UNIVERSITY
OF LONDON

Computer security

R. Shipsey

CO3326

2009

Undergraduate study in
Computing and related programmes

Goldsmiths
UNIVERSITY OF LONDON

This guide was prepared for the University of London by:

R. Shipsey

This guide was produced by Sarah Rauchas, Department of Computing, Goldsmiths, University of London.

This is one of a series of subject guides published by the University. We regret that due to pressure of work the author is unable to enter into any correspondence relating to, or arising from, the guide. If you have any comments on this subject guide, favourable or unfavourable, please use the form at the back of this guide.

In this and other publications you may see references to the 'University of London International Programmes', which was the name of the University's flexible and distance learning arm until 2018. It is now known simply as the 'University of London', which better reflects the academic award our students are working towards. The change in name will be incorporated into our materials as they are revised.

University of London
Publications Office
32 Russell Square
London WC1B 5DN
United Kingdom

london.ac.uk

Published by: University of London

Contents

Preface

This is a level three half-unit subject for the BSc Computing and Information Systems programme. It aims to serve as an introduction to some aspects of computer security. One of the major roles of computers in the 21st century is the generation, storage and communication of data. This data may be sensitive or confidential and unauthorised disclosure – whether accidental or malicious – can cause major problems. There are many examples of the need for security ranging from the private to the global. For example, your medical records and bank details will be stored on a computer somewhere. These details are personal and confidential and you would only want people with the appropriate authority to see this information about you. In many countries there are data protection laws that are supposed to enforce the privacy of any personal data stored for whatever reason by a company or government agency. On a larger scale, it may be of the utmost importance that military secrets are kept confidential and only accessible by those who have the highest security clearance.

This guide will introduce you to some important techniques of computer security including:

- passwords and identification
- encryption
- access controls
- symmetric key cryptosystems
- asymmetric key cryptosystems
- digital signatures
- key management
- hash functions
- internet protocols.

Some characters will appear frequently throughout the subject guide. These are Alice and Bob who are always sending each other messages using the cryptographic protocols and methods described in the subject guide. Alice and Bob just represent any sender and receiver, A and B, but it is often helpful to think of them as people. Another character, the bad-guy, who makes a frequent appearance is Charles who represents a cryptanalyst or hacker.

It is the intention of this subject that you become familiar with the need for security in computing systems, and learn about some particular computer security techniques that are currently in use. You should know how these techniques can be applied and the range of problems where they are applicable. Learning outcomes are given at the start of the chapter. These are a summary of what you should learn from the chapter. However, be aware that the examination questions may cover any of the material in the guide and any additional material covered in the coursework.

Each chapter also includes learning activities and sample examination questions which can be used to test your understanding. Solutions to the examination questions are given at the back of the guide. Some of the activities do not have

solutions either because they are practical exercises, or because there is no 'right' answer.

Examination questions are likely to test material from more than one chapter. Therefore the examination questions at the end of each chapter may be partial rather than complete questions. An entire examination paper with solutions is included at the end of the guide so that you can see the type and level of questions to expect in the examination.

Prerequisites

No particular computing units are required as prerequisites for this subject. However, as this is a third year subject, it is assumed that students have some mathematics and computing experience. Please do not be put off by the mathematics. You need to be able to apply the given mathematics to particular problems in computer security, but you do not need to reproduce any proofs.

The ability to program in java will be useful, particularly when completing the coursework. However, remember that the coursework is designed to test your understanding of computer security and not your programming ability. Examiners are not looking for elegant programming solutions or fancy data input screens.

Method of assessment

There will be one examination lasting 2 hours and 15 minutes at the end of the academic year. This examination consists of five questions; candidates have to answer **any three out of** these **five** questions to achieve full marks. The examination is worth 80 per cent of your final mark for this subject.

You are also expected to complete two coursework assignments. These each count as 10 per cent of your final mark. The coursework assignments typically include some independent research and practical work.

Recommended reading

The subject guide is a complete account of the subject. However, you should be aware that developments in the world of computer security means that the subject guide can never be completely up-to-date. You are therefore advised to access further reading wherever possible to keep abreast of the current state of technology in this field.

Following is a list of books that are recommended. By no means do you need to have copies of all of these books but a selection of your choice would complement the material covered in the subject. Some of the books are available to download free of charge; where appropriate I have given the website addresses.

Anderson, R. ***Security Engineering – The Book.***

Six sample chapters of this book are available to download free of charge from the website – `http://www.cl.cam.ac.uk/$\sim$rja14/book.html`. The first five chapters deal with protocols, passwords, access control and cryptography and are particularly relevant to the subject.

Ferguson, N. and B. Schneier ***Practical Cryptography.*** (New York; Chichester: Wiley, 2003) [ISBN 0471223573](pbk).

As the title suggests this book is a practical guide to choosing and using

cryptographic tools. Some students may prefer this practical approach over the more academic approach of traditional text books.

Gollmann, D. ***Computer Security.*** (Hoboken, NJ: Wiley c2006) second edition [ISBN 0470862939(pbk)].

This book provides useful further reading on identification and authentication and access control, including a chapter on Unix security. There are lots of thought provoking and practical exercises.

Menezes, A., P. Van Oorschot and S. Vanstone ***Handbook of Applied Cryptography.*** (Boca Raton, Fla.; London: CRC, 2001) fifth edition.

Chapters of this book are available to download free of charge from `http://www.cacr.math.uwaterloo.ca/hac/index.html`. This book is intended for professional cryptographers and is the ultimate reference book for cryptography. Thorough details on all aspects of cryptography are given. This book is over 700 pages long and is not recommended for a light read but it is well worth looking at the website.

Pfleeger, C. and S. Pfleeger ***Security in Computing.*** (Upper Saddle River, NJ; London: Prentice Hall 2007) fourth edition [ISBN 9780132390774].

In this book the authors introduce the core concepts of computer security and then identify and assess the threats currently facing programs, operating systems, database systems and networks. Attacks on RSA, SHA and DES are discussed.

Piper, F. and S. Murphy ***Cryptography: A Very Short Introduction.*** (Oxford: Oxford University Press, 2002)[ISBN 9780192803153].

This really is a very short book and a great introduction to cryptography. The ideas behind symmetric key and public key cryptography and their uses are clearly explained.

Schneier, B. ***Secrets and Lies: Digital Security in a Networked World.*** (Wiley, 2004) new edition [ISBN 0471453803].

Computer security from a business world perspective. This book examines the necessity for computer security in the real world. It is written more like a reading book than a textbook and gives an interesting background to the subject of computer security.

Stallings, W. ***Network Security Essentials: Applications and Standards.*** (Upper Saddle River, NJ; Harlow: Pearson Education, 2008) second edition [ISBN 9780132303781(pbk)].

This book covers internet security tools and applications. It includes good descriptions of symmetric cryptosystems including DES, 3DES, AES, IDEA, Blowfish and RC5. There are lots of exercises and test questions.

Chapter 1

Security

1.1 Introduction

In this chapter, we will introduce the notion of computer security, provide some basic definitions and discuss the features that a good security system should provide.

Supplementary reading

Chapter 1 of *Computer security* by Gollmann gives a good introduction to the notion of computer security.
Part 1 of *Secrets and Lies* by Schneier is easy to read and puts computer security into context.

