Key management in distributed systems

A comparison of Kerberos and Sesame

Per Kristian Gjermshus

Cand. scient thesis

August 2000
Key management in distributed systems
A comparison of Sesame and Kerberos

August 2000

Per Kristian Gjermshus
<pergj@ifi.uio.no>
Preface

This thesis is submitted to the Department of Informatics, University of Oslo in partial fulfillment of the requirements for the degree of Candidatus scientiarum (Cand. scient).

My interest in the field of computer security started when I read the book “Cyberpunk: Outlaws and Hackers on the Computer Frontier” by Katie Hafner and John Markoff [19]. This was years before my formal computer science studies started. While I never wanted to break into computers belonging to others, I found it interesting to find out how this could be done. My first encounter with cryptographic software was PGP [65], this introduced me to the basic concepts of public key cryptography.

One thing makes work in computer security fundamentally different from work in most other parts of computer science. In computer security theory comes before practice. In software development, testing plays an important role, if a product works during testing it most probably will continue to do so when deployed in the field. In computer security testing does not help much. Functional testing cannot help you find security problems, it is not possible to know in advance which problems to test for. The security system must be able to protect against unknown threats and attacks. Security systems must therefore be developed using sound software engineering practices. The protocols and algorithms must be analyzed thoroughly before one can be sure about their security. I find this close relationship between theory and practice, where theory can be said to be at least as important as practice, fascinating.

Even though I am a native Norwegian, I have chosen to write this thesis in English. I initially tried using my native tongue, but the terminology of the field does not adapt easily to the Norwegian language. The reader must be the judge on whether this decision was right.

My advisor on this thesis have been Leif Nilsen at UNIK (Universitetsstudie på Kjeller) and Thomson-CSF Norcom. Thanks to him for making me feel that I was on the right track and for having someone to discuss issues of computer security with.
I want to thank several people for their helpful comments. Without them this thesis would have had many more mistakes. Peder Klingenberg, Johan Seland and Olav Andree Brevik have offered helpful suggestions and corrected many mistakes. Hans Arild Runde has been particularly helpful. There is no doubt that his criticisms have made the end result much better. All the remaining mistakes are my own.

Finally I want to thank my parents Brit and Per. They have always supported me, and have given me the foundation on which it was possible to build my academic work.

Per Kristian Gjermshus
August, 2000
Contents

1 Introduction 1
  1.1 Goals and results of this thesis 2
  1.2 Overview of the thesis 3

2 Cryptography 5
  2.1 Encryption schemes 6
    2.1.1 Conventional cryptography 7
    2.1.2 Public key cryptography 9
  2.2 Integrity check functions 11
    2.2.1 Cryptographic hash functions 11
    2.2.2 Message Authentication Codes 12
  2.3 Digital signature schemes 13
  2.4 Attacks on cryptographic primitives 13

3 Cryptographic Protocols 15
  3.1 Protocol failures 16
  3.2 Properties of protocols 17
    3.2.1 Trust considerations 18
    3.2.2 Third parties 18
    3.2.3 Computational efficiency 19
    3.2.4 Communicational efficiency 19
    3.2.5 Need for maintenance of state 20
    3.2.6 Key control 20
    3.2.7 Impact of key compromise 21
  3.3 Designing protocols 22
    3.3.1 Data Confidentiality 22
    3.3.2 Modification Detection 22
    3.3.3 Use of identifiers 23
    3.3.4 Replay Detection / Timeliness 23
    3.3.5 Entity authentication 25
    3.3.6 Data Origin Authentication 25
    3.3.7 Proof of delivery 26
  3.4 Entity authentication 26
3.4.1 Properties of authentication protocols ......... 27
3.4.2 Strong authentication .......................... 28
3.4.3 Weak authentication .......................... 33

3.5 Key exchange ........................................ 35
3.5.1 Properties of key exchange protocols ........... 35
3.5.2 Example protocols .............................. 36

3.6 Ways of attacking protocols .................... 39
3.6.1 Known key attack ............................. 39
3.6.2 Impersonation .................................. 39
3.6.3 Dictionary and forward search attacks ........... 39
3.6.4 Replay attack ................................ 40
3.6.5 Active attacks ................................ 40

4 Key Management ................................. 43
4.1 Classification of keys ............................ 44
4.2 Key protection ................................. 44
4.2.1 Key layering ................................ 45
4.2.2 Cryptoperiods ............................... 46
4.2.3 Controlling key usage ......................... 46
4.2.4 Physical protection .......................... 48

4.3 Key life cycle .................................... 49
4.4 Services provided by a key management system .... 50
4.4.1 Generate key ................................ 50
4.4.2 Register key ................................ 51
4.4.3 Create key certificate ......................... 52
4.4.4 Distribute key ................................ 52
4.4.5 Install key .................................. 52
4.4.6 Store key .................................. 52
4.4.7 Derive key .................................. 53
4.4.8 Archive key ................................ 53
4.4.9 Revoke key ................................ 53
4.4.10 Deregister key ............................. 53
4.4.11 Destroy key ................................ 53

4.5 Public key infrastructures ....................... 54
4.5.1 Public key certificates ....................... 54
4.5.2 Certification Authorities ...................... 55
4.5.3 Certificate revocation ......................... 55
4.5.4 Certification hierarchies ...................... 56
4.5.5 Webs of trust ................................ 57

5 Kerberos .............................................. 59
5.1 Introduction ........................................ 59
5.2 The Kerberos architecture ....................... 60
5.3 The Kerberos protocols .......................... 61
<table>
<thead>
<tr>
<th>CONTENTS</th>
<th>vii</th>
</tr>
</thead>
</table>

5.3.1 Data elements .................................. 62
5.3.2 Message exchanges .............................. 63
5.3.3 Cross realm operation .......................... 67

5.4 Kerberos assumptions .............................. 68
5.4.1 Requirements for cryptographic primitives ........ 68
5.4.2 Key generation ................................. 69
5.4.3 Environmental assumptions ...................... 69

5.5 Key management in Kerberos ....................... 70
5.5.1 Key hierarchy .................................. 71
5.5.2 Key control .................................... 72
5.5.3 The life cycle of a Kerberos key ............... 72

5.6 The Kerberos trust model .......................... 74

5.7 Kerberos limitations ............................... 74
5.7.1 No control over the usage of session keys ....... 75
5.7.2 Dependence on synchronized clocks ............... 75
5.7.3 Password guessing ............................... 75
5.7.4 Trust in local workstation ...................... 76
5.7.5 Same key used for authentication and encryption .. 76

6 SESAME ............................................ 77
6.1 Introduction ....................................... 77
6.2 Goals of SESAME ................................. 78
6.3 The SESAME architecture ......................... 79
6.3.1 Initiator components .......................... 80
6.3.2 Target components .............................. 81
6.3.3 Trusted Third Parties ......................... 82
6.3.4 Generic components ............................ 82
6.3.5 Public Key Management Services ............... 83

6.4 The SESAME protocols ............................ 84
6.4.1 Cryptographic primitives ..................... 84
6.4.2 Data elements .................................. 85
6.4.3 Credentials ................................... 85
6.4.4 PAC Protection Methods ........................ 86
6.4.5 Message exchanges ............................. 87
6.4.6 The Kerberos-compatible protocol ............. 90
6.4.7 Inter-domain operation ......................... 92

6.5 Key management in SESAME ......................... 92
6.5.1 Asymmetric keys ............................... 93
6.5.2 Symmetric keys ................................. 93
6.5.3 Kerberos compatibility keys .................... 94
6.5.4 Key control .................................. 95
6.5.5 The life cycle of a SESAME key ............... 95
6.5.6 Key hierarchy ................................. 96

6.6 The SESAME trust model .......................... 97
6.7 Sesame limitations ........................................ 98
   6.7.1 The PFV is trusted .................................. 99
   6.7.2 Critical keys stored on the user’s workstation .... 99
   6.7.3 Complicated architecture ............................. 100
   6.7.4 Weak Public Key Infrastructure ..................... 100
   6.7.5 Encryption before integrity protection .............. 101
   6.7.6 Unpolished implementation .......................... 101

7 Comparing Kerberos and SESAME .......................... 103
   7.1 Architectural differences .............................. 104
   7.2 Protocols ............................................... 105
      7.2.1 Initial authentication ............................. 105
      7.2.2 Privileges ......................................... 105
      7.2.3 Authentication to the target ..................... 106
      7.2.4 Sending of protected data ......................... 106
      7.2.5 Inter-domain operation ............................ 106
      7.2.6 Renewal of credentials ............................ 107
      7.2.7 Handling of forwardable credentials ............. 107
   7.3 Key management ......................................... 108
      7.3.1 Key storage ....................................... 109
      7.3.2 Key control ....................................... 110
   7.4 Concluding remarks ..................................... 111
List of Figures

2.1 Conventional cryptography .......................... 8
2.2 Public key cryptography ............................ 11

4.1 The key life cycle ................................. 50
4.2 A certification hierarchy ........................... 57

5.1 Kerberos components and messages ............... 60
5.2 The key hierarchy of Kerberos ..................... 71

6.1 Overview of the SESAME components .............. 80
6.2 The key hierarchy of Sesame ....................... 97
List of Protocols

3.1 ISO-9798-2 one pass authentication .................. 29
3.2 ISO-9798-2 authentication with random numbers .......... 29
3.3 ISO-9798-2 mutual authentication with random numbers .... 29
3.4 Challenge response based on asymmetric cryptography .... 30
3.5 Unilateral one pass authentication ........................ 31
3.6 Fiat-Shamir ............................................. 33
3.7 Needham-Schroeder – first part .......................... 36
3.8 Needham-Schroeder – second part ......................... 37
3.9 Diffie-Hellman key exchange .............................. 37
3.10 Man in the middle attack ............................... 38
3.11 Station-to-station ....................................... 39
3.12 Protocol vulnerable to a reflection attack .................. 41
3.13 Reflection attack ......................................... 41
3.14 Protocol vulnerable to an interleaving attack ............... 42
3.15 Interleaving attack ...................................... 42
5.1 Kerberos Authentication Service Exchange .................. 64
5.2 Kerberos Client/Server Authentication Exchange .......... 65
5.3 Kerberos KRB_SAFE Exchange ............................ 66
5.4 Kerberos KRB_PRIV Exchange ............................. 66
6.1 Sesame SES-AS .......................................... 88
6.2 Sesame SES-PAS .......................................... 89
6.3 Sesame SES-INIT-CTXT .................................. 89
6.4 Sesame SES-DATA (integrity only) ........................ 91
6.5 Sesame SES-DATA (integrity and confidentiality) .......... 91
6.6 Sesame SES-DATA (improved version) ...................... 101
List of Tables

5.1 Trust relationships of Kerberos ................. 74
6.1 SESAME Abbreviations ..................... 81
6.2 Trust relationships of SESAME ................. 98
Chapter 1

Introduction

In the infancy of computing, security was simple. When each computer only had one user, and this user had to be sitting right next to the computer, it was sufficient to protect the computer hardware itself. The fact that the computer was an extremely costly piece of equipment made it apparent that it had to be adequately protected. The time-proven art of physical security was enough to secure any computer system. To be able to utilize the resources in a better way, time sharing systems were developed. This made it possible for several users to use a central computer at the same time. Users hooked up to the computer using terminals. These terminals were usually located in the same building as the computer and physical security was still adequate.

As the years passed and multi-user systems became more and more common, it became apparent that users sharing a computer had to be protected from each other. The data belonging to one user should be protected from that of another, and processes belonging to different users should not be allowed to interfere with each other. A lot of research was done in the field of operating systems during the 60s and many lessons were learned.

Nowadays most computers are connected to a network. The concept of one single computer providing all the computing needs of one organization was abandoned long ago. Networks make it possible to interconnect computers at physically different locations. Autonomous networks previously inaccessible from the outside are now connected to the global network of networks called the Internet. Coulouris, Dollimore and Kindberg [6] define a distributed system as a collection of autonomous computers linked by a network, with software designed to produce an integrated computing facility.

Obviously these systems give rise to new classes of security threats. New
threats require new methods of protection. Physical security is no longer enough. Physical security comes natural to humans, we have been accustomed to protecting our belongings for a long time. People know that if something is locked down it is protected. We also have an intuition that can guide us in choosing the right form of protection for a given object. When it comes to information security or security in computing most people can’t see the threat, consequently they don’t think it exists.

Security in computing is about ensuring that three specific goals are met [53]. These goals are confidentiality, integrity and availability. Confidentiality ensures that no one gets to read information he is not authorized to read. Integrity is about unauthorized modification of information and availability is about making sure that information is available to those who are authorized when they need the information. It is often said that the most secure computer is turned off. A turned off computer is not particularly available.

So what do people do about these problems? Many people just close their eyes pretending the problem is not there. Some conclude that the Internet is dangerous and refuse to connect their computers to it. Others install firewalls screening the traffic between them and the Internet and hide behind them. Although the market for firewalls has become quite large during the last few years, firewalls are not the solution to all security related problems. Firewalls assume that all the bad guys are on the outside of the firewall. In most large organizations and especially in university networks, this assumption is simply not true. A recent survey [60] indicated that as much as 35% of theft of proprietary information was committed by discontented employees.

The majority of security experts realize that the use of security systems employing cryptographic techniques for data integrity protection and authentication is the only viable solution in the long term. The recent “I Love You” virus has shown how malicious code can easily sneak trough the firewall with the help of unknowing users. Systems securing the network from attacks from the inside are therefore clearly needed.

1.1 Goals and results of this thesis

Kerberos and SESAME are two systems providing integrity protection and authentication in distributed computer systems. The goal of this thesis is to compare these two systems from the perspective of key management. This is an interesting subject because Kerberos and SESAME try to solve the same problems, but have chosen to base the technologies on two different
types of cryptographic primitives. Kerberos uses symmetric key technology while SESAME uses asymmetric cryptography. How the management of the keys differ because of these design choices will be the main topic of this thesis. As we will see in later chapters, the use of cryptographic techniques require cryptographic keys. The security of the entire system will rely on the security (both the integrity, confidentiality and availability) of these keys. Key management is therefore of extreme importance to the security of the system.

To be able to compare Kerberos and SESAME a subgoal is to give an overview of existing cryptographic protocols for key distribution and entity authentication. The topic of key management will also be given extensive treatment. This will give a background for the comparison and analysis in the last chapters.

SESAME has not been as thoroughly analyzed as have Kerberos, and I present several weaknesses of SESAME, which to the best of my knowledge, have not previously been described in the literature.

The result of the comparison is that the Kerberos architecture is less scalable and require more central control. SESAME is more distributed, and scales better. The price SESAME must pay for this is the burden of requiring the users to protect critical keys locally.

1.2 Overview of the thesis

The structure of this thesis is as follows: The chapter following the introduction serves as a quick primer on general cryptographic techniques, focusing on the types of functions needed later. In the third chapter I describe cryptographic protocols in general. The common failure modes of protocols are described along with properties that must be considered when evaluating protocols. Specific protocols for entity authentication and key transport are presented.

The fourth chapter is a description of key management techniques. How keys are classified and how they can be protected from misuse is the focus of the first part of the chapter. Then the life cycle of a key and the services that a key management system provides are explained. The last part covers public key infrastructures.

Chapter five and six are descriptions of Kerberos and SESAME. In the first part of both chapters the architecture and protocols are explained, while the last part consists of a discussion of how key management is done and a presentation of some weaknesses that have been found.
Chapter seven is a comparison of Kerberos and SESAME. The focus is on how the choice of asymmetric cryptography in SESAME and symmetric cryptography in Kerberos influence the key management in these two systems.
Chapter 2

Cryptography

This chapter introduces the basic cryptographic techniques used to build secure computer systems. I will not go in depth about specific cryptographic algorithms, but will rather describe the different classes of algorithms in use. In the next chapter we will see how the algorithms are used as elements in cryptographic protocols.

The basic building block of secure computer systems is cryptography. The word cryptography has its roots in the Greek word “kryptos” which means “hidden” and “graphein” which means “writing”. Menezes, Oorschot and Vanstone [38] define cryptography as “the study of mathematical techniques related to aspects of information security such as confidentiality, data integrity, entity authentication, and data origin authentication”. Four services are then defined to be the basic objectives of cryptography.

Confidentiality This service is used to keep information hidden from those who are not authorized to see it. Secrecy or privacy are often used as synonyms for confidentiality.

Data integrity A service used to protect data from unauthorized alteration. To detect such alteration a method able to detect whether a set of data has been tampered with is needed. Tampering can be insertion, deletion or substitution.

Authentication In many applications it is important to be able to identify the entity you are talking to. The authentication service covers both entity authentication and authentication of information. For example date of origin, data content etc.

Non-repudiation This service is used to prevent an entity from denying previous communication. In on-line banking for example it is crucial
that the bank can prove that the customer wanted to transfer a certain amount of data.

To be able to provide these services several cryptographic primitives are used. I will describe encryption schemes, hash functions and digital signature schemes. The cryptographic services go a long way in meeting the goals of computer security in general. They can help ensure confidentiality and integrity. The problem of how to ensure availability can not easily be solved by using cryptographic techniques. In many cases using cryptography will be counterproductive to the goal of availability.

2.1 Encryption schemes

The goal of a cryptosystem is to enable two participants (usually called Alice and Bob), to communicate securely over an insecure channel. This means that an eavesdropper (often called Eve), who is listening on the communication channel should not be able to read the messages. The information Alice wants to communicate to Bob is called the plaintext \((x)\). The plaintext and the key is input to the encryption function. The operation of encrypting with the key \(k_e\) is denoted as \(y = e_{k_e}(x)\). The result of this function is called the ciphertext \((y)\). The ciphertext is communicated over the insecure channel and is received by Bob. Bob applies a decryption function \((x = d_{k_d}(y))\) to the ciphertext, the result is the plaintext.

Kerckhoffs’ Principle: The security of a cryptosystem must not depend on keeping secret the crypto-algorithm. The security depends only on keeping secret the key.

— Kerckhoffs (1883)

As Kerckhoffs realized in the 1880s the security of an encryption scheme should not lie in the secrecy of the algorithm. The only part assumed to be secret should be the key. In other words one should assume that a potential attacker has access to the complete description of the algorithm. Even though military cryptographic algorithms are seldom published they are always designed with this in mind.

Because the key is the only secret parameter of a cryptosystem, the most obvious way of attack is to try to guess the key. The size of the keyspace (the number of possible keys), is therefore of extreme importance. With a large keyspace the number of trials needed to guess the right key will be large. Trying all keys is often called brute force attack or exhaustive search.
attack. It should be noted that a large keyspace is not enough to make an algorithm secure, but it is necessary. There might be weaknesses in the algorithm making it possible to obtain the plaintext by other methods than trying all the keys.

When speaking about cryptographic algorithms cryptographers often say that a cryptosystem is *computationally secure*. What this means is that the best currently-known attack on the system has so large computational complexity that an adversary will not be able to carry out the attack with the resources he is in possession of. The adversary is usually assumed to have access to a lot of resources, and a large margin is added as well.

Computational complexity theory is usually concerned with the worst case or the average case complexity of a problem. NP completeness is not a very useful notion when speaking about complexity for cryptosystems, since NP completeness is always about worst case. An NP complete problem only needs to contain one instance with large complexity. Having high average case complexity is not enough either. The problem can still contain certain instances with low complexity.

An attack is said to be *computationally infeasible*, if the fastest computers use considerable time to carry out the attack. Statements like “This cryptosystem is secure if the integer $n$ cannot be factored”, are quite common. In this case breaking the cryptosystem has been reduced to the integer factorization problem and it has been proven that if one can factor integers effectively one can also break the cryptosystem. If it can be proved that the only way of breaking a system is to solve some well-known and difficult problem, the system is said to be *provably secure*.

If a system cannot be broken no matter how much computation the attacker is allowed to do the system is called *unconditionally secure*. While it would be comforting to know that a system was unconditionally secure the reality is that such a system would have to use keys at least as long as the plaintext. The first example of a cryptosystem providing unconditional security was the Vernam One-Time Pad invented and patented by Gilbert Vernam in 1917. Claude Shannon later proved [57] that the One-Time Pad did provide unconditional security and that a system only could do so if the length of the keys were at least as long as the plaintext.

### 2.1.1 Conventional cryptography

A conventional cryptosystem, or a symmetric cryptosystem is a cryptosystem in which it is computationally easy to compute the decryption key knowing only the encryption key. In most cases $k_e = k_d$. 
Figure 2.1 gives a schematic presentation of a conventional cryptosystem. It is absolutely crucial to the security of the system that Eve does not gain access to the key. The key will therefore have to be transmitted over a secure channel between Alice and Bob before encryption can be used. Knowing the key is the only computationally feasible way to compute $x$ from $y$.

Conventional cryptosystems are typically divided into two classes. Block ciphers and stream ciphers. A block cipher is an encryption scheme that breaks the plaintext up into blocks before encryption and encrypts one block at a time. A stream cipher encrypts one bit of data at a time and each step is dependent on the steps before.

The concept of cryptography predates the electronic computer and worldwide computer networks. The best source of information about the early days of cryptography is David Kahn’s Codebreakers [30]. The most well-known symmetric cryptosystems are block ciphers, and the most widely used of these is DES (Data Encryption Standard) [45]. The key size of DES is 56 bits, and this is considered too small for most current applications. The algorithm has nonetheless withstood most attacks and has played an important role in the development of other block ciphers such as IDEA [36]. The American National Institute of Standards and Technology (NIST) has started a selection procedure that will ultimately choose the block cipher that will be the standard for the future. This standard is called the AES (Advanced Encryption Standard). Several interesting algorithms have been submitted and they will all be put through a very thorough evaluation. Up-to-date information can be found on the NIST AES website [51].

Although there is much theoretical knowledge about stream ciphers, very few algorithms have been published. Examples of unpublished stream cipher algorithms are the A5 algorithm used in GSM mobile telephones and RC4 designed by Rivest.
2.1 Encryption schemes

Modes of operation

The material to be encrypted is often very much longer than the block size of a block cipher. DES for example has a block size of 64 bits. One therefore has to break the plaintext up somehow and feed it to the encryption algorithm. There are several ways of doing this. I will describe some of the modes of operation defined for DES.

The most obvious way of attacking the problem is to encrypt each block of plaintext separately. In a sequence of blocks \( x_1 x_2 \ldots \), each block \( x_i \) would be encrypted using the same key and produce the ciphertext \( y_1 y_2 \ldots \). This mode is called electronic codebook mode or ECB. In this mode one looks at the encryption algorithm as one large codebook where one can look up each block and find the corresponding ciphertext.

In cipher block chaining mode (CBC) each plaintext block is \( x \)-ored with the previous ciphertext block before encryption. The first block is \( x \)-ored with a value called the initialization vector (IV). This gives the encryption rule
\[
y_0 = IV, \quad y_i = e_K(y_{i-1} \oplus x_i), \quad i \geq 1
\]

The two other operation modes of DES essentially use DES as a keyed random number generator for a stream cipher. These modes are called output feedback mode (OFB) and cipher feedback mode (CFB). More information on the different modes of DES and the advantages and disadvantages of each can be found in [46].

