of $K_{lkdc}$ can be changed). However, if a PAT was used instead, then access would still be granted until the PAT expired (or authorisation information changed, but that is beyond the scope of this thesis). Hence, some KDCs may want a lower expiry time in order to minimise the damage in case a remote KDC is known to be have been breached.

The PAS could also be reconfigured to reject any incoming authentication requests from the suspect domain. This would allow the local KDC to keep a copy of the PAT indefinitely, as requests will be denied on receipt by the server. However, this would make the PAS more complicated, as it would have to store information on suspected domains, and the small gain may not be worth the effort.

## 3.9 Summary

The proposal in this chapter improves Kerberos performance between multiple realms, by allowing the local KDC to produce tickets on behalf of the remote KDC. This overcomes the limitation in the standard Kerberos protocol, in that it requires the local KDC to contact the remote KDC for every cross realm ticket request. This is slow, and more likely to fail (eg if the remote KDC is not contactable for some reason).

This is achieved by allowing the local KDC contact the remote KDC in advance, in order to get a proxy authentication ticket, which the local KDC can use to create tickets on behalf of the remote KDC.

# Chapter 4

# Scalable Public Key Distribution

Scalable public key distribution is an important part of any protocol that uses asymmetric keys for authentication. Unfortunately, existing methods suffer from at least one serious limitation: there is no easy way to identify the correct public key over a fraudulent public key, hence allowing the 'man in the middle attack', where the wrong user is identified.

The proposal given in this chapter allows unlimited distribution of public keys in a secure way on a large scale network. There are two ways this could be used:

1. The secret inverse keys each belong to a KDC, and a proposal like PkCross is used, so that keys do not have to be allocated to individual end users (see Section 2.3, page 18 as to why this can be a bad idea).
2. Each user has a secret inverse key on a protected smart card (eg. fingerprint detection[31]).

In each case, there are a large number of keys that must be securely distributed throughout the system. For the remainder of this chapter, case 1 is used, although in practice, case 2 could be used, too.

This thesis does not aim to solve all problems associated with key distribution in one go, but proposes a unique method to encourage future research into this important area.

## 4.1 Introduction

Today, most modern computer security protocols (eg. SSH[20] [47] with RSA authentication, PGP E-Mail encryption, and Kerberos[52] [61] with the proposed public key extensions: PkInit[52] or PkCross[35]) rely on public key encryption methods. However, there is still no scalable method of securely transporting a public key without risk of it being tampered with along the way. Even CA (certification authorities) do not solve the problem, as there is no way the certification authority can be certain it is signing the correct public key.

Another solution to this problem is to construct a path of signatures, ie as X trusts Y and Y trusts Z, X must trust Z. However, this involves a single point of failure. For instance, assume Y gets a job for a major competitor and is no longer trustworthy. If you continue trusting keys that Y signs, it would result in a breach of security.

The obvious solution is to require multiple paths in order to verify the signature from multiple parties. However, requiring more paths requires more manual work in order to verify each path remains valid. e.g. no chain should ever (now or in the future) cross at the same point, or the single point of failure will be reintroduced.

For instance, assume that X wishes to communicate with host Z, and needs Z's signature. X may not have any prior knowledge of Z, and Z may not have any prior knowledge of X. Without any prior knowledge it is difficult to obtain one path of signatures, but many paths are required for maximum security[62].

## 4.2 Metric System

This chapter proposes a new protocol designed to automate the process of maintaining and distributing paths of signatures. That is, signatures are distributed in advance, without any manual intervention.

This is done using Reiter and Stubblebine's proposal for measuring the assurance provided by a set of paths[62]. The metric system represents the keys and signatures as a directed graph.

Each node assigns a trust level to the keys it signs, before the results are distributed to other nodes. When required, a node will calculate the metric level based on the trust level assigned by each node along the chain of signatures. The final metric value for a given unique path will be the minimum trust level any key has along that path.

Hence, it is impossible for any one key to increase the final metric value. This is why this system is preferable to alternatives. If multiple unique paths exist, then the final metric value will be the sum of the metric value for each path. More generally, the metric value will be the maximum flow in the directed graph from the source key to the destination key.

The minimum metric level required the trust level could be set in a language similar to PolicyMaker[63]. For example, an online transaction worth $10 would not require such a large number compared with an online transaction worth $1,000,000.

Keys can be distributed and authenticated via multiple trusted chains over a large scale network, making it easy to manage. This, in effect, is a large scale key server. It is distributed over many systems, and automatically manages signatures for individual keys as well as the keys themselves.

## 4.3 Method

Each KDC in a region is initially set up with public keys for three other KDCs. This initial distribution must be done manually, eg. face-to-face meeting. Each key received by the KDC is automatically signed by that KDC to prevent tampering. The signature is kept separate from the key so each one can be distributed separately. Each key must identify how that realm can be contacted, and can also have an expiry date.

Every hour, every KDC will place a request to a selection of trusted KDCs. This request will contain a list of all keys that the requesting KDC has, and associated signatures. This will inform the remote KDC what keys and signatures not to send. It does not need to be encrypted in anyway, as security does not depend on it remaining secure or confidential.

The reply will contain a list of all keys and signatures that the remote KDC knows will be of value to the local KDC (ie that the local KDC does not already have).

Upon receipt of the public keys and signatures, the local KDC will do the following:

- Calculate the metric value for each key[62]. This is done by modelling the list of keys and signatures into a graphical form. Each key is represented by a node. A signature by one key to another key, is represented as a arc, a point from the first key to the second key. Each arc has a capacity, a number used when the signature was created. For now, assume this to be a constant number, e.g. 10. The metric value is the maximum "flow" through the network, from the key representing the current KDC to the end key. This will be 0 for disconnected parts of the graph, which cannot be verified.

- Check for duplicate keys. If duplicate keys exist, then calculate the metric value of the signatures. Trust the key with the highest metric value. If the metric values for two keys for the same KDC are the same, then trust neither key.

- Does not sign the incoming key, as signing should normally be restricted to when the local KDC is positive that the key is not a fake (i.e. some external means is used to verify it is correct).

If, when validating a signature, it is found to be invalid, then the key must have been tampered with during the last transmission, and is simply discarded. Alternatively, a message may be displayed to the system administrator warning of the problem, in case there is an indication that the Kerberos realm is under attack.

## 4.4 Example

An example is illustrated in Figure 4.1. The blocks indicate different KDCs in the network, and the arrows between the blocks represent key pairs that have been pre-configured (and are assumed to be correct). For instance, A has W's public key so W can send secure (unforgeable) messages to A. Similarly, W and X both have the shared public key for each other, so secure communication is possible in either direction. This example deliberately has a number of problems (some of them might already be obvious) which will be explored later.

This model was simulated, on the assumption that every KDC contacts each neighbour (vertical as well as horizontal) in every iteration. In practise, the algorithm which determines what pairs of KDCs contact each other can be more sophisticated.

The arbitrary set of signatures at each node is represented by Figure 4.2. Each public key is represented by a node[62]. Each arc (eg. $a = (A, B)$) represents a signature made by the public key at the tail of the arc ($A$) on the public key at the head of the arc ($B$). The result is a directed graph

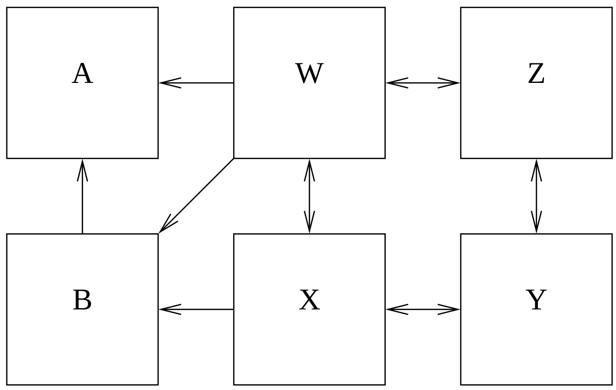


Figure 4.1: Simple network structure: The boxes represent KDCs, and the arrows represent the direction in which keys are initially copied.

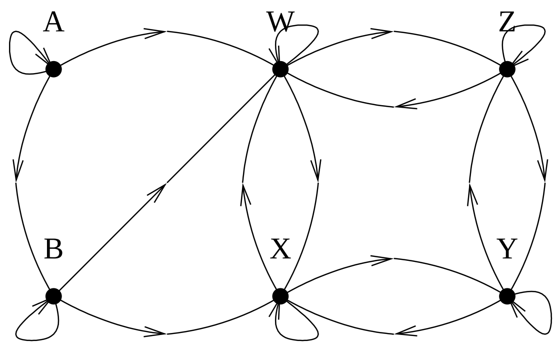


Figure 4.2: Set of all Signatures

illustrating the signatures between the public keys. It is assumed that these signatures have not yet been distributed, so that any given node will only hold signatures which it created.

Figure 4.3 lists which signatures have been copied for each iteration, starting from the initial configuration. This is from the perspective of Y (the other KDCs will differ slightly). Figure 4.3(d) represents the ideal final state.

For improved clarity, the number representing the capacity of each arc has been omitted from the figure. It is 0 if the signature isn't known to the KDC (i.e. no arc exists), or 10.

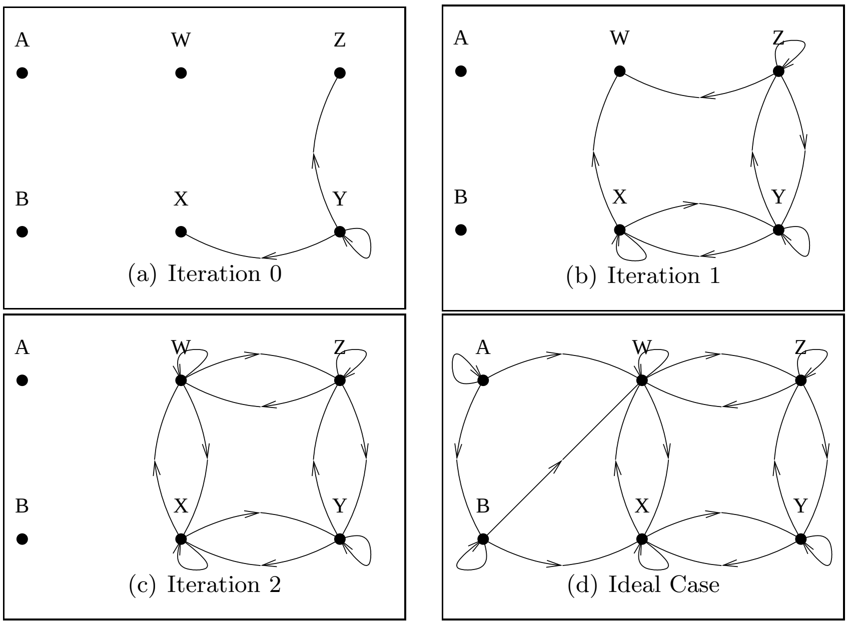


Figure 4.3: Y's Signature graphs

The results of the metric analysis for each iteration is listed in Table 4.1. Note that after the first iteration, W has a very high metric value of 20, and Y has an even higher metric value of 30. This is because there are three unique paths from Y to Y and 2 unique paths from Y to W.