After studying this chapter and the additional reading, you should be able to:

- Recognise the need for computer security and describe how computer security differs from security in the traditional sense.
- Define what is meant by the terms integrity, availability, non-repudiation, authentication, accountability and access control with regards to computer security.
- Discuss the various types of attack that may threaten a security system.
- Discuss the many design considerations to take into account when designing a security system.

1.2 What is security?

In the broadest sense security can be defined as the protection of assets. There are three main aspects to security:

- prevention
- detection
- reaction.

Consider security in the traditional sense – for example, securing your house against burglary. You may take steps to **prevent** a burglary such as locking the doors and windows and installing a burglar alarm. If a burglary did occur, you would be able to **detect** this because items would be missing and the burglar may have caused damage to your house while breaking in. You might **react** to the burglary by reporting it to the police, working out what had been stolen and making an insurance claim.

1.2.1 How is information security different?

Although the definition of security given above still applies when we are talking about information, there are some major differences between traditional security and information security.

- Information can be stolen – but you still have it.

  If a physical item such as a car is stolen then the thief has possession of the car and you no longer have it. If a thief steals a file from your computer, he will probably make a copy of the file for himself and leave the original on your computer. Hence you still have the file but it has also been stolen.
- Confidential information may be copied and sold – but the theft might not be detected.

  If your car has been stolen it is not hard to detect the fact – the car is missing! However as mentioned above, a thief who steals computer files may leave the files on your computer and only copy them for himself. Nothing appears to have changed on your computer so you may not be aware that anything untoward has happened.
- The criminal may be on the other side of the world.

  If a thief steals your car you at least know where he was when he stole the car. However, it is possible to hack into computer systems remotely from anywhere in the world. This makes it very hard to know who is responsible for catching a computer criminal. Is it the police in the country where the computer is, or the police in the country where the criminal is?

Although there is no single definition of computer security, we can say that:

> Computer security deals with the prevention and detection of unauthorised actions by users of a computer system.

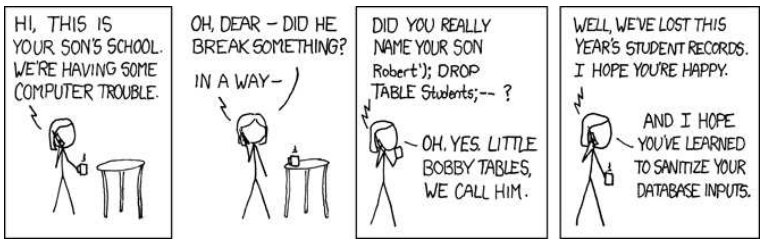
This subject deals with the theory of computer security. You should be aware that unfortunately things that are great in theory do not always work in practice. As Schneier says in *Secrets and Lies*:

> *Theory works best in ideal conditions and laboratory settings. We can design idealised operating systems that are provably secure, but we can't actually build them to work securely in the real world. The real world involves design trade-offs, unseen variables and imperfect implementations.*

Schneier kept a log of 'security events' for the first week of March 2000. He recorded approximately 100 events during this time including hackers launching denial-of-service attacks, leakage of personal data from supposedly secure websites, email worms and viruses, and websites being defaced. Most of these attacks and vulnerabilities were the result of the perpetrator bypassing the security mechanism, or exploiting a weakness in the system such as an overflowing buffer.

Learning activity

The following cartoon by Randall Munroe is taken from xkcd.com.



Do an Internet search on SQL injection vulnerabilities to find out why this is funny!

1.3 Features of a security system

In order to prevent and detect unauthorised actions by its users a good security system should provide (some of) the following features:

- confidentiality
- integrity
- availability
- non-repudiation
- authentication
- access controls
- accountability.

We will look at each of these features in turn. Note that different authors on computer security disagree as to which of these features are the most important. It will depend on the main purpose of the system – is confidentiality paramount or is the prevention of denial of service attacks more important? This will depend on the system in question. For example a computer system which holds personal medical records must certainly provide access controls in order to ensure that personal information does not fall into the wrong hands, and integrity to ensure that the information stored is accurate. Other features such as non-repudiation and availability may not be so important in this case. On the other hand, it is essential for a computer system which transfers money electronically to guarantee non-repudiation and accountability in order to prevent and/or detect dishonest transactions occurring.

In this context, the term *unauthorised* implies not only malicious or criminal, but could also be accidental. For example, a breach of confidentiality arises maliciously if a spy deliberately hacks into a computer and looks at confidential material stored there. It happens accidentally if the material is left out on a desk and is seen by the office cleaner.

1.3.1 Confidentiality

Confidentiality is the prevention of unauthorised disclosure of information.

In other words, confidentiality means keeping information private or safe. Confidentiality may be important for military, business or personal reasons. Confidentiality may also be known as *privacy* or *secrecy*.

1.3.2 Integrity

Integrity is the prevention of unauthorised writing or modification of information.

Integrity in a computer system means that there is an external consistency in the system – everything is as it is expected to be. *Data integrity* means that the data stored on the computer is the same as what is intended.

1.3.3 Availability

Availability is the prevention of unauthorised with-holding of information.

Information should be accessible and usable upon appropriate demand by an authorised user. *Denial of service* attacks are a common form of attack against computer systems whereby authorised users are denied access to the computer system. Such an attack may be orchestrated by the attacker flooding the system with requests until it cannot keep up and crashes. Authorised users are unable to access the system. Consider the damage that such an attack may cause to an electronic commerce site such as an internet shop.

1.3.4 Non-repudiation

Non-repudiation is the prevention of either the sender or the receiver denying a transmitted message.

A computer security system must be able to prove that certain messages were sent and received, who sent the message, who received the message and perhaps what the message said. For example, suppose a dishonest trader sends an electronic message to a stock broker telling him to buy £2,000 worth of shares in CryptoCom. The next day the price of CryptoCom shares soars. The trader now pretends that his original message said to buy £20,000 worth of shares. Conversely if the share price fell he might pretend that the original message said to buy shares in KryptoCom instead. Non-repudiation means that the trader is not able to deny his original message.

Non-repudiation is often implemented by using *digital signatures* (see section 7.3).

1.3.5 Authentication

Authentication is proving a claim – usually that you are who you say you are, where you say you are, at the time that you say it is.

Authentication may be obtained by the provision of a password or by a scan of your retina for example. See Chapter 2 for further methods of authentication.

1.3.6 Access controls

Access controls provide the limitation and control of access to authorised users through identification and authentication.

A system needs to be able to identify and authenticate users for access to data, applications and hardware. In a large system there may be a complex structure determining which users and applications have access to which objects. See Chapter 3 for further details on access control models.

1.3.7 Accountability

Accountability means that the system is able to provide audit trails of all transactions.

The system managers are accountable to scrutiny from outside the system and must be able to provide details of all transactions that have occurred. Audit trails must be selectively kept (and protected to maintain their integrity) so that actions affecting security can be traced back to the responsible party.

Learning activity

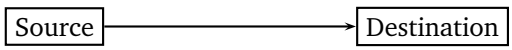
Consider the following scenario and think about the questions at the end.