2.1.2 Public key cryptography

_We stand today on the brink of a revolution in cryptography._

— Diffie and Hellman

When the development of large computer networks started it became clear that it was impractical to be dependent upon a secure channel to be able to exchange keys. In many potential applications of cryptography the participants have never met in person, and do not have the resources needed to exchange keys manually.

The concept of public key cryptography was therefore introduced in 1976 by Whitfield Diffie and Martin Hellman in their article “New Directions in Cryptography” [8]. They describe a system in which the encrypting key is different from the decrypting key. Their idea is very simple, but at the time their article was published, it was still unclear whether such a system was implementable in practice. Three years passed until Rivest, Shamir
and Adleman [54] presented the first functioning public key cryptosystem. This system has later become known as the RSA cryptosystem.

It is rather interesting to note that the English CESG (Communications-Electronics Security Group) claims to have invented public key cryptography (they called it non-secret encryption) in 1970 [14]. James Ellis struggled with the key distribution problem and managed to prove the existence of public key cryptosystems. In 1973 Clifford Cocks, discovered the first workable implementation of the new system. His algorithm was essentially equivalent to RSA.

A public key cryptosystem can be described as follows. Instead of having one key as in conventional cryptography, two keys are used. One is used for encryption, this key is called the public key. The other is used for decryption, this key is called the private key. The security of the system relies on the assumption that it is difficult to derive the secret key from the public key.

Let’s use the RSA system as an example of a public key cryptosystem. It relies on the difficulty of factoring large integers, and is the most well known and most widely used public key system. RSA has two public variables \( n \) and \( b \), forming the public key. The private part of the key consists of three secret variables \( p \), \( q \) and \( a \). \( p \) and \( q \) are large primes and \( n = pq \). \( a \) and \( b \) are related by the equation: \( ab \equiv 1 \mod \phi(n) \). The encryption and decryption operations are as follows:

\[
e_k(x) = x^b \mod n
\]
\[
d_k(y) = y^a \mod n
\]

I will not say more about why RSA works, the purpose of the example is just to illustrate what the keys of a cryptosystem could be like. A good description of RSA and its mathematical properties can be found in [63].

Although the problem of distributing the key through a confidential channel is removed when using public key cryptography, there are still problems related to key distribution. To trust a public key system you must be sure that the public key you are using is really belonging to the person with whom you are communicating. This means that the channel used to distribute the key must be integrity protected such that the key cannot be changed in transit. In addition the channel must be authentic; one must be sure that the party at the other end is who he is pretending to be. A figure illustrating this can be found in figure 2.2. Note Bob is the source of the key, rather than Alice as the case was with symmetric cryptography.

How to distribute public keys in a system in a way that ensures the authenticity and integrity of the keys is the job of a **public key infrastructure**
2.2 Integrity check functions

Integrity check functions are used to detect unauthorized alteration of data while in transit. Integrity check functions include cryptographic hash functions and message authentication codes.

2.2.1 Cryptographic hash functions

A hash function is a function taking an arbitrary binary string as input and generating an output string of a constant size. The generated string is often called a message digest. To be of any use in cryptographic applications a hash function has to satisfy several properties. Some potential properties of a hash function $h()$ are listed on the next page.
1. **compression** – $h()$ maps an input string $x$ of arbitrary finite bitlength, to an output $h(x)$ of fixed bitlength $n$

2. **ease of computation** – given $h()$ and input $x$, $h(x)$ is easy to compute

3. **preimage resistance** – for essentially all pre-specified outputs, it is computationally infeasible to find any input which hashes to that output

4. **2nd-preimage resistance** – it is computationally infeasible to find any second input which has the same output as any specified input

5. **collision resistance** it is computationally infeasible to find any two distinct inputs $x, x'$ which hash to the same output

The first two properties are necessary for a function to be classified as a hash function. If $h()$ satisfies property 1, 2, 3, and 4 it is called a **one-way hash function**. Functions that satisfy 1, 2, 4 and 5 are called **collision resistant hash functions**.

### 2.2.2 Message Authentication Codes

In addition to the normal hash functions described above, there exists a special kind of hash function called Message Authentication Codes (MAC). These are hash function taking a key as an additional argument. The process of computing a MAC on the message $m$ using the key $k$ is denoted as $MAC_k(m)$. If two entities want to compute a MAC of the same data, they will have to supply the same key as input to get the same result. A MAC can be used to provide assurance about the source and integrity of a message.

A MAC can be implemented by using the CBC and CFB modes of a block cipher. If one plaintext block is altered in one of these modes all the remaining ciphertext blocks will also be different. One can therefore define the last ciphertext block in the sequence to be the MAC of the plaintext. To verify that the MAC is correct one can go through exactly the same operation as when computing the MAC and check that the last block is equal to the MAC.

Other possibilities of creating message authentication codes is to use a hash function. The straightforward method is to append or prepend a secret key to the data and compute a hash over both the data and the key. These methods have problems (see for example Menezes, Van Oorschot and Vanstone [38]). The use of the HMAC [35] method is therefore advised. Further information on how to create message authentication codes can be found in
2.3 Digital signature schemes

ISO-9797 [26]. The first part describes methods for creating message authentication codes using block ciphers, while the second part uses hash functions.

2.3 Digital signature schemes

A digital signature is the computer security equivalent of a handwritten signature on a document. A signature scheme consists of two algorithms, the signature algorithm and the verification algorithm. The signature algorithm produces a signature given a secret key and a message. The verification algorithm is used to verify that the signature is correct, and can be carried out without knowing any secret information. The secret key of a signature system is usually called the signature key and the public part is called the verification key. Signing a message \( m \) using the signature key \( k \) is denoted as \( \text{SIGN}_k(m) \).

Digital signature schemes can be divided into two classes. Digital signature schemes with appendix require the original message as input to the verification algorithm, while digital signature schemes with message recovery can be verified without the original message. In the latter case the message is recovered from the signature.

The RSA cryptosystem [54] can be used as a signature scheme. There are also systems specifically designed to be used as signature schemes, for example the ElGamal signature scheme [13] and the Digital Signature Standard [47] which is a modification of ElGamal.

2.4 Attacks on cryptographic primitives

Attacking cryptographic primitives and trying to deduce the key or the plaintext by exploiting weaknesses in the algorithms is called cryptanalysis. The attacks can be categorized according to what the cryptanalyst is allowed to do.

Ciphertext only In this type of attack the attacker has access to the encrypted ciphertext only.

Known plaintext If the attacker has access to some ciphertext-plaintext pairs he can mount a known plaintext attack.

Chosen plaintext Even more powerful methods can be used if the attacker is allowed to choose some plaintexts and obtain their decryptions.
Adaptive chosen plaintext In this case the attacker is not only allowed to choose plaintexts and have them encrypted, but he can also obtain new pairs after looking at what he has already got.

Chosen ciphertext If the attacker can choose the ciphertext himself and obtain the corresponding plaintext the attack is called a chosen ciphertext attack.

Adaptive chosen ciphertext If the attacker is allowed to chose new ciphertexts after having looked at plaintexts from previous requests the attack is called adaptive chosen ciphertext.

A ciphertext only attack is the most difficult attack to mount, whereas an adaptive chosen plaintext gives the attacker a lot more to work on. Adaptive chosen plaintext attacks are especially easy to mount on public key systems, since one can use the public key to obtain the encryption of any plaintext.

These attacks apply to encryption schemes, but they can also be adapted to message authentication codes and signature schemes. For signature schemes based on public key systems it is important that the system is resistant to adaptive chosen ciphertext attacks since a verification of a signature is the same operation as decryption with the public key system.
Chapter 3

Cryptographic Protocols

A protocol is a distributed algorithm involving two or more parties, designed to accomplish some common objective. For example the secure transfer of a key from one party to another. Cryptographic protocols use the cryptographic primitives such as encryption and hash functions described in the previous chapter to accomplish more complex than the primitives can accomplish in isolation.

There is a subtle distinction between a cryptographic protocol and an ordinary communication protocol. When designing communication protocols one is focused on a small set of identifiable problems to be solved, the goal is for example to transfer one block of data from one party to another and be sure that it has been received correctly. The protocol designer might be concerned about issues such as packet loss and round-trip time. The threats to cryptographic protocols are much more subtle and cannot be easily identified in advance. A malicious intruder might do all sorts of strange things to confuse or trick the protocol. The threat might not even come from the outside but from one of the participants in the protocol. The designers of communication protocols have traditionally not been very concerned about security issues. This has led to security problems that are a direct result of the design of protocols. Bellovin [3], for example, describes several security related flaws in the TCP/IP protocol suite used on the Internet.

The focus in this chapter will be on protocols for key distribution. In order to effectively manage keys in a distributed system one needs to be able to establish keys securely. I therefore discuss key exchange protocols. The question of entity authentication is important when distributing keys, it does no good to distribute a key securely if you do not know who you are distributing the key to. I therefore examine entity authentication protocols.
Before starting to study the protocols proper, it is instructive to look at how past protocols have failed to accomplish the goals they promised to fulfill. I will also describe some properties that one should have in mind when choosing what protocol to use. Section 3.3 gives an overview of primitives that are commonly used in cryptographic protocols.

### 3.1 Protocol failures

When designing a protocol on top of the cryptographic primitives several new possibilities of security flaws arise. When a cryptographic protocol fails to accomplish its security goals, not because the underlying cryptographic primitives are broken, but because of flaws in the protocol itself, this is called a protocol failure or mechanism failure. Handbook of Applied Cryptography [38] lists the following reasons for protocol failures:

1. weaknesses in a particular cryptographic primitive which may be amplified by the protocol or mechanism
2. claimed or assumed security guarantees which are overstated or not clearly understood
3. the oversight of some principle applicable to a broad class of primitives such as encryption

An example of the first weakness is the common modulus protocol failure and the small exponent protocol failure of the RSA system. The source of these failures is inherent in the mathematical properties of the RSA system, and occurs if system parameters are chosen carelessly. More information on these failures can be found in the articles by Simmons [58] and Moore [43].

Sometimes protocols are proposed that do not have well understood security guarantees. For example a protocol used for manipulation detection when using DES encryption was proposed as a US federal standard. The proposal had several flaws, one of them being that it was possible to rearrange individual blocks without detection. A protocol assuming that all blocks were ordered correctly could be broken by exploiting this flaw.

In some applications the number of possible messages going over a communication channel can be low. The information theoretical term used for measuring the amount of information contained in a single data element is entropy. If there are few possible messages the entropy of the channel is low. The message being sent can for example be the answer to a simple
yes-or-no question. If this type of message is encrypted using a public key system an attacker only has to encrypt the two possible messages using the public key and then compare these two encryptions with the data transmitted. Although it is not hard to see the problem in this specific case, more subtle attacks using the same method have been documented. For example a system for secure telephony was broken because spoken language did not utilize the whole frequency spectrum available on the channel. It was therefore possible to precalculate many plaintext-ciphertext pairs and use these known pairs to interpolate the data. The result was a scrambled but still understandable voice. This attack is described by Simmons and Holdridge [59].

Moore [43] gives a survey of some known protocol failures and identifies two steps that are essential to follow when designing cryptographic protocols.

1. Identify all assumptions in the protocol

2. For each assumption determine the effect on the security objective if that assumption is violated.

It is important to be explicit on the assumptions that are made, as violation of these will often lead to a total breakdown of the protocol and perhaps also the entire security system. Typical assumptions could be the trustworthiness of trusted third parties (see section 3.2.2) or that the same key should not be used for different purposes. It can also be assumptions of mathematical nature, such as specific mathematical properties of keys. If the assumptions are well documented and the effect of breaking them are well understood it will be easier to develop a new protocol if, for some reason, one of the assumptions cannot be met. By looking at the assumptions it will also be easier to pinpoint critical parts of the protocol. The purpose of each assumption should also be carefully analyzed, it might be that one would be better off making a small change to the protocol and removing the assumption. An ideal protocol should contain as few assumptions as possible and those that are made should be easy to understand and not too hard to fulfill in practice.

### 3.2 Properties of protocols

As we have seen, designing a cryptographic protocol from scratch is error-prone and difficult. Many of the protocols described in the literature and used in real life systems have later been found to have fatal flaws. The best
approach will therefore be to look at the available protocols and choose the one that fits the needs at hand best. There are several properties that should be considered before making a choice. The cryptographic protocol often makes assumptions about the environment it is going to operate in, these assumptions should be compatible with the demands of the specific application.

In many cases the properties are conflicting. A trade-off where one property must be prioritized over another will often have to be done.

### 3.2.1 Trust considerations

The components of a security system that are so important that a compromise will destroy much of the security, are called trusted. The trusted components form the foundation of the system security. Being explicit about the components that are trusted is an important part of documenting a security architecture. The implementor can then take measures to protect the trusted components. Formal evaluation of trusted components should be considered for environments high security requirements. Since each trusted component is an obvious point of attack, the number of such components should be kept as low as possible.

Trust is a relation. One component trusts another. The trusting component is dependent on the trusted component to carry out its objective. To complicate things even further, one component can trust another which in turn trusts another component. We can therefore talk about chains of trust. Such chains should be clearly documented in the architecture description.

### 3.2.2 Third parties

Closely related to the concept of trust are third parties. Some protocols require a third party to be involved in the protocol. These parties can be trusted by the entities carrying out the protocol. In this case the party is called a trusted third party (TTP for short). The trusted party can have many different roles, the most common being to act as a key distributor or an issuer of public key certificates (see section 4.5.1). The third party can either be on-line, or off-line. An on-line trusted party takes actively part in the protocol during each run. The off-line trusted party typically issues some information before the protocol starts and does not participate in the protocol in real time.

Whether the use of a third party is acceptable or not depends on many factors. First of all, in a large system having one central server (which an
on-line trusted party often is) could be a serious bottleneck and prevent the system from scaling to the required size. It might be possible to create more than one trusted server and distribute the third party tasks. The downside to this is that each new server in the system is a new possible point of attack for an adversary and a threat to the system as a whole.

Trusted third parties also require that all the participants in the system can agree on one party that they all trust. In many situations this will be infeasible. Inside one organization this is usually no problem, but if one wants to scale to global settings it becomes impossible. Moreover, trusted parties often imply a large amount of central control, and this might be unacceptable.

### 3.2.3 Computational efficiency

Both the complexity of the cryptographic operations performed during the run of the protocol and the number of such operations is an important aspect to consider in a protocol. In many applications today it is expected that many clients will connect to one server. This is common in e-commerce applications where many customers will connect to the server of the service provider. Most servers are overloaded in the first place, and the cost of adding real security should not be so high that people choose to drop the security altogether. If the complexity of carrying out the cryptographic operations is too high the server will use too much of its resources on the cryptographic services instead of doing real work.

In some protocols the computational burden is not evenly divided among the participants. One example of this is the Digital Signature Algorithm [48], where the verification of the signature is more intensive than the generation. In some applications such behavior can be desirable, for example if one of the parties in the protocol is a smart-card with limited cpu power and limited memory.

### 3.2.4 Communicational efficiency

The amount of communication that has to be done varies among protocols. Both the number of message passes, and the size of each protocol message must be considered. A protocol using many messages will be inefficient on a high latency network, while large messages will be a problem on low bandwidth networks. A communicationally efficient protocol uses fewer protocol passes and requires less data to be transferred.

The latency issue must not be underestimated. Users don’t like to wait
while computer systems work, and high bandwidth networks often have considerable latencies. Round-trip times above 0.5 seconds are not uncommon on the Internet today. If the protocol uses lots of passes this will quickly add up to a noticable delay for the user. Protocols designed to run over a low-latency network might make other considerations. The round-trip time of a local network is often in the sub millisecond range, and it can make sense to use a protocol with many message passes.

Keeping the amount of data transferred during each round of the protocol low is usually desirable. Most networks today have large capacities, and the capacity can be expected to increase with time. Some applications will still need protocols that are light-weight with respect to bandwidth requirements. For example in smart cards where the bandwidth between the computer and the card is limited.

### 3.2.5 Need for maintenance of state

Some protocols need to keep some per-session state for each connection, the amount of such state should ideally be kept as low as possible. If the number of connections are large keeping a lot of state information can become a serious burden. Session keys (see section 4.2.1) will almost always have to be stored as state information. It is important that the amount of state stored after an authentication protocol has been started, but before it has completed, is low. Otherwise denial of service attacks will be much easier to carry out. Imagine trying to start a lot of sessions without finishing the authentication. If the server has to store much state for each unfinished connection there is a danger that it will run out of memory.

### 3.2.6 Key control

The question of key control is important in protocols that exchange keys, or have key exchange as a side effect. If one party is able to choose the key, that party is said to have key control. There are three possibilities here. One is that any one of the clients chooses the key. The second is that a trusted third party chooses the key, and the third is that the key is computed mutually. When a key is computed mutually none of the parties is able to choose the final key.

The entity that has key control must be trusted to generate good keys. As described in section 4.4.1, key generation requires a good random number generator. Support for strong random number generation is becoming common in many operating systems. Nevertheless the key generation
3.2 Properties of protocols

process is a tempting point of attack, and moving the key generation to a trusted server will make these types of attacks much less likely.

If one of the parties chooses the key the number of messages needed can often be reduced. This assumes that the party choosing the key is competent. If none of the parties trust each other the only possibility is to use a protocol that computes the key mutually. Some cryptosystems have keys that are weak and should not be used. The question of which party will suffer if a weak key is established should be taken into consideration when choosing the protocol.

3.2.7 Impact of key compromise

When considering key establishment protocols it is important to analyze the impact of key compromise. A key compromise is something that should be avoided, but being aware of what consequences such a compromise has is important.

There are two types of compromise that should be considered. The first, and most serious is the compromise of a long-term key. The second is the compromise of a session key. Session keys are keys that are only used for a short period of time, typically for one communication session. long-term keys are keys that are active for a longer period of time, and are typically used for authentication and generation or distribution of session keys. More information on how keys are classified can be found in section 4.1. The compromise of a used key is much more serious than the compromise of an unused key. If the key has not been used it can be replaced with a new key, and no damage will have been done.

The compromise of a long-term key gives inspiration to the term perfect forward secrecy, introduced by Günther [18]. Perfect forward secrecy is a property of a protocol which assures that compromise of a single long-term key will not compromise past session keys. Even though past communications cannot be read, the compromise of a long-term key is a serious event and the key should immediately be taken out of service to prevent an eavesdropper to be able to read future transmissions.

If a session key is compromised and this makes it possible to either compromise future session keys or to carry out a successful impersonation, the protocol is said to be vulnerable to a known key attack.

If the probability of a session key being compromised is non-negligible the protocol being used should not be vulnerable to a known key attack. Session keys can often be more vulnerable to compromise than long-term keys.
They are often used with weaker cryptographic algorithms, and used for more data. This makes it more likely that the key should be found by cryptanalysis. Some protocols even deliberately uncover session keys after use.

### 3.3 Designing protocols

Several primitives are common in many protocols. They perform the same basic tasks, and if you look at many different protocols a pattern can be seen. The work of Fumy and Munzert [16] presents a modular approach to designing key distribution protocols. Many of the modules they present are general enough to be used not only for key distribution protocols, but in other protocols as well. The description given here is inspired by Fumy and Munzert, see their article for a more rigorous treatment.

When analyzing or creating a protocol it is important to keep in mind the assumptions that are made and the goals one wants to accomplish. By modularizing one can select the useful parts and by following certain rules be reasonably sure that the protocol is secure. The modular approach is not a substitute for the formal methods developed (an example is the so called BAN logic [5]), but an addition. The method is useful when designing a protocol from scratch, and when trying to understand existing protocols.

I will go through each of the generic requirements for cryptographic protocols and list some building blocks that can be used to accomplish the requirement.

#### 3.3.1 Data Confidentiality

To protect the confidentiality of a data item, encryption is used. The algorithm used can either be symmetric or asymmetric. In the symmetric case the protocol parties must share a key beforehand. When using asymmetric techniques the public key of the recipient is used.

#### 3.3.2 Modification Detection

The goal of modification detection is to enable the recipient of a message to detect unauthorized modification of a data item. The basic idea is that some redundant information computed using a cryptographic hash function is added to the message. The process must also involve some secret parameter to prevent forgery.
3.3 Designing protocols

First, one can concatenate the data with a MAC using a key shared between A and B: $D||MAC_{K_{AB}}(D)$. Instead of using a MAC one can encrypt the hash of $D$ with a shared key: $D||e_{K_{AB}}(h(D))$. If A has a public key-pair the secret key part can be used for encryption, allowing A to send modification protected messages to any recipient: $D||e_{K_{A}}(h(D))$. Modification detection can also be obtained without using encryption at all by calculating the hash of the data and a shared secret. By doing this a MAC is created using a hash function as a primitive as described in section 2.2.2. By encrypting the data concatenated with a hash, one can obtain both modification detection and confidentiality in one step: $e_{K_{AB}}(D||h(D))$.

3.3.3 Use of identifiers

Names of entities should be made explicit if the name of an entity is essential to the meaning of the message. Several flaws found in cryptographic protocols have been possible because messages have been re-used for communication with other parties than the intended recipient. The identifiers should be bound cryptographically to the message to prevent forgery. Abadi and Needham [1] mention this principle, and give several examples of protocols that have been victims to the lack of explicit naming of entities.

3.3.4 Replay Detection / Timeliness

Replay detection is used to prevent unauthorized copying of messages. It is closely related to timeliness which assures that a message is recent and not a playback of an earlier message.

In order to detect playback of a message and to ensure that a message is timely one needs to use some sort of challenge and response system. This system can either be explicit or implicit using for example timestamps. A replay detection code can either be a timestamp, a counter value or a random number. The inclusion of a replay detection code is denoted as $D||rdc$. Including such codes in a message does not make much sense unless they are protected by modification detection. If the data item requires modification detection the concatenation of the replay detection code and the data item must be protected from separation, for example by binding the items together by encryption. Just sending a time variant parameter unencrypted as part of a protocol message offers no added security. Anyone with access to the communication channel could easily forge such a parameter.

Time variant parameters are usually divided into random numbers, sequence numbers and timestamps. These terms are quite loosely defined
and the exact properties needed by a time variant parameter differ from one protocol to another. Such parameters are called *nonces, unique numbers* or *non-repeating numbers*. All these terms imply that the time variant parameter should be used only once.

**Random numbers**

When using random numbers the verifier must be the same entity as the provider of the parameter. This implies that random numbers can only be used in challenge response protocols using at least two messages. One example of use of random numbers is as follows. One entity sends a random number to another, the receiver then returns a message containing this number cryptographically sealed in such a way that the message could only be created with knowledge of the random number. The message received is then considered to be recent if the random number matches up with the previously generated number.