Note that the metric value for A's and B's key is 0. This is because there is no path in the graph leading from Y to A or Y to B (it goes in the wrong direction. This means that A and B have been cut out of the network, and their keys, cannot be used. This problem is because no KDC has produced a signature that can be trusted by W.

This could be solved by W signing B's key. However, this would give a metric value of only 10 for both keys, as there would only be one valid path. More signatures are required for a higher metric value.

| iteration | A | B | W | X | Y | Z |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 10 | 10 | 10 |
| 1 | 0 | 0 | 20 | 10 | 30 | 10 |
| 2 | 0 | 0 | 20 | 20 | 30 | 20 |
| 3 | 0 | 0 | 20 | 20 | 30 | 20 |

Table 4.1: Y's Metric Values

Now assume that X has produced a false signature for W. This means Y will have two separate paths to W: the real one (as delivered from Z) and the false one (as delivered from X). As Y has no way of knowing which is the correct key (both have the same metric value), there is no way of knowing which one is valid.

Suggested solution: Each KDC should initially have three links to other KDCs. If one KDC produces bad results, the other two should (in theory) still match.

Also note, if the correct decision can be made as to which path is correct, Y has all the evidence it needs (i.e. X's signature) to prove that X is guilty of signing a false key. Y may want to lower the capacity it has given to all of X's signatures, and possibly notify other KDCs of the problem.

## 4.5 100 Keys

This example has 100 keys, each owned by one KDC. Each KDC is placed in a ring, and knows the keys to 4 other KDCs:

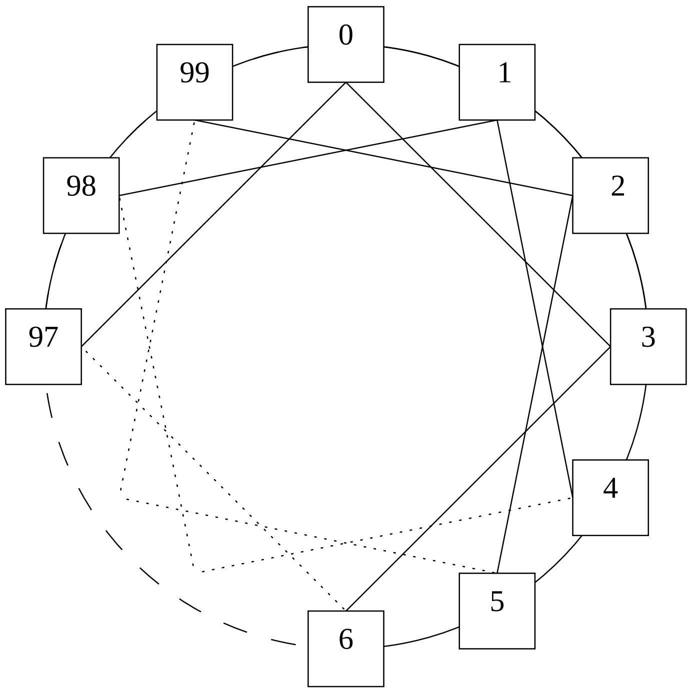


Figure 4.4: 100 node network structure

- 2 adjacent KDCs.
- 2 KDCs exactly two hops away.

See Figure 4.4 for illustration.

Assuming each KDC contacts its four most immediate neighbours for each iteration, the total update would take $100/4 = 25$ iterations. If this is too large, it can be reduced by allowing each KDC to contact more remote KDCs.

Distribution of keys has minimal overhead as a key is approximately 0.8 kbytes (PGP RSA based public key). However, very few keys will be distributed in each iteration (i.e. 2), as keys are only distributed to neighbours, and only if the neighbour does not already have a copy of the key.

Transferring signatures is potentially a bigger problem. In this example, there are 500 signatures in total. As each of the 100 nodes signs 4 other nodes and itself directly, the maximum amount to be transferred between any set of KDCs will be 500 signatures. This, in practise will be lower, as the KDCs can negotiate not to send duplicate signatures. As a signature is approximately 150 bytes long (PGP RSA based detached signature), this would mean 75 kbytes are transferred between any two KDCs. This is very small, and well within reasonable limits. Also, it could be argued that one set of signatures does not enhance security in anyway (i.e. the signature by the same key), nor does it add anything to the metric value (for remote hosts). By not distributing these signatures, a saving of 15 kbytes is achieved.

The length of each path is also significant, as the maximum length of a given path may be 100 nodes (for a path back to the same KDC). Alternatively, the maximum length of any path to a different KDC would be 99 nodes. This should not be a problem given an optimum algorithm for determining maximum network flow.

## 4.6 Scalability

For the above example, eventually each KDC will have to store 400 signatures. This should be well within reason for large networks (remember that each KDC can cope with a single realm, and each realm can be as large as required). It is definitely not reasonable for very large networks, e.g. with 1 million KDCs, 4 million signatures would eventually be required (assuming same distribution rules apply). This would consume 781 megabytes for keys, and 572 megabytes for signatures.

For very large networks (eg the Internet), it is possible to scale down the information required. For instance, public keys can be obtained (either directly or indirectly) via the domain name service (DNS), using a similar mechanisms as in[58]. They can be down-loaded on demand, as required. Once a key is down-loaded, it can be cached for as long as it may be needed. This already eliminates the need to transfer 781 megabytes of data.

An alternative might be to look signatures up using DNS, so everything can be distributed on demand. However, this means hosts must also search for trusted paths, slowing the process down.

Instead of transferring each signature, one at a time, a label can be distributed instead. While the label (typically a short number, eg. 15 bytes) cannot be trusted by itself, it would indicate where the trusted signature can be down-loaded from. This would reduce the memory required for keys down from 572 megabytes to 57 megabytes. In addition, this could be significantly compressed, making it even lower.

However, this may have the drawback of requiring a lengthy down-load to obtain all keys and signatures in the paths required. This only needs to occur once though, in order to cache the required information. No search is required to find which paths can be trusted by a given KDC, as the KDC already has this information immediately available.

While signature information is not protected in anyway, this is not trusted by the KDCs until verified by down-loading the real signature. Hence, denial of service attack is possible, where the correct KDC is prevented from receiving the correct key (by altering the signature information), but denial of service attack is already a possibility (e.g. by blocking network channels).

## 4.7 Security

This proposal is a tradeoff between two risks:

- The risk of the administrator making a mistake, when manually maintaining a large number of keys, that affects network security.
- The risk of a KDC being tricked into trusting a false key.

The first risk will increase as the number of keys required goes up. It could be as simple as a certification authority (CA) accidently signing a fraudulent key. When the CA signs the bad key, KDCs will automatically trust it, affecting network security.

The second risk is less likely with some degree of automation, as administrators have more time to concentrate on verifying the identity of remote hosts, before signing their keys. Also, administrators only need to sign keys for a limited number of remote hosts, further reducing workload.

## 4.8 Summary

Existing methods of scalable public key distribution suffer from serious limitations. There is no easy way to identify the correct key over a fraudulent key. This allows the 'man in the middle attack', where the wrong user is identified.

This is despite the fact that scalable public key distribution is an important part of any protocol that uses asymmetric keys for authentication.

The proposal given in this chapter allows unlimited distribution of public keys in a secure way on a large scale network.

While this may not be a final solution to the problem, it is a unique solution in order to encourage future research into this important area.

# Chapter 5

# Non-Repudiation

The previous two chapters both described methods to make Kerberos more scalable in large networks. This chapter describes how to incorporate an important, and frequently required feature into Kerberos — non repudiation. This is achieved without comprising security in anyway, and does not require any extra work on behalf of the users (ie. it can be automated to such a degree that no extra effort is required).

This chapter is not relevant if an asymmetric authentication scheme is used, as in this case the user already has a secret inverse key which can be used to sign messages.

## 5.1 Introduction

Non-repudiation is the property where you can prove to a third party the creator of a message[19]. It is often considered that this cannot be achieved without using asymmetric encryption, but this chapter describes not only how it is possible with symmetric encryption, but why it is preferred in certain cases.

For instance, with normal symmetric encryption, a receiver of a message (eg. E-Mail) cannot prove to a third party that the message is genuine without exposing the secret inverse key, and once the secret inverse key is made public, its proof is meaningless (ie. anyone can forge a message).

This is possible using asymmetric encryption, however, as the sender of a message can 'sign' it with their secret inverse key. This signature can be verified by anyone who has a copy of the public key. This property is called 'non-repudiation'[19].

Non-repudiation is important to authorisation, as a frequent requirement of authorisation protocols is to have one person authorise another to have access to a resource. Non-repudiation allows the first person to give the second person a signed certificate, which can then be independently verified by the resource. This removes the need for the resource to be configured beforehand.

## 5.2 Combining PGP and Kerberos

A technique that has been suggested is to combine Kerberos and PGP[64]. This method allows public keys to be signed by a trusted server (CA) that authenticates users by Kerberos, without requiring any manual intervention. However, that could require a large number of requests to the one CA.

This could also be combined with the scalable KDC key distribution as follows: Each KDC authenticates $U$ using the proposed scalable Kerberos solution, and receives the public key for $U$ at the same time. The KDC signs the key with its secret inverse key (the associated public key has already been widely distributed), which anyone can check to ensure that they have a valid copy of $U$'s key. This is illustrated in Figure 5.1.

### 5.2.1 Ensuring Security is not Compromised

Asymmetric authentication has a number of potential security problems when used by end users (see Section 2.3, page 18). These must be taken

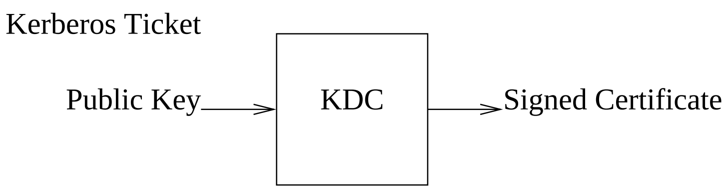


Figure 5.1: KDC Signs User's Public Key

into consideration here.

Some possible improvements:

- The KDC could set or require an expiry date/time set on public keys, forcing users to regularly update their keys. The exact rules for enforcement could vary depending on the administration of the KDC, and the desired level of security.

- Any person or program that verifies the signature must keep in mind that the signature is only secure if the secret inverse key has not been stolen. This should not be used as a means to protect very sensitive information unless the owner of the secret inverse key is confident that the key will not be stolen. The level of paranoia required here would depend on how sensitive the information is. Also keep in mind that access to some systems may lead to easier access to other systems, so just because a computer does not have any sensitive information does not mean security can be ignored.

These items would eliminate some of the known problems with asymmetric authentication[12], (see list in Section 2.3, page 18). Authentication of the user to a distant CA (Problem 1) is solved using Kerberos. Ensuring that the CA's signature is valid (Problem 2) can be achieved with the secure public key distribution described in this thesis. Revoking public keys (Problem 3) is solved with the use of time and date stamps. However secret inverse key management (Problem 4) and pass phrase quality (Problem 5) have not been solved. For these problems, there are two possible solutions:

- The KDC could set the public key, before signing it, so it expires when the user's ticket expires. This may be sufficient for some applications, eg. see chapter 6 on authorisation. However, other applications require a long expiry date (eg. signing E-Mail), and this would clearly be insufficient.