A student suspects there is a vulnerability on a system in a university public access laboratory. She tests this by trying to exploit the vulnerability. She succeeds, and obtains privileges that she would not normally have. She reports both the hole and her exploiting it to the system staff, who in turn report it to the manager of the laboratory. The manager files charges of breaking into the computing system against the student. The student has to appear before the Student Judicial Authority – she is in trouble!

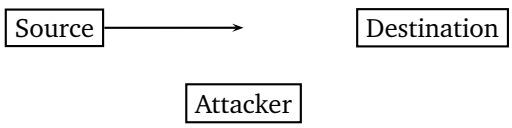
1. Did the student act ethically by testing the system for the security hole before reporting it?
2. Did the manager act ethically by filing charges against the student?
3. The manager told the system staff not to bother fixing the hole, because the action taken by the SJA would deter any further break-ins through the hole. Was the manager's action appropriate?

1.4 Security attacks

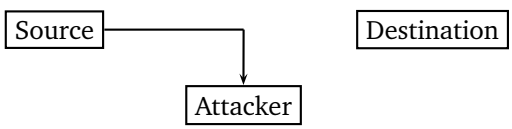
There are a number of ways in which an attacker can disrupt communications. Normally, information goes from the source to the destination.



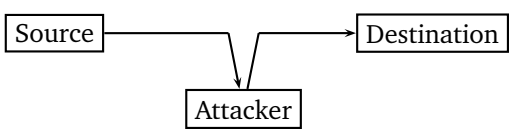
Communication is *interrupted* if the attacker does not allow the information to reach the destination.



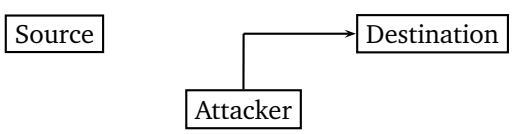
Communication is *intercepted* if the attacker interrupts the communication and receives the source information.



Modification occurs when the attacker intercepts the communication, alters it in some way, and then sends it on to the destination. The attacker intends to deceive the destination into thinking that the modified communication has come directly from the source. This is also known as a *Man-in-the-middle attack*.



An attacker may also make up a communication and send it to the destination pretending that it has come from the source. This is called *fabrication*.



1.5 Security systems

A computer security system is not just a computer package. It also requires security conscious personnel who respect the procedures and their role in the system. For example, an access control system may be rendered worthless by employee Fred Smith who chooses user-name *Fred* and password *Smith* and therefore leaves the system open to abuse by password hackers (see section 2.3.1). Conversely, a good security system should not rely on personnel having security expertise.

1.5.1 Risk analysis

When designing or implementing a computer security system it is very important to bear in mind the level of risk involved and the value of the information that is to be protected. As an illustration, consider that you may be willing to leave £50 in a changing room locker, but you would not be likely to leave £5,000 unattended. You would assess the risk involved before deciding whether to leave the money or not. On the other hand, it would be foolish to pay someone, say £20, to look after your £50, but this might be a good investment in the case of the £5,000 (assuming that you totally trust the person charged with keeping your money safe of course!).

In terms of computer security, the disadvantages of security systems are that they are time consuming, costly, often clumsy, and impede management and the smooth running of the system. *Risk analysis* is the study of the cost of a particular system (in terms of effort and time as well as cost) against the benefits of the system (the level of security offered).

Learning activity

Think about your own circumstances and where you might be affected by computer security. For example, do you use a password or PIN for any purpose? Are there medical or employment records about you? What features of a security system are involved with each example?

Consider a computer system that you are familiar with; for example, perhaps you have a networked system where you work or study, or a PC at home that is used by more than one person. How good is the security of the system? How easy is it to access other people's files or to read their emails? How difficult would it be to add extra security to the system?

1.5.2 Design considerations

There are a number of questions which need to be considered when designing a security system. We will pose five design questions here. See Gollmann, Chapter 1 for further discussion of these questions.

- Does the system focus on the data, operations or users of the system?

  For example, is it more important to have a data focused rule such as: *Only data of type A can be inserted in data box A* or a user focused rule such as: *Only section managers are able to access the information in data box A*?

- What level should the security system operate from?

  The security system may consist of a software package that runs on top of the operating system, such as Norton Internet Security which runs on top of Windows. Alternatively, it may be part of the hardware and have physical control over the data such as where it is stored and how it is manipulated, for example Security Enhanced Linux (SELinux).

- Should the security system be simple or sophisticated?

  As discussed above, there are disadvantages to having a security system, not least in terms of time and cost. The more sophisticated a system the costlier it is likely to be. On the other hand, a system which is too simple may not provide the necessary level of security. It is obviously not a good idea to spend millions

of dollars on a state of the art security system which is to be used to protect data that is not of high importance or value.

- In a distributed system should the security be centralised or spread?

  Should a security manager have ultimate control, for example over access control issues (this will make it easier to achieve a consistent and rigorous approach, but may cause time delays if the security manager has to be applied to for every change of access rights)? Alternatively, should individual users be allowed to choose who has access to their files? See section 3.3.3 for a description of how SELinux implements mandatory access control.

- How do you secure the levels below the level of the security system?

  An attacker may manage to gain access to the operating system and from there make alterations to access control limitations giving themselves access to other parts of the system. The logical access controls of the system may be by-passed by gaining direct access to the physical memory. It is therefore important to ensure that physical security measures are in place as well as the logical computer security mechanisms.

1.6 Security models

Computer security protects the computer system and the data it processes. Success depends on the implementation of security controls designed for the system. A *security model* is a means of formally expressing the rules of the security policy. The model should:

- be easy to comprehend
- be without ambiguity
- be possible to implement
- reflect the policies of the organisation.

Different systems require different models. We will look at the theoretical Bell-LaPadula security model and the practical Unix security model in Chapter 3.

Learning activity

Do an Internet search for some examples of definitions of security concepts. A good starting point is the web site of the UK National Technical Authority for Information Assurance: `http://www.cesg.gov.uk`. Other governments have similar sites, and many major IT companies also have pages discussing security.

1.7 Summary

In this chapter we have introduced some important concepts and definitions regarding computer security. We have discussed features that a computer security system may provide including confidentiality, integrity, availability, non-repudiation, authentication, accountability and access controls. We have also looked at the different ways in which an attacker may threaten a security system including

interrupting, intercepting, modifying and fabricating communications. We have discussed the many design questions that need to be taken into account when designing a security system.

1.8 Learning outcomes

After studying this chapter and the additional reading, you should be able to:

- Recognise the need for computer security and describe how computer security differs from security in the traditional sense.
- Define what is meant by the terms integrity, availability, non-repudiation, authentication, accountability and access control with regards to computer security.
- Discuss the various types of attack that may threaten a security system.
- Discuss the many design considerations to take into account when designing a security system.

1.9 Sample examination questions

Question 1

a) The following are seven features that may be provided by a security system. For each write a sentence describing what is meant by the feature:

   i. confidentiality
   ii. integrity
   iii. availability
   iv. non-repudiation
   v. authentication
   vi. access control
   vii. accountability.

[7]

b) A University department has a file called *exam marks* which contains a list of examination marks indexed by student names in alphabetical order. A student manages to access the exam marks file. The student cannot read the file since it is encrypted. However they can work out the position of their own mark making use of the fact that the students are listed in alphabetical order. The student swaps their mark with that of the student who is always 'top of the class'.