Choosing the random number must be done carefully to prevent an adversary from predicting the future numbers based on those that have been previously used. A random number generator does of course not guarantee that the same number is not generated multiple times, but if the range is large the probability of this happening will be very low. Many of the same issues apply to the generation of random numbers in protocols as when generating keys. The exact requirements depends on the particular protocol. See section 4.4.1 for more information on random number generators.

**Timestamps**

Using a timestamp as a time variant parameter is intuitively simple and straightforward. The clock increases monotonically and if the clock functions correctly each value occurs only once as long as the clock has sufficient resolution. When a message is sent, the time is read from the local clock and then included in the message. When the message arrives at the receiver the timestamp is compared with the receiver clock, and if the difference between the clock and the timestamp lies within a prespecified acceptance window the message is accepted as valid. The receiver can additionally check that the sender has not previously used the same timestamp. This can be done by keeping a list of previously used timestamps within the acceptance window or by only allowing strictly increasing values.

The problem with using timestamps as nonces is that the security depends
on having synchronized clocks all over the network. If an adversary can change the clock of the machine checking the timestamp he can cause old messages to be accepted as recent by turning the clock back. The problems involving depending on synchronized clocks is discussed in more detail by Gong [17].

A protocol using timestamps can always be converted to using random numbers and vice versa. Which solution to choose depends on a trade-off between less state and fewer protocol exchanges versus the burden of keeping the clocks synchronized in a secure manner.

**Sequence numbers**

Predictable sequence numbers can also be used as time variant parameters. It is easy to ensure that sequence numbers are unique, a simple counter is all that is needed. Sequence numbers are used just like random numbers in challenge response protocols. Because sequence numbers can be predicted the numbers have to be encrypted during transmission.

A timestamp can detect forced delays, where an adversary delays a message for some time. Forced delays are much harder to detect when using sequence numbers, and requires at least a protocol with two exchanges.

When sequence numbers are used, more state have to be kept on both sides. The use of sequence numbers should be recorded in such a way that valid numbers can be distinguished from invalid ones.

### 3.3.5 Entity authentication

Authentication protocols are used to authenticate the identity of parties, and will be described in more depth in section 3.4. Although authentication protocols are valuable on their own they are often also used as building blocks in protocols that are more complex. Some of the authentication protocols are quite simple and can be easily used as a part of another protocol. The most common forms of authentication used as components are the those that prove possession of a key.

### 3.3.6 Data Origin Authentication

To be sure that the source of some data is the one claimed data origin authentication is not needed. Data origin authentication does not necessarily
include modification and duplication protection, but data origin authentication is often implemented by combining sender authentication and modification detection. If the data being sent includes some redundant information, origin authentication can be obtained by encryption. This is basically authentication by proving knowledge of a key. Origin authentication can also be implemented by signing the data item with a cryptographic signature algorithm.

3.3.7 Proof of delivery

To be sure that a message was received correctly by the receiver proof of delivery is needed. For example a key distribution protocol often wants to provide key confirmation (see section 3.5.1). Using proof of delivery on the key data is one way of accomplishing this.

Proof of delivery of a key can be provided by encrypting some data using the key. If the data decrypts correctly proof of knowledge of the key is provided. The data must contain some sort of replay detection code to prevent replay of messages.

3.4 Entity authentication

Entity authentication protocols are used to assure one party of the identity of the other participant(s) in the protocol. The party that has its identity proven is called the claimant or prover and the other party is called the verifier. An important aspect of entity authentication protocols is that they are real-time. This is in contrast to message authentication protocols where no timeliness guarantees are made. The claimant has to be active during the protocol run to have its identity proven.

The entities that participate in the protocol can be both human beings and general purpose computers. The problem of authenticating a human is more complex because a human cannot be expected to be able to remember long cryptographic keys or to perform complex mathematical operations.

The goal of an entity authentication protocol is to either accept the identity of the claimant as authentic, or to terminate the protocol without acceptance. Handbook of Applied Cryptography [38] lists the following objectives for an authentication protocol:

1. If both the claimant and the verifier are honest parties, the protocol will complete with the verifier having accepted the claimant’s identity.
3.4 Entity authentication

2. The verifier cannot use the identification exchange with the claimant to impersonate the claimant with respect to another verifier.

3. The probability that a party $A$ can trick the verifier into believing that he is $B$ is negligible. The precise definition of negligible depends on the particular application.

4. The last point remains valid even if the adversary has observed a (polynomially) large number of previous authentications or has participated in previous authentications with any party. The adversary is also allowed to run multiple instances of the protocol involving any party at the same time.

The security of an authentication protocol can depend on several parameters, and the protocols can be divided into three groups according to these parameters. First of all the security can depend on something known. Examples of this are standard passwords, PIN-numbers and secret or private keys. The second possibility is to depend on something possessed, for example a magnetic stripe card or a hand-held password generating calculator. It can be argued that the devices possessed are just storage devices for cryptographic keys, and that there is little difference between this group and the first group. The last methods depend on something inherent, for example written signatures, retinal patterns, voice patterns and other biometric information.

Proving knowledge of a key is a common authentication method. If one party is known to be in possession of a key, and it is known that no one besides that party knows that key, the party can be authenticated by doing a cryptographic operation using the key. When public key techniques are used, proving possession of the secret key serves the same purpose.

There are several possible methods of proving key possession, all of them needing some kind of challenge and response. The challenge and response can either be an explicit random number or an implicit time variant parameter (for example a timestamp or a counter). The advantages of using implicit methods is that the number of protocol passes can be reduced. The problem is that these protocols assume some kind of synchronization. Depending on synchronized clocks can sometimes lead to security problems as discussed in section 3.3.4.

3.4.1 Properties of authentication protocols

Some properties must be considered before choosing which authentication protocol to use. I will discuss some common properties before describing concrete authentication protocols.
First of all the protocol can provide authentication of one of the participants in the protocol or both. In the former case the protocol is said to provide \textit{unilateral authentication}, and in the latter \textit{mutual authentication}.

Authentication protocols always involve some sort of secret information. How to store this information, and how it needs to be protected is influenced by the protocol. The information can for example be stored on a smart card, on a local disk, or on a piece of paper.

Different protocols can offer different kinds of security guarantees. Examples are provable security and zero knowledge properties (see section 3.4.2).

### 3.4.2 Strong authentication

Authentication protocols providing authentication without revealing any secret information to the verifier are called strong authentication protocols. In some cases the secret is known to the verifier in advance, but no information about the secret is revealed during the protocol run. The information sent over the communication channel is different with each protocol run, and an eavesdropper should not be able to retrieve any information that makes it possible to impersonate the claimant.

Strong authentication can be implemented using either public key algorithms, symmetric key algorithms, digital signatures or zero knowledge algorithms. This distinction is made by the ISO 9798 [27] standard for authentication protocols.

**Using symmetric key algorithms**

The authentication protocols using symmetric key cryptography corroborate the identity of the prover by proving knowledge of a shared key. The key can either be shared a priori, or it can be obtained from a trusted third party with which both parties share a key. The protocols can use timestamps, in this case only one pass is necessary. If random numbers or sequence numbers are used two passes are required. To provide mutual authentication one can always add one additional pass.

The encryption algorithms used in the protocols described below need to be able to detect forged or manipulated data. There are several ways to do this. If the encrypted data contains enough redundancy, it is enough to check that this redundancy is present after decryption. Another possibility is to concatenate the data with a message authentication code before
3.4 Entity authentication

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>$e_{K_{AB}}(t_A, B)$</td>
<td></td>
</tr>
</tbody>
</table>

Protocol 3.1: ISO-9798-2 one pass authentication

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r_B$</td>
<td>$e_{K_{AB}}(r_B) \leftarrow$</td>
</tr>
</tbody>
</table>

Protocol 3.2: ISO-9798-2 authentication with random numbers

encryption. It is advisable to use different keys for authentication and encryption. The initial key $K$ is therefore often used to generate two key variants $K'$ and $K''$, where $K'$ is used for encryption and $K''$ is used to calculate the MAC.

I will describe some example protocols taken from part two of the ISO 9798 standard [29]. The first example (protocol 3.1) is a simple protocol using only one pass and providing unilateral authentication.

$B$ receives the message and decrypts it. If the timestamp $t_A$ is recent, $B$ can be sure that the message originated from $A$, provided that $B$ is sure that only $A$ knows the key $K_{AB}$. The timestamp can be replaced with a sequence number, and in this case the sequence number is checked according to a policy agreed upon beforehand. Mutual authentication can be provided by adding one more step where $B$ authenticates to $A$ in the same way as $A$ did to $B$.

Protocol 3.2 uses random numbers and requires one more step to accomplish the same goal as the previous protocol. It should be noted that this protocol opens for a chosen plaintext attack. $B$ can send any number to $A$ and have it encrypted. To prevent this, $A$ can insert an additional random number in the reply as done in protocol 3.3. The identifier of $B$ in the last message is included to prevent reflection attacks (see section 3.6.5 and section 3.3.3). If this cannot occur the identifier can be removed.

To provide mutual authentication, one additional message can be added to the previous protocol, resulting in protocol 3.3. The respective parties

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r_B$</td>
<td>$e_{K_{AB}}(r_A, r_B) \leftarrow$</td>
</tr>
</tbody>
</table>

Protocol 3.3: ISO-9798-2 mutual authentication with random numbers
check that the random numbers they have generated are the same as in the encrypted messages. \( B \) will check that the random number received in the second message is the same as sent in the first. \( A \) will check that \( r_A \) received from \( B \) in the last message is the same as sent to \( B \) in the second, additionally \( A \) checks that \( r_B \) is the same as sent by \( B \) initially.

Using public key algorithms

Another class of authentication protocols use public key algorithms. The goal of these protocols is to demonstrate knowledge of the secret part of a public key-pair key held by the claimant. This can be done in two ways. Either by decrypting a challenge encrypted using the claimant’s public key or by signing a challenge.

Special caution should be exercised if the public key algorithm used is not resistant to chosen ciphertext attacks. An adversary can send as many challenges he likes and have them encrypted. If the algorithm has such problems, a random number (a confounder) should be inserted into the reply to make each message unpredictable.

In protocol 3.4 \( B \) encrypts a random challenge using the public key of \( A \) and sends it to \( A \). \( A \) proves its knowledge of the secret key by decrypting the random number and sending it back to \( B \). The hash \( h(r) \) over the random number is included to prevent chosen-plaintext attacks, and proves \( B \)'s knowledge of \( r \). \( A \) should only reply if the hash calculated over the received random number equals the hash received from \( B \).

The next protocol described uses public key signatures. A number of protocols using signatures are standardized in part 3 of ISO-9798. It is assumed that the verifier is in possession of the claimant’s public key. The verifier can either obtain the key a-priori from a key distribution center or a public key certificate can be included in the first protocol message. Public key distribution is covered in more depth in section 4.5.

Protocol 3.5 is similar to the one pass protocol in the symmetric key case (protocol 3.1). It provides unilateral authentication of \( A \) to \( B \) and does this with only one message. \( B \) checks that the timestamp is valid, that the identifier is its own and verifies the signature. As in the symmetric key case the
timestamp can be exchanged with a sequence number.

As was done when using symmetric keys, reliance on timestamps can be dropped and random numbers can be used instead. The cost is one additional protocol message. Analogously mutual authentication can be obtained by having $B$ generate a random number and include it in the reply, or reply with a timestamp. More details on this can be found in part 3 of the ISO-9798 standard [28].

**Using zero knowledge**

A very interesting class of protocols can be implemented using zero knowledge techniques. Let’s first describe what a zero knowledge protocol is.

The goal of an authentication protocol is to prove the knowledge of something that identifies the verifier. If one could trust anyone, the question could be as simple as: “Do you know the secret password?”. The reply could only be one bit, a “1” for yes and “0” for no. It would be nice if it were possible to design a protocol where no information other than whether the password was known or not was transferred. The verifier gains the knowledge that the prover knows something and nothing more. This is exactly what a zero knowledge protocol aims to accomplish.

A zero knowledge protocol is an example of an interactive proof system. An interactive proof system is a protocol where a claimant tries to prove the truth of a certain assertion to a verifier. The verifier can either accept or reject the proof at the end of the protocol. The proof is not a mathematical proof in the traditional sense, but an interactive game where proofs are probabilistic. By increasing the number of rounds in the protocol, the chances of a proof being incorrect can become arbitrarily close to zero. The protocol messages of an interactive proof system consist of several rounds where each round consists of the following three steps:

1. receive a message from the other party
2. perform a private computation
3. send a message to the other party
Typically the verifier will in each round generate a challenge, the prover will perform some kind of computation based on this challenge and send a response back to the verifier.

A zero knowledge protocol has the following properties:

**completeness** Given an honest verifier and an honest prover, the protocol succeeds with an overwhelming probability.

**soundness** If a dishonest prover can with non-negligible probability successfully execute the protocol with the verifier, then there must exist a polynomial time algorithm that can extract knowledge from the prover that will make it possible to subsequently successfully execute the protocol.

**zero-knowledge** There exists a polynomial time algorithm that, given the assertions that are to be proven, can generate transcripts that are indistinguishable from those resulting from a real protocol execution involving a live prover and verifier.

The completeness property says that the protocol works as intended when nobody tries to cheat. The soundness property is a little more involved. The essence is that being able to cheat in the protocol is equivalent to knowing the secret, or the secret can at least be derived in polynomial time. The protocol provides a proof of knowledge since completing the protocol successfully is equivalent to knowing the secret.

The zero knowledge property guarantees that the protocol does not give away any knowledge about the secret. Anyone can generate transcripts of a successful run of the protocol without knowing the secret. This also implies that someone looking at the transcripts gains no assurance whatsoever about whether the prover knows the secret or not. To gain this knowledge one must be involved in the protocol real-time.

To illustrate how a zero knowledge protocol works, let’s discuss a simple protocol. This protocol is called Fiat-Shamir [15], and involves a trusted third party in addition to the prover and verifier. Before the protocol begins the third party generates a RSA-like modulus; \( n = pq \), \( n \) is published while \( p \) and \( q \) is kept secret. The prover selects an integer \( s, 1 \leq s \leq n - 1 \), and computes \( v = s^2 \mod n \). This \( v \) is the prover’s public key, and is registered with the trusted third party. \( s \) is the private key and is what the protocol aim to prove the knowledge of.

The protocol is then run for a number of rounds, each round is as shown in protocol 3.6.
3.4 Entity authentication

<table>
<thead>
<tr>
<th>$A$</th>
<th>$B$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x = r^2 \mod n$</td>
<td>$e \in {0, 1}$</td>
</tr>
<tr>
<td>$y = rs^e \mod n$</td>
<td>$\leftarrow$</td>
</tr>
</tbody>
</table>

Protocol 3.6: Fiat-Shamir

$A$ chooses a random number $r$, squares it modulo $n$ and sends it to $B$, this number is called the commitment. $A$ commits to using $r$ in future computations. $B$ then selects a random bit ($e$) and returns this as a challenge to $A$. The response from $A$ is $rs$ if $e = 1$, or $r$ if $e = 0$. When $B$ receives $y$ he checks the congruence $y^2 \equiv x v^e \mod n$. There are two cases here, the first if $e = 0$. The reply from $A$ in the last protocol message is then $y = r$, $B$ computes $y^2 \equiv x \mod n$ and accepts the proof. If $e = 1$, $A$ replies with $y = rs$, $B$ then computes $y^2 \equiv x v \equiv r^2 s^2 \mod n$, and accepts the proof. It is $A$’s knowledge of $s$ that makes it possible to perform the computation of $y$ when $e = 1$.

A dishonest prover ($D$) could try the following procedure to try to trick $B$ into believing that he knows $s$. $D$ chooses by picking any $r$ and setting $x$ to be $r^2 \frac{v}{x}$. He will then be able to answer the question from $B$ in the case were $e = 1$ with $y = r$. For the case were $e = 0$ he would have to know the square root of $x \mod n$, without the factorization of $n$ this is a computationally hard problem.

The fact that the zero knowledge properties hold for the Fiat-Shamir protocol can be proven. The reader is referred to the original paper by Fiat and Shamir [15] for details.

### 3.4.3 Weak authentication

Protocols using conventional passwords are usually said to provide weak authentication. Passwords are usually rather short and consist of characters that are possible to type on a computer keyboard. This makes it easy for a human user to remember the password and enter it on the keyboard.

The authentication of a user usually proceeds as follows. The user is prompted for his user-name. The user-name is a system-wide unique identifier. He is then prompted for the password. The system finds the entry for the user in a database, and checks that the password is correct. If it is, the user is allowed access to the system.

The easiest way to check the correctness of a password is to store each password in plaintext in the user database. If the database is compromised,
all the user passwords will be compromised as well. To prevent this, the passwords are usually stored in encrypted form. I will describe one way of doing this that has traditionally been used in Unix systems [44]. In this scheme, only the user knows the password, the password itself is not stored in the system.

In the password database a one way hash of the password is stored. To check the validity of a password, the one way function is applied to the password supplied by the user and the result is compared to the entry in the database. If they match the user is authenticated. A password database of this type does only require protection from unauthorized writing. Unix password files are usually readable by all users of the system. To prevent users from discovering that they have equal passwords and to make it harder to mount a dictionary attack (see 3.6.3 below) a method called salting is used. In Unix 12 bits are taken from the system clock at the time of password creation and is used to alter the one way function. This salt is stored with the hashed password in the password file. The same password will therefore have 4096 different encryptions.

If a key is used for authentication and this key is derived from a password the authentication method is said to be weak. This is true no matter how strong the authentication protocol used with the resulting key is. Keys derived from passwords are often called passkeys, and the function used to derive the key is called a key derivation function. The most serious problem with this approach is that ideally the password has to have larger entropy than the resulting passkey, or it will be easier to do an exhaustive search of the password space than of the passkey space. Salting can be used in the same way as described above to make dictionary attacks harder. In addition one can make the key derivation process more time consuming, for example by iterating the same key generation function multiple times. The PKCS # 5 standard [55] describe key derivation functions and both salting and iteration. The Kerberos standard [34] contains information on how the key derivation in Kerberos systems is implemented.

Users often use the same password in different contexts, even though they are told not to. The same password used in two different contexts (for instance when authenticating to machines in different administrative domains), should not give the same passkey. When salting is not used some site-specific information should be included to prevent this. It is also beneficial that the password derivation function is one-way. If one passkey is compromised it should not be possible to derive other passkeys with the same password. If passwords cannot be derived from passkeys this becomes much harder.
3.5 Key exchange

We have seen that the public key cryptosystems promise to make it unnecessary to exchange secret keys. But in reality this is not so. The public key systems are much slower than conventional cryptosystems. Public key systems are therefore usually used to set up a key used for encryption with a conventional cryptosystem. There can also be reasons for avoiding public key cryptography altogether, for example due to patent problems. Protocols distributing secrets securely are therefore needed.

3.5.1 Properties of key exchange protocols

As was the case for the authentication protocols, key exchange protocols have several properties that needs to be taken into account when choosing the proper protocol.

A distinction is usually made between what is called key agreement protocols and key distribution protocols (also called key transport). In a key agreement protocol the two parties involved agree on a key without any one of them being able to choose a specific key. In a key distribution protocol one of the parties chooses the key and the purpose of the protocol is to transfer the key securely from one to the other. The party that chooses the key is said to have key control (see 3.2.6).

Rueppel and Oorschot [56] describe six important characteristics of key exchange protocols, and categorize several different protocols according to these characteristics. The first characteristic is the number of message passes required. Fewer passes give a simpler protocol that takes less time to execute.

They then consider the concept of authentication. Key exchange protocols usually provide authentication. The properties from section 3.4.1 therefore apply to key exchange protocols as well.

The concept of entity authentication is only concerned with the identity of the communicating parties, not whether they really received the key correctly or not. Key confirmation is the property that assures that a key really was received. Key confirmation can be both mutual and unilateral.

If a protocol provides assurance that no one, except for an identified second party could have acquired the key, the protocol is said to provide implicit key authentication.

The key freshness concept is concerned with whether the key derived during
a protocol run is guaranteed to be a new key that has never been used before. The freshness guarantee can either be provided by one of the parties (unilateral freshness control) or by the parties in combination (mutual control).

The last characteristic considered is the type of *public key certificate* used in the protocol. This can either be a conventional certificate, an identity-based certificate or a self-certifying certificate. More information on public key certificates can be found in section 4.5.1.

### 3.5.2 Example protocols

I will now discuss some of the most well known key exchange protocols. Diffie-Hellman key agreement is a protocol where none of the parties have key control, while Needham-Schroeder is a key transport protocol.

**Needham-Schroeder**

The Needham-Schroeder protocol uses a key distribution center as a trusted third party. A key is generated by the third party, and is distributed securely to the two communicating parties. The protocol was proposed by Needham and Schroeder [49] and has been an important inspiration for other protocols such as Kerberos (described in chapter 5). The protocol has one specific problem, and should therefore not be used in its original form, the problem will be described after the protocol itself.

The protocol involves a trusted party \( T \) and two parties \( A \) and \( B \). Both \( A \) and \( B \) share a key \( (K_{AT} \) and \( K_{BT} \)) with \( T \) a priori. The protocol provides authentication of \( A \) to \( B \) and key establishment with key confirmation.

In protocol 3.7 \( A \) first communicates with the trusted party in order to receive a session key for communication with \( B \). When \( T \) receives the message from \( A \) it generates a fresh session key \( k \). This key is then returned to \( A \) encrypted with the key shared between \( A \) and \( T \). The nonce \( N_A \) is included to prevent replays of old messages from \( T \) to \( A \). The copy of the key intended for \( B \) is included in the message encrypted under the key \( (K_{BT}) \).
3.5 Key exchange

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>$e_{K_B T}(k, A)$</td>
<td></td>
</tr>
<tr>
<td>$e_k(N_B)$</td>
<td>$e_k(N_B - 1)$</td>
</tr>
</tbody>
</table>

Protocol 3.8: Needham-Schroeder – second part

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha^{r_A}$</td>
<td></td>
</tr>
<tr>
<td>$\alpha^{r_B}$</td>
<td></td>
</tr>
</tbody>
</table>

Protocol 3.9: Diffie-Hellman key exchange

shared between $B$ and $T$.

$A$ can now continue the protocol as seen in protocol 3.8, communicating directly with $B$. $A$ first sends the encrypted session key received in the first step to $B$. $B$ decrypts this key upon receipt and sends $A$ an encrypted nonce ($N_B$), this number serves as a challenge to $A$. $A$ replies with $N_B - 1$. This last two messages provide entity authentication of $A$ to $B$. If the nonce $N_B$ contains enough redundancy, $A$ can check this and obtain entity authentication of $B$.

The problem with this protocol is that $B$ has no way of knowing that the session key $k$ is fresh. If an adversary gains access to an old session key, he can impersonate $A$ by replaying the first message in the second part of the protocol and (because he knows $k$) compute a correct response to the challenge from $B$ in the last message. Kerberos solves this by limiting the validity of specific session keys to a certain time-interval.