- Or, the solution in the next Section could be used, instead.

## 5.3 Without End-user Asymmetric Keys

The goal of non-repudiation without end user asymmetric keys, surprisingly, is easy to achieve using a very minor modification to the above routine - instead of the user $U$ giving the KDC a copy of their key to sign, $U$ gives the KDC (or many KDCs) a copy of their message to sign (as illustrated in Figure 5.2). If $U$ wishes to send a private message, then the KDC could sign a 'message digest[1]' instead of the full message. This would give the end user the security of non-repudiation without the security risks associated with keeping a secret inverse key secure. Furthermore, the KDC could include the time and date (ie. a nonce) on the message, making replays easy to detect.

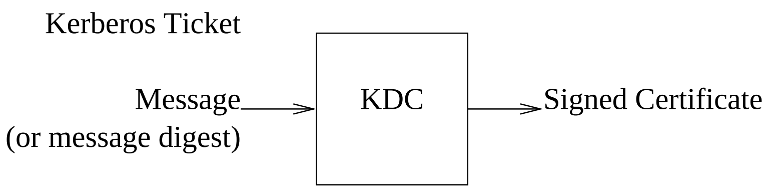


Figure 5.2: KDC Signs User's Message

With this in place, all security issues have been addressed, as the end user no longer needs to keep a asymmetric secret inverse key. The only key that the user must keep is the symmetric key (ie. password) used for

[1]A hashed representation of the message[19]

initial authentication to the KDC. This password can be memorised without needing to write it down anywhere.

## 5.4 Implementation

The ideas presented in this chapter can be implemented quickly with the standard Kerberos programming library (or higher level libraries such as GSSAPI or SASL). These tasks can be made automatic so the end-user is not aware of how non-repudiation is achieved, only that it is there, and it can be relied on.

If $U$ continues using a secret inverse key and has the public key signed by the KDC, then the only difference noticed will be in requesting the local KDC sign the key. If on the other hand, the user requests that the KDC sign individual messages, then this could be done in a similar manner to signing messages with PGP, except that a high level program (eg. mail user agent) does not need to prompt the user for a password or store a password in memory — it can use the existing Kerberos ticket instead.

Checking the signature is done using standard techniques, and requires no modification. The public key used may vary though. In either version, the public key of the KDC is required. This can be used to check either the public key of the sender, or the message itself.

## 5.5 Scalability

The ideas presented here are as scalable as the authentication system used, Kerberos. While Kerberos is not scalable, this has already been addressed in this thesis.

The first method (signing $U$'s public key) combines the scalability features of user based public keys, where everything is up to the user, and a scalable Kerberos implementation (some of the signatures are managed by Kerberos). However, the second method (signing $U$'s message) is based only on signing the message, in order to eliminate the chances of the user mismanaging their secret inverse key.

## 5.6 Security

While it is recommended that most people rely on the local KDC signing their individual messages, there may be some people who do not like the idea of trusting the KDC. For this reason, the first method is still available, having the KDC sign the user's public key, instead. This means that the user can use conventional asymmetric key type authentication, and have other entities (eg. CAs or other people) sign their keys, as well as the KDC.

This way, the level of security that can be obtained is up to the end-user, even though a centralised security system is used.

## 5.7 Summary

This chapter has described a scalable and secure method of non-repudiation. There are two versions provided for different situations: one where the central KDC automatically signs messages for users and another where the central KDC automatically signs public keys for end users. This choice allows users and/or administrators to pick the best choice that fits the requirements of the site.

# Chapter 6

# Authorisation

Authorisation is an important issue that has not been addressed by Kerberos, nor was Kerberos designed to address this issue.

This chapter discusses issues involved in current Authorisation protocols, and well known current implementations.

## 6.1 Methods

Authorisation can be implemented using one of the following techniques[1]:

- Determine the user's access strictly based on a password (eg. Windows 98), as shown in Figure 6.1. This is the simplest form of authorisation, but least secure, as there is no way to hold individual users accountable for their actions. Typically, each resource would have a different password, which users would have to remember when accessing that resource.

- Use the result of the authentication (eg. password based), as illustrated in Figure 6.2. In practise, this can only be used if the authentication system returns the identity as a number (UID or User Identifier) that

[1]The diagrams in this section are based on [1].

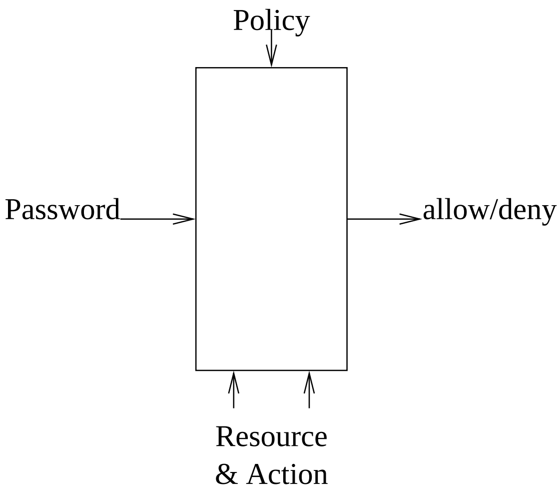


Figure 6.1: Password Based Authorisation

is specific to a network or computer.

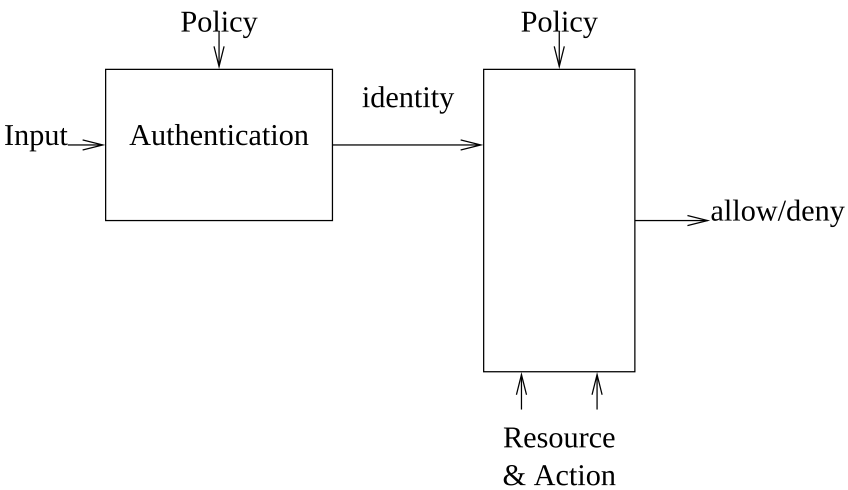


Figure 6.2: Authorisation Based on Authentication system

- Lookup a table based on result of the authentication (eg. Kerberos [9]), as illustrated in Figure 6.3. Here the identity returned by the authentication is turned in to a UID by a lookup table.

- Lookup a table based on what groups the user is in. Any number of users can be assigned to a group, which can be used to simplify

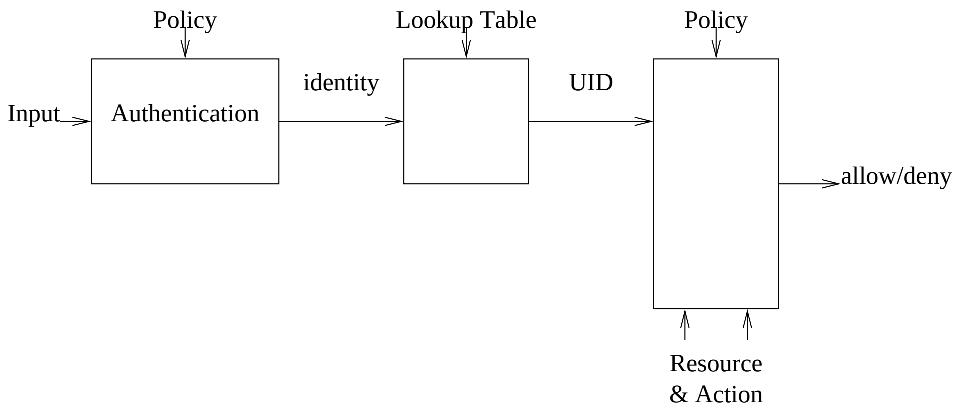


Figure 6.3: Authorisation with a Lookup Table for Authentication

authorisation. In most implementations, the list of groups for a given user is directly determined based on the results of the authentication. An example is shown in Figure 6.4, where groups are identified by a GID (group identifier).

Both MIT[44] and the Heimdal[54] implementations of Kerberos implement a variation of the theme, where the user $U$ submits the UID with the request. The server $S$ checks the Kerberos identity to see if that the user $U$ really is allowed to log in as UID, based on a lookup table.

Windows 2000 is similar, except that SID (security identifiers) are used[65].

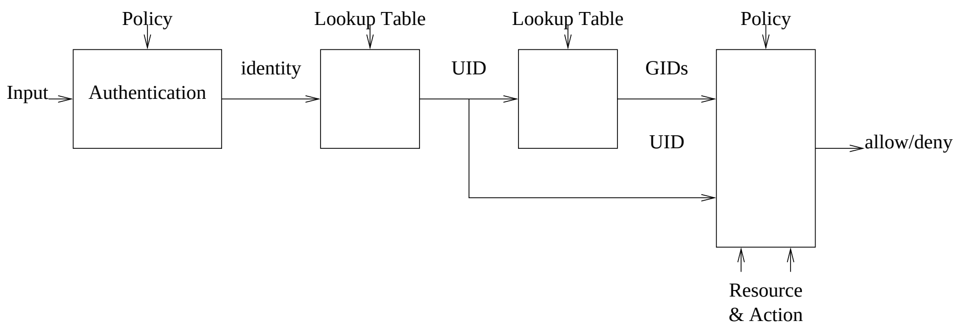


Figure 6.4: Authorisation with Lookup tables for Groups

All of these methods have limitations. Sometimes users require temporary access to a resource, for instance when copying a file from one user's account to another user's account. This is not possible without allowing everybody access to the file unless one user gives his/her password to the other user. Sharing passwords around in this manner is bad, because it makes changing the authorisation policy difficult (for instance, denying a user access means a new password needs to be distributed to all authorised users).

Flexibility of the authorisation protocol is important, as otherwise it may not be possible to implement an organisation's security policy exactly, leading to simplifications that reduce overall security. For example, Unix does not allow sharing of files without the help by the system administrator. For this reason, many people share passwords in order to work around this limitation, hence reducing security of their accounts.

## 6.2 Kerberos

Kerberos was not designed to be an authorisation protocol, but the result of the Kerberos authentication can be used as the input of the Authorisation process. See Figure 6.4 for an illustrated example of how this is often done.

In this system, authorisation decisions are made purely by the information given in the standard Kerberos ticket, the user's principle. See Figure 6.5 for an example. This is inflexible, as the resource must have a list of users that are allowed to access it.