   Write a paragraph discussing which of the security features listed in part a) have been breached.

[5]

Question 2

Three aspects of security are prevention, detection and reaction. Write a paragraph explaining why methods used for the prevention of, detection of and reaction to, theft of physical property may not be appropriate when the crime involves the theft of digital information.

[6]

Chapter 2

Identification and authentication

2.1 Introduction

In this chapter, we will consider why identification is an extremely important aspect of computer security and look at some methods that can be implemented in order to identify computer users. We will also consider the ways in which an identification system can be abused and methods that can be used to minimise the threats to security.

Supplementary reading

Chapter 2 of *Computer security* by Gollmann is a good introduction to identification and authentication.
Do an Internet search on *password crackers*. Analysis of a password cracking program will give you an insight on how fast these programs run and the size of the dictionaries that they use.

After studying this chapter and the recommended reading, you should be able to:

- show familiarity with the concepts of identification and authentication
- describe how a user-name/password system works
- understand that a user may prove their identity using *something they are*, *something they know* or *something they have*
- identify different kinds of threats such as password guessing attacks, password spoofing attacks and attacks on the password file; and give measures that can be used to prevent or detect these attacks
- understand the importance of educating system users so that they do not choose a weak password that may undermine the security of the system
- describe how to protect a password file by using a one-way function to encrypt the passwords
- know that *one-time passwords* can be implemented to reduce the risk of a password being discovered by an attacker
- show familiarity with alternative methods for identification and authentication and know that it is important to assess the security need when choosing which method to apply in a given situation.

2.2 User-names and Passwords

When a computer system has to verify a user's identity, there are two basic questions that have to be asked and answered appropriately. The first is:

Who are you?

The computer system has to establish somehow *who* is trying to gain access to its files. This is usually done by use of a *user-name* which, although probably unique to the user, is not a secret. The user-name is often simply produced using all or part of the user's actual name. For example, the user-name of John Smith might be *JSmith* or *johnsmith*.

When John Smith correctly enters his user-name, the computer can establish, by looking in a database of authorised user-names, that John Smith is an authorised user of the system. However, a second question now has to be asked:

How do I know that you are who you say you are?

The computer must now establish that the person logging into the system as John Smith actually is John Smith. Since the user-name is not a secret, anyone could try to log into the system using the identity of John. The person logging in must somehow prove that they are the genuine John Smith. This is usually done by using a *password*. The password is a secret and is only known to the genuine user John Smith. By entering this secret password, in conjunction with his user-name, John proves to the computer that he is an authorised user and is allowed access to the system.

Thus there are typically two stages in the process of identification.

1. A *user-name* is used to establish identity.
2. A *password* is used to establish authentication of identity.

Your own name is an example of *something you are*. If you apply for a bank loan or a travel visa you will have to prove you are who you say you are by showing, for example, your passport. Your passport is an example of *something you have*. Another example of *something you have* might be a bank card. In order to use the bank card to withdraw money at a cash machine, you do not have to prove who you are, but you do have to prove that you are authorised to use the card. This is done by using *something you know*. In this case the PIN number associated with the bank card. It is possible (although not recommended) for a person to lend their bank card to someone else. However, the second person will only be able to withdraw money using the bank card if they know the correct PIN number. A password is another example of *something you know*. Thus, a person can identify themselves by using *something they are, something they have*, or *something they know*.

2.3 Threats

A basic identification system consists of a database of passwords indexed by user-names. This is called the *password file*. When a user logs into the system, the computer checks that the user-name and password input match an entry in the password file. If a match is found, the process is complete and the user is allowed access to the system. If not, access is denied although the user may be given another chance to enter their user-name and password.

There are various ways in which a user-name/password identification system can be abused. The simplest attacks include the hacker looking over the user's shoulder when they are typing in their password, or finding a written note that the user has

made of their password. In the following sections we will consider some further possible attacks that might be used and defences that can be employed to either prevent, or detect, an attack.

2.3.1 Password guessing

Suppose that a hacker wants to access a system which is protected by a user-name/password identification system. We will assume that the hacker knows the user-name of an authorised user since this information is not generally secret. Therefore if the hacker can guess the user's password he will gain access to the system. There are several ways in which the hacker can find out the user's password. These include:

Guessing using personal knowledge of the user

Many people use passwords which relate to them personally. For example, they may use the name of their spouse or child or pet. They may use their football team or street name or birth date. If the hacker can find out personal information about the user, then they may be able to guess a personal password without too much difficulty.

This attack will fail if the user is careful not to use a password which is personally related to them in any way.

Dictionary searching

Another favourite method of generating easy to remember passwords is for the user to choose a word, usually in their own language. If the hacker cannot directly guess the user's password then he may set up a *dictionary attack*. This means that he will run a computer programme which tries every word in a dictionary as the password of the user until he finds a match.

This attack will fail if the user does not use a word which appears in a dictionary as their password.

Intelligent searching

Some user-name/password systems insist that the user's password contains a mix of letters and numbers. The most common thing for a user (who has not been educated in password security) to do is add a number onto the end of a word. For example, using a password such as banana1. An intelligent dictionary search might try all words with numbers added. Thus if the hacker knows that a particular password system insists that passwords are a minimum of six characters long and must contain at least one number, then the hacker may try all five letter words with each of the digits 0,..,9 attached. Thus apple0, apple1, apple2,....,apple9, apply0, apply1,... and so on would form part of this search.

If this attack does not succeed, the next step might be to capitalise the first letter of each word in the dictionary. Other intelligent dictionary modifications include capitalising each letter of the word in turn, including a number at the front of the

word, including a number in any position in the word or replacing letters which are similar to numbers with that number. For example, replacing the letter l with the number 1 or the letter o with the number 0.

Exhaustive searching

If the user has been clever enough to use a random, meaningless string of characters as their password, then the hacker may have to resort to trying an *exhaustive search attack*. An exhaustive search is similar to a dictionary search, but in the exhaustive case, the computer programme used by the hacker will try every possible combination of permissible characters as the password in order to find a match. Thus if searching for a six character password, the hacker might try aaaaaa, aaaaab, aaaaac, ..., aaaaaz, aaaaa0,...., aaaaa9, aaaaa*, etc. and move systematically through all possible permutations.

This attack will always succeed eventually. Since *every* possible password is tried in turn sooner or later a match will be found. However, there are ways of making an exhaustive search so time consuming for the hacker that it is not successful during the life of the password (i.e. before the exhaustive search is successful the password has been changed). Some password systems insist that the users change their passwords every three months, for example.

Learning activity

A hacker is trying to find a password in order to get access to a computer system. He does not have any personal information about the system users, but he knows that all passwords are at least eight characters long and can contain any upper or lower case letter, digit, or other keyboard character. What kind of attack do you think the hacker should attempt?

2.3.2 Number of passwords

An intelligent attacker will carry out dictionary and intelligent or modified dictionary attacks before attempting an exhaustive search. This is because, although an exhaustive search is bound to succeed eventually and a dictionary search may fail, if it succeeds, the dictionary search is much faster.

Suppose that passwords are six characters long.

If the password is made up only of lower case letters, then there are 26 choices for each character in the password. Hence there are $26^6 = 308,915,776 \approx 3^8$ possible passwords of six lower case letters.