Diffie-Hellman Key agreement

The goal of the Diffie-Hellman protocol is to let two independent parties compute a key jointly such that only the two parties know the key after the protocol is finished.

Before the protocol starts, the parties must agree on some non-secret system parameters $p$ and $\alpha$. $p$ is a prime and $\alpha$ is a primitive element of $\mathbb{Z}_p$. $A$ and $B$ then generate two separate random numbers $r_A$ and $r_B$.

The protocol description of Diffie-Hellman can be found in protocol 3.9. Upon receipt of the message from $A$, $B$ computes $K = \alpha^{r_A r_B} \mod p$, and similarly $A$ computes $K = \alpha^{r_B r_A} \mod p$. They now have access to the same key $K = \alpha^{r_B r_A} \mod p$. 
Protocol 3.10: Man in the middle attack

In this form the protocol is vulnerable to a man in the middle attack. Let’s see how this vulnerability can be exploited. To be able to carry out this attack the attacker must have the capability to intercept messages between A and B and replace them with new ones. Let’s call our interceptor E. The protocol can be seen in protocol 3.10.

E intercepts the message $\alpha^r_A$ from A and replaces it with a new message where he knows the random number. When the message from B arrives, it too is changed with a new message. After the completion of the protocol E is in possession of one key $\alpha^r_A\alpha^r_B$ used for communication with A and one key $\alpha^r_A\alpha^r_B$ used for communication with B. When a message from A arrives E can decrypt it and re-encrypt it before sending it to B.

The earlier mentioned problem with this protocol, is that A has no guarantee that it is speaking to B and vice versa. The protocol provides no entity authentication or key authentication. The protocol can be easily adapted to provide entity authentication using a signature algorithm.

### Station-to-station protocol

Protocol 3.11 is building on Diffie-Hellman and provides mutual entity authentication and explicit key authentication. The difference from the ordinary Diffie-Hellman protocol is that after receiving the first message from A, B replies with $\alpha^r_B \mod p$ and the two numbers $\alpha^r_B, \alpha^r_A$ concatenated and signed using the signature key of B. This message is encrypted using the freshly generated session key $k = \alpha^{B\cdot A} \mod p$. After generating the session key, A can decrypt the message and check that the signature belongs to B. To do this A will have to possess the signature verification key of B, this can either be previously obtained and verified, or downloaded from a certificate storage server. Another option is to include B’s certificate in the protocol exchange. At last A replies with the same two numbers signed using its signature key. B can then go through the same verification that A did to become sure about A’s identity.

By adding one extra protocol pass and some expanded messages, additional requirements can be met. Data origin authentication of the random exponents is obtained using techniques described in section 3.3.6.
3.6 Ways of attacking protocols

Although the threats that can lead to a protocol failure can be subtle and difficult to identify, known threats can be divided into various categories. The most obvious is called a **passive attack**. In this case the attacker does not take part in the protocol but only listens in on the communication and tries to derive secret information from what he can hear of the communication. An **active attack** is more powerful, it allows the attacker to not only listen to the traffic but to actively participate in the protocol. For example by inserting rogue messages or pretending to be someone else. There are many possibilities. I will now give an overview of attacks that have been described in the literature.

### 3.6.1 Known key attack

If an adversary gains access to a previously used key it will be possible to mount a **known key attack**. Neither keys in use, nor old keys should normally be compromised. It is nevertheless beneficial for a protocol to be resistant to these types of attacks. It should not be possible for an attacker to derive new keys using knowledge of old, obsolete keys.

### 3.6.2 Impersonation

An attack where the adversary assumes the identity of another party is called an **impersonation attack**. There are usually mechanisms in the system to prevent such attacks, so a successful impersonation will probably use other types of attacks to subvert the authentication mechanisms.

### 3.6.3 Dictionary and forward search attacks

Dictionary attacks are most commonly mounted against authentication protocols or authentication systems using passwords. Instead of guessing all possible passwords and trying them against the system, only words
from a dictionary are tried. While the number of possible passwords can be rather large, users tend to choose passwords that are easily remembered. If passwords are 8 characters long and consist of lower case characters only, the number of possible passwords is $26^8 = 208,827,000,064,576$. A typical dictionary contains only 150,000 words. It is therefore self-evident that trying words from a dictionary is a smart strategy of attack.

The encrypted password file can often be obtained. Either directly through the file system as with the standard Unix `/etc/password` file or encrypted password entries can be collected over time by listening in on network traffic. The dictionary can be encrypted in advance and compared rapidly to the collected passwords.

One common way of preventing password guessing attacks is to routinely run a password cracking program on the local password file. If any passwords are found, the passwords are changed by the system administrator.

Closely related to the dictionary attack is the *forward search attack*. Carrying out a forward search attack involves precomputing encrypted data, using this to get around the protocol. This attack is identical to the attack described in section 3.1, where a low entropy data item is encrypted using a public key cryptosystem.

### 3.6.4 Replay attack

A replay attack is an attack where the communication of a previous protocol run is recorded and replayed later in malicious intent. The replay can be played out against the original recipient or against a completely different target.

Replay attacks can be avoided by including a time variant parameter in the protocol as described in section 3.3.4.

### 3.6.5 Active attacks

What I choose to call *active attacks*, are attacks where the attacker participates in the protocol and tries to subvert the protocol. I will describe several well known attacks of the active type.

**Reflection attack**

In a reflection attack, the attacker sends information back to the party carrying out the protocol. The information is sent directly back without change.
3.6 Ways of attacking protocols

The classic reflection attack, originally described by Mitchell [42] proceeds as follows.

Let's first describe a flawed protocol that is vulnerable to a reflection attack (protocol 3.12). A and B share a symmetric key and they want to prove knowledge of this key to each other by encrypting a random challenge.

As can be seen in protocol 3.13, the adversary E can impersonate B without knowing the key k by reflecting what A sends back to A. E first reflects A's challenge $r_A$. When A receives this challenge he thinks that this is the start of a new authentication session. A therefore encrypts the challenge concatenated with a new challenge $r'_A$ and sends this back to E. E will now reflect this as the answer to the original challenge $r_A$ from A.

There are two simple solutions to this problem. First, A and B can use separate keys for each direction. Adding an extra element identifying the party sending the message will enable the receiver to detect the attack.

**Man in the middle**

The man in the middle attack is an active attack that requires that the adversary is able to remove, modify and insert generated messages. The most basic active attack is the man in the middle attack that was described for the Diffie-Hellman key agreement protocol discussed in section 3.5.2. Other more elaborate schemes are possible.
Protocol 3.14: Protocol vulnerable to an interleaving attack

<table>
<thead>
<tr>
<th>A</th>
<th>E</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>( r_A )</td>
<td>( r_{A} )</td>
<td>( r_{B}, \text{SIGN}<em>B(r</em>{B}, r_{A}, A) )</td>
</tr>
<tr>
<td>( r_{B}, \text{SIGN}<em>B(r</em>{B}, r_{A}, A) )</td>
<td>( r_{A}', \text{SIGN}<em>A(r</em>{A}', r_{B}, B) )</td>
<td>( r_{A}', \text{SIGN}<em>A(r</em>{A}', r_{B}, B) )</td>
</tr>
</tbody>
</table>

Protocol 3.15: Interleaving attack

**Interleaving attack**

In an interleaving attack the attacker participates in several executions of the same protocol at the same time. Let’s illustrate the concept with an example. I will first describe the flawed protocol 3.14.

The random numbers provide freshness, while the signatures provide entity authentication. The problem is that \( E \) can initiate a protocol with \( B \) and pretend to be \( A \). \( E \) will then initiate a new run of the protocol with \( A \) and use those messages to complete the protocol with \( B \) thereby tricking \( B \) into believing that \( E \) is \( A \). This can be seen in protocol 3.15.

The problem is that the messages containing the signatures are symmetric. This enables \( E \) to complete the protocol without having to fake any signatures. If \( r_{A} \) was used instead of \( r_{A}' \) in the last message the problem would be avoided. Another solution would be to add a sequence number to the messages, this would allow \( B \) to detect that the last message from \( E \) was not intended to be used as a last message, but rather as a response to the first protocol message. This is an important principle, a protocol message should only be allowed to be used for its intended purpose in the protocol.
Chapter 4

Key Management

In the previous chapter I discussed cryptographic protocols and among these key exchange protocols. These protocols transfer keys securely from one party to another. But this is not enough. All these protocols assume that the initial cryptographic keys are already in place. How to initialize these keys and how to properly manage keys through the lifetime of the system is the job of a key management scheme.

In part two of the OSI security architecture [24], key management is defined as “The generation, storage, distribution, deletion, archiving and application of keys in accordance with a security policy”. Key management is not a component that exists in one specific part of the system. As indicated in the definition, key management will influence all parts of the security system dealing with cryptographic keys. The security policy of the system will in turn influence the choice of how to manage the keys. A security policy is the criteria for the provision of security services.

It is not hard to understand that proper key management is crucial to a security system. If the keys are compromised, all other security mechanisms will be worthless. Just consider how ordinary physical keys are managed in most organizations. There usually is some kind of system that keeps track of who is in possession of each key, and each individual is expected to report if a key is missing. What to do in case a key gets lost depends on the security policy of the organization. If security is important locks will have to be changed, and new keys will have to be issued. Management of cryptographic keys should be similar.

The implementation of key management schemes are mostly influenced by practical experience, as opposed to the theoretical approach used in most other parts of cryptography. Key management schemes depend on
the types of keys managed, the applications that are to be supported and the environment in which the system is to be employed.

### 4.1 Classification of keys

Cryptographic keys can be classified according to the cryptographic service they are intended to be used with. It is tempting to classify the keys according to the type of algorithm they are intended to support. This could for instance be symmetric keys used for conventional cryptography, and private and public keys for asymmetric cryptosystems. This classification is of course correct and needed, since an asymmetric key cannot be used for a symmetric cryptosystem. However keys are usually classified in an even more fine grained manner.

There are several rules that should be enforced when dealing with cryptographic keys, for example a key used for encryption should not be used for making signatures. A key used for authentication should not be used for normal data encryption. Each cryptographic objective should have a separate key. Examples of cryptographic objectives could be confidentiality, data origin authentication, key agreement and entity authentication. Each of these objectives could be met by using both symmetric and public key algorithms.

The proper management of a key depends very much on which class a key belongs to. Loosely speaking a private key should be protected very rigidly and no one but the rightful owner of the key (and perhaps some trusted third parties) should have access to it. A public key should be distributed widely and should be integrity protected. Symmetric keys should only be known by the parties communicating using that particular key.

### 4.2 Key protection

In any system involving cryptographic keys, the keys are the most sensitive data in the system. They are even more sensitive than the actual data stored in the system. Every measure available must therefore be employed to protect the keys.

Protection of key can be done by cryptographic methods. Encryption protects keys from disclosure and unauthorized used. Integrity check functions can be used to stop modification. Non-cryptographic methods can also be used. Time stamps are used to limit the use of a key to a specified
4.2 Key protection

period. Physical protection of keys is also an important aspect of key protection.

4.2.1 Key layering

One key can be used to encrypt another. This suggests a hierarchy in which certain keys are used to encrypt other keys. The keys on the lowest levels are in turn used to encrypt data. These concept were first introduced by Ehrsam, Matyas, Meyer and Tuchman [12].

**master keys** These keys are at the top of the hierarchy. They are not encrypted, and have to be protected by other means.

**key-encrypting keys** These keys are used for the transport and protection of other keys. Key encryption keys are used in key transport protocols to encrypt data keys during transport.

**data keys** These keys are used to encrypt the actual data sent over the wire. They are usually used only for a short period of time. Data keys used for instances of communication sessions are called *session keys*.

The compromise of a master key is very serious. All keys protected by that key will also become compromised. Master keys will therefore have to be very carefully guarded and they should not be distributed widely. Protection in hardware should be considered, and the keys should be used as little as possible.

There are several advantages in using a key layering scheme. First of all the number of keys that have to be manually distributed and installed is reduced. Only a small number of key-encrypting keys or master keys will have to be installed on each computer, the rest of the keys can be encrypted and sent directly over the communication channel. Key layering can also be used to limit the damage in case of the loss or compromise of a key. If a session key is lost, only the data encrypted using that particular key is affected, data encrypted using other keys is unaffected.

System administrators have much power over computer systems. These administrators cannot always be trusted totally. While the administrator will most likely have to access most of the keys, a key layering scheme can reduce the risks. Only a very limited number of people should have access to actual master keys, and the knowledge of these keys could even be distributed among several people. For example by dividing the key or using secret sharing schemes.
Asymmetric keys are often used for authentication. The security of the authentication is never stronger than the security of the private part of the key. The private key could be stored on a smart card as described in section 4.2.4, or the key could be encrypted with a key derived from a password typed in by the user. The encrypted key is stored on disk and unencrypted when it is needed.

### 4.2.2 Cryptoperiods

The cryptoperiod of a key is the time in which the key is valid for use. Cryptoperiods are used to limit the information available for cryptanalysis, and to limit the damage if a key gets compromised. Cryptanalytic attacks become easier if more encrypted information is available. Cryptoperiods are enforced by changing keys on a regular basis.

Keys are usually classified as long term or short term keys. Master keys, key-encrypting keys and keys used for key agreement are called long term keys. Session keys are called short term keys. Keys used for encryption of data communicated over insecure channels are usually short term keys while long term keys are used to encrypt stored data.

A key will often have to be stored and protected for a much longer time than the cryptoperiod of the key. How long depends on the confidentiality requirements of the data encrypted with the key.

### 4.2.3 Controlling key usage

There is little use in classifying keys according to the intended usage and dividing keys into layers if there are no procedures in the system which ensures that keys are used only for their intended purpose. The same holds for controlling cryptoperiods.

Handbook of Applied Cryptography [38] lists the following information as relevant to store in connection with a key.

1. owner of key
2. validity period
3. key identifier
4. intended use
5. specific algorithm
Incorrect use of keys can lead to security flaws. In a system that employs cryptographic services in a trusted component (usually in hardware), unencrypted keys are usually unavailable to the software. This is ensured by the trusted component. For data the case is different, the software can ask the component to decrypt data and the plaintext will be returned. If the component could be tricked into using a data key instead of a key encryption key for key encryption the software could easily decrypt the key by having the trusted component decrypt it as data.

Several methods have been described in the literature to provide key separation that will avoid the problems described above. The simplest method is called *key tags*. A key tag is a bit-string that describe the key. The bit-string is encrypted concatenated with the key using a key-encryption key. The same bit-string is decrypted at the same time as the key. The trusted component performing the cryptographic operations can decide whether the key is valid or not. An obvious shortcoming of this scheme is that it is not possible to gain access to the key tag without knowing the key-encrypting key.

Another possibility is to use different key-encrypting keys for each type of key. These key variants can be derived from a single master key by using non-secret parameters and a non-secret function. This method is called key-offsetting. It is important that the process of key derivation is non-reversible. If a derived key is compromised and the key derivation process is reversible, the derivation key will also become compromised. The concept of using master keys and key variants to separate different types of keys was introduced by Ehram, Matyas, Meyer and Tuchman [12].

The method accepted today to be the most flexible way to separate keys is *control vectors*. A control vector is a set of data that is stored with the key and bound cryptographically to the key. This binding ensures that one control vector can not be replaced with another. Control vectors work by exclusive-ORing a hash of the control vector with the key encryption key before encrypting the key. When a key is to be decrypted, the specified vector is again XORed with the key encryption key and used as the key to the decryption algorithm. If the vector used for encryption of a key differs from the one used for decryption the key will not decrypt correctly. For more details on control vectors see the article by Matyas [37].
Control vectors have several advantages compared to the other alternatives. First of all the length of the control vector is not limited. Secondly the control vector can be inspected without decrypting the key. This is a great advantage to application programs and other untrusted components handling keys. It is also easy to add new fields to the vector without changing the whole system (only the fields that are relevant are checked).

### 4.2.4 Physical protection

Cryptographic methods can protect keys during storage or transport. When it comes to protection during actual use, no cryptographic techniques can help. The key has to be decrypted before usage. The use of the key can be inside a general purpose computer using a software encryption algorithm, or it can be inside a special purpose cryptographic processor.

In military applications physical protection of keys have traditionally been something that have been given much consideration. Commercial systems have not seen these aspects as very important. There have been some developments, most notably the FIPS-140-1 U.S federal standard [50]. This standard is currently being revised.

FIPS-140 describes requirements for cryptographic modules. A cryptographic module is hardware and/or software that implements cryptographic functions. The module is contained within a physical boundary. A cryptographic module can for example be a plug-in card for a computer or a smart card. The standard describes methods for designing tamper-proof systems and divides products into four security levels.

Smart cards are often seen as being a solution to the key storage problem. If a key can be stored on a small card the size of a credit card, the card can be used as a token allowing access. To prevent disasters when a smart-card is lost, there has to be a mechanism to authenticate the user to the smart-card. The authentication mechanism can for example be a PIN-code.

If every cryptographic operation is done inside a cryptographic module, there is no reason why the keys should ever leave the module in an unencrypted form. Master keys are the only exception. They should be loaded into the module at system initialization, for example by using a smart card. When the module is initialized, the smart card could be stored in a safe. Loading of master keys should not be legal during normal operation, and the optimal solution is to use a specific interface that is only used for this purpose.

Where an unencrypted key is should be known at all times. Is the key in
memory? Is it on a disk? Is it in transit over a network? Modern operating systems use virtual memory, and virtual memory systems swap unused parts of the computer memory to disk. If a key is stored in memory and the power is removed the key is destroyed. If the key is swapped to disk it can easily be recovered after a power failure. Designers of cryptographic software should be aware of these issues and take precautions to avoid them.

4.3 Key life cycle

In addition to controlling what a key is used for and where in the system it is stored, it is necessary to control the period in which a key is in use. This gives birth to the concept of key life cycle. A key is brought into use at a particular time, and will at a later time be removed from use. This is an important part of enforcing cryptoperiods.

A key goes through four states during its life cycle. Each of these states consists of several different sub states.

1. Pending active
2. Active
3. Post active
4. Obsolete

The key is in the pending active state before it is used for cryptographic services. This phase includes tasks such as generating the key and registering the entity owning the key to the system.

In the active state the key is registered with the system and installed before it is used for the intended purpose. When the key is taken out of normal service it enters the post active state. If local policy mandates archival of old keys the key is now archived. An archived key should no longer be used, but is still available should it be necessary to recover it for special purposes. In the obsolete state all records of the key are destroyed. The first three states are special, because in these states the key material is present in the system and has to be protected. A diagram illustrating the flow between the states can be found in figure 4.1.
4.4 Services provided by a key management system

The key management system will provide several services ensuring the proper management of keys. The key management system should ensure that the local key management policy is followed. It should not be possible to break the policy by misusing the key management services.

I will describe the key management services as specified by the ISO-11770 key management standard [22].

4.4.1 Generate key

The first step is the key generation. Secure key generation involves picking a random number in an unpredictable manner. Each cryptographic algorithm has special requirements for the keys, but they all have one thing in common; the key generation process should not have any bias, every legal key should be as likely as another. Some keys might be considered to be illegal by the algorithm. DES for example has several weak keys with unwanted properties that should be avoided. The key generation process often involves operations such as generating large random prime numbers. These operations can be computationally quite intensive.
As have been said several times before the security of any cryptographic system lies in the security of the keys. The key generation process is the source of new keys. Weaknesses within the key generation process will propagate throughout the entire system and subvert the security. The key generation process should therefore generate keys that cannot be guessed easily by attackers.

Generating random numbers using a highly predictable computer is a contradiction in terms. Numbers generated by computer algorithms are therefore called pseudo-random numbers. The algorithm used to generate pseudo-random bits for pseudo-random numbers is called a pseudo-random bit generator. A pseudo-random bit generator uses a small random bitstring as input (called the seed) and generates more data with a random distribution. The random output can be tested using various statistical tests for randomness. Knuth's volume two of the Art of Computer Programming [32] describes pseudo-random number generators in depth.

Statistical randomness is not enough when it comes to generating cryptographic keys. What we are looking for is unpredictability. A cryptographically secure pseudo-random bit generator (CSPRNG) is a random bit generator where there is no polynomial time algorithm which given the first \( l \) bits of a random sequence can predict the \( l+1 \)th bit with probability greater than \( \frac{1}{2} \). When generating keys cryptographically secure random bit generators should be used. An even better solution is to use external hardware that can generate random numbers. Such hardware can easily be designed using thermal noise, or a radioactive decay source. At the moment such hardware is not common, but RFC 1750 [9] gives more information on how to generate random numbers with computers using various sources of randomness existing in a modern computer.

### 4.4.2 Register key

After the key is generated it has to be registered with the system. Registration is especially important for keys used for asymmetric cryptography, but some symmetric keys will often have to be registered as well. The entity conducting the registration is called the registration authority.

The registration process involves creating a binding between a cryptographic key and a system entity. This entity can be a specific machine, a process or a human being. When a key is registered, it is of utmost importance to check correctness of the binding. If the registering entity is a human being, this can for example be done by physically presenting some kind of proof such as an identity card or driver’s license. In small organizations such arrangements can be implemented quite easily, but as the size of the
organization grows the administrative burden involved with key registration can become very large.

4.4.3 Create key certificate

For public keys a certificate that binds the public key to the owner of the private key part has to be made. This operation is carried out by a certification authority. A key certificate should only be created if the certification authority is absolutely sure that the key belongs to the user it claims to. There is much more involved in generating a certificate than creating a digital signature over some keying data. The key registration process is the crucial part here. A strong link exists between the registration process and the generation of the certificate. More about digital certificates and public keys can be found in section 4.5.1.

4.4.4 Distribute key

After the key has been registered and a certificate has been created the key can be distributed throughout the system. Several protocols for key distribution have been described in chapter 3.

It is important to note that the key distribution does not have to be carried out immediately after the key has been created. It is often the case that the key is stored somewhere (for example at a publicly available keyserver), and is distributed when the need arises.

4.4.5 Install key

Installation of the key is needed before it can be used for cryptographic purposes. If special cryptographic hardware is used, this step will involve loading the key into the hardware and possibly decrypting the key before it is stored in some kind of protected memory. The key should always be guarded closely to protect it from getting compromised.

4.4.6 Store key

A key often has to be stored for a period of time. The key store should be easily accessible, and it should ensure that no unauthorized parties gain access. A key can be stored in all but the obsolete state.
4.4 Services provided by a key management system

4.4.7 Derive key

Keys are often derived from other keys for various reasons. This service uses the methods for key derivation as described in section 4.2.3.

4.4.8 Archive key

After a key has been taken out of normal use it might still be needed. To check a signature the public part of the signature key must be available, usually for many years, after the signature key has been taken out of normal use. A key archive is not very different from a key storage except that keys are intended to be stored in an archive for much longer time. Key archives do not have the same accessibility requirements as normal key storage and might use different implementations such as off-line storage on magnetic tape.