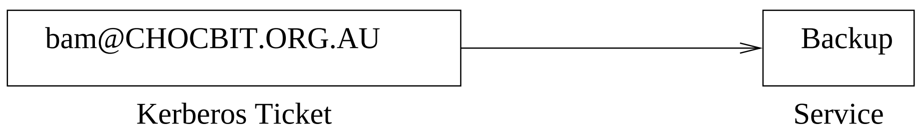


Figure 6.5: Authorisation with Kerberos

## 6.3 Roles

A role is a function or a position. An authorisation system can be modelled as a set of roles. Each user is assigned to one or more roles, and each role is mapped to a set of privileges that are required for the user to perform the functions of that role. A role is similar to a group, except roles are usually based on the functions that users need to perform, where as groups are simply a number of users that have been grouped together for convenience.

Assigning roles[1] to users is a considerable improvement. Roles can be given to individual resources. For instance, a role could be given to a shared project. Anyone who should have access to the shared project is assigned to the appropriate role.

Roles can be implemented in different ways, for example, using groups in Unix. However, this is restricted, because the Unix groups are statically assigned based on a lookup table when the user logs in.

## 6.4 Extensions to Kerberos

So far, most protocols (eg. Windows 2000[65], SESAME[66], and DCE [57]) that claim to add authorisation to Kerberos, do so by adding an extra field, typically known as the Privilege Attribute Certification (PAC) to the authentication ticket. This lists which lists groups that the user may belong to.

These groups contain any special privileges (groups or roles) that the user may have, eg. backup access, access to adding new users, access to shared Project X, etc. See Figure 6.6 for details.

However, this method has a number of serious limitations:

- Users cannot distribute authorisation to other users, without transferring their identity, too. Instead, authorisation has to be maintained at the server, $S$. This is a problem, because policy may dictate that au-

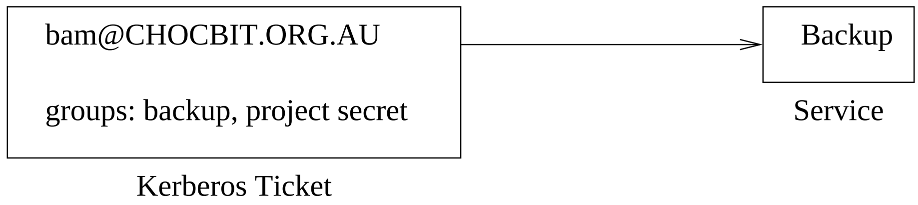


Figure 6.6: Authorisation with Kerberos Extensions

thorised users need to delegate access to other users. Proxiable tickets, under Kerberos 5[9], solve this problem to some degree, but requires contacting the KDC to obtain a separate proxy ticket, for each remote service.

- It is not possible to have a privilege assigned independently from the authentication. This means that privileges are centrally adminstrated from the KDC, and cannot be distributed by end users.

- Anonymous transactions are not possible. Anonymous access is often desirable, because users do not want their actions tracked. For instance, in order to eliminate unsolicited E-Mail, users sometimes choose to send E-Mail anonymously, via an anonymous Re-mailer. This server, could be setup in such a way to that it only allows users having the correct privilege to send anonymous E-Mail, but not track the users identity. User's can feel safer that such a system is secure if they don't have to submit their identity but only proof of the required privileges.

  Sometimes a compromise is required: users are initially anonymous, but can be identified at a later date, eg. for identifying criminals[5].‘

- Similarly, it is not possible to log on to a remote system without admitting to that system what your privileges are. Some of these privileges may have no relevance to the remote system, but could be used by the remote system for undesirable reasons. For example, if a user has access to a bank account, there is no reason why this should be revealed when that user tries to access computer time from another provider.

- When capability[23] based operating systems become more common, it will be possible (in theory) to assign different sets of authorisation to different processes, without forcing the user to log in multiple times. This means, if for instance, you do not trust program X (perhaps it was downloaded from an untrusted website, or received via untrusted E-Mail), then you can ensure program X does not have access to given authorisation certificates.

A better scheme would be to use the authentication ticket for authentication, nothing more, and nothing less.

## 6.5 Delegation

Delegation allows an authorised person to tranfer authorisation rights to somebody else[14], typically for a limited period of time.

Delegation can be split into two ways[67]:

- **Delegation of rights**: Allow another user to access a resource. This is how Kerberos proxiable tickets[9] work. One user can "pretend" to be another user for the duration of the ticket.

- **Delegation of responsibility**: Allow another user to access a resource, but do it in such a way that he/she is held accountable for their actions. That is, if X delegates responsibility to Y, and Y fails to comply, there is no way that Y can blame X for it.

Delegation is important as it means that the service does not need to know in advance everybody who may have access.

A primitive example of delegation is proxiable tickets, as defined in Kerberos (as distinct from the proxy authentication ticket proposal described earlier in this thesis). This allows a user to obtain a Kerberos authentication ticket with restricted priviliges that can then be given to third party services (eg. a printer service which requires direct access to the users files). This is restrictive though, because it is only delegation of rights, and not delegation of responsibility, as the user using the ticket cannot be identified.

## 6.6 Keynote

Keynote, proposed by Blaze et al[68], is a trust management system that supports delegating responsibility via certificates. The authors claim that keynote is simpler and easier to implement than alternatives.

However, Keynote is simple because it does not have any authentication system to verify public keys. Rather, it relies on public keys to be validated via some other mechanism. As such shortcuts are likely to be taken[12] (see Section 2.3, page 18), resulting in the incorrect users being authorised access to privileged resources.

This problem is addressed when using a naming system like the Scalable Public Key Infrastructure standard (SPKI). See next chapter for details.

## 6.7 Example

This section is describes an example that demonstrates some of the key concepts found in this chapter.

Consider a university setting, with the following protected resources: Printers, student's name, address, private work and marks.

With Amy, Bob, and Charles for students in Project 1; David, Eugene, and Frank for students in Project 2; Kate, Luke, and Mike for university lecturers; Ursula is the course coordinator; Victor is a senior member of the administration staff. There are three subjects: Maths, English, and DSP (digital signal processing), which are taught by Kate, Luke, and Mike respectively. Frank also tutors Amy and David in English. The list of students and what subjects they take is shown in Table 6.1.

| *Student* | *Subjects* |
|---|---|
| Amy | Maths, English, DSP |
| Bob | Maths, DSP |
| Charles | Maths, English |
| David | Maths, English, DSP |
| Eugene | Maths, DSP |
| Frank | Maths, English |

Table 6.1: Example: Students

This example is based around resources required for students, and does not attempt to deal with all issues that could arise in the given scenario.

These resources would need to be protected, according to a policy. For instance, consider the policy given in Table 6.2.

| *Resource* | *Who* | *Access* |
|---|---|---|
| Address | The student. | `R-` |
| | The student's subject's lecturers. | `R-` |
| | Any administration Staff. | `RW` |
| Private Work | The student. | `RW` |
| Group Work | The Group. | `RW` |
| Marks | The student's subject's tutor. | `RW` |
| | The student's subject's lecturer. | `RW` |
| | Other lecturer. | `R-` |
| Printers | 10 cents a page, pre-pay | |

Table 6.2: Example: Security Policy

There a number of ways this authorisation could take place. These methods are described in the following sections.

### 6.7.1 Password Based Authentication

Each resource has two passwords (one for read-only and one for read/write), and the users are expected to remember the password for each resource. This is the approach taken by Windows 95 and Windows 98 when sharing files via the network.

However, people cannot be held accountable for their actions, and each user would have to remember a long list of passwords. This is clearly not scalable. This scheme is also useless for the printers, as there is no way to know who should be charged for using the resource.

See Table 6.3 for an example.

### 6.7.2 Conventional Access Control List

Access Control Lists (ACL) are supported by Windows NT, Windows 2000, and currently being developed for Linux based operating systems. There are several ways that an ACL can be used in this example.

- Each resource has a list of users who are allowed to access it. An example of this scheme is given in Table 6.4. However, this list of user's would have to be maintained centrally for the owner of each resource, and would not support delegation. This seriously limits scalability, as any errors in updating the lists (eg. forgetting to update some resources) could lead to security breaches.

- Each resource can be controlled to a separate group. However, the number of groups required would be enormous. Each user would require at least 9 groups (read-only address, read-write address, read-only private work, read-only marks for each subject, read-write marks for each subject), and each group project would require a group (read-write access). These groups must be maintained by the system administrators.

| *Resource* | *Password* | |
|---|---|---|
| | `R-` | `RW` |
| Amy's Address | S9S41q1h | S9GmrPw |
| Bob's Address | EICz6zs | BqiLg5b |
| ⋮ | | ⋮ |
| Frank's Address | sRrTLy | MCm9gf |
| Amy's Private Work | | VC8DNGwD |
| Bob's Private Work | | wIuGF0b |
| ⋮ | | ⋮ |
| Frank's Private Work | | scPwnjSv |
| Project 1 | | PwScqxyJ |
| Project 2 | | meXdeo |
| Amy's Marks (Maths) | novytw | q5dPGe1 |
| Bob's Marks (Maths) | 8HGiaf | FjN15oXf |
| ⋮ | | ⋮ |
| Frank's Marks (Maths) | ISpAcmPF | SpGGA63 |
| Amy's Marks (English) | 3m6yaga | eqbm3IxF |
| Bob's Marks (English) | NVUsscY | d0y2mF |
| ⋮ | | ⋮ |
| Frank's Marks (English) | YK7b9oG | b9jgnCE1 |
| Amy's Marks (DSP) | YGtqyAQj | 94aarTp |
| Bob's Marks (DSP) | f2dNiox | D5LbDvz |
| ⋮ | | ⋮ |
| Frank's Marks (DSP) | T5z8c4T | FXdnzJPt |

Table 6.3: Example: Password Based Authorisation

| *Resource* | *Users* | |
|---|---|---|
| | R- | RW |
| Amy's Address | Amy, Kate, Luke, Mike | |
| Bob's Address | Bob, Kate, Mike | |
| ⋮ | ⋮ | |
| Frank's Address | Frank, Kate, Luke | |
| Amy's Private Work | | Amy |
| Bob's Private Work | | Bob |
| ⋮ | ⋮ | |
| Mikes's Private Work | | Mike |
| Project 1 | | Amy, Bob, Charles |
| Project 2 | | David, Eugene, Frank |
| Amy's Marks (Maths) | Luke, Mike | Kate |
| Bob's Marks (Maths) | Luke, Mike | Kate |
| ⋮ | ⋮ | |
| Frank's Marks (Maths) | Luke, Mike | Kate |
| Amy's Marks (English) | Kate, Mike | Luke, Frank |
| Bob's Marks (English) | Kate, Mike | Luke |
| ⋮ | ⋮ | |
| Frank's Marks (English) | Kate, Mike | Luke |
| Amy's Marks (DSP) | Kate, Luke | Mike |
| Bob's Marks (DSP) | Kate, Luke | Mike |
| ⋮ | ⋮ | |
| Frank's Marks (DSP) | Kate, Luke | Mike |

Table 6.4: Example: User Based ACL

This example would look similar to Table 6.4, except that each cell would be replaced with a unique group name.

- Alternatively, roles could be assigned to different users. However, this will quickly look very similar to the previous setup. While some roles would not be required (eg. read-write address could be replaced by a role for all students), this is at the expense of flexibility. It would be difficult to give an administrator temporary privilege to alter only the address of one student.

  This example would look similar to Table 6.4, except each cell would be replaced with a role name. Unlike in the previous case though, roles can be shared between resources which are similar (eg. all marks for maths could belong to a distinct role).