If we include lower and capital letters, there are $52^6 \approx 2^{10}$ possible passwords.

Adding in digits as well, gives a choice out of 62 for each character in the password and there are now $62^6 \approx 5.7^{10}$ possible passwords.

Finally if we allow any keyboard character including ¡ ¿ * & etc. there are approximately 100 different choices for each character in the password and hence there are $100^6 = 10^{12}$ possible passwords.

In general, if a password is n characters long and is made up from an alphabet of A different characters, then there are A^n possible different passwords.

Now suppose that a hacker has written a computer program which can try 10,000 passwords per second.

The hacker has a dictionary file which contains 1,000,000 common six letter words. First he runs a dictionary attack trying every word in his dictionary. This will only take him $1,000,000/10,000 = 100$ seconds to complete.

Making modifications to the dictionary, for example capitalising each word is easy and it only takes the hacker a few more minutes to run modified dictionary searches.

If a dictionary search is not successful then the hacker may try every combination of lowercase letters. This will take $26^6/10,000$ seconds, which is just over 8.5 hours to try every combination of lower case letters.

In comparison, if the hacker attempts an exhaustive search using all 100 possible characters in every combination, it will take $100^6/10,000 = 10^8$ seconds to complete and this is over three years!

Note that the **average time** for a hacker to find a particular kind of password is only **half** the time taken to do a complete search (i.e. if a user has chosen a dictionary word as their password, then the hacker will, on average, only have to search through half of the dictionary in order to find the password). Likewise, on average, a hacker using an exhaustive search will only have to search through half of the possible passwords before finding a match.

Learning activity

1. How many different passwords of lower case letters are there if the password is of length four? How many if the password is of length eight?
2. How many different alphanumeric (any letter or digit) passwords are there if the password is of length four? How many if the password is of length eight?
3. On average, how long will it take a hacker to find a password of length eight which is made up entirely of lowercase letters:
   (a) if the hacker tries only combinations of lowercase letters?
   (b) if the hacker tries all alphanumeric combinations?

   Assume that the hacker can try 10,000 passwords per second.

2.3.3 Password spoofing

A *spoofing attack* is when the user is fooled into giving the hacker their password. Spoofing attacks may be very simple or very sophisticated.

Asking the user

This may sound unlikely, but it is a fact that a lot of people will tell you a password if you can convince them that you need to know it.[1] For example, the hacker may phone the user, and tell them that he is from their office computer staff and that there is a problem with the files. All backed-up information is going to be lost so he needs the user password in order to recover the data. Sometimes an approach as simple as this will work and the user is fooled into giving the hacker their password.

This attack will fail if the user has been educated in computer security and refuses to reveal their password.

Fake log-in screens

A more sophisticated spoofing attack is when the hacker sets up a fake log-in screen which exactly resembles the genuine log-in screen for the system. The user is presented with this log-in screen and unsuspectingly enters their user-name and password. The hacker captures this information and then typically gives the user an error message saying that they have incorrectly typed in their password. The genuine log-in screen is then displayed. The user cannot be sure that they did not make a typing mistake, so they type in their user-name and password again and gain access to the system. The user may have no idea that they have been the victim of a spoofing attack.

This attack will fail if the user notices that there is something wrong with the log-in screen and so does not enter their user-name and password. Some log-in interfaces contain patterns or pictures which are impossible to replicate accurately. The attack can be detected (although not prevented) if the user is informed, at every log-in, of the time of the last failed log-in attempt. After a spoof attack, the user thinks that they had a failed log-in. If when the user successfully logs in, the system does not inform them of this failed log-in then the user is alerted to the fact that they may have been the victim of a spoof attack.

Phishing

Phishing is similar to the above. Communications such as emails or instant messages purporting to be from reliable sites such as eBay, PayPal or online banks direct users to a fake website which looks very like the genuine one. Here the user is asked to input their username, password and perhaps their bank details.

Phishing is a growing problem and attempts to deal with it include legislation, user training, public awareness and technical security measures.

[1] In 2004 an experiment was done by a small group of researchers at a London railway station. They asked the commuters at the station to reveal one of the passwords that they used at work in exchange for a bar of chocolate. Over 70 per cent of the commuters gave a password away! There was no check that the passwords were genuine so wily individuals may have given false passwords. However, it is very likely that many genuine passwords were revealed. You can find more information about this experiment by doing an Internet search on *password for chocolate*.

2.3.4 User and system defences

There are various things that users can do in order to minimise the risk of a hacker getting hold of their password. When a user-name/password system is implemented, it is important that the users are informed of the following measures:

- The user should always set up a password and not leave the password option as blank.
- The user should change the default password.
- The user should change their password frequently.
- The user should not use the same password for all systems.
- When changing a password, the user should not just add a digit onto the end of the old password.
- The user should not choose a password that relates to them personally such as their date of birth or the name of their child.
- The user should not choose a dictionary word as their password.
- The user should not choose a password that is too short.
- The user should choose a password that contains a mix of letters and numbers.
- The user should not write their password down or reveal it to anyone.

Some of these measures can be enforced by the system. Things that the system can do in order to minimise the risk of attack include:

- insist that the user creates a password
- provide the user with a default password
- enforce the user to change the default password at the first log-in
- enforce the user to change their password at frequent intervals (say every three or six months depending on the security need)
- check password choices against a dictionary and reject weak passwords
- insist that passwords contain an alphanumeric mix of characters
- insist that passwords are at least a minimum length (say 6+ characters depending on the security need)
- limit log-in attempts (a maximum of three attempts is usual) after which time the system administrator will have to reset the password for the user
- inform users of each unsuccessful log-in attempt.

Learning activity

Suppose a user has an account on a user-name/password system and that they want to change their password. Write a protocol for a secure procedure that should be followed to enable this.

2.4 Attacking the password file

The *password file*, where the system stores the data for verifying passwords, is very sensitive to attack. In an insecure system, the password file will be a list of passwords indexed by user-name. A hacker with access to this file has potential knowledge of every password. It is therefore essential that the password file is protected.

There are essentially two ways in which the password file can be protected:

- using cryptographic protection
- implementing access control over the password file.

Ideally, the password file should be both encrypted and protected from unauthorised access by the implementation of access controls.

2.4.1 Cryptographic protection

A password file can be encrypted by using a *one-way function*. After encryption, the password file is just a list of garbled characters. Even if a hacker manages to view the file, it will not help him to gain access to the system.

One-way functions

A problem is said to be *one-way* if it is easy to do one way but hard to do in reverse. A non-mathematical example is making a cup of instant coffee. It is easy to put coffee granules, boiling water and milk into a mug and stir them together to make a cup of coffee. However, given a cup of coffee, it is difficult to reverse the operation and retrieve the separate components of milk, coffee granules and water.

In cryptography, the one-way problems used are mathematical functions. A good example of a mathematical one way function is multiplying/factorising.

A one-way function is a function $f : X \rightarrow Y$ which satisfies the following two properties:

- Given x in X it is easy to compute $y = f(x)$ in Y.
- Given y in Y it is very difficult to find an x in X such that $f(x) = y$.

A good example of a mathematical one-way function is multiplying/factorising. It is very easy (especially given a computer or calculator) to multiply together two integers, even if those integers are very large. However, given the resulting number, it is very hard (even with access to a computer) to find the two numbers that were originally multiplied together. In this example, both X and Y are the set of positive integers.