4.4.9 Revoke key

If a key is known to be compromised it must not be used any more. The key revocation service will assure this. When a key is revoked it must not be used for encryption or signing. Decryption and checking of signatures might still be acceptable.

There are several strategies for handling revoked keys. If the revoked key resides in a key distribution center and clients contact this center to obtain keys, the center can stop sending out the revoked key. For public keys certified by a certification authority, it is common to maintain a list of revoked keys. This is further explained in section 4.5.3.

4.4.10 Deregister key

The deregister key service removes the binding between the key and the entity it is associated with. This service is carried out by the registration authority and is a part of the key destruction process.

4.4.11 Destroy key

This service removes all records of a key. If copies of the key is kept off-line they will have to be deleted as well. Before a key is destroyed it must be checked that no stored material protected by this key is needed again.
4.5 Public key infrastructures

The management of public keys is different from the management of cryptographic keys used with symmetric cryptosystems. The private part of the key is managed and protected in much the same ways as ordinary secret keys. The purpose of the key management scheme is to protect the key from compromise. For the public part of the key the situation is completely different. The key should be distributed as widely as possible while still maintaining the integrity of the key. A system designed to accomplish this goal is called a Public Key Infrastructure, or PKI for short.

Public Key Infrastructures have received a lot of attention lately. This is no surprise, the growth of the Internet has fueled an enormous interest in electronic commerce. To secure commercial transactions transmitted over the Internet, public key cryptography is the only realistic solution. To be able to use public key cryptography one needs to obtain an authentic copy of the public key of the recipient. In small applications it might be enough to simply ask the recipient for the key, and ascertain the authenticity through non-electronic means. If you receive the key on a diskette personally from the recipient, you can be reasonably certain that the key on the diskette belongs to the person you received it from. This approach does not scale at all, more flexible and scalable methods are needed.

4.5.1 Public key certificates

To store and transmit public keys over insecure channels public key certificates are used. The certificates protect the key from manipulation. The certificate consists of two parts. The data part and the signature part. The data part contains the public key and a string identifying the entity that holds the private part of the key. The signature part is a digital signature over the data part, effectively binding the name to the public key. The data part can contain additional information, for example the validity period of the key, serial numbers or the identity of the party signing the key. The certificate is effectively a document saying “This key belongs to entity X, signed Y”.

To enable interoperability between cryptographic systems developed by different software vendors, ISO have standardized a format for public key certificates in the ISO-9594 standard (more commonly known as X.509) [25].
4.5 Public Key Infrastructures

4.5.2 Certification Authorities

The entity performing the actual signing of public key certificates is called the Certification Authority, or CA for short. The CA is responsible for checking that the key belongs to the entity listed in the certificate, and then carrying out the actual signing. The CA vouches for the identity of the party identified in the certificate. This identity is represented by a unique name in the system (a distinguished name), X.500 distinguished names are commonly used. How the CA checks the identity of the entity requesting the certificate is a matter of the policy of the CA. Different CAs can enforce different policies. A stringent policy could be to require the entity to show up in person and present identification papers. In the other end of the specter the authority could register entities by e-mail address and only check that the address is valid and that there is a sane response when mail is sent to the address.

The public key of the CA must be distributed to all entities in the system as a part of setting up the system. By using this key, the signature on the certificates can be checked, and thereby the validity of the binding between the public key and the entity.

Before the CA can sign the data part of the certificate, a key must be generated. The key can either be generated by the CA, and the private key transferred to the owner over a secure channel, or the key can be generated by the owner and the public part transferred to the CA for signing. In the latter case the CA must check that the entity listed in the certificate is in possession of the private part of the key. This prevents cheating by trying to obtain a certificate on the public key of another party. If the CA generates the key, more trust must be placed in the CA by the owner of the key. The CA will in this case have access to the actual key, and will be able to generate valid signatures and messages looking like they come from the owner of the key.

4.5.3 Certificate Revocation

There has to be a way of undoing the signing of a public key. The private part of a public key can for example become compromised, or it can simply be lost. The certification authority therefore maintains a list of certificates that are revoked. This list is called the Certificate revocation list (CRL). The CRL is usually issued at regular intervals, and is signed by the CA to prevent rogue parties to issue manipulated lists. It is the duty of the user of a public key to check that a key is not revoked before usage. To prevent the CRL to grow indefinitely, all certificates contain a validity period. There-
Before the revocation list only has to contain revoked certificates that are still valid according to the validity period.

Because the revocation lists are issued at regular intervals, there is a possibility that a key is compromised immediately after a new revocation list is issued. In this case it will be possible to use the compromised key until a new revocation list is issued. To reduce the problem new, revocation list can be issued more frequently, or the list can be issued each time there is a change.

4.5.4 Certification hierarchies

Having one central Certification Authority that certifies the public keys for all entities all over the world is clearly infeasible. This is both a question of whether people and organizations want to trust such a central authority and a question of scalability. Therefore that more than one CA will be needed on a world wide basis. Certification can be carried out by national bodies, for example the postal organizations. Large international organizations will probably want to certify the keys they use and avoid being dependent on other organizations. A problem then arises when a user wants to encrypt a message and send it to a person that he does not share CA with. One possibility is to obtain the public key of the CA that has signed the key of the other entity, but then the key could have been directly obtained in the first place. If the number of certification authorities is large we run into the same kind of scaling problems that triggered the invention of public key certificates in the first place.

The problem can be avoided by having one CA sign the public key of another CA. For example if CA $A$ signs the key of CA $B$, then $Y$ can obtain and trust a certificate belonging to $Z$ if $Y$ holds the key of $A$ and $Z$’s certificate is signed by $B$ and $A$ has signed the key of $B$. In this setting a chain of trust is established from $Y$ to $Z$ through the CAs $A$ and $B$.

To improve scalability, certification authorities can be organized into a tree-like hierarchy. An example of such a hierarchy is given in figure 4.2. The certification authority at the root of the tree is called the root CA. Each CA will sign the certificates of the CAs below it in the tree. The key of the root CA is certified by no one besides the root CA itself, such keys are called self certified. Each user is then given the key of the root CA instead of the key of the CA immediately above it in the hierarchy. Certificates can then be checked by checking all the certificates from the root and down to the certificate for the user.

Another possibility is to give each user the public key of the CA directly
Figure 4.2: A certification hierarchy

above him. In this model each CA signs both the certificates of CAs below and above it in the tree. To verify a certificate, the signature on the certificates of the CAs up to the least common ancestor and down to the recipient is verified.

It would be desirable, from a technical viewpoint, to have one root CA for the entire world. This CA could for example be operated by an independent international organization, for instance the United Nations. As described above, this approach is not feasible for political reasons. A more realistic approach is to have several different roots, and have the root CAs certify each other. This approach is called cross certification.

It is important to remember that public key certificates are a method for transferring trust. When a CA signs a key, the trust that the CA has in the validity of the key is transferred to all the trusting the CA. When using certification hierarchies, the root CA effectively delegates the responsibility of certifying entities to others. The policy that a CA uses to decide whether to accept a certification request or not is therefore of extreme importance. A CA must both accept the policy and the competence of another CA before deciding to accept certificates issued by the other CA.

4.5.5 Webs of trust

Certification hierarchies and certification authorities imply some sort of central control over the certification process. For some individuals this is not acceptable, perhaps because they detest central authorities for political reasons, or because they are extremely selective about whom they trust. Another problem with certification hierarchies is that they take a lot of resources to set up. To be of any use, a certification hierarchy must be wide spread; if the party with whom you want to communicate does not have a registered key you are out of luck. In many situations there is no available certification authority. Setting one up yourself is usually not an alternative.
The PGP cryptographic software [65] was designed to operate in environments where there are no central authorities that can act as a CA. PGP therefore uses a model called a web of trust. In this model every principal in the system can act as a certification authority. All users have the possibility of signing other keys. If you receive a key from a friend, and make sure that the key really belongs to your friend, you can vouch for this association and create a certificate. You can then choose to accept your friend as an introducer of keys. This means that if your friend has signed the key of some other person you can trust that this key is authentic. The amount of trust you have in some other person to act as an introducer is divided into tree levels; complete, marginal or untrusted. The acceptance of a key requires either one completely trusted signature or two marginally trusted signatures (these are the default values, they can be changed by the user).

A web of trust is a more general model than the purely hierarchical certification tree. Certification trees are a subset of webs of trust. A certification tree is a web of trust where some principals use their keys for signing only, effectively acting as certification authorities. In addition keys used for ordinary encryption is only found at the leaf nodes and there is a root key that is signed only by itself.
Chapter 5

Kerberos

After having devoted the last chapters to general descriptions of cryptographic protocols and algorithms it is now time to move on to describe a concrete implementation.

I will start by introducing the history of Kerberos. After that the Kerberos architecture and the protocols are described. The last part of the chapter consists of a description of the assumptions, the key management system and the trust model of Kerberos. The chapter concludes with an overview of the known limitations of Kerberos.

5.1 Introduction

Kerberos is a system for authentication in distributed computer systems. It provides strong authentication of clients and server using conventional cryptography. It also protects the confidentiality and integrity of communications over an insecure network. Kerberos operates on the application layer, providing end-to-end security.

The Kerberos system was developed at Massachusetts Institute of Technology as a part of project Athena. One implementation of Kerberos is freely available from MIT, and Kerberos has also been integrated into commercial products such as Windows 2000. Kerberos is actively developed, the current version is version 5, containing security relevant improvements over version 4. Where these versions differ I will stick to version 5. The Kerberos architecture and protocol system is thoroughly described in RFC 1510 [34].

When testing Kerberos implementations I have mostly used the Heimdal
Kerberos is designed to operate in an environment where the network itself and the workstations are not trustworthy. An adversary is assumed to be able to read, insert and modify packets on the network.

5.2 The Kerberos architecture

The Kerberos architecture is designed around a data element called a ticket. The ticket is a set of cryptographically sealed data that lets a client authenticate to a server. The Kerberos system uses two trusted on line services. These services are the authentication service and the ticket-granting service.

The Authentication Server (AS) implements the authentication service, and is responsible for authenticating users and giving them tickets usable for contacting the Ticket-Granting Server (TGS). The ticket-granting server implements the ticket-granting service, and is responsible for issuing tickets usable with servers implementing other services on the network. The combination of the AS and the TGS is called the key distribution center (KDC). A figure illustrating the relationships between the components can be found in figure 5.1.

Both the TGS and the AS are trusted services, and are assumed to be physically secure. Each organizational unit has one TGS and one AS. Each such unit is called a realm. Kerberos supports authentication of users across realm boundaries. Which realm a user belongs to is a part of the user’s
name. Trust between realms can be established through inter-realm keys.

5.3 The Kerberos protocols

The goal of the Kerberos protocols is to authenticate a party A to another party B. As a result of this authentication, a session key usable for further communication is established. Kerberos does not require the authentication to be mutual, but optional mutual authentication is available. A and B communicate with a trusted third party to achieve this goal. The basic Kerberos protocol is based on the Needham-Schroeder protocol described in section 3.5.2, and modifications proposed by Denning and Sacco [7].

A user wanting to log on to the network first contacts the authentication server asking for credentials (step 1 in figure 5.1). These credentials are returned to the client (step 2). The credentials are called a ticket-granting ticket and can be used to contact the ticket-granting server to obtain a ticket for a specific server. The ticket-granting ticket is then transferred to the ticket-granting server (step 3). The ticket-granting server can now generate a ticket for the server the client wants to contact, and send this back (step 4). In the last step the client sends the ticket to the server (step 5).

Kerberos defines two methods for asking for tickets. A client can send a cleartext request to the authentication server for a ticket to the TGS. The ticket is returned encrypted with the client’s key. This method is used for initial authentication when the client is not in possession of a ticket. The ticket is in this case called a ticket-granting ticket, and serves as proof that the user has been successfully authenticated.

The credentials obtained in step 1 is later presented to the TGS, the TGS will then issue tickets for other servers. This method is the second method that Kerberos supports; credentials can be obtained by presenting previously obtained credentials. The reason for using this double step is that it is beneficial to reduce the time in which the user’s key is available in clear on the client workstation. The user’s key is derived from his password, and because the entropy of the password is low it is not advisable to use this key as a primary key for encryption and authentication.

Tickets are not protected from replay. A ticket must therefore be accompanied by an authenticator. Authenticators are encrypted using the session key and contains a timestamp to ensure that it is recent. The encryption with the session key proves that the authenticator was generated by someone who knows the session key. This guarantees the identity of the client since
session keys are generated by the KDC and they are never transmitted in the clear.

5.3.1 Data elements

I will now describe in detail the two most important data elements in Kerberos, namely the ticket and the authenticator.

\[ ticket_{C-S} = C, client\_addr, e_{K_S}(C, K_{C-S}, t, flags) \]

A ticket is a record that lets a client authenticate to a server. The ticket above transports a session key for use between the client \( C \) and the server \( S \). The key is generated by the ticket issuer, and is shared between \( C \) and \( S \). The ticket is encrypted using the key of the target server.

The ticket is valid for use from the machine in \( client\_addr \). The ticket also contains the identity of the client, a timestamp and some flags. Since the ticket does not contain anything avoiding the replay of a ticket it must be accompanied by a fresh authenticator to be valid.

A ticket can have several flags set, these flags indicate attributes of the ticket. Attributes are requested by the client when obtaining the ticket or they can be set or unset by a Kerberos server. The most important flags are the “renewable” flag, the “may postdate” flag and the “proxiable” flag.

The “renewable” flag indicates that the ticket is renewable, in this case the ticket can be presented to the key distribution center before it expires, and a new ticket with a new session key will be returned. If it were not possible to renew tickets, the user would have to enter the password again every time his ticket-granting ticket expired.

If a user wants to schedule a job for later processing the ticket must be usable not at the moment when the job is submitted, but at the time the job will be processed. If valid tickets are held in the batch queue, there is a risk that they will become stolen and misused. A postdated ticket is a ticket that is usable at a later time than the time of issuing.

Since the source address is included in the ticket, a ticket can only be used from one specific source. Forwardable tickets are used to avoid this problem. A ticket can be marked as forwardable using the “forwardable” flag. During initial authentication the user asks for a forwardable ticket. The forwardable ticket can then be used to obtain a new ticket valid for another source address. If a user logs in on another machine on the network he can then obtain a new ticket valid from the new machine without entering his password again.
Sometimes a client wants some other entity to perform a service on its behalf. The other entity must then be able to use the client’s identity for a specific task. In this case the client can grant a proxy to the other entity. When the “proxiable” flag is set, the ticket-granting server is allowed to issue a new ticket with another source network address. The new ticket has the “proxiable” flag set to distinguish it from tickets used directly without proxying. The ticket will be valid for the same server as it was initially issued for, but the source may be different.

\[ \text{authenticator}_{CS} = e^{K_{c,s}}(C, k, t, \text{seqno}) \]

The authenticator provides authentication by proving knowledge of a key, and key confirmation. It is encrypted using the key shared between the server and the client. If the server can decrypt the authenticator and verify that the timestamp is recent, it can be sure that it is talking to the correct client. Key confirmation is provided, because the client cannot generate a correct message without knowing the key.

A key can be generated and included in the authenticator, in this case this new key can be used as a session key instead of the key contained in the ticket that is transferred with the authenticator. These keys are called sub-session keys. The sequence number included in the authenticator is intended to be used with the KRB_Safe and KRB_PRIV messages, serving as an initial sequence number to prevent replays.

### 5.3.2 Message exchanges

Now that the most important data elements have been described, we can proceed to describe the actual message exchanges.

**Authentication Service Exchange**

The first message exchange is between the client (C) and the authentication server (AS), and is shown in protocol 5.1. The AS exchange is the first exchange a client takes part in, it is used to obtain credentials for a server when the client holds no credentials. In the first message the client asks for credentials to contact the Ticket Granting Server (TGS). Encryption is done using the client’s key (KC), which is derived from the user’s password. There are two situations in which a client will have to use this type of authentication, either at the first login or when contacting a service that requires the user’s private key to be used. Such a requirement is needed
for the password changing service. To prevent someone from walking up to an unattended terminal and changing the password of a user, a recent proof of knowledge of the password is needed.

The identity of the client and the server for which the credentials are requested is sent to the Authentication Server in cleartext. In reply to this message a ticket for the requested service is returned. The key contained in the ticket (\(K_{C\rightarrow S}\)) will be used for communication between the server and the client, and is returned encrypted with the client’s key. In addition a nonce \(N_C\) is included to prevent replay. This nonce is intended to be a random number generated by the client, but the specification allows a timestamp to be used if the client can be sure that its clock is monotonically increasing.

The authentication service can not know whether or not the client knows the password at this stage, so the client is not authenticated to the AS yet. Authentication of the AS to the client is provided because the AS is the only entity assumed to know the client’s key.

The Ticket-Granting Service (TGS) Exchange

The ticket-granting service exchange is used when a client wants to obtain a ticket for a specific server and holds only a ticket for the ticket-granting server.

The message exchange is almost identical to the AS Exchange except that the encryption key used is not the client’s key but the key obtained in the ticket-granting ticket.

The Client/Server Authentication Exchange

When credentials have been obtained using the AS exchange or the TGS exchange, the client can use these credentials to authenticate itself to the target server. The complete protocol can be found as protocol 5.2.

The authentication accomplished by the Client/Server Authentication Exchange can either be one-way, or mutual. The client can request mutual
5.3 The Kerberos protocols

<table>
<thead>
<tr>
<th>Client</th>
<th>Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>→</td>
<td></td>
</tr>
<tr>
<td>$ticket_{C \rightarrow S}$, $auth_{C \rightarrow S}$, options</td>
<td></td>
</tr>
<tr>
<td>$e_{K_{C \rightarrow S}}(t, k, seqno)$</td>
<td>←</td>
</tr>
</tbody>
</table>

Protocol 5.2: Kerberos Client/Server Authentication Exchange

authentication by setting a special flag in the request to the server. If the
authentication is one-way and no session-specific key is to be negotiated,
the last message of this exchange can be omitted.

As described earlier the ticket provides no replay protection guarantee, the
client therefore prepares an authenticator in addition to the ticket. The au-
thenticator is constructed using the current system time and the session key
obtained with the ticket.

When the server receives the request for authentication, it first decrypts the
ticket, and at the same time an integrity check of the ticket is performed. It
is assumed that the encryption method used is able to detect modification
of ciphertext. The authenticator is then decrypted using the session-key
included in the ticket. The timestamp in the authenticator is compared
with the system clock of the server, if they differ by more than a predefined
amount the authentication is rejected. The server is in addition required to
reject authenticators that it has seen before. Authenticators are considered
to be equal if the server name, client name and timestamp are equal. To im-
plement this the server must keep a list of all received authenticators within
the allowable clock skew. Additionally some sanity checks on the ticket is
performed, if the ticket was generated before the current time (adjusting
for allowed clock skew) or if the tickets life-time has expired it is rejected.

The reply from the server to the client is then constructed, it is encrypted
using the key contained in the ticket. The reply consists of a timestamp
an optional subkey and a sequence number. The timestamp used must
be the client’s timestamp exactly as obtained in the authenticator received
from the client. The sequence number is intended to be used as the initial
sequence number for messages going from the server to the client.

When the client receives the message, it decrypts it using the key from the
credentials for the server, even if a new subkey is negotiated the new key
is not used. The client checks that the timestamp is the same as used in the
original authenticator. If it is the server is authenticated to the client. If the
subkey or sequence number are present, they are saved for later use.

The client and the server now share a key that can be used for further com-
munication. This key can either be the key contained in the ticket, or it
can be a subkey derived from the subkeys transferred between the parties.
How this subkey is derived is not specified by the Kerberos standard.
Protocol 5.3: Kerberos KRB_SAFE Exchange

\[
A \rightarrow data, t, seqno, s - address, r - address, MAC_{k_{A-B}}()
\]

Protocol 5.4: Kerberos KRB_PRIV Exchange

\[
A \rightarrow e_k(data, t, seqno, s - address, r - address)
\]

The KRB_SAFE Exchange

The KRB_SAFE message (protocol 5.3) is used to transfer a message with integrity protection. This is done by including a message authentication code in the message. The code is computed using the session key negotiated in the Client/Server Authentication Exchange. The Kerberos standard defines several collision proof checksum algorithms that can be used for this purpose. An application can either choose to use a timestamp or a sequence number to ensure uniqueness of the message. If individual messages cannot be lost, the application is advised to use sequence numbers. If it is decided that the loss of messages is acceptable, the use of a timestamp is a better solution. Timestamps are also preferable in multicast applications where all messages might not be sent to all of the participants.

When the message is received the receiver checks that the sender and receiver addresses in the message corresponds with what the local operating system believes. Then the sequence number and/or the timestamp is checked. The sequence number must be larger than the last sequence number seen. The timestamp is checked as for authenticators. At last the receiver computes the checksum over the data and the control information and compares it with the checksum included in the message.

The KRB_PRIV Exchange

The KRB_PRIV message (protocol 5.4) is used to secure the confidentiality and integrity of messages.

The data to be sent and some control information is encrypted and sent to the receiver. The key used for the encryption is the last key negotiated, or the session key if no such subkey has been agreed upon. When the message is received, it is decrypted. After decryption the same checks on the control information as for the KRB_SAFE exchange is performed.
5.3 The Kerberos protocols

The KRB_CRED Message

The KRB_CRED exchange is used to transfer a ticket with the corresponding session keys. This message essentially serves as a means to transfer credentials from one client to another. The method is used with forwardable or proxiable tickets as described earlier.

The tickets and the session keys are transferred encrypted with a session key previously negotiated between the clients. I will not describe the exact data formats here, for more details see the Kerberos RFC.

5.3.3 Cross realm operation

Kerberos supports the concept of different security domains. Each security domain is called a realm in the Kerberos terminology. Each realm has one authentication server and one ticket-granting server. Every principal registered in the realm shares a key with the AS. Sometimes a user will want to contact a server that belongs to another administrative domain. One option is to get a key and a username for the remote realm. In many situations this can be a serious burden for the users.

To enable users to operate across realm boundaries more easily, the administrators of two realms can generate inter-realm keys enabling users in one realm to authenticate to services in the other. Such keys are generated by registering the ticket-granting server as a principal in the remote realm. In this way a client in one realm can obtain a ticket-granting ticket for the remote realm from the local ticket-granting service. The ticket-granting ticket is encrypted using the inter-realm key, the remote ticket-granting server can then be sure that the ticket was issued by the local ticket-granting server.

The typical sequence that a client will go through to contact a server in a foreign realm will be to first contact the local ticket-granting server and ask for a ticket for the remote ticket-granting server. The client will then use this ticket to ask the remote ticket-granting server for a ticket for the remote server. When the client has the ticket for the remote server, he can communicate directly with the remote server. For the target server there is no difference in the authentication of a local user or a user coming from a remote realm. The ticket being presented is in both cases generated by the same ticket-granting server. It is possible for the target server to know that the user belongs to a remote domain, since the domain name is a part of the user’s name.