## 6.8 Summary

This chapter has demonstrated that existing techniques of authorisation schemes are clearly insufficient for large scale networks, for many different reasons, such as no support for delegation of rights or responsibilities, no allowances for privacy, and inflexible to the requirements of the site's security policy.

These issues are largely due to the requirement that the resource must contain one or more, potentially large, centrally administrated, tables for controlling access. The next chapter, describes a potential solution, SPKI.

# Chapter 7

# SPKI

The previous chapter described existing methods of authorisation that have been implemented in programs like Kerberos, Unix and various versions of Windows. This chapter describes a better authorisation system: SPKI, the state-of-the-art access control protocol. It also describes three extensions to the standard which enhance its usability in large scale networks, such as the Internet.

The draft SPKI certificate structure standard has been included in Appendix A.

## 7.1 SPKI

As shown in Figure 7.1, SPKI[3] has two main inputs to aid the authorisation process, namely name certificates and authorisation certificates. These certificates are a standard set of S-expressions that have been digitally signed. This allows the server to prove that the authenticity of each certificate is valid, and that no-one has tampered with it along the way (non-repudiation).

Each name certificate binds a single user to a name or a group. This is similar to other standards that use certificates for authentication, eg. the ITU X.509 standard, and PGP.

Authorisation certificates, in contrast, bind a privilege to a given user or a given group. This is done with the tag expression, which is a set of attributes that are understood by the server as specific privileges to give the user. For instance, the tag expression `(tag (address (* set read)))` might be interpreted as allow read access to the address field of the database.

If the authorisation certificate allows delegation, then any of the valid users may delegate his/her rights to another user or group.

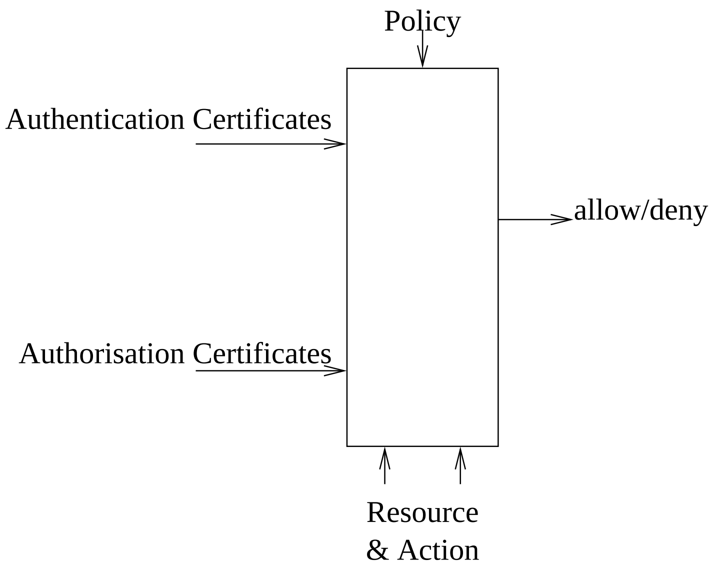


Figure 7.1: Authorisation with SPKI

Most importantly, name certificates allow users to be identified by a trusted third party without requiring a global scale certification authority. For instance, a university could set up a university wide naming service that identifies students and staff inside the university. This would mean individual staff do not need to verify students but can trust the university wide naming service instead.

Name certificates could also be used as roles. One user can belong to a role without needing to know what other users become part of the role. However, these certificates (unlike authorisation certificates) need to be issued by the owner of the group.

SPKI is very flexible in that users can create and sign certificates for groups of people and/or allow delegation of privileges. See the next section for an example based on a real life situation. A copy of the SPKI standard[14] has been included as Appendix A.

SPKI, allows for delegation of responsibilities. This is done by requiring [69] an authorisation certificate to contain a subject field. Only the person holding the private key corresponding to the public key in the subject field may use the authorisation certificate, hence ensuring that this user is accountable for their actions.

While the SPKI standard is based on S-expression syntax[69], a standard also exists that permits XML encoding of the certificates[70]. Only the basic S-expression syntax will be used in this thesis, simply because the XML expression syntax is more verbose. While this is normally an advantage (ie. easier to validate), here it is a disadvantage (ie. uses more paper).

SPKI (similar to Keynote) is based on certificates. While it suffers from the same problems as Keynote does (eg. end users likely to take shortcuts that compromise security), SPKI supports name certificates, so that a user can trust a certificate that is signed by another trusted user. This means that the server does not need to manually verify each certificate that is to be used (this requires a face to face meeting with the owner).

## 7.2 Asymmetric Keys

SPKI is a protocol that uses asymmetric authentication for both authentication and authorisation.

This means, SPKI has similar problems to other protocols that use asymmetric keys. Users have to be trusted to manage private keys in a sensible manner, and must be careful trusting (ie. signing) a key from an outside source, unless they are sure they know who the real owner is.

It could be argued that this is not authentication, but a form of *authorisation*. That is, the holder of the private key gets the privileges regardless of his/her identity, meaning no-one can be held accountable for the misuse of the privileges.

However, SPKI only requires the private asymmetric key in order to provide non-repudiation. Alternative methods of achieving this are possible, and have already been mentioned in Chapter 5.

Distribution of public keys is still a problem. The use of SPKI does not change that. However, SPKI is designed to be more distributed than any other system described in this thesis.

This means, for instance, that organisations can issue certificates for users within the organisation. Each organisation can have their own guidelines for how much verification is required when signing the certificate.

While this is possible using other systems, eg. neither PGP or SPKI use a global name space. Instead, users are identified via a chain of signatures (eg. 'Ka Henry Brian' would refer Brian's key that has been signed by Henry's key. Henry's key has been signed by the Ka key). Different users can have separate keys, without having to worry about assigning each key a unique global name. This means that different organisations can allocate the same person different keys and not have to worry about assigning the key a unique name.

However, a new problem here is that only one chain of paths is allowed, so if Henry started signing fraudulent keys, there would have no alternative method of validating the key, even if 'Ka X Brian' was another trusted path to the final key. This issue is addressed below.

The use of local name space also means that the technique described earlier on scalable public key distribution cannot be used, as this technique relied on each key having a unique and meaningful name within a global name space.

## 7.3 Limitations

There are at least three limitations with SPKI:

- Asymmetric authentication is the only authentication allowed. This means that there is an extra step required (asymmetric authentication) which is redundant if a good authentication system already exists (eg. Kerberos), especially for users who do not need to sign their own certificates.
- As mentioned in Section 7.2, on Page 78, only one chain of signatures can be used. This is a serious limitation, as there is no way to confirm that the signature provided by the single source is correct.
- Some of the suggested authorisation certificates[2] (or rather the tag field within the authorisation certificate) are very specific to the protocol being used. While this means that tags can be application specific, it has the disadvantage that the issuer of the ticket must know in advance what protocol the user will use to access the resource.

## 7.4 Authentication

It is desirable to be able to use SPKI in conjunction with another authentication protocol, eg. Kerberos. This is easy to achieve, by having a name certificate bind a name to a Kerberos principle instead of a public key. For example, instead of having a name certificate bind to the hash of the user's public key[14]:

```
(cert
  (issuer (name (hash sha1 <...>) brian))
  (subject (hash sha1 <brian's key>)))
```

The certificate would refer to the user's Kerberos realm instead:

```
(cert
  (issuer (name (hash sha1 <...>) brian))
  (subject (kerberos victor@MONASH.EDU.AU)))
```

While this is converting the global name space into a local name space, it is very flexible as it allows the local user to trust only given principles.

Alternatively, a trusted third party could be set up that automatically issues certificates for a given Kerberos principle. The server, $S$ would then be able to get a certificate for an unknown remote user without any prior contact. This third party service can issue the certificate using the Kerberos principle as the name, and the server $S$ would refer to the key using the third parties' public key.

However, these name certificates could not be used as an intermediate step for other name certificates, because this would require a public key in order to sign the certificate.

## 7.5 Multiple Chains

It is also desirable to have multiple chains leading up to the final destination. This would mean there is some redundancy in the chains, yielding greater security.

This typically can be done by changing the format of the name expression. The standard format is:

```
(name (hash sha1 <key>) <user1> <user2> <user3>)
```

where the key is the initially trusted public key, to find user1's public keys, which is used to find user3's public key. This format does not support multiple chains. A better format would be

```
(name (chain (hash <key1>) <user1> <user2>)
      (chain (hash <key2>) <user3> <user4>))
```

where two chains exist: (key1 user1 user2) and (key2 user3 user4) which must produce an identical result (eg. an identical public key or an identical Kerberos principle) before it can be trusted. This name could exist anywhere a name can normally appear, eg. in a name certificate or an authorisation certificate, to allow for maximum flexibility.

This has the tradeoff that both chains must produce the same result. If for instance, user1 updated the key for user2, user3 would have to do the same. Otherwise, the above expression would break. At least, in this case, you would minimise your chances of using a false key.

## 7.6 Tag Expressions

A better system for tag expressions would be to name the resource (eg. project1) rather than the protocol (eg. ftp, telnet, rsync, or SSH). In this case, some sort of mapping is required to map the name in the tag expression to the name of the physical resource (eg. `/home/projects/project1/*`).

The mapping required may depend on the rest of the tag expression, for example you could specify `(* set source-only)` which could limit the previous expression to `/home/projects/project1/*.c`.

Ideally, this mapping would occur after the ACL (access control list) is processed, so that the ACL can specify restrictions in the tag expression that get mapped to a restriction in what physical resource can be accessed.

## 7.7 Example

The best approach to the example given in the previous chapter is to use a decentralised SPKI based system. Each user is registered in the Kerberos system. The Kerberos system allows users to create SPKI certificates using the methods in Chapter 5. User's can be assigned authorisation certificates that allow access to resources.

The address database could be setup to only allow access by Victor. The following ACL for the database would achieve this:

```
(acl
  (entry
    (hash sha1 <Victor's key>)
    (propagate)
    (tag (address (* set read write))
         (subject (*))
         (student (*))))
)
```

Victor could delegate restricted rights to the university lecturers. Victor would create the following certificate, and give it to Luke. This would give Luke read-only access to the addresses of any students doing English – obviously the address database would also need to know what subjects each student is enrolled in. While each student could also be named individually inside the certificate, this method has the advantage that students may change subjects without invalidating the certificate.

```
(cert
  (issuer (hash sha1 <Victor's key>))
  (subject (hash sha1 <Luke's key>))
  (propagate)
  (tag (address (* set read))
       (subject (* set English)))
)
```

If the Public Key Protected File System[2] (PKPFS) protocol was implemented, the computer centre could also authorise Victor to allocate disk space to students. The computer centre could have the following ACL:

```
(acl
```

```
  (entry
    (subject (hash sha1 <Victor's key>))
    (propagate)
    (tag (pkpfs (* prefix "//ftp.uni.edu.au/students/"))
         (pkpfs-quota (* range le "50000")))
  )
)
```

Victor could delegate restricted disk space rights by creating and distributing a certificate like the following:

```
(cert
  (issuer (hash sha1 <Victor's key>))
  (subject (hash sha1 <Frank's key>))
  (propagate)
  (tag (pkpfs (* prefix "//ftp.uni.edu.au/students/frank/"))
       (pkpfs-quota (* range le "10000")))
)
```

Frank, in turn can delegate restricted rights to students in his project group, Project 2, by distributing certificates like the next one to each student in his group:

```
(cert
  (issuer (hash sha1 <Frank's key>))
  (subject (hash sha1 <David's key>))
  (propagate)
  (tag (pkpfs (* prefix "//ftp.uni.edu.au/students/frank/project2/"))
       (pkpfs-quota (* range le "1000")))
)
```

A similar setup could also be used for print quotas too. If, for example Frank had never used the printer before, but paid for 100 pages in advance, he would get the following certificate, issued by the computer centre's administrators:

```
(cert
  (issuer (hash sha1 <Administrator's key>))
  (subject (hash sha1 <Frank's key>))
  (propagate)
  (tag (printer (* set black-white))
       (printer-account "frank")
       (printer-quota (* range le "100")))
)
```

In order for this to work, the printer server would have to keep track of the total number of pages printed by the user. The certificate specifies what account this total number of pages is stored in. The above certificate would allow Frank to print pages until his total page count exceeds 100.