See Chapter 8 for more details on how the factorisation problem can be used as the basis for encryption.

2.4.2 Encrypting the password file

The password file can be protected by using a one-way function $f(x)$ to encrypt the stored passwords as follows:

To create a new user-name/password pair:

- The user inputs their user-name and password x.
- The system computes $f(x)$.
- The password file does not store x but instead stores $f(x)$ indexed by user-name.

To verify a user:

- The system asks for the user-name and password.
- The system computes $f(x')$ where x' is the password entered by the user.
- The system checks to see if there is a match between the $f(x)$ stored for the given user-name and $f(x')$ just computed.
- If $f(x) = f(x')$ then $x = x'$ and the user is verified. If $f(x) \neq f(x')$ then the password entered by the user is incorrect and access to the system is denied.

Attacking an encrypted password file

If a hacker manages to access a password file which has been encrypted using a one-way function, all he will see is the encrypted passwords, indexed by user-names. These encrypted passwords will not enable the hacker to access the system, and the actual passwords are not stored anywhere.

The function used to encrypt the passwords is not usually a secret, so the hacker may try to find an actual password by running a computer program that encrypts a dictionary list or an exhaustive list of passwords and then check to see if the result matches any of the stored encrypted passwords. If a match is found then the hacker has a password and can now gain access to the system.

This type of attack can be thwarted by using a relatively inefficient function to encrypt the passwords. Consider that the hacker may have to encrypt millions of possible passwords before a match is found. If each encryption takes one or two seconds then this will take many days. However, for a genuine individual user a time lapse of a few seconds each time they enter their user-name and password is negligible.

Rainbow tables

If a well known function, such as a secure hashing function, is used to encrypt passwords then pre-computed *rainbow tables* can be used to find passwords very quickly.

A rainbow table is a table that stores the encryption of all possible passwords of a given format. For example, all passwords that are eight characters long and contain lower case letters and digits. These rainbow tables are huge and require a large amount of storage space and initially a lot of time to compile. However, once they

are built they can be searched very quickly to find password matches. These tables are used to retrieve lost user passwords and they are very useful for this purpose. However, in the wrong hands they can obviously be used to find passwords for malicious purposes.

To avoid pre-compiled rainbow tables being used on a security system, the function used to encrypt the passwords should be somehow unique to the system. Pre-compiled tables will therefore not be available. If a user loses or forgets their password it will be irretrievable. An alternative secure method for resetting the lost password to a new value will have to be devised.

2.4.3 Password salting

Password salting is a process used to ensure that all passwords in a system are unique. Most systems insist that all user-names are unique. If a new user tries to create an account with a user-name that is already in use, they will be informed that the user-name is already used and that they should choose another. However, the system cannot inform a new user that the password they have chosen is already in use – that would be a gift for a hacker! Instead, the system adds some *salt* which is another piece of information such as the user-name to all the passwords before encryption. This ensures that every password is unique.

Learning activity

Why is it important that every password should be unique? Suppose a hacker found two encrypted passwords that were the same – how could he use this information?

2.4.4 One-time passwords

Given enough time and attempts, a *static* password (i.e. a password which remains the same) may be accessed by an unauthorised attacker. To counter this, some systems are now making use of *one-time passwords* or OTP. By constantly changing the password, the risk of the password being discovered is greatly reduced. Furthermore, an attacker who does find a password, will only be able to use it to gain access to the system once. The next time the attacker tries to use the password it will be rejected.

One-time passwords typically work in one of three ways.

- A mathematical algorithm is used to generate a new password based on the previous password.
- A time synchronisation protocol is used between the authentication server and the client providing the password.
- A mathematical algorithm is used to create each new password based on a challenge such as a random number chosen by the authentication server and a counter.

To implement a OTP, users generally have a *token* (similar to a small electrical keyring, for example) which generates the passwords either based on a

mathematical algorithm, or if the token contains a clock synchronised with the authentication server, using the current time. Work is currently being done on the use of mobile phones as tokens. This would be practical and cost effective since most Internet users also have a mobile phone.

2.4.5 Alternative methods for authentication

There are many alternative methods used for identification and authentication. Some are used when the risk is low and others where security is of paramount importance. Of course, in general, as the level of security increases so does the cost, so it is sensible to assess the risk before deciding on the level of security required. Alternative methods include:

- Answering a question that only you are likely to know the answer to such as your mother's maiden name or date of birth. This information is not that hard for a hacker to acquire so provides only a low level of security.
- Presentation of something that you have, such as a credit card or passport. These can be forged or stolen but in general are a good means of identification and authentication.
- Use of finger prints, retina patterns or palm prints. This is a high cost solution, but fingerprints, etc. are fairly hard to replicate and are not something that the genuine user can lose or forget! However, a determined attacker with adequate financial resources can replicate these physical attributes leading to a catastrophic failure of a supposedly high security identity system.

2.4.6 Authentication failure

An identification and authentication system can fail in two ways. Firstly, it can accept an unauthorised user. In this case, the security may be too weak. Secondly, it can reject an authorised user. This may be because security is too high. For example, if the system insists that user passwords are 15 characters long and include digits and letters then it is likely that genuine users will forget or mistype their passwords leading to authentication failure.

2.5 Summary

In this chapter, we have discussed how user-name/password systems are used to provide identification and authentication. We have described various attacks that may threaten such systems and measures that can be taken in order to prevent or detect these attacks. We have looked at one-way functions and seen how they can be used to encrypt a password file.

2.6 Learning outcomes

After studying this chapter and the recommended reading, you should be able to:

- show familiarity with the concepts of identification and authentication

- describe how a user-name/password system works
- understand that a user may prove their identity using *something they are*, *something they know* or *something they have*
- identify different kinds of threats such as password guessing attacks, password spoofing attacks and attacks on the password file; and give measures that can be used to prevent or detect these attacks
- understand the importance of educating system users so that they do not choose a weak password that may undermine the security of the system
- describe how to protect a password file by using a one-way function to encrypt the passwords
- know that *one-time passwords* can be implemented to reduce the risk of a password being discovered by an attacker
- show familiarity with alternative methods for identification and authentication and know that it is important to assess the security need when choosing which method to apply in a given situation.

2.7 Sample examination questions

Question 1

After reading a newspaper 'scare story' about password security, Walter has decided to implement strict rules regarding the passwords used by the staff in his company. Walter insists that:

- Staff passwords are of length 15 characters or more.
- Staff change their passwords at least once a week.
- Every password contains a mix of letters and digits.

a) Explain why Walter's password policy is likely to make the password system at his company less rather than more secure.

[3]

b) Write a more suitable password policy explaining the importance of each rule you suggest.

[10]

c) Assuming that a password cracking program can check 10,000 passwords per minute, calculate the average amount of time that it would take to find a password based on the policy that you have written in part b).

[4]

Question 2

a) What two properties are required for a *one-way* function?

[2]

b) Describe how a one-way function can be used to protect password files.

[4]

c) Explain why the one-way function used to protect a password file should not be an efficient function.