Two realms can communicate if they share an inter-realm key or if they share a key with an intermediate realm. The extension of this is called a
certification path. The ticket contains a field that lists the name of each realm transited. This field can be used to determine how much a remote ticket can be trusted.

To improve scalability realms are usually organized hierarchically, each realm share a inter-realm key with the realm above it in the hierarchy. To construct a certification path when two realms do not share a key directly a path can easily be constructed by walking the tree towards the root, finding the least common ancestor. Shortcuts between realms communicating frequently can be added to prevent long authentication paths.

5.4 Kerberos assumptions

The Kerberos system makes several assumptions that must be fulfilled for the security guarantees to hold. Some of the assumptions have to do with the general environment in which Kerberos is to be employed, and others have to do with the cryptographical primitives used.

5.4.1 Requirements for cryptographic primitives

Authentication in Kerberos is done through proof of knowledge of a secret key (see section 3.3.7). The proof of knowledge of this key is done by encrypting some data, and the receiver must have a method for determining whether this data was tampered with during transmission or not. The various Kerberos messages do not include a checksum. It is assumed that the underlying cryptographic primitive can detect tampering.

Since integrity protection is required, a checksum of the plaintext must be added to the transmission if the cryptosystem itself does not deliver this protection. The exact nature of this checksum is specified as a part of the encryption method.

Several encryption methods are defined in the Kerberos standard, and new encryption methods can easily be added. Kerberos has traditionally used DES and a CRC-32 checksum. The CRC-32 checksum is not collision proof and this makes it possible to mount an attack as described by Stubblebine and Gligor [64]. The use of CRC-32 is therefore no longer mandated by the Kerberos specification.
5.4 Kerberos assumptions

5.4.2 Key generation

The initial key used for the first authentication of a user is derived from the user’s password. Poorly chosen keys are therefore a threat. Bad passwords make it possible to reduce the time needed for a successful brute-force attack. The use of passwords makes it easy for a user to log on to the system, and no additional equipment is needed on the client computer to support it. Kerberos assumes that the passwords chosen by users, or generated by the system, are random and not easily predicted. The problem can be reduced by having the password changing program check the new password against a dictionary and specify requirements for the password. Spafford [62] gives an overview of how to check passwords and how this can be done efficiently.

5.4.3 Environmental assumptions

The Kerberos system makes several assumptions about the environment in which it is to be used. If these assumptions are violated, the security of the system cannot be guaranteed by Kerberos.

Denial of service Kerberos does not try to prevent denial of service attacks.

Secrecy of secret keys If the secret key of a principal is stolen, the intruder can easily masquerade as the principal.

Password guessing It is possible to guess the password of a user off-line. Care must therefore be taken so that weak passwords are not chosen.

Synchronized clocks Since some Kerberos messages contain timestamps to prevent replay attacks, Kerberos must assume that the clocks of the machines on the network are loosely synchronized. The problems involved with this are described in section 3.3.4.

Recycling of principal identifiers The access control system in Kerberos is based on access control lists (ACLs). If an identifier is reused for a new principal before the ACLs in the system are updated, the new principal will get access to resources that he was not intended to have access to. Kerberos therefore assumes that principal identifiers are not reused.
5.5 Key management in Kerberos

All keys in Kerberos are conventional symmetric keys. There are some proposals adding public key functionality, for example [61], but they are not discussed here.

**master key** The Kerberos master database can be encrypted. The Kerberos RFC does not require this, but it suggests that it can be done and describes the implementation used in the MIT implementation. The key is used to encrypt the keys in the database, and is an extra precaution in case the database is compromised while the master key is not.

**user key** The key that the user uses for initial authentication to the TGS. This is considered to be weak since it is derived from the user’s password. It should therefore be used as little as possible.

**cross realm key** A cross realm key serves as a link between two realms. The key is shared between the KDCs of the two realms and is used to encrypt ticket-granting tickets of clients in one realm who wants to contact servers in another realm.

**server key** Each host acting as a server providing a service needs to have a server key. This key is used for communication with the ticket-granting server. More specifically the key is used to encrypt tickets received from clients contacting the server. The key is shared with the key distribution center.

**tgt key** Tgt keys are session keys exchanged between the user and the KDC. They are used in communication with the KDC to obtain new tickets that can be used to authenticate to servers.

**session key** Session keys are used to encrypt communication between a client and a server. It is also used to authenticate the client, and optionally to authenticate the server. The session key is generated by the key distribution center.

**sub-session key** A client and a server can optionally negotiate a separate sub-session key instead of using the session key provided by the key distribution center. The exact details on how this is done is not standardized.

When looking at the keys used, some points are worth noticing. The initial authentication of a user exchanges the weak user key with a much stronger automatically generated tgt key. In addition no unencrypted long-term keys are stored on the user’s workstation.
5.5 Key management in Kerberos

![Key Hierarchy Diagram]

Figure 5.2: The key hierarchy of Kerberos

5.5.1 Key hierarchy

Kerberos uses a key layering model as described in 4.2.1. A schema displaying the hierarchy can be found in figure 5.2. At the top of the hierarchy is the Kerberos master key. This key is used as a safety measure as described earlier. A sensible implementation will try to keep as few keys as possible in memory at the same time and decrypt keys in the database only when they are needed.

Below the master key in the hierarchy are the server keys, cross realm keys and user keys. These keys are all stored in the Kerberos database at the KDC. The user key is special. In addition to being stored in the master database it is also stored in the brain of the person owning the key.

In addition to being stored in the master database, server keys are stored on the server to which they belong. Cross realm keys are stored on the KDC of the two realms it is used for.

The tgt key is just a special session key. It is used for communication between a user and the KDC after initial authentication. It can also be the result of a cross realm authentication. In the first case the tgt key is first encrypted using the user key and then it is included in the ticket-granting ticket encrypted with the key of the KDC.

The ultimate result of a Kerberos protocol run is to establish a session key. The session key is therefore at the bottom of the hierarchy. In addition it is possible for two communicating parties to establish a sub-session key.
5.5.2 Key control

If the question of key control is considered, it easily be seen that all keys but
the user key are generated by the KDC. Neither the server nor the client has
key control. Since the exact process of sub-session keys negotiation is not
specified, the question of key control for these keys is open.

5.5.3 The life cycle of a Kerberos key

The Kerberos architecture does not go into details about how keys are man-
aged. The architecture will give some constraints however, for example all
Kerberos tickets have a expiration time. This effectively limits the period in
which a key is in the active state. When a ticket expires the client will have
to go to the KDC to renew the ticket.

master key The Kerberos RFC does not mention how this key is admin-
istered. In MIT Kerberos it is stored on the local disk and is generated
when the database is initialized. There is no reason why the key could
not be changed on a regular basis. This would require the database
to be decrypted and re-encrypted with the new key.

user key A new user key is generated when a user is registered with the
system. A random password is generated, and this password is trans-
ferred to the user. This can be done by giving the user the password
on a piece of paper. New users are advised to change their pass-
words. The new password is chosen by the user, and the key derived
from the password is therefore generated by the user. An implement-
ation is free to implement password aging where the user is forced to
change the password at regular intervals. The password changing is
implemented by initiating a secure connection to the KDC using the
old password, the user can then enter a new password on the key-
board.

A copy of the derived key is stored at the AS. Since the key is only
used for initial authentication, it does not need to be stored on the
user’s computer. The password is entered at the keyboard when
needed, and the key is derived from it. The key is deleted when the
authentication is done.

When the key is deactivated it is simply removed from the database.
The AS is the only party with access to this key except from the user
himself. This makes it trivial to avoid use of obsolete and post-active
keys.
5.5 Key management in Kerberos

server key  Server keys are generated during the initialization of a new server in the system. If a server key is changed, the old key must be held in the post-active state, and be stored at the server for as long as the maximum lifetime of any tickets encrypted with this key. It can then be destroyed. The AS can destroy old keys immediately, and start issuing tickets encrypted with the new key.

A server key is installed on the server by copying it from the key database of the KDC. How this is done is implementation defined. In MIT Kerberos and Heimdal a secure connection to the KDC is established, using the key of a privileged user, and the key is copied to the local filesystem.

cross realm key  A cross realm key is generated when the administrators of two realms decide that they want to trust each other and make it possible for users to authenticate across the realm boundaries. The key is generated at the KDC of one of the domains and is transferred manually to the other realm. No protocol support to automate this process exists. In practice a procedure as the one described for server keys will be used.

If a cross realm key is changed the old key must stay in the post-active state until all tickets issued under the key has expired. The ticket-granting server of both realms can then destroy the key.

tgt key  This key is generated by the AS when a user authenticates to the system. It has to be stored at the KDC for the duration of the life-time of the particular ticket-granting ticket. After it expires it is no longer needed and can be destroyed. If a user logs out of the system his keys should be deleted from the local workstation. The KDC has no way of knowing when this happens.

session key  A session key is generated by the AS when a client wants to contact a server. After the generation this key is not needed at the KDC. It is distributed to the client and the client sends the key to the server contained in a ticket. Since all tickets have an expiration-time the key can only be used for authentication for a limited period of time. However, there are no mechanisms limiting the use of the key after that.

sub-session key  The administration of sub-session keys is left completely to the application. How these are generated and destroyed is an implementation issue.

As can be seen from the above descriptions, all keys used by Kerberos are generated at the key distribution center. They are then distributed to the servers where they are to be used.
As have been seen, the Kerberos architecture consists of three main parts, the client, the KDC, and the server. The KDC consists of the AS and the TGS. There might of course be more than one server and one client in a system, but in this discussion regard all servers and all clients are regarded as one component. The trust relationships in Kerberos are quite simple. The KDC components are the only components having to be 100% trusted. The client trusts the TGS not to reveal session keys to any parties but those who are authorized to have them. The server trusts the TGS to only generate valid tickets, and to keep session keys secret. The KDC is also trusted not to reveal the key database. Since all key generation is done on the KDC, this is the only entity that has to be able to generate good keys (see section 4.4.1. Table 5.1 gives an overview of the trust relationships.

When authenticating across realm boundaries, things get a little more complicated. When a cross realm key is set up, the KDCs agree to trust the competence of each other. If a user is authenticated in one realm, the other realm trusts that this authentication was carried out correctly. Any server will know if a user was authenticated remotely. This will be visible to the servers by looking at the user identifier contained in the ticket. It will therefore not be possible for a cheating KDC in another realm to impersonate local users.

One important thing to remember is that the user must trust the local workstation he logs on to. The user gives away his long term key to the workstation, and the cryptographic operations are carried out inside the computer outside the user’s control.

### 5.7 Kerberos limitations

Several limitations found in Kerberos have been described in the literature. I choose to call them limitations rather than weaknesses since a weakness is usually meant to be a security problem that can be exploited. The prob-

<table>
<thead>
<tr>
<th>Trusting</th>
<th>Trusted</th>
<th>For what</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client / Server</td>
<td>KDC</td>
<td>Not reveal keys</td>
</tr>
<tr>
<td>Server</td>
<td>TGS</td>
<td>Issue correct tickets</td>
</tr>
<tr>
<td>TGS</td>
<td>AS</td>
<td>Authenticate users</td>
</tr>
</tbody>
</table>

Table 5.1: Trust relationships of Kerberos
lems found in Kerberos will not compromise the security of the system under most circumstances. The most thorough criticism of Kerberos was done by Bellovin and Merrit [4], they suggest several improvements to the protocols.

5.7 Kerberos limitations

5.7.1 No control over the usage of session keys

From a key management point of view there are several features that Kerberos does not implement that would give better control over the keys in the system.

When a connection has been established, a key is shared between the server and the client. The period in which the key and the ticket can be used for authentication is limited by the life-time of the ticket. The time in which the key itself is used for data transfer after this is not handled by Kerberos. No cryptoperiods are enforced for session keys at all. Applications using Kerberos are free to implement this as they see fit, but my opinion is that this should be a part of the protocol. This burden should not be put on the application developers, but should be implemented in the security architecture.

5.7.2 Dependence on synchronized clocks

As have previously been said, Kerberos depends on synchronized clocks. The fact that the clocks are synchronized is a fundamental assumption of the Kerberos architecture. One can always question the assumptions of a security system makes, and this particular dependency has received a lot of attention. Removing this dependency does not come without costs, and the designers of Kerberos have chosen not to remove it.

5.7.3 Password guessing

All password-based systems will be vulnerable to bad passwords. Bad passwords are passwords that can be guessed, such as passwords that resemble words in a dictionary. Attacks on passwords can in Kerberos be carried out off-line. An adversary can try to authenticate once to a KDC and receive an encrypted key. The adversary can then try one possible password after another without having to contact the KDC again.

This limitation is listed as one of the environmental assumptions of Kerberos. It deserves mentioning here because password guessing attacks are
common in large networks with many users.

5.7.4 Trust in local workstation

The fact that the user has to trust the local workstation to handle his keys cause several problems. The Kerberos login program could for instance be exchanged with a Trojan. An attacker could install software or hardware recording the user’s keystrokes. There are no solutions to this problem if special trusted hardware is not used. This hardware should ideally perform all cryptographic operations on behalf of the user. A compromise would be to use this trusted component for authentication only, and let the workstation take care of the session keys.

5.7.5 Same key used for authentication and encryption

In Kerberos the same keys are used for both authentication to the server as well as for encryption of data. As explained in section 4.1 keys used for different purposes should not be mixed. It is possible for a Kerberos application to negotiate sub-session keys derived from the authentication key, but this is not required.
Chapter 6

SESAME

SESAME tries to solve the same problems as does Kerberos. They differ mostly due to the fact that SESAME has chosen to use asymmetric technology extensively, while Kerberos uses conventional symmetric cryptographic techniques. In addition SESAME employs role-based access control, while Kerberos uses an identity based access control scheme.

This chapter has roughly the same structure as the Kerberos chapter. I first describe the architecture and the protocols, while the last part is an analysis of the key management, trust model and limitations of SESAME. This last section presents some limitations not previously described in the literature. My description of SESAME is mostly based on the book by Ashley and Vandenwauver [2].

6.1 Introduction

During the 1980s most of the standardization of network technology took place in the OSI working groups of ISO. The OSI standards for Open Systems Networking [23] were developed, and as a part of this the OSI security architecture [24]. This standard proposed different security services and the layers of the OSI reference model on which they were to be implemented. Later the OSI Security frameworks [21] were developed and the concepts were further refined. These standards gave a firm foundation for implementing security in networked systems, but there were still a lot of work left in developing a commercial product.

As a result of the work on the X500 directory services [25] the X509 authentication framework was standardized. The most important contribution here was the certificate format and some authentication protocols.
Many vendors were disappointed with the speed of the standardization process. The European IT industry therefore decided to do work within ECMA (European Computer Manufacturers Association). As a result, a standard called “Security in Open Systems – Data Elements and Service Definitions” [11] was produced. This standard pointed out the services and data elements needed to secure a distributed system.

In addition Bull, ICL and Siemens decided to implement a security system based on the ECMA work. The result was called SESAME (Secure European System for Applications in a Multi-vendor Environment). The project started in 1989 and was founded by the three partners and the European Commission through the RACE project. The first stage was finished in 1991, and demonstrated that the architectural ideas and principles of the ECMA standards were feasible and practical. The second stage opted for developing security components usable in commercial products. This stage produced SESAME V2 and V3, which were released to selected sites for non-commercial purposes. The current version is V4, and is freely available on the Internet for non-commercial evaluation.

The first versions of SESAME were based on the Kerberos sources from MIT. Because of licensing and export problems the dependency on the Kerberos sources were removed in version 4 of SESAME. SESAME still chose to be compatible with Kerberos, and it is possible for Kerberos clients and servers to interoperate with SESAME.

### 6.2 Goals of SESAME

Kaijser [31] describes the process behind the SESAME development. He states that the first emphasis was to develop an architecture that could support a variety of security policies and security mechanisms. Authentication and access control in an open distributed network was the main focus. It was also a goal from the start to support delegation in a controlled manner. The system also had to support secure communication between different security domains, and thus key distribution not only within a security domain, but also between different domains.

It was realized that system administration was usually more costly than purchase price. Management was therefore considered the most important aspect besides the ones mentioned above. Moreover ease of use was seen to be an important factor.

To make it possible to distribute the responsibility of management in the system, the architecture was designed to consist of a set of well-defined
components. These components could be implemented on the same com-
puter or be distributed in the network.

A trust model was developed based on a set of assumptions. These as-
sumptions guide the development of the architecture. The most important
assumption was that the underlying communication infrastructure was in-
secure. SESAME does not depend on any security provided by the un-
derlying layers. SESAME thus provides true end-to-end security on the
application layer.

The workstations in the system were considered to be an easy target for
hackers or malicious users. They were therefore given limited trust. A
tampered end-system should not make it possible for an attacker to com-
promise the security of the entire system.

A security system is dependent on the cryptographic services it uses. The
architecture was designed to be independent of the underlying crypto-
graphic techniques. SESAME can utilize both symmetric and asymmetric
cryptographic algorithms in addition to one-way functions. To avoid prob-
lems in countries where the use of cryptography to achieve confidentiality
is restricted, SESAME tries to avoid encryption of data as much as possible.

6.3 The SESAME architecture

The SESAME architecture is thoroughly described by Parker and Pinkas
[52]. I will here give an overview of the architecture with emphasis on the
concepts relevant for key management.

The standards describing the SESAME architecture define several terms
needing explanation.

principal  Humans or system entities that can authenticate themselves to
the system are called principals.

initiator  When a principal is in an active role, for example requesting ac-
cess to a resource, it is called an initiator.

target  A principal that is contacted by an initiator, and thus acting in a
passive role, is called a target.

To be able to meet the goal of an easily administered system, the SESAME
designers chose to implement a role-based security system. This means
that initiators are not allowed access according to who they are, but which
role they play. When an initiator is contacting a target, he presents a certificate containing his privileges. This differs from how this is done in Kerberos, where the certificate presented to the target contains the identity of the initiator. The certificate used by SESAME is called a Privilege Attribute Certificate (PAC).

The components of SESAME can be divided into initiator and target components. In addition there are some components that are used by both initiators and targets. These are either on-line or off-line. See figure 6.1.

The SESAME description uses a lot of acronyms. This can be very confusing for a reader unaccustomed with the architecture. A table explaining the acronyms used in this chapter can be found in table 6.1, it can be useful to refer to later.

6.3.1 Initiator components

The user sponsor is the part of the system that is responsible for acting on behalf of a human user. A human user is not able to perform complex cryptographic operations, or to send data through a computer network. The user therefore interacts with the user sponsor and the sponsor interacts with the rest of the system. The SESAME user sponsor supports login, logout and changing of roles.
6.3 The SESAME architecture

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAC</td>
<td>Privilege Attribute Certificate</td>
</tr>
<tr>
<td>APA client</td>
<td>Authentication and Privilege Attribute (server) client</td>
</tr>
<tr>
<td>SACM</td>
<td>Secure Association Context Manager</td>
</tr>
<tr>
<td>PVF</td>
<td>PAC Validation Facility</td>
</tr>
<tr>
<td>AS</td>
<td>Authentication Server</td>
</tr>
<tr>
<td>PAS</td>
<td>Privilege Attribute Server</td>
</tr>
<tr>
<td>KDS</td>
<td>Key Distribution Server</td>
</tr>
<tr>
<td>PKM</td>
<td>Public Key Management</td>
</tr>
<tr>
<td>CSF</td>
<td>Cryptographic Support Facility</td>
</tr>
<tr>
<td>SMIB</td>
<td>Secure Management Information Base</td>
</tr>
<tr>
<td>CA</td>
<td>Certification Authority</td>
</tr>
<tr>
<td>LRA</td>
<td>Local Registration Authority</td>
</tr>
<tr>
<td>CAA</td>
<td>Certification Authority Agent</td>
</tr>
<tr>
<td>PPID</td>
<td>Primary Principal Identifier</td>
</tr>
<tr>
<td>PV/CV</td>
<td>Protection Value / Check Value</td>
</tr>
<tr>
<td>DTQ</td>
<td>Delegate Target Qualifier</td>
</tr>
</tbody>
</table>

Table 6.1: SESAME Abbreviations

The Authentication and Privilege Attribute Client (APA Client) is a library of functions used for authentication. The library hides the interactions between the user and the Domain Security Server.

The Secure Association Context Manager (SACM) is responsible for the authenticity and optionally the confidentiality of the communication between the client and the server.

### 6.3.2 Target components

The most important of the target components is the PAC Validation Facility (PVF). The PVF is used by the SACM to handle a number of important tasks:

- Establishment of basic keys
- Validate PACs
- Extract and return relevant information from PACs
- Obtain dialogue keys
- Provide information and support needed for delegation
The PVF is trusted by the application it does access control for, it is also trusted not to release information contained in the PAC to the target application when the target is not authorized for it.

The SACM of the target has the same responsibilities as the SACM of the initiator.

6.3.3 Trusted Third Parties

The trusted third parties of SESAME, except for the PVF, are all part of the domain security server.

The Authentication Server (AS) is responsible for authenticating a principal. In SESAME there are two ways to do this: a public key based system and a Kerberos-compatible password authentication system. The AS authenticates the principal using one of these techniques and returns a ticket for the Privilege Attribute Server (PAS).

SESAME uses role-based access control. A Privilege Attribute Certificate (PAC) is used to prove that a principal has the required privileges to perform a certain action. The role of the PAS is to manage the access privileges of the principals in the system, and issue signed PACs according to this information on request from the principals. In addition the PAS provides keys that clients can use for communicating with a Key Distribution Service (KDS).

The KDS is an important component when using the Kerberos compatible symmetric key based protocol. The most important task of the KDS is to deliver basic keys used for communication between an initiator and the target PVF. It also manages long term secret keys shared with the PVFs of its security domain. It also manages the mapping between target application names in its security domain and their PVFs. This makes it possible for the KDS to find the right key for an application. Finally the KDS is responsible for supporting inter-domain working by making use of the public keys of the KDS of other security domains.

6.3.4 Generic components

These components are used in various places within the architecture. They are all accessed as library routines.

The Public Key Management (PKM) provides support for handling public key certificates. It is used by target and initiator components, the PVF, the
6.3 The SESAME architecture

PAS and by the KDS. The library currently implemented in SESAME is relatively simple, but is designed to be easily extended.

The Cryptographic Support Facility (CSF) provides algorithmic cryptographic services to the other SESAME components. New cryptographic algorithms can be added, by hooking them up through the CSF.

The Secure Management Information Base (SMIB) is the database of the security server. The AS, PAS and KDS use this database store and retrieve information on users, keys, privileges, roles and other data used for security management. The SMIB is administered through a control program.

Many events in a system is considered to be relevant for the security of the system. The audit facility is responsible for recording these events.