Frank could delegate this to his friends, for instance if somebody wanted to print something urgently but did not have the required print credit, with a certificate like this one:

```
(cert
  (issuer (hash sha1 <Frank's key>))
  (subject (hash sha1 <Amy's key>))
  (propagate)
  (tag (printer-daily-quota (* range le "2")))
  (not-before "2000-12-25_00:00:00")
  (not-after "2000-12-26_00:00:00")
)
```

One limitation with this scheme is that if Frank has already printed pages 1 and 2, the above certificate becomes useless. It would be better if the certificate contained the number of pages that could be printed, but implementing this correctly, in a scalable manner so that delegation is supported, may prove to be difficult.

The problem here is that a certificate might be only partially used (eg. 10 pages printed when the certificate allows printing of 20 pages), and the print server would some how have to keep track of the credit remaining for each certificate.

Finally, the marks database could be setup to allow only the subject coordinator access, using this ACL:

```
(acl
  (entry
    (subject (hash sha1 <Ursula's key>))
    (propagate)
    (tag (marks (* set read write))
         (subject (*))
         (student (*)))
  )
)
```

The subject coordinator could, in turn delegate restricted per subject rights to the lecturer of each subject. This could be done with the following certificate:

```
(cert
  (issuer (hash sha1 <Ursula's key>))
  (subject (hash sha1 <Luke's key>))
  (propagate)
  (tag (marks (*))
       (subject (* set English)))
)
```

Each lecturer in turn could delegate restricted per person rights to each tutor, for each student (note: it is assumed that the database has no knowledge of tutors and what students each tutor has, hence the need to specify each student).

```
(cert
  (issuer (hash sha1 <Luke's key>))
  (subject (hash sha1 <Frank's key>))
  (tag (marks (*))
       (student (* set Amy David)))
)
```

In practise, this key would have a restricted expiry date (not shown), since students changing tutors would invalidate the current certificate. Also, this certificate could be even more restrictive, so the tutor can only change marks for a given project. However, in this case the marks database needs to know what projects exist.

This solution has been deliberately simplified, and does not make use of certain features, for instance expiry dates on certificates or named certificates. In a real situation, both of these should be used extensively.

The use of expiry dates would depend on university policy, and is a trade-off between how often the certificate should be re-issued and the risk involved if somebody misuses it.

Name certificates should be used extensively too. For instance, the following name certificates could be used:

```
(name (hash <department's key> ...) victor)
(name (hash <university's key> ...) victor)
```

So that ACL's no longer need to refer explicitly to Victor's key. Instead, references to:

```
    (hash sha1 <Victor's key>)
```

can be changed to

```
(name (chain (hash <department's key>) victor)
      (chain (hash <university's key>) victor))
```

or, if Kerberos is used, this could be used instead:

```
(name (kerberos bam@CHOCBIT.ORG.AU))
```

## 7.8 Summary

SPKI is a sophisticated protocol for distributed authorisation. It allows access control delegation, which allows it to be used in large scale environments. It also discards the concept of the global name space, which some say is not achievable or even required[14].

However, limitations exist in SPKI. SPKI does not allow usage with other authentication protocols, nor does it allow for multiple chains to refer to the final user's key. Also, the examples in [2] suggest that authorisation tickets should have the protocol names hard-coded. This makes the SPKI certificates needlessly very protocol specific.

Solutions were proposed to allow other authentication protocols to be used (in this case Kerberos) and allowing multiple chains of signatures. These involved minor changes to the S expression grammar used by SPKI. An example is also given where SPKI certificates do not have to hard-code protocol specific information.

# Chapter 8

# Evaluation

This chapter evaluates the proposals made in the previous chapters.

## 8.1 Authentication

Some features of the protocols that are used for the formal criteria:

**Scalable key distribution.** Most protocols assume that the keys have already been distributed, without attempting to address the problem. This task differs slightly depending on whether private/secret inverse keys or public keys require distribution. Private/secret keys must be kept secret at all times. Public keys can be made public, but must be protected to to prevent alterations.

Even though public keys can be signed to form secure certificates, this is not a statisfactory end solution, as some means is still required to ensure that the issuer of the certificate signs the correct key.

None of the protocols described within this thesis have a scalable method of distributing asymmetric keys, except for the proposed scalable key distribution protocol. Even this is limited though, because it is not suitable for distributing SPKI asymmetric keys (as SPKI does not use a global name space) or private keys used by Kerberos.

**End-user cache.** Most protocols support caching of authentication information per each user, without having to request it each time. This means slightly different things for each protocol. Asymmetric key protocols can cache the public key (as per scalable key distribution), at the remote server. Kerberos tickets are cached locally by each client. This cache should automatically expire after a preset time or condition, ie. caching passwords is not good enough, as a stolen password may be used indefinitely.

All authentication protocols discussed can support this to some degree, but only the password protocol and biometrics are unable to do it securely (ie. the cached data can be used indefinitely if stolen).

**Non-repudiation.** Non-repudiation, could be added to any of the protocols, using a similar method to what was proposed in this thesis. Non-repudiation is important for SPKI (see Chapter 7).

The different protocols, and how well they meet the formal criteria (section 1.4, page 9), are shown in Table 8.1. For rating the facility of use and robustness, it is assumed that all the keys have already been distributed. This is because the lack of a good system for distributing keys already rates poorly for scalability.

### 8.1.1 Password Based Authentication

Password authentication fails all the criteria for scalable authentication methods. It is not *scalable*, as the more systems a user has access to, the more passwords he/she needs to memorise.

It can be *secure*, but often is not, because of insecure transmission links, or abuse from users (eg. not choosing a good password). It does not feature *facility of use*, as users must remember a separate password for each computer. Abuse is easy (eg. by choosing weak, easy to remember passwords). It is not *robust*, as once a password has been stolen, it may not even be immediately obvious that the password has been stolen.

Remote users can be authenticated, but all links in between need to be trusted, especially if encryption is not used. Support for disconnected computers is also possible, but only if you trust people with physical access not to try and apply dictionary attack methods to work out passwords for other people (depending on how many people have access to the computer).

### 8.1.2 Biometrics

Using Biometrics for authentication can be *secure*, features *facility of use*, and mistakes are hard to make (assuming database is kept accurate) making it *robust*. It is not *scalable* (as a secret database is required).

However as previously discussed biometrics, cannot be used for general remote authentication or for disconnected use.

### 8.1.3 Asymmetric Authentication

Asymmetric authentication methods (eg. SSH) are not *scalable* as no mechanism has been defined for securely transferring the public keys[17]. Assuming the keys have been correctly transferred in advance, it is *scalable*. It is not *secure* as end users need to manage their own keys.

Once initially setup, asymmetric authentication methods feature *facility of use*, as serious misakes are difficult. However, making asymmetric authentication protocols *robust*) is difficult, as there is no easy way to check or revoke public keys should a problem arise.

Asymmetric authentication, by itself, cannot be used for local authentication, because some form of trusted storage space is required to store the secret inverse key. Unlike passwords used for symmetric authentication, secret inverse keys cannot be memorised, because the length of a asymmetric key is usually considerably longer than that of the symmetric key.

### 8.1.4 Kerberos

Kerberos is *secure* and features *facility of use* (there is no need to take shortcuts with passwords, as only one is required). It is forgiving of most mistakes (*robustness*, due to the expiry time on tickets), however, the KDC must be kept secure. If the KDC is not kept secure then all other computers in the realm controlled by the KDC could be compromised. Kerberos, as defined in the standard[9], is not *scalable*. Nor can it be used securely on off-line computers.

There are a number of extensions for Kerberos. Some of them are important, where as others are questionable. However, even with these extensions, some issues still remain. PkInit requires end users manage asymmetric keys, and PkCross requires but does not provide a scalable key distribution technique.

### 8.1.5 Smart-Cards

Smart cards are probably the best system for authentication, as it is impossible to steal a user's identity without stealing the card. With further enhancements (eg. authentication of the user to the smart card first), then even this risk is minimised.

Smart cards are not a final solution to access control or authentication. Rather, they must be used in conjunction with another protocol (eg. asymmetric authentication). While the problem of storing confidential user information for authentication is solved, other issues still remain (eg. distributing public keys). For this reason, smart cards have not been included in Table 8.1, as it depends on what protocol they are used with.

There is no single standard that can be used for smart-cards, nor is there wide deployment of smart card-readers. Both of these issues must be solved before smart-cards can become a general solution for scalable access control.

### 8.1.6 Proposed

The proposed method uses Kerberos for authentication and non-repudiation and SPKI for authorisation.

Kerberos with PkCross extensions with a method for scalable key distribution. Hence it meets all the desired criteria. Kerberos can be used with SPKI, either by allowing SPKI naming certificates to support Kerberos, or by allowing Kerberos to support features of non-repudiation, as previously discussed.

SPKI supports delegation, eliminating the need for a large centrally administrated ACL. It doesn't require end users manage their private keys. These mean that SPKI has *scalability*, *security*, *facility of use*, and *robustness*.

| | *Security* | *Scalability* | *Facility of Use* | *Robustness* |
|---|---|---|---|---|
| Password | $\surd$ | $\times$ | $\times$ [a] | $\times$ [b] |
| Biometrics | $\surd$ | $\times$ | $\surd$ | $\surd$ |
| SSH | $\times$ | $\times$ | $\surd$ | $\times$ |
| Kerberos | $\surd$ | $\times$ | $\surd$ | $\surd$ |
| PkInit | $\times$ | $\times$ | $\surd$ | $\surd$ |
| PkCross | $\surd$ | $\times$ | $\surd$ | $\surd$ |
| Proposed | $\surd$ | $\surd$ | $\surd$ | $\surd$ |

Table 8.1: Comparison of Authentication Methods

[a] requires memory skills

[b] passwords must be kept secret

## 8.2 Authorisation

Some authorisation protocols suffer from similar security problems as the authentication protocols. For instance, password based authorisation systems, such as Windows 98, require the user remember a separate password for each service. This prevents the protocol from being easy to use. Also, there is no scalable method of transporting passwords for each service, hence limiting scalability of the protocol.

Most authorisation protocols rely on a separate authentication system, such as SPKI which uses name certificates for authentication. Hence the above problems are not an issue (assuming that the separate authentication system is secure).