[4]

Chapter 3

Access control

3.1 Introduction

In this chapter, we will consider access control, why it is vital and how it can be implemented. We will introduce the terminology of access control and look at different ways of stating and illustrating access control policies. We will study the Bell-LaPadula theoretical model of access control, and see how the Unix system implements access control in practice.

Supplementary reading

Chapter 3 of *Computer security*, by Gollmann covers access control and Chapter 4 discusses various security models including Bell-LaPadula. Chapter 6 deals with Unix security.
Chapter 8 of *Secrets and Lies* by Schneier is about access control and how hard it is to achieve.

After reading this chapter and the recommended reading, you should be able to:

- understand how important it is to get access control right when designing security systems
- identify *subjects, objects, permissions, modes, operations* in access control scenarios
- interpret and write access control lists or matrices
- use groups appropriately to simplify access control
- draw and interpret graphs illustrating access control
- describe a situation when it is appropriate to use a protection ring to model access control
- know the difference between *mandatory access control* MAC and *discretionary access control* DAC
- discuss the Bell-LaPadula security model and how it gets around the restrictions and problems that its no-read-up and no-write-down policies bring about
- describe how Unix implements access control in practice.

3.2 Access control

In Chapter 1 we said that *access controls provide the limitation and control of access to authorised users through identification and authentication*. Access control is crucial in computer security. All of the features that we would like a security system to provide (confidentiality, availability, integrity, non-repudiation, authentication and accountability) depend upon the proper implementation of access controls. Broadly

speaking people who have the proper authority should be able to do whatever it is (and only whatever it is) they are authorised to do. Nobody else should be able to do anything on the system.

3.2.1 Objects and subjects

To talk about access control we need to make some definitions.

- A multi-user distributed computer system offers access to ***objects*** such as resources (memory, printers), data (files) and applications (software).
- The system offers this access of ***subjects*** such as users, processes and other applications.

Subjects and objects represent respectively the active and passive parties in a request for access. In defining access controls, we can either specify:

- *what* a *subject* is allowed to do; or
- what may be done *with* an *object*.

For example, suppose that Alice, Bob and Charles are subjects and a database is an object. We could either say that Alice has write and read access, Bob has read only access and Charles does not have any access to the database. Or we could say that only the figures in Column 2 of the database can be altered manually and all the other figures will be updated automatically in response.

3.2.2 Operations and modes

Operations that the system may offer include:

- read
- write (which may or may not automatically include read access)
- append
- execute
- delete.

Modes that the system may offer include:

- observe (look at the contents)
- alter (change the contents).

Operations are defined by the security model. Modes are basic notions of what can be done to an object. The relationship between operations and access modes can be summarised as follows:

	Read	Write	Append	Execute	Delete
Observe	√			√	
Alter		√	√		√

3.2.3 Permissions

Permissions for files may include:

- read
- write
- execute
- append
- delete
- change permission
- change ownership.

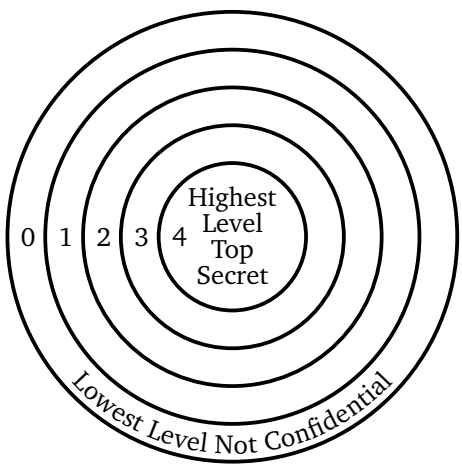
3.3 Stating and illustrating access control permissions

There are different ways of expressing access control permissions including lists, matrices and graphs. Which of these is the most suitable depends upon the complexity of the security model.

3.3.1 Protection ring model

If access control is strictly hierarchical, this can be illustrated by a simple ***protection ring*** model. In this model, every subject and object is given a security level. Subjects can access all objects at their own level or lower. A security level may involve operations as well. For example, read and write permission may be at a higher level than read only.

Figure 3.1: Protection ring model



The protection ring model shown in figure 3.1 has five security levels {0, 1, 2, 3, 4} with security level 4 being the highest. A subject with security level 4 would be able to access any object in the system. A subject with security level 2 would be able to access any object with a security level classification of 2, 1 or 0.

3.3.2 Access control lists, matrices and graphs

Access control is often not as hierarchical or clear cut as in the example above and so a protection ring may not be a suitable model. Instead we might write an *access control list* which gives details of a subject's particular access writes to an object, or an *access control matrix* which combines details of different subjects and objects and defines the rights of the subjects over the objects. We can also illustrate these access control policies by drawing an *access control graph*.

For example, suppose we have two subjects called Alice and Bob, three objects called prog1, database1 and database2 and access control permissions {read, write, execute}. The access control rights of Alice and Bob can be given as an *access control list*:

Alice: prog1{execute}; database1{write, read}; database2{read}
Bob: prog1{write, read, execute}; database1{read}
We could also show these rights in an *access control matrix* as in Table 3.1.

	prog1	database1	database2
Alice	{e}	{w,r}	{r}
Bob	{w,r,e}	{r}	{ }

Table 3.1: Access control matrix

Groups and negative permissions

In a large organisation, it is likely that several subjects will all have the same access control permissions. These subjects can be grouped together and the group access permissions listed.

For example, the five subjects $\{S_1, S_2, S_3, S_4, S_5\}$ have access permission rights over the four objects $\{O_1, O_2, O_3, O_4\}$ as given in the following access control matrix in Table 3.2.

	O_1	O_2	O_3	O_4
S_1	$\surd$	$\surd$	$\surd$	
S_2	$\surd$	$\surd$	$\surd$	
S_3	$\surd$	$\surd$	$\surd$	
S_4			$\surd$	$\surd$
S_5			$\surd$	$\surd$

Table 3.2: Access control matrix showing subjects and objects

Since subjects S_1, S_2 and S_3 all have the same access rights we can group them together into a group G_1. Likewise we can put subjects S_4 and S_5 into a group G_2.

Now the access control matrix is simplified as shown in Table 3.3.

These group memberships and access control permissions may also be illustrated using a graph as in Figure 3.2.

We can also express *negative permissions* on the graph. For example, suppose S_1 has

	O_1	O_2	O_3	O_4
G_1	$\surd$	$\surd$	$\surd$	
G_2			$\surd$	$\surd$

Table 3.3: Access control matrix with groups

all the permissions of G_1 except that S_1 does not have access to O_1. This can be shown on the graph as in figure 3.3.

Learning activity

Express the access control permissions of the subjects in the graph shown in Figure 3.4 as:

i) an access control matrix,

ii) an access control list.

3.3.3 Ownership policy

We have not yet considered who has the authority to decide which subjects have which permissions over which objects. The ownership policy may either be *discretionary* or *mandatory*.

- If the ownership policy is *discretionary* the owner of the resource decides who has access permission. For example, I could write a web page and post it openly on the Internet so that everyone has access. Alternatively, I could post the web page with a password access control system and then decide to whom I give the password.
- If the ownership policy is *mandatory* then the security system manager allocates permissions according to the security policy of the organisation.

SELinux

A real life example of access control is SELinux (Security-enhanced Linux). SELinux is an implementation of a *mandatory access control mechanism*. This mechanism is in the Linux kernel and checks for allowed operations after standard Linux *discretionary access controls* are checked.