6.3.5 Public Key Management Services

The SESAME architecture includes a basic PKI. I will here describe the parts of this PKI. There is no reason why SESAME could not utilize a separate external PKI.

The Certification Authority (CA) is responsible for certifying the public keys of the principals in the system. To minimize the chances that the CA will be compromised, it is implemented as an off-line service.

When a principal needs a certified public key, it first interacts with the Local Registration Authority (LRA). The LRA is contacted and a public key-pair is generated. The public part is then sent, protected with the signature of the LRA to the Certification Authority Agent (CAA). The CAA is an on-line service that spools certification requests. The request is later forwarded to the CA using a manual method, for example on a diskette. The CA checks that the request is correct and signs the certificate using its private key. The certificate is then returned to the CAA and forwarded to the principal.

This process is somewhat simplified in the current SESAME implementation. The off-line CA component is not available, and a simple Unix-based program is used in its place. The key-generation process is therefore done at the principal side. After the key is generated, it is copied to the CA where the signing is done. It is not checked that the principal initiating the key process is in possession of the private part of the key.

The SESAME PKI does not include a directory service. Each principal therefore holds the certificates of the entities with whom they want to communicate in a local database.
6.4 The SESAME protocols

Now that the components making up the SESAME system have been described, it is time to look at how these components interact to provide a secure system. In the protocol descriptions some non-cryptographic elements have been omitted for clarity.

6.4.1 Cryptographic primitives

This is a description of the cryptographic primitives used by SESAME. SESAME itself does not specify the specific algorithms used, so this description only covers the semantics.

In the description below, a basic key shared between the principals $A$ and $B$ is denoted as $BK_{AB}$. $PK_A$ is the public key belonging to $A$, and $PK_A^{-1}$ is the corresponding secret part.

$\text{SEAL}_{BK_{AB}}(m)$
This primitive provides integrity protection and data origin authentication for the message $m$. SESAME is dependent on being able to recover the message without knowing the key. This primitive is therefore currently realized by concatenating the message $m$ with a MAC computed on $m$ using $BK_{AB}$ as the key.

$e_{BK_{AB}}(m)$
The message $m$ encrypted using a symmetric-key technique. In SESAME this operation must ensure both confidentiality and integrity. The current SESAME implementation achieves this by encrypting the message concatenated with a one-way hash.

$e_{PK_A}(m)$
The message $m$ encrypted using the public-key of $A$. The message is assumed to contain enough redundancy to distinguish valid messages from invalid ones.

$\text{SIGN}_{PK_A^{-1}}(m)$
The signature of $A$ on the message $m$ using the secret part of $A$’s public key-pair. Note that while the current SESAME implementation uses RSA for signatures, there is no reason why a signature scheme not usable as an encryption scheme could not be used. The same key-pair is never used for both encryption and making signatures.
6.4 The SESAME protocols

6.4.2 Data elements

The following data elements are used in the protocols. They have specific purposes and are described here to make it easier to follow the description of the protocol exchanges.

\[ AuthSK_{A \rightarrow B}(data) = e_{BK_{AB}}(B, t_A, data) \]

A proves to B that it knows the session key \( BK_{AB} \). The freshness of the message is ensured by the timestamp \( t_A \). This message is equal to the Kerberos authenticator.

\[ AuthPK_{A \rightarrow B}(data) = \text{SIGN}_{PK_{A^{-1}}}(B, t_A, data) \]

This is an authenticator authenticating the entity A to B using public key technology. This is done by signing the identity of B concatenated with a timestamp generated by A. In addition to providing authentication of A to B, it provides non-repudiation of origin of the data element. The data element will typically contain a key.

\[ KeyPK_{A \rightarrow B \rightarrow C}(data) = e_{PK_B}(BK_{BC}, t_A, VP, data) \]

This message transports a session key \( BK_{BC} \) from A to B. B can later use this key to communicate with C. VP is the validity period of the key. The timestamp \( t_A \) is added to prevent replay.

\[ KeyPK_{A \rightarrow B}(data) = e_{PK_B}(BK_{AB}, VP, data) \]

The same as above except that the key is transported from A to B and the key is intended for use between A and B.

6.4.3 Credentials

SESAME can use two types of credentials. The credentials can be seen as certificates that allow the holder to perform a specific task.

\[ TGT_A = e_{LK_{AS\rightarrow PAS}}(A, U, VP, BK_{U\rightarrow PAS}) \]

The TGT (Ticket Granting Ticket) is a certificate used as evidence of the identity of a principal. It is returned from the authentication server upon successful authentication of a user. A basic key used for communication between the user sponsor and the PAS (\( BK_{U\rightarrow PAS} \)) is included. It also contains the identity of the user hiding behind the user sponsor (\( U \)) and the validity period for this TGT (\( VP \)). The whole package is encrypted using the long-term key shared between the Authentication Server and the Privilege Attribute Server.
$Ticket_{I-T} = e_{LK_T}(U, I, TGS, T_{TGS}, L_I, BK_{I-T})$

This is the standard Kerberos ticket that enables the initiator $I$ to authenticate to the target $T$.

$PAC_A = SIGN_{PK_{PAS}}(attrs, PPID_A, PV_A, DTQ_A, data)$

This credential contains the privilege attributes given to the user $A$. $PPID_A$, $PV_A$ and $DTQ_A$ are data used for the protection methods described below. The PAC is signed by the PAS. The PAC is presented to the PVF of the target, the PVF can then check that the initiator is in possession of the required privileges to access the target.

### 6.4.4 PAC Protection Methods

The SESAME privilege attribute certificate is integrity protected by a signature. This signature does not protect the PAC from being replayed or misused in other ways. The ECMA-219 standard [11] describes several ways to provide additional protection for a privilege attribute certificate. Three of these are used in the SESAME PAC. Each protection method serves a different purpose.

#### Primary Principal Qualification

The first method is called “Primary Principal Qualification”, and ensures that only the intended receiver of the PAC can use it. The method requires that each principal in the system has an identifier ($PPID$). If public key technology is used, the $PPID$ is the certificate identifier of the X.509 certificate used by the initiator requesting the PAC.

The protection method works as follows: Since the principal initiating a connection is authenticated, the PVF of the target only needs to check that the $PPID$ inside the PAC matches the identifier of the X.509 certificate of the initiator. Note that the target can not misuse the PAC because it can not establish new associations pretending to be the original owner of the PAC.

#### Protection Value / Check Value

It is possible in SESAME to let a target use the privileges of the initiator to contact other targets. This is called proxying. The “PV/CV” protection method makes this possible. In this case a protection value ($PV$) is inserted into the PAC. This value is the result of applying a one-way function to a
check value ($CV$) generated by the privilege attribute server. Those who know the $CV$ are entitled to use the PAC.

When a PAC is requested, the $CV$ is sent along with it encrypted using the session-key shared between the user and the privilege attribute server.

When contacting other servers, the $CV$ is distributed encrypted using the session key established between the initiator and the target PVF. The target PVF can then apply the one-way function to the $CV$ and check that the result matches the $PV$ contained in the PAC.

SESAME can distinguish between targets allowed to be delegates and those who are not. If the target is not allowed to be a delegate, the $CV$ is not returned from the PVF to the target. Remember that the PVF is trusted, so one can count on it not to return the $CV$ if the checks are not passed.

**Delegate Target Qualifier**

It is also possible to limit the use of a PAC to certain targets or trust groups. A trust group is a group of targets that trust each other mutually. If the $DTQ$ field exists in a PAC, the primary principal qualification and “PV/CV” tests are first performed. If they are passed, the PVF checks that the target is one of the targets listed in the $DTQ$, or is member of one of the listed trust groups.

**6.4.5 Message exchanges**

The ultimate goal of the SESAME protocols is to establish a key shared between two communicating parties, and to mutually authenticate the parties. This basic key is then used to derive one key used for protection of authenticity and one for confidentiality.

This is done in three steps. In the first step the initiator authenticates to the authentication server and receives a ticket as proof of its identity. In the next step, the client contacts the privilege attribute server and gets a certificate (the PAC) as proof of the privileges it has. The last step is to contact the target and present the credentials. As a result of this last step the session keys are generated. The first two steps can be done once at login time, and the same PAC can be used to access many targets.

In the description below both $I$ and $U$ is used to denote the initiator. $U$ is when long term keys belonging to the user are used, while $I$ is used when talking about short term secrets generated by the secure association context.
Authentication step

The purpose of the first protocol step (protocol 6.1) is to authenticate a principal to the authentication server (AS). When the AS is sure of the identity of the user a TGT (Ticket Granting Ticket) is returned as a proof of identity. This ticket can then be presented to other parties. Since the access control is role-based the principal will not use the TGT directly, but use it to authenticate itself to the PAS. The PAS will then return a certificate containing proof that the principal has a certain set of privileges.

The user sponsor (US) sends a packet containing the identifier of the user (U), a random number $R_U$, the requested lifetime of the TGT ($RL$), the user’s certificate, and an authentication token. The authentication token authenticates U to the Authentication Server.

Upon receipt of the packet the Authentication Server verifies the signature on the certificate and uses the public key of $U$ to verify the authentication token. The AS then replies with the new TGT and a key package that contains a key shared between the user sponsor and the PAS.

Before using the TGT, the user sponsor verifies the authentication token to be sure that it was talking to the correct AS. This prevents an intruder to set up a rogue AS. The key package is then decrypted and the key $BK_{U-PAS}$ is obtained. This key will later be used for authentication to the PAS.

Obtain privileges

Now that the principal is authenticated and is in possession of the TGT, it is time to ask the PAS to generate a privilege attribute certificate. This is done using protocol 6.2.

The user sponsor sends a message containing TGT along with an authenticator authenticating $U$ to the PAS. In addition the requested lifetime of the PAC ($RL$), and the requested privileges ($ReqPriv$) is included. To stop

<table>
<thead>
<tr>
<th>US</th>
<th>AS</th>
</tr>
</thead>
<tbody>
<tr>
<td>$U, RL, Cert_U, AuthPK_{U-AS}(R_I)$</td>
<td>$TGT_U, Cert_{AS}, AuthPK_{AS-U}(KeyPK_{AS-U-PAS}(), R_I, R_{AS})$</td>
</tr>
</tbody>
</table>

Protocol 6.1: Sesame SES-AS manager of the client are used.


6.4 The SESAME protocols

<table>
<thead>
<tr>
<th>US</th>
<th>PAS</th>
</tr>
</thead>
<tbody>
<tr>
<td>( RL, R_I, TGT, AuthSK_U-PAS, ReqPriv )</td>
<td>( PAC, e_{BKU-PAS}(R_I, CV) )</td>
</tr>
</tbody>
</table>

Protocol 6.2: Sesame SES-PAS

<table>
<thead>
<tr>
<th>( SACM_I )</th>
<th>( SACM_T )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \rightarrow ) ( SEAL_{IK_{I-T}}(SEAL_{BK_{I-PVF}}(M), I, t_I, n_I) )</td>
<td>( \rightarrow ) ( SEAL_{BK_{I-PVF}}(M) ) ( PVF )</td>
</tr>
<tr>
<td>( \rightarrow ) ( IK_{I-T}, CK_{I-T}, CV )</td>
<td>( \leftarrow ) ( SEAL_{IK_{I-T}}(t_I, n_T) )</td>
</tr>
</tbody>
</table>

Protocol 6.3: Sesame SES-INIT-CTX

playback attacks a random number generated by \( I \) is also included.

When the PAS receives the request from \( I \), it first checks that the authenticator is valid. It then proceeds to check the validity of the TGT. If the TGT is valid and the principal is authorized for the requested privileges, a PAC and an encrypted check value \( CV \) are returned. This PAC contains the requested privileges.

Set up the secure context

Now that the user sponsor have obtained its privileges, it can proceed to establish a connection with the remote server using protocol 6.3. The protocol follows the ECMA-206 [10] standard for association context management.

To make the following protocol exchanges easier to read, the message \( M \) is defined as follows

\[
M = \begin{align*}
\text{PAC} \\
\text{Key package} \\
\text{check value} \\
\text{Authenticator} \\
\text{Misc} \\
\text{Random offsets} \\
\text{Target application}
\end{align*}
\]

\[
\begin{align*}
\text{PAC} & : KeyPK_{U-PVF} \\
\text{check value} & : e_{BK_{I-PVF}}(CV) \\
\text{Authenticator} & : AuthPK_{U-PVF} \\
\text{Misc} & : Cert_U, r_I, t_I \\
\text{Random offsets} & : r_{s_1}, r_{s_2} \\
\text{Target application} & : T
\end{align*}
\]

The first step of the session establishment is to generate the session key used for communication between the initiator (\( I \)) and the PVF of the remote end. The initiator first sends a message containing \( M \), its identifier, a random number and a timestamp. When this message is received by
the target, the sealed message $M$ is forwarded to the PVF. The PVF will perform the access control. The message from the target to the PVF is not protected. The PVF is running on the same machine as the target and it is assumed that the underlying operating system will provide sufficient integrity, confidentiality and authentication.

The PVF starts by decrypting the key package $KeyPK_{U-PVF}$, thereby obtaining the basic key $BK_{I-PVF}$. It then performs the following checks:

- The authenticator $AuthPK_{U-PVF}$ is validated.
- The user $U$ is authenticated by validating $AuthPK_{U-PVF}$ using the public key contained in the certificate $Cert_U$.
- The seal on $M$ is verified using the key $BK_{I-PVF}$.
- It is checked that the identifier $T$ of the target application matches the application that the PVF is currently serving.
- The signature of the PAS on the PAC is verified.
- The $PPID$ check is performed.
- The check value package $e_{BK_{I-PVF}}(CV_U)$ is decrypted.
- The access control checks on the PAC are performed.

If all steps are successful, the integrity and confidentiality keys ($IK_{I-T}$ and $CK_{I-T}$) are returned to $SACMT_T$. The check value $CV_R$ is returned if $T$ is allowed by the PVF to act as a delegate.

**Sending of protected data**

The secure association managers of both $I$ and $T$ now has the session keys used for confidentiality ($CK_{I-T}$) and integrity ($IK_{I-T}$). They can now start sending protected messages. Protocol 6.4 is used when only integrity protection is required, and protocol 6.5 when both integrity and confidentiality is required.

**6.4.6 The Kerberos-compatible protocol**

SESAME was designed to be backwards compatible with Kerberos. Protocols are therefore implemented that allows Kerberos clients to connect to
6.4 The SESAME protocols

Protocol 6.4: Sesame SES-DATA (integrity only)

\[
\begin{array}{c|c|c}
SACM_I & \text{SEAL}_{IK_{1-T}}(t_I^1, n_I^1, m) & SACM_T \\
\rightarrow & \text{SEAL}_{IK_{1-T}}(t_T^1, n_T^1, m) & \leftarrow \\
\end{array}
\]

Protocol 6.5: Sesame SES-DATA (integrity and confidentiality)

\[
\begin{array}{c|c|c}
SACM_I & \text{SEAL}_{IK_{1-T}}(t_I^1, n_I^1, e_{CK_{1-T}}(m)) & SACM_T \\
\rightarrow & \text{SEAL}_{IK_{1-T}}(t_T^1, n_T^1, e_{CK_{1-T}}(m)) & \leftarrow \\
\end{array}
\]

SESAME servers, and also SESAME clients to connect to Kerberos servers. SESAME implements version 4 of the Kerberos protocols. The protocols are essentially identical to those described in chapter 5, so I will not describe the protocols extensively here. The implications on the SESAME architecture and the interactions between the SESAME and the Kerberos protocols will be stressed.

Authentication step

In SESAME users have two possible ways of authenticating to the security server. The first is the native SESAME method as described in section 6.4.5. The second method is to use the Kerberos authentication protocol. SESAME users can be initialized to have both asymmetric keys and password-based symmetric keys. The password-based key can be used to authenticate to Kerberos servers, but it can also be used to authenticate to SESAME servers. It is also possible for a user holding only Kerberos keys to authenticate to SESAME servers.

A user holding a symmetric key can send a request to the authentication server asking for a ticket-granting ticket. The protocol in this case is identical to the Kerberos V4 protocol. A ticket-granting ticket is returned along with a session key used for communication with the security server. The session key is encrypted with the user’s key. In this chapter the SESAME notation will be used. The user-key is therefore called \( LK_U \), and the session-key is called \( BK_{U-PAS} \).

The result of this is exactly the same as if the native SESAME protocol were used. A ticket for the PAS and a session key are obtained.
Obtain privileges

A SESAME compatible client will now contact the privilege attribute server to obtain a privilege attribute certificate. The client can do this independently of the authentication method used in the previous step. A Kerberos client will not be able to receive SESAME credentials, but must ask for a Kerberos ticket for the target server.

A SESAME client wanting to contact a Kerberos server can ask for Kerberos credentials. These can then be used for communication with a Kerberos server.

Set up the secure context

A client holding a Kerberos ticket can authenticate directly to the PVF of the target. In this case it is not possible for the PVF to use role-based access control. Traditional identity based access control will therefore have to be done by the PVF.

6.4.7 Inter-domain operation

A PAC from one security domain can be accepted by the PVF of another domain directly if the initiating privilege attribute server is trusted by the local CA. To enable the target PVF to check the signature on the PAC it must have access to the public key of the PAS of the domain generating the PAC in the first place. A trusted PAS has a public key certified by a CA trusted by the PVF. Additionally the PAC must pass the normal validation checks that all PACs go through.

One problem is that the attributes of the initiating domain might be different from those in the target domain. To fully support inter-domain operation, SESAME would have to implement an inter-domain service capable of translating between attributes of different domains. This functionality is not implemented in the current version of SESAME.

6.5 Key management in SESAME

In the core SESAME protocols, asymmetric keys are used for authentication and key transport. The symmetric keys are used as session keys. The keys
6.5 Key management in SESAME

used for Kerberos compatibility are in addition to these two groups of keys. I will describe each group separately.

6.5.1 Asymmetric keys

All the asymmetric keys in SESAME are long term keys. They are used for three purposes; signing certificates and PACs, transporting keys and authenticating principals.

\( PK_{CA} \) The signature key of the certification authority. This key is extremely important, as it is used to sign all other public keys.

\( PK_{AS} \) The long term public key-pair of the Authentication Server. It is used to authenticate the AS to users during initial authentication. If this key is compromised, it is possible to set up a rogue authentication server and trick the PAS into issuing privilege attribute certificates.

\( PK_{PAS} \) The long term public key-pair of the Privilege Attribute Server. The secret part of this key is used for signing PACs. An adversary having access to the secret part of this key can create PACs that will enable him to gain access to any other service on the network.

\( PK_{PVF} \) The public key-pair of the PVF. Each application server has one of these. It is used to encrypt symmetric keys while in transit from an initiator to the PVF of the target.

\( PK_{U} \) This key is the long term public key-pair of the user U. It is used to authenticate the user to the authentication server. It is also used to authenticate the user to the PVF of the target.

6.5.2 Symmetric keys

Most of the secret keys used are session keys used only during a single session.

\( LK_{AS-PAS} \) This is a long term key shared between the Authentication Server and the Privilege Attribute Server. It is used to encrypt the TGT produced by the authentication server. This means that this key is used to encrypt the key \( BK_{U-PAS} \), used for communication between the initiator and the PAS.
$BK_{U-PAS}$ A key shared between the user and the PAS. This key is used to authenticate the user to the PAS. It is contained in the ticket-granting ticket and is generated by the AS.

$BK_{I-PVF}$ This is a session key generated by the SACM of the initiator and sent to the PVF of the target. It is used to protect the communication between the initiator and the PVF. Its most important task is to be used as a basis key for derivation of keys used to encrypt communication between the initiator and the target.

The message $M$ is integrity protected using $BK_{I-PVF}$. The key is also used to encrypt the check value contained in this message. If this key is compromised an attacker will be able to generate the integrity and confidentiality keys, and therefore be able to listen to and impersonate communication between $I$ and $T$.

$IK_{I-T}$ and $CK_{I-T}$ These keys are used to respectively protect the integrity and confidentiality of the communication channel. They are derived from $BK_{I-PVF}$ by applying a one-way function. A compromise of one of these keys will not enable an attacker to derive any other keys. The hash function is applied by the PVF and the keys are delivered to the target as described above.

### 6.5.3 Kerberos compatibility keys

Some keys are only used to support the Kerberos compatibility of SESAME. These keys all have a counterpart in the Kerberos architecture.

$LK_U$ This is the long term secret key used by a user to authenticate himself. It is stored at the AS and is derived from the user’s password. The corresponding Kerberos key is called the user key.

$LK_T$ The long term key of the target application. Used to encrypt the Kerberos ticket. This key is shared between the KDS and the target. In the Kerberos description this key was called the server key.

$BK_{I-TGS}$ This key is called the tgt key in the description of the Kerberos protocols. The key is generated by the KDS.

$BK_{I-T}$ This is the session key shared between the initiator and the target. It is established by the KDS and is essentially identical to the session keys of Kerberos.
6.5 Key management in SESAME

6.5.4 Key control

In SESAME the initiator usually has key control. The basic key used for communication between the initiator and the target PVF is generated by the initiator. The key derivation functions used by the PVF to derive the confidentiality key and the integrity key are public, and the random offsets used as inputs to those functions are generated by the initiator.

The public keys of a principal is generated by the principal itself. The public key part of the key is transported to the CA for signing, but the principal itself is in control of the actual key generation.

As for the keys used for Kerberos compatibility the key control issues of these keys have been explained in section 5.5.2. Everything said there is relevant for SESAME as well.

6.5.5 The life cycle of a SESAME key

The life cycle of the keys used in the SESAME system will now be described. I will first explain the life cycle of the asymmetric keys and then the symmetric keys.

Asymmetric keys

The asymmetric keys in SESAME are generated by the entity holding them. The CA key is generated by the CA, and user keys are generated by the local workstation of the user. When a certificate is generated by the CA an expiration date is included in the certificate. After this date the key is expired. By default the expiration date is set to be one year into the future.

When a key expires it can no longer be used for authentication. If an attempt to use an expired key is detected the target will reply with a message saying that the key has expired, in addition the event is logged. The current SESAME implementation contains no functionality to automatically request new keys when the old expire. Neither is there any automatic deletion of expired keys from the key database. This will have to be done manually by the administrator.

In the current implementation distribution of asymmetric public keys is done using manual and automatic techniques. Keys are distributed in public key certificates. Initiators must obtain the certificates for the CA, KDS, AS, PAS and the PVFs of the targets they want to access through manual methods. The certificates of an initiator is pushed to the target.
therefore no need for targets to hold the certificates of initiators.

All principals need to obtain a copy of the public part of the CA key through an integrity protected channel. Currently this operation is done by manually copying the CA key from the CA to the principals. Ideally a hash of the key should be published, and the principals should check that their key matches the published hash before trusting the CA key. Currently no such functionality is implemented in SESAME, it is therefore difficult for a principal to check that he is in possession of the correct CA-key.