However flexibility is an important issue. If the authorisation protocol is not flexible, then a given security policy for a site can not be effectively implemented, without making compromises that limit security of the overall site.

The most flexible protocol described here is the SPKI protocol. It is flexible in that the resource does not need to have any prior knowledge of who may have acess. Instead. features such as issuing of certificates, delegation of rights, and restrictive tags allow for decentralised access control.

# Chapter 9

# Conclusion

This chapter summarises the results and contributions to scalable acess control.

The main contributions of this work are:

- Fast cross realm authentication, as presented in Chapter 3, demonstrates how inter-realm information can be cached in order to speed up inter-realm ticket requests. This was published in [17].

- Scalable public key exchange is required by any protocol that uses or requires public keys. An example of how this could be achieved is given in Chapter 4. This was published in [11]. This issue is often neglected, as protocols expect users to come up with some external solution.

- Non-repudiation is important to state-of-the-art authorisation protocols. Chapter 5 shows that non-repudiation can be provided for even in authentication systems that do not use public keys, such as Kerberos. A selection of different solutions are described.

- SPKI (an access control protocol) has a number of limitations. Chapter 8 lists proposed enhancements to SPKI, which allow SPKI to be used with Kerberos, to use multiple chains of signatures, and to be generalised so that certificates do not need to be specific to each protocol.

In performing this work, I found that major issues of security are often overlooked or ignored. Some of these have raised potential security problems, eg. no satisfactory framework for securely distributing public keys for asymmetric authentication and authorisation systems.

While satisfactory methods of authentication already exist, such as Kerberos, none of these are currently scalable to very large networks such as the Internet, nor does Kerberos deal with issues such as Non-Repudiation.

Kerberos is a state-of-the-art authentication protocol. It authenticates users by granting them a ticket, which acts as proof of identity when passed to a server. However Kerberos by itself has issues that limit its scalability in large scale networks.

A number of extensions have already been proposed in attempt to deal with this problem. These include symmetric encrption (PkCross and PkInit) and the use of smart cards.

PkInit adds asymmetric authentication capabilities to Kerberos for end-users[52]. While this is an interesting idea, and is required for the PkCross proposal, it suffers from all of the problems (see Section 2.3, page 18) that asymmetric authentication schemes suffer, without adding any significant benefits.

PkCross is a proposed extension to Kerberos that allows it to use asymmetric authentication for cross realm authentication while continuing to use asymmetric authentication for end users[35]. As end users do not need to worry about administration of the public keys, the compliance defects[12] (see Section 2.3, page 18) are not longer relevant.

A better approach for authentication is to use smart cards[51]. With a smart card system, it is easy to determine if a smart card has been lost or stolen. If the card is not lost or stolen, unauthorised use is not possible.

However, the issues of scalability still remain unresolved, as there is no scalable method of distributing private symmetric keys used by Kerberos or public asymmetric keys used by its extensions, such as PkCross and PkInit.

The proxy authentication ticket solves the problem of distributing private Kerberos keys, by allowing a KDC to grant tickets for other realms without having to contact the remote KDC for each ticket issued.

The scalable key distribution solves the problem of distributing public asymmetric keys, by providing a protocol which attempts to automate this tedious task.

Finally, access control also requires an authorisation scheme. SPKI is a flexible access control protocol (ie. it covers both authentication and authorisation). In the conventional implementation, a central administrator (sometimes known as the owner of the resource) needs to set up the ACL, which specifies who is allowed to access what resource. This means that the administrator must keep track of when to add new entries (eg. new staff member), and when to delete old (eg. staff member who got fired). This also implies that the central administrator must always be informed of any changes of status in staff. In a big company with possibly thousands of staff, this could get tedious. It also requires significant processing power (eg. a smart card based door lock) just to store and process the long ACL list.

SPKI provides an alternative. The central ACL only needs to contain one entry (eg. CEO is allowed access). This entry would in turn delegate responsibility (eg. middle manager is allowed access), all the way until it reaches the individual employers. Each boss would be responsible for granting authorisation certificates to his/her employees, removing the need to do this centrally.

An improved version of SPKI is shown that allows the flexibility of SPKI but without compromising the security. It is shown how to make SPKI use

Kerberos, multiple chains, and a better naming system for tags.

However, SPKI is not useable without some form of non-repudiation. In the long term, the best solution may be use of normal asymmetric keys stored on secure smart cards. However, until smart card readers are widely deployed, and good standards are developed, use of asymmetric keys means users have to maintain their own asymmetric keys. As discussed, this will cause security problems as users can take shortcuts with key management.

Kerberos can also provide the level of non-repudiation required, as discussed. It is demonstrated that non-repudiation can be achieved securely on a large scale by combining the previous proposals together. While certain tradeoffs were made in the design process that could potentially decrease security, this design should increase overall security.

# Appendix A

# SPKI Certificate Structure

This appendix is quoted from the Internet draft standard[69].

# Appendix B

# Glossary

**API** Application Programming Interface — a standard library of routines that can be called by an application.

**authentication** "The process of reliably determining the identity of a communicating party."[19]. See section 1.2, page 6.

**authorisation** Grant access to authenticated user to restricted resources and services. See section 1.2, page 6.

**CA** Certification Authority — sign public keys to allow unlimited and safe distribution.

**GnuPG** GNU Privacy Guard — Open source asymmetric encryption program. See also *PGP*.

**GSSAPI** Generic Security Service Application Program Interface — Abstract API for implementing authentication.

**KDC** Key distribution centre — distributes secret keys.

**non-repudiation** The property where you can prove to a third party who sent a message[19], perhaps for legal reasons[13].

**nonce** "A number used in a cryptographic protocol that must (with extremely high probability) be different each time the protocol is run with a given set of participants in order to ensure that an attacker can't usefully inject messages recorded from a previous running of the protocol. There are many ways of generating nonces, including suitably large random numbers, sequence numbers, and time-stamps,"[19].

**PAS** Proxy Authentication Service[11] — allows a KDC to act as a proxy for the KDC at the remote realm and issue tickets on its behalf. Not to be confused with proxy tickets, which can be given to a server to allow it to act on the user's behalf. See also *PAT*.

**PAT** Proxy Authentication Ticket[11] — allows a KDC to act as a proxy for the KDC at the remote realm and issue tickets on its behalf. Not to be confused with proxy tickets, which can be given to a server to allow it to act on the user's behalf. See also *PAS*.

**PGP** Pretty Good Privacy[46] — Asymmetric encryption program. See also *GnuPG*.

**PkCross** Extension to Kerberos for cross realm asymmetric key authentication[35].

**PkInit** Extension to Kerberos for asymmetric key authentication of end users[52].

**SASL** Simple Authentication and Security Layer — Abstract API for implementing authentication.

**SPKI** Scalable Public Key Infrastructure[14]

**SSH** Secure Shell[20]. A program similar to telnet, but supports asymmetric authentication. See also *telnet*.

**SSL** Secure sockets layer. An encryption standard commonly used for HTTP transfers. HTTP is beyond the scope of this document. The latest version of SSL is called the Transport Layer Security (TLS)[36].

**SSL-telnet** A version of telnet that encrypts data using SSL. See also *telnet* and *SSL*.

**telnet** A program[33] to run text based programs on a remote computer as if it were done on the local computer.

**TGS** Ticket Granting Service — Kerberos service to allow a client to obtain a ticket for any other service without requiring the user to re-enter a password.

**TGT** Ticket Granting Ticket — Kerberos ticket to use the TGS service. See also *TGS*.

**Thesis** The document you are reading.

**UID** User Identifier

**XML** Extensible Markup Language

# Bibliography


[1] A. Herzberg, Y. Mass, J. Mihaeli, D. Naor, and Y. Ravid, "Access control meets public key infrastructure, or: Assigning roles to strangers," in *1000 IEEE Symposium on Security and Privacy*, (Berkeley, California, USA), pp. 2–14, IEEE Computer Society, May 2000.

[2] C. M. Ellison, B. Frantz, B. Lampson, R. Rivest, B. M. Thomas, and T. Ylonen, "SPKI examples," Internet-Draft draft-ietf-spki-cert-examples-01.txt, Internet Engineering Task Force, Mar. 1998. Work in progress.

[3] A. J. Maywah, "An implementation of a secure web client using SPKI/SDSI certificates," Master's thesis, Massachusetts Institute of Technology, May 2000. `http://www.toc.lcs.mit.edu/~cis/sdsi.html`.

[4] D. Davis, "Workstation services and Kerberos authentication at project Athena," Feb. 1990. `http://world.std.com/~dtd/`.

[5] D. Boneh and M. Franklin, "Anonymous authentication with subset queries," in *6th ACM Conference on Computer and Communications Security*, (Kent Ridge Digital Labs, Singapore), pp. 113–119, ACM SIGSAC, Nov. 1999.

[6] M. Blaze, J. Feigenbaum, J. Ioannidis, and A. D. Keromytis, "The role of trust management in distributed systems security.," in *Secure Inter-*

*net Programming: Security Issues for Distributed and Mobile Objects* (J. Vitek and C. Jensen, eds.), (New York, NY, USA), pp. 185–210, Springer-Verlag Inc, 1999.

[7] S. Muftic, A. Patel, P. Sanders, R. Colon, J. Heijnsdijk, and U. Pulkkinen, *Security Archictecture for Open Distributed Systems*. John Wiley & Sons, 1994.

[8] D. Gollman, "On the verification of cryptographic protocols — a tale of two committees," in *Workshop on secure architectures and information flow* (S. Schneider and P. Ryan, eds.), vol. 32, (Royal Holloway, University of London), Dec. 1999.

[9] J. T. Kohl, B. C. Neuman, and T. Y. Ts'o, "The evolution of the Kerberos authentication system," *IEEE Computer Society Press*, pp. 78–94, 1994. `ftp://athena-dist.mit.edu/pub/kerberos/doc/krb_evol.PS`.

[10] F. Bergadano, B. Crispo, and G. Ruffo, "High dictionary compression for proactive password checking," *ACM Transactions on Information and System Security*, vol. 1, pp. 3–25, Nov. 1998.

[11] B. May and H. R. Wu, "Making Kerberos scalable," in *IFIP/SEC2000: Information Security* (S. Qing and J. H. P. Eloff, eds.), (Beijing, China), pp. 144–147, IFIP, International Academic Publishers, Aug. 2000.

[12] D. Davis, "Compliance defects in public key cryptography," in *Proceedings of the 6th Usenix Security Symp*, Springer, New York, 1996. `http://world.std.com/~dtd/`.

[13] A. McCullagh and W. Caelli, "Non-repudiation in the digital environment," *First Monday*, vol. 5, Aug. 2000. `http://www.firstmonday.dk/issues/issue5_8/mccullagh/`.

[14] C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, and T. Ylonen, "SPKI certificate theory," Experimental RFC 2693, Internet Engineering Task Force, Sept. 1999.

[15] G. I. Gaskell, "Integrating smart cards into kerberos," Master's thesis, Information Security Research Centre, School of Data Communications, Faculty of Information Technology, Queensland University of Technology, Feb. 2000.

[16] M. Looi, *Authentication for Applications in Computer Network Environments using Intelligent Tokens.* PhD thesis, Information Security Research Centre, School of Data Communications, Faculty of Information Technology, Queensland University of Technology, June 1995.

[17] B. May and H. R. Wu, "Scalable public key distribution," in *Papers presented at MADE2000* (H. Cardfelt, ed.), (Göteborg, Sweden), Chalmers University of Technology, May 2000.

[18] B. May and H. R. Wu, "Scalable authorisation with SPKI," *Submitted to Journal of Computer Security*, Mar. 2001.

[19] C. Kaufman, R. Perlman, and M. Speciner, *Network Security: Private Communication in a Public World.* Prentice Hall, 1995.

[20] T. Ylonen, T. Kivinen, M. Saarinen, T. Rinne, and S. Lehtinen, "SSH protocol architecture," Internet-Draft draft-ietf-secsh-architecture-03.txt, Internet Engineering Task Force, Feb. 1999. Work in progress.

[21] T. Wu, "A real-world analysis of Kerberos password security," in *Proceedings of the 1999 Network and Distributed System Security Symposium*, 1999. `http://www.isoc.org/ndss99/proceedings/`.

[22] W. Belgers, *UNIX Password Security*, Dec. 1993. `http://sunsite.nstu.nsk.su/sun/inform/whitepapers.html`.

[23] A. S. Tanenbaum, *Modern Operating Systems*. Prentice Hall, 1992.

[24] J. F. Haugh, "Shadow password suite," in *3rd USENIX Security Symposium*, Sept. 1992.

[25] D. R. S. D. K. Hess and U. W. Pooch, "A unix network protocol security study: Network information service," tech. rep., 1992. `http://citeseer.nj.nec.com/12427.html`.

[26] Sun Microsystems, Inc, *Network Information Service Plus (NIS+): A White Paper*, 1991. `http://sunsite.nstu.nsk.su/sun/inform/whitepapers.html`.

[27] M. Wahl, T. Howes, and S. Kille, "Lightweight directory access protocol (v3)," Internet Official Protocol Standard RFC2251, Internet Engineering Task Force, Dec. 1997.

[28] S. Pankanti, R. M. Bolle, and A. Jain, "Biometrics: The future of identification," *Computer*, pp. 46–48, Feb. 2000.

[29] S. V. Solms and B. Trait, "Biometric single sign-on: An authentication server approach," in *IFIP/SEC2000: Information Security* (S. Qing and J. H. P. Eloff, eds.), (Beijing, China), pp. 1–4, IFIP, International Academic Publishers, Aug. 2000.

[30] M. Vandenwauver, R. Govaerts, and J. Vandewalle, "Overview of authentication protocols: Kerberos and SESAME," in *Proceedings 31st Annual IEEE Carnahan Conference on Security Technology*, pp. 108–113, 1997. `https://www.cosic.esat.kuleuven.ac.be/sesame/html/sesame_links.html`.

[31] J.-S. Kim, B.-H. Cho, I.-G. Bae, and K.-Y. Yoo, "Authentication using a fingerprint in a smart card," in *IFIP/SEC2000: Information Security* (S. Qing and J. H. P. Eloff, eds.), (Beijing, China), pp. 53–56, IFIP, International Academic Publishers, Aug. 2000.

[32] B. Kaliski and J. Staddon, "PKCS #1: RSA cyrptography specifications," Request for Comments RFC2437, Internet Engineering Task Force, Oct. 1998.

[33] R. Housley and T. H. anf P. Yee, "TELNET authentication using DSA," Standards Track RFC2943, Internet Engineering Task Force, Sept. 2000.

[34] C. Ellison and B. Schneir, "Ten risks of pki: What you're not being told about public key infrastructure," *Computer Security Journal*, vol. 16, no. 1, pp. 1–7, 2000. `http://www.counterpane.com/pki-risks.html`.

[35] B. Tung, T. Ryutov, C. Neuman, G. Tsudik, B. Sommerfeld, A. Medvinsky, and M. Hur, "Public key cryptography for cross-realm authentication in Kerberos," Internet-Draft draft-ietf-cat-kerberos-pk-cross-05.txt, Internet Engineering Task Force, Nov. 1998. Work in progress.

[36] T. Dierks and C. Allen, "The TLS protocol," Standards Track RFC2246, Internet Engineering Task Force, Jan. 1999.

[37] T. Berners-Lee, R. Fielding, and H. Frystyk, "Hypertext transfer protocol — HTTP/1.0," Informational RFC1945, Internet Engineering Task Force, May 1996.

[38] C. Newman, "Using TLS with IMAP, POP3, ACAP," Standards Track RFC2595, Internet Engineering Task Force, June 1999.

[39] T. Ebringer, P. Thorne, and Y. Zheng, "Parasitc authentication to protect your e-wallet," *Computer*, pp. 54–60, Oct. 2000.

[40] R. M. Needham and M. D. Schroeder, "Using encryption for authentication in large networks of computers," *Communications of the ACM*, vol. 21, pp. 993–998, Dec. 1978.

[41] M. Burrows, M. Abadi, and R. Needham, "A logic of authentication," *ACM Transactions on Computer Systems*, vol. 8, pp. 18–36, Feb. 1990.

[42] D. Davis, "Kerberos with clocks adrift: History, protocols, and implementation," *USENIX Computing Systems*, vol. 9:1, Jan. 1996. `http://world.std.com/~dtd/`.

[43] S. M. Bellovin and M. Merritt, "Limitations of the Kerbero authentication system," in *Proceedings of the Winter 1991 Usenix Conference*, Jan. 1991. `http://web.mit.edu/kerberos/www/papers.html`.

[44] Massachusetts Institute of Technology, USA, *Kerberos: The Network Authentication Protocol*. Official Implementation of Kerberos. `http://web.mit.edu/kerberos/www/`.

[45] J. G. Steiner, C. Neuman, and J. I. Schiller, "Kerberos: An authentication service for open network systems," in *Proceedings of the Winter 1988 Usenix Conference*, Feb. 1988. `ftp://athena-dist.mit.edu/pub/kerberos/doc/usenix.PS`.

[46] S. Garfinkel, *PGP — Pretty Good Privacy*. O'Reilly & Associates, Inc., 1995.

[47] T. Ylonen, "The SSH (secure shell) remote login protocol," Internet-Draft draft-ylonen-ssh-protocol-00.txt, Internet Engineering Task Force, Dec. 1995. Work in progress.

[48] D. Davis, "Keberos plus RSA for world wide web security," in *Proceedings of the 1st Usenix Workshop on Electronic Commerce*, July 1995. `http://world.std.com/~dtd/`.

[49] G. Bella and L. C. Paulson, "Kerberos version IV: inductive analysis of the secrecy goals," in *Fifth European Symposium on Research in Computer Security: ESORICS 98* (G. Bella and L. C. Paulson, eds.), pp. 361–375, 1998. `http://www.cl.cam.ac.uk/users/lcp/papers/protocols.html`.

[50] R. Oppliger, "Authentication and key distribution in computer networks and distributed systems," in *Communications and Multimedia Security* (R. Posch, ed.), (Australia), pp. 148–159, Chapman & Hall, 1995.

[51] N. Itoi and P. Honeyman, "Smartcard integration with Kerberos V5," Technical Report 98-7, Center for Information Technology Integration, 1998.

[52] B. Tung, C. Neuman, M. Hur, A. Medvinsky, S. Medvinsky, J. Wray, and J. Trostle, "Public key cryptography for initial authentication in Kerberos," Internet-Draft draft-ietf-cat-kerberos-pk-init-09.txt, Internet Engineering Task Force, June 1999. Work in progress.

[53] P. Ashley, "Authorization for a large heterogeneous multi-domain system," in *Australian Unix and Open Systems Group National Conference*, pp. 159–169, 1997. `https://www.cosic.esat.kuleuven.ac.be/sesame/html/sesame_links.html`.

[54] *Heimdal*. Sweden. Alternative Implementation of Kerberos. `http://www.pdc.kth.se/heimdal/`.

[55] E. Gkioulekas, *Learning the GNU development tools*. Free Software Foundation, 1 ed., Sept. 1998. `http://www.amath.washington.edu/~lf/tutorials/autoconf/tutorial.html`.

[56] *SESAME V4 Administrator's Guide*, 1 ed., Dec. 1995. `https://www.cosic.esat.kuleuven.ac.be/sesame/html/sesame_documents.html`.

[57] W. Hu, *DCE Security Programming.* O'Reily & Associates, Inc., July 1995.

[58] K. Hornstein, "Distributing Kerberos KDC and realm information with DNS," Internet-Draft draft-ietf-cat-krb-dns-locate-05.txt, Internet Engineering Task Force, June 1999. Work in progress.

[59] V. Kessler and G. Wedel, "AUTLOG — An advanced logic of authentication," in *The Computer Security Foundations Workshop VII*, (The Franconia Inn, Franconia, New Hampshire), IEEE Computer Society Press, Washington, June 1994.

[60] D. Longley and S. Rigby, "An automatic search in key management schemes," *Computers & Security*, vol. 11, no. 1, pp. 75–89, 1992.

[61] J. Trostle and M. M. Swift, "Extension to Kerberos v5 for additional initial encryption," Internet-Draft draft-ietf-cat-kerberos-extra-tgt-01.txt, Internet Engineering Task Force, June 1999. Work in progress.

[62] M. K. Reiter and S. G. Stubblebine, "Authentication metric analysis and design," *ACM Transactions on Information and System Security*, vol. 2, pp. 138–158, May 1999.

[63] M. Blaze, J. Feigenbaum, and J. Lacy, "Decentralized trust management," in *1996 IEEE Symposium on Security and Privacy*, (Oakland, California), pp. 164–173, IEEE Computer Society Press, May 1996.

[64] J. I. Schiller and D. Atkins, "Scaling the web of trust: Combining Kerberos and PGP to provide large scale authentication," in *Proceedings*

*of the Winter 1995 Usenix Conference*, 1995. http://www.mit.edu:8001/people/warlord/warlord.html.

[65] Microsoft, "Windows 2000 Kerberos authentication," white paper, Microsoft, Jan. 2000. http://www.microsoft.com/technet/win2000/win2ksrv/technote/kerberos.asp.

[66] *Sesame — A Secure European System for Applications in a Multi-vendor Environment*.

[67] B. Crispo, "Delegation of responsibilities," in *Security protocols : 6th international workshop*, vol. 6, (Cambridge, UK), pp. 118–130, Springer, New York, Apr. 1998.

[68] M. Blaze, J. Feigenbaum, and A. D. Keromytis, "Keynote: Trust management for public-key infrastructures," in *Security protocols : 6th international workshop*, vol. 6, (Cambridge, UK), pp. 59–66, Springer, New York, Apr. 1998.

[69] C. M. Ellison, B. Frantz, B. Lampson, R. Rivest, B. M. Thomasn, and T. Ylonen, "Simple public key certificate," Internet-Draft draft-ietf-spki-cert-structure-06.txt, Internet Engineering Task Force, July 1999. Work in progress.

[70] J. Paajarvi, "XML encoding of SPKI certificates," Internet-Draft draft-paajarvi-xml-spki-cert-00.txt, Internet Engineering Task Force, Mar. 2000. Work in progress.