**Symmetric keys**

There is one long term symmetric key in the SESAME system. This key \((LK_{AS-PAS})\) is used to encrypt the TGT during initial authentication. This key will normally not be changed, and there is no support in SESAME to automatically change this key. Since the TGT has a limited validity period, it is possible to change this key regularly without changing the underlying protocols.

The other symmetric keys are session keys. These keys are transported using a key package. The key packages contain the validity period of the key within the package. This validity period is checked upon the initial validation of the package.

When the validity period runs out the key should no longer be used. The SESAME documents do not give any information on how this situation is handled. One solution is to carry out the authentication process once more. The currently open connections could then start to use the new key.

### 6.5.6 Key hierarchy

A diagram of the SESAME key hierarchy is shown in figure 6.2. The ovals in the figure are asymmetric keys while the rectangles are symmetric keys. I have chosen to include both signing, encryption and derivation in the figure.

At the top of the hierarchy is the \(CA\) key. This key is used to sign other public keys in the system. These other keys fall into one of four groups. The first is the public key of the privilege attribute server \((PK_{PAS}\) in the figure). One such key exists in each SESAME domain. The next is the public key of the PAC validation facility. This key is used to encrypt the session key exchanged between the initiator and the remote PVF. The \(PK_{AS}\) key is used to authenticate the authentication service to the user such that the AS
cannot be impersonated.

The last public key is the user key (\(PK_U\)), it encrypts the \(BK_{U-PAS}\) key that is used to authenticate the user to the privilege attribute server. The \(BK_{U-PAS}\) key is transferred between the AS and the PAS under encryption by the \(LK_{AS-PAS}\) key. If Kerberos authentication is used \(BK_{U-PAS}\) is encrypted with the password-derived user-key \(LK_U\). It should be noted that the key \(BK_{U-PAS}\) can be encrypted using either \(PK_U\) or \(LK_U\). But it will not be the case that the same key is encrypted using both keys at the same time.

The keys used for Kerberos compatibility are mostly separated from the rest of the keys. Of all the long term keys stored in the master database and encrypted with the key database master key, only the key \(LK_{AS-PAS}\) is used when the native SESAME protocols are used. \(LK_T\) corresponds to the server key in Kerberos and \(LK_U\) to the user key. \(BK_{I-TGS}\) serves the same purpose as the Kerberos tgt key.

Note that no keys are used for both signing and encryption. This is in correspondence with good cryptographic practice.

6.6 The SESAME trust model

I have now described the different components of the SESAME system, and it is time to look at the trust relationship between them.

The initiator components and the target components are trusted by their principals and their target applications.
There is a strong link of trust between the AS and the PAS. The AS authenticates the user and the PAS will have to trust this authentication to be able to issue privilege attribute certificates.

Each target application is sponsored by a PVF. The PVF of the target is responsible for performing access control decisions on behalf of the application. The target application therefore trusts its PVF unconditionally.

To be able to make access control decisions, the PVF must trust the principal who issued the credential used for access. There are two distinct cases here. If asymmetric cryptography was used the credential used is a PAC. The PAC is generated and signed by the PAS and consequently the PVF must trust the PAS. The identity of the PAS is guaranteed by the signature of the CA on the certificate of the PAS. The PVF therefore must trust the CA. When using Kerberos compatible symmetric cryptography, the link of trust is between the PVF and the KDS. The ticket used for authentication is generated by the KDS, and the PVF must trust the association made by the KDS between the session key in the ticket and the identity of the initiator.

To avoid delegation of privileges not intended to be delegated, the PVF plays a crucial role. If the target is not allowed to act as a delegate the PVF of the target should not return the check value to its application. This means that an initiator has to trust the target PVF, even if the target application is not trusted.

Table 6.2 gives an overview of the trust relationships described in this section.

### 6.7 Sesame limitations

SESAME has not been as thoroughly analyzed in the literature in the same manner as Kerberos. Most of the limitations described in this section have been revealed during my own analysis of the system, except where noted.
otherwise. None of these limitations have catastrophic effects on the security of SESAME.

### 6.7.1 The PFV is trusted

Each server needs to have its own trusted PVF component. In the current implementation the PVF runs as a separate process. If a target computer is compromised it will be easy to work around the delegate target qualifier protection method. This protection method relies on the PVF not to give the $CV$ away to principals who are not allowed to act as delegates.

Server processes often must run as a trusted user on the local machine. Since trusted processes have the ability to bypass access control on disks and memory, it will not be difficult to gain access to the $CV$, even if the PVF does not give it away voluntarily.

The SESAME documentation [52] mentions that future implementations might separate the PVF from the applications it supports. This will solve this problem. If this approach is to be used the protocol used between the SACM and the PVF must be changed. As it is now it assumes that the communication channel is secure.

Such a solution will not be without problems. An important advantage of the current solution is that it is possible to verify a PAC without contacting any trusted on-line servers. The verification of a PAC will take longer if a remote server must do the job.

### 6.7.2 Critical keys stored on the user’s workstation

Two critical keys needs to be stored on the user’s workstation in the SESAME architecture. These keys are the public key of the CA used for verification of certificates, and the secret part of the user’s public key-pair.

The SESAME V4 implementation stores the private part of public keys unencrypted in plain files in the user’s home directory. In many configurations this will destroy the security of the system completely. It is common for home-directories to reside on remote servers, using unencrypted network file systems. An adversary could pick up the key while in transit from the server to the workstation. Even if the key is stored on a local disk, this situation is unacceptable. Other people could gain physical access to the workstation and fetch the key directly from the disk, or the workstation could become cracked by outsiders. The easiest solution would be to encrypt the key on the disk using a key derived from a password.
It is common for users to move from one computer to another, and expect their files to move with them. Authenticating access to home directories is needed to do this in a secure manner. Since SESAME stores the key on the local workstation this is not possible. One possible solution to this problem is to store the user’s key on a token the user can carry with him, there is no support for this in the current version of SESAME.

The public key of the CA does not need to be protected from compromise like the user’s key. It must be protected from tampering. If an attacker can change the public key part of the CA key and replace all the other keys with keys signed by the new CA key, it will be possible for the attacker to spoof the entire SESAME domain. The user will think that he is talking to the correct AS, PAS and target, while in reality he could be talking to someone else. It will not be possible to listen in on communication to legitimate targets because it will not be possible for the attacker to create a PAC acceptable by a legitimate target.

6.7.3 Complicated architecture

The decision to base the SESAME protocols on the Kerberos protocols has made the architecture unnecessarily complicated. While this can be seen as a wise decision from a political and marketing point of view, it adds complexity to the protocols. Complexity is never a good thing when it comes to security, simple systems that are easy to analyze and implement must be the goal.

Several keys exist in the system only to support the Kerberos compatibility. This results in more keys that can become compromised, and more possible ways of attack for an adversary.

The Kerberos compatibility of SESAME is an additional feature, and it looking at it as a limitation might be strange. In any case the Kerberos compatibility cause all the limitations mentioned for Kerberos to apply to SESAME as well. Specific problems that might apply to the Kerberos that the SESAME protocols do not have, will creep into the system anyway.

6.7.4 Weak Public Key Infrastructure

The PKI implemented in SESAME lacks several security relevant features. The PKI has no support for certificate revocation. The PKI is not intended to be complete, and the modularity of the architecture makes it easy to replace the PKI. Handling certificate revocation is an important part of any system
6.7 Sesame limitations

<table>
<thead>
<tr>
<th>$SACM_I$</th>
<th>$SACM_T$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\rightarrow$</td>
<td>$e_{CK_I-T}(m, t_I^t, n_I^t, SEAL_{IK_{I-T}}(m))$</td>
</tr>
<tr>
<td>$e_{CK_I-T}(m, t_T^t, n_T^t, SEAL_{IK_{I-T}}(m))$</td>
<td></td>
</tr>
</tbody>
</table>

Protocol 6.6: Sesame SES-DATA (improved version)

using asymmetric cryptography. Keys will get lost and become compromised, there must be routines in the system that can prevent damage when these unfortunate situations happen.

As described in section 4.5.2, the CA should check that the is in possession of the private part of the key he is requesting certification for. In the SESAME PKI no such checks are done. The key is generated by the user, and a copy of the public key is sent to the CA. The CA has no way of knowing whether the secret part of the key is held by the principal requesting certification.

6.7.5 Encryption before integrity protection

If the SES-DATA message is observed, we can see that encryption is done before integrity protection. Good cryptographic practice is to reverse these operations. Vandcnwauver and Ashley [2] propose to replace the message with protocol 6.6.

6.7.6 Unpolished implementation

The current implementation of SESAME is not intended for general usage. It can only be considered to be a research prototype

Setting up a SESAME system can be time-consuming and error-prone. Abnormal situations are not reported directly back to the user. The SESAME software has a lot of debugging output enabled, this output is written to specific files. To find the cause of an error a lot of debug messages must be investigated, most of these report normal situations. Finding the exact lines that report the error can be difficult. For example if the key of a user has expired, the SESAME client will only say that the authentication failed. The debugging logs must be examined in order to find the real reason for the failure.

The administrative utilities are crude and not very user friendly. Setting up the keys and adding users to the system requires running several different programs. Files must be copied between accounts and keys must be
exported from key databases. Even if the user has a thorough understanding of the architecture it is not always easy to understand exactly what is happening.
Chapter 7

Comparing Kerberos and SESAME

Kerberos and SESAME both have the same basic goal, they want to provide reliable security in a distributed network. SESAME was designed after Kerberos, and has chosen to solve some problems differently. The main differences are that SESAME has chosen to use asymmetric cryptography extensively while Kerberos uses conventional cryptography. Another important difference is that SESAME provides role-based access control, while the access control in Kerberos is identity-based.

This chapter will focus on the differences and equalities of SESAME and Kerberos, and the influence their differences have on the key management and key distribution subsystems.

The design process of the two technologies was done differently. SESAME was designed by an industry consortium, adding to work done in ISO and ECMA. Kerberos on the other hand was designed as a solution to a specific problem inside a single organization. MIT needed a system that could protect the emerging Athena distributed computing system. Kerberos has since its inception been actively used and improved upon. The software is easy to install, and the administrative procedures are well developed. The Kerberos software has been freely available on the Internet for a long time, and it has a large user base.

SESAME is freely available, but the license does not allow for non-experimental installation without consent from the SESAME partners. This has resulted in fewer installations and contributions to SESAME from the outside. The SESAME partners have built upon the lessons learned from the
SESAME process, and have developed new security products. These products are not described in the open literature like SESAME itself is. The SESAME implementation available on the Internet is a reference implementation, it does not attempt to be complete. This is particularly apparent when it comes to administration and installation. The SESAME distribution is much more rough on the edges and less polished than Kerberos. While Kerberos has continued to evolve after the initial design period and has been adapted to the needs of the users, SESAME has not been in open active development since release of V4 in 1995.

The fact that the implementations differ, both in stability and in development model should not be a reason to judge the merits of the security architectures. Security is about both theory and practice. While a good implementation cannot make up for a bad architecture, a bad implementation could easily compromise the security of a sound architecture. The SESAME implementation was never designed to be deployed unaltered in the field. The implementation might have problems that are not inherent in the architecture. I will be careful to distinguish between problems that exist in the architecture and in the implementation.

7.1 Architectural differences

The architecture of SESAME is much more rigorously specified than the architecture of Kerberos. This is mostly due to the fact that SESAME owes its heritage to standardization work done in ISO and ECMA. The basic components are the same, but the designers of SESAME have chosen to break some of the components up into sub-components, resulting in an architecture with more parts.

The three main components of both Kerberos and SESAME are the client, the server and the security server. The security server of Kerberos contains the authentication server (AS) and the key distribution server (KDS). These components also exists in SESAME. SESAME additionally has a privilege attribute server (PAS). The PAS is an important component of the SESAME role-based access control system. Since Kerberos does not implement role-based access control, this component is not needed for Kerberos.

Since SESAME uses asymmetric technology, the SESAME architecture includes a PKI. Kerberos does not need this service since only asymmetric keys are used.
7.2 Protocols

The protocols are the core of the architecture. While SESAME is Kerberos compatible and can utilize the Kerberos protocol, I will here compare the native SESAME protocols with the Kerberos protocols. The focus is on issues that are relevant to how key management is done.

7.2.1 Initial authentication

SESAME supports the Kerberos authentication method, and additionally provides a native public key based protocol. The SESAME authentication protocol provides mutual authentication of the client and server. This means that a user logging on to the system can know immediately whether he is in contact with the correct authentication server or not. Likewise a SESAME server is able to determine if the user is successfully authenticated or not. This is not possible with the Kerberos protocol.

7.2.2 Privileges

The handling of privileges is where SESAME and Kerberos differ the most. The decision to support role-based access control in SESAME has had large implications on the underlying protocols.

In Kerberos a user is issued a ticket containing his identity. In SESAME a privilege attribute certificate is issued. The certificate contains the user’s privileges. The problem of checking whether a user is granted access to a resource is therefore much simpler in SESAME. In Kerberos each service must have a list of all the principals that are allowed access, while in SESAME it will suffice to have a list of roles. This list will be much shorter.

In section 5.4.3 it was mentioned that the Kerberos authentication system will break down if principal identifiers are reused. SESAME does not exhibit this particular problem. If an identifier is recycled, only the privilege attribute list stored at the privilege attribute service will have to be changed.

Another important difference between the two systems is that the SESAME PAC is valid for any target, while the Kerberos ticket is only valid for one specific target. This means that a SESAME client doesn’t have to contact the PAS every time he wants to contact a new server. It will suffice to get one PAC and this will be valid for all targets. Another implication of this is that there are fewer keys having to be protected on a SESAME client. The
PAC itself contains no keys, and the only other key that is needed is the public key of the client. In Kerberos one key must be stored for each ticket the client holds.

7.2.3 Authentication to the target

The authentication of the initiator to the target is done using asymmetric cryptography in SESAME, and symmetric cryptography in Kerberos. If a Kerberos client wants to contact a server that it has not been in contact with before, it must obtain a ticket for the server. In SESAME this step is not needed since the PAC contains all the keys needed to contact any server. Immediately it seems like the SESAME solution will scale better because the number of times a client will have to be in contact with the security server will be lower. In fact a SESAME client will only have to contact the security server at the start of each login session, and when the PAC expires. How big this advantage is depends very much on the usage pattern on the network. If it is common to contact a large number of separate servers the SESAME method will be better. In most networks there are a relatively low number of servers, and the difference might not be large in practice.

7.2.4 Sending of protected data

As described in section 6.7.5, SESAME encrypts data before it is integrity protected. In Kerberos the data is encrypted using a shared key, and the underlying encryption primitive is assumed to provide integrity protection. In practice this means that a checksum is added to the message before encryption.

Kerberos does not distinguish explicitly between keys used for integrity protection and keys used for confidentiality protection. In SESAME two keys are derived from one session key. Since both integrity protection using a MAC and encryption is done in SESAME, this makes sense. In Kerberos where the encryption function itself is assumed to provide the required integrity protection, the derivation of separate keys is left to the cryptographic layer.

7.2.5 Inter-domain operation

A domain is called a realm in Kerberos, but they are essentially the same thing. I will use the term domain for both concepts in this section.
In Kerberos inter-domain operation requires that a special cross-realm key is established. In SESAME the CA of one domain must sign the public key of the PAS of the other domain. Both solutions require an agreement between two domains in advance. An authentication across domain boundaries in SESAME will require less resources than it will in Kerberos. The reason for this is that no security servers have to be contacted in order to carry out the authentication. In Kerberos the local ticket-granting service will first have to be contacted. The ticket obtained in this step can then be used to contact the remote ticket-granting service and then finally a ticket for the target server can be obtained. It is clear that the SESAME solution is more elegant, and more scalable.

Another advantage with the SESAME approach is that no new keys are needed to support inter-domain operation. Each new key added to the system is one more key that can be lost or become compromised.

7.2.6 Renewal of credentials

Both the tickets of Kerberos and the PACs of SESAME have a limited validity period. It is common for users to stay logged in for a long period of time. Many users don’t log out when they leave the office.

Having to enter the password once every day is awkward for the user. Credentials that expire can also be a problem for processes that run for a long time. Kerberos has added the renewable flag to the ticket to solve these problems. In SESAME it is not possible for a SESAME client to renew the credentials without going through the authentication process again. If strong authentication using private keys is used, this can be done without user intervention provided that the private part of the public key-pair of the user is stored unencrypted.

How to handle renewal is a matter of security policy. Manual renewal ensures that credentials residing on workstations not in use expire, for example if a user stays logged in when going on an extended vacation. Automatic renewal of credentials defeat some of the intentions in having the credentials expire.

7.2.7 Handling of forwardable credentials

Both SESAME and Kerberos have chosen to implement forwarding of credentials.

Forwarding is useful when one entity in the system wants to delegate ac-
cess rights to another entity. For example, a print server could be given the right to access files on the home directory of a user.

The actual implementation of proxiable credentials in Kerberos and SESAME differs a lot. In Kerberos the tickets are marked as forwardable or proxiable as described in section 5.3.1. New tickets that are valid for specific targets can be issued. As usual in Kerberos it is the KDC that is in control in these operations, and the KDC gets to determine whether a new ticket should be issued or not.

In SESAME there is no need to contact any security server in order to enable proxying. The SESAME solution is to use the PV/CV protection method of the PAC (see section 6.4.4). The client is in control of which entities who are allowed to use a PAC. The client can do this by choosing whether or not to forward the CV.

Again the SESAME solution scales better. There is no need for a SESAME client or server to contact any security services in order to do forwarding of credentials.

### 7.3 Key management

The aspect of key management has not attracted much attention in neither the SESAME nor the Kerberos documents. Technical solutions are in place that distribute keys as part of the protocol, but not very much is said about how the keys should be administered properly. In my opinion the key management services described in chapter 4 should be incorporated as a natural part of the architecture. These functions might have most influence on the implementation, but the architecture should nevertheless be aware of these issues.

When protocols suites such as Kerberos and SESAME mature and their security implications are well understood, the most obvious point of attack will be on the key management systems and routines. An attacker will consider the protocols so well analyzed that he will not try to attack them directly. Since secure handling of keys is largely a question of administrative routines and the attitude of each individual, the secure management of keys will be harder to implement in organizations than the protocols themselves. The software must help in making this process easier.

Where the keys are stored should, at all times, be obvious to the administrator. In Kerberos and SESAME this is currently not the case. A thorough understanding of the architecture is needed to understand how the key management works. The key management issues should be clearly pointed
7.3 Key management

out in the documentation, and the software itself should also be as helpful as possible. System administrators can not be expected to have knowledge of cryptographic protocols. Without this knowledge they can easily make mistakes with security implications. Normal administrative tasks such as making backups of filesystems could, for example, easily subvert the security if unencrypted keys are carelessly handled by the backup system.

7.3.1 Key storage

Where keys are stored is to a large extent determined by the architecture. Neither the Kerberos RFC nor the SESAME architecture documents have anything to say about how keys are stored.

User keys

In SESAME each user has a public key-pair being used for authentication. The secret part of this key must be stored at the user’s workstation. Kerberos uses a key derived from a password for initial authentication. This means that Kerberos does not have to store any keys at the user’s workstation. The key is indirectly stored in the user’s mind.

There are advantages and disadvantages to each of these approaches. Using a password-derived key has the obvious advantage that no security sensitive data has to be stored at any workstation. The use of passwords does definitely make sense in the setting that Kerberos was developed. One of the main goals was that the workstations should not have to be trusted. This makes sense both from a security and an administrative perspective.

The SESAME V4 implementation stores the private part of public keys unencrypted in plain files in the user’s home directory. In many configurations this will destroy the security of the system completely. It is common for home-directories to reside on remote servers, using unencrypted network file systems. An adversary could pick up the key while in transit from the server to the workstation. Even if the key is stored on a local disk, this situation is unacceptable. Other people could gain physical access to the workstation and fetch the key directly from the disk, or the workstation could become cracked by outsiders.

A solution to this problem would be to encrypt the key using a password derived key. The question is whether the use of public key technology for initial authentication offers any considerable security advantage to using the traditional approach of Kerberos. If the situation is as in the SESAME implementation, where public keys are stored unencrypted, the security
of the system is considerably lower. If keys are stored encrypted under a password, many of the same problems as with the password scheme in Kerberos arise. If the adversary can get access to the encrypted key, the system becomes vulnerable to the same password guessing attacks as described in section 3.4.3. Making a scheme for password protection of private keys that does not allow off-line password guessing is difficult. You can always know when you have the correct key by trying to encrypt a piece of data and see if decryption with the public key yields the same data.

It is common for users to move from one computer to another, and expect their files to move with them. Authenticating access to home directories is needed to do this in a secure manner. Since SESAME stores the key on the local workstation this is not possible. One possible solution to this problem is to store the user’s key on a token the user can carry with him, there is no support for this in the current version of SESAME.

Server keys

Long term server keys have to be stored at the server. They can be stored in encrypted form, but this will need some kind of manual intervention during boot to supply the master key. One solution is to have a person enter the key at the keyboard during boot. The optimal solution is to store important server keys in cryptographic hardware modules as described in section 4.2.4.

The server key in Kerberos is a symmetric key, while in SESAME it is an asymmetric key. In practice this will have little impact on how the server key is stored and managed at the server.

7.3.2 Key control

In Kerberos the issue of key control is as earlier mentioned (see section 5.5) quite simple. The KDS is always in control, except when negotiating sub-session keys. Even the session keys used for communication between a client and a server are generated by the KDS.

In SESAME the session keys are generated at the initiator side. Even the public key of an entity is generated by the entity itself. As I said in section 3.2.6 key generation implies a certain form of trust in the generating entity. In Kerberos the only place in the system where key generation is done is at the KDC. In SESAME every client must be able to generate good keys.
Both Kerberos and SESAME solve the problems they were designed to solve. Kerberos has been in use at MIT and other sites around the world for over ten years, and SESAME has been the basis for implementations of other security related products. None of the limitations and weaknesses that have been found in these systems have had catastrophic impact on the security of the systems.

The use of symmetric cryptography in Kerberos has led to an architecture where security decisions are made centrally. Access control issues and cryptographic keys are managed centrally. In contrast the SESAME architecture is much more distributed in nature. While access decisions are done centrally (role-based access control), the management of asymmetric long term keys and symmetric session keys is distributed to the clients and servers in the network. The SESAME solution scales better because it is possible to set up secure associations between two parties without involving an online trusted third party. As described in section 6.7.2 this does not come without a cost. SESAME clients must store critical keys locally. If the users’ keys are not stored in a tamper-proof module a SESAME system cannot be made secure.

When it comes to the actual implementation, Kerberos has a definitive advantage. The specification of Kerberos is intended to support multiple independent implementations, and several do exist. They are solid and they are being actively developed and maintained. The freely available SESAME implementation is not good enough for anything but research use.
Bibliography