The following extract is from the Red Hat SELinux Guide:

> *To understand the benefit of mandatory access control (MAC) over traditional discretionary access control (DAC), you need to first understand the limitations of DAC.*
>
> *Under DAC, ownership of a file object provides potentially crippling or risky control over the object. A user can expose a file or directory to a security or confidentiality breach with a misconfigured chmod command and an unexpected propagation of access rights. A process started by that user, such as a CGI script, can do anything it wants to the files owned by the user. A compromised Apache HTTP server can*

Figure 3.2: Access control graph

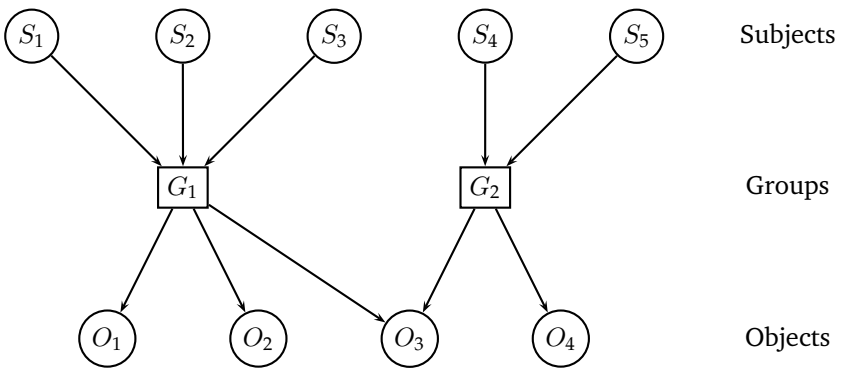


Figure 3.3: Access control graph with negative permissions

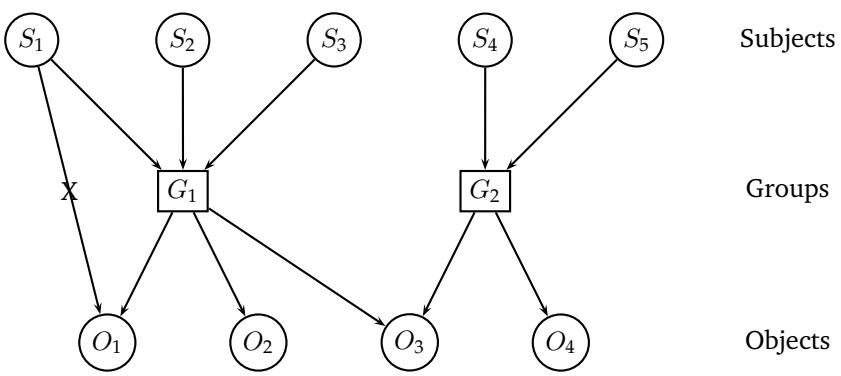
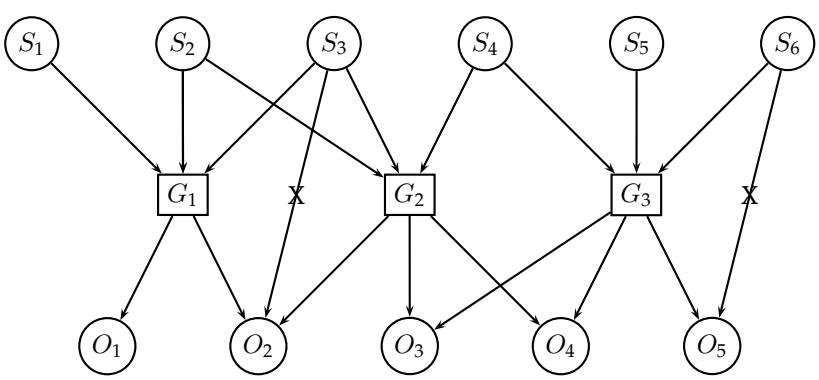


Figure 3.4: Access control graph



perform any operation on files in the Web group. Malicious or broken software can have root-level access to the entire system, either by running as a root process or using setuid *or* setgid*.*

Under DAC, there are really only two major categories of users, administrators and non-administrators. In order for services and programs to run with any level of elevated privilege, the choices are few and course grained, and typically resolve to just giving full administrator access. Solutions such as access control lists can provide some additional security for allowing non-administrators expanded privileges but for the most part a root account has complete discretion over the file system.

A MAC or non-discretionary access control framework *allows you to define permissions for how all processes (called* subjects*) interact with other parts of the system such as files, devices, sockets, ports, and other processes (called* objects*). This is done through an administratively-defined security policy over all processes and objects. These processes and objects are controlled through the kernel and security decisions are made on all available information rather than just user identity. With this model, a process can be granted just the permissions it needs to be functional. This follows the principle of* least privilege*. Under MAC, for example, users who have exposed their data using* chmod *are protected by the fact that their data is a kind only associated with user home directories, and confined processes cannot touch files without permissions and purpose written into the policy.*

All interactions between subjects and objects are disallowed by default on a SELinux system. The policy specifically allows certain operations. To know what to allow, an access control matrix is used. The matrix is derived from the policy rules. The matrix clearly defines all the interactions of processes and the targets of their operations.

3.4 Security models

There are many ***security models*** which can be used to describe how access control is to be managed. These include:

- the Bell-LaPadula Model
- the Harrison-Ruzzo-Ullman Model
- the Chinese Wall Model
- the Biba Model
- the Clark-Wilson Model.

Most of these are designed for use with military data and provide multilevel security with multiple levels of classification within a single system. For example, some documents (objects)are classified as top secret and only a person (subject) with top secret clearance would be able to access them. Other documents may have unclassified status and can be accessed by everyone on the system. We will look at the Bell-LaPadula Model which is a classic (theoretical) example of a security model. We will also see how access controls work in practice by looking at Unix security.

3.4.1 The Bell-LaPadula model

The Bell-LaPadula security model (BLP) is one of the most famous security models. It was developed by Bell and LaPadula and aims to provide a secure multi-user

operating system. Access permissions are defined by an access control matrix and security levels. The basic idea is that information cannot flow from a high security level to a lower security level.

We have the following sets:

- a set of subjects S
- a set of objects O
- a set of access operations $A = \{execute, read, append, write\}$
- a set of security levels L with a partial ordering $\leq$.[1]

Every subject is allocated a maximum security level f_s and a current security level f_c. Every object has a security classification.

The following rules are applied. Think of the subjects as people and objects as documents.

1. A subject can read an object only if the subject's security level is greater than or equal to the objects classification. This is a *no-read up* policy.
2. To get around the no-read up policy, a subject at a low level might persuade a subject at a higher level to copy part of a high level document into a new document with a lower security level. This might be done without the knowledge of the high level subject, for example by use of a Trojan horse program. To prevent high level objects being copied into a lower level document and then accessed at that lower level, there is also a *no-write down* policy. This means that a subject cannot write (or append to) an object which has a security classification lower than their own.
3. An access control matrix M is defined and subject s can only perform operation a on object o if (s, o, a) is (ticked) in the access control matrix. This is called a *secure transition*.

Rules 1 and 2 mean that subjects could in theory write a document which they cannot read. Also a subject at a high security level cannot send messages to a subject at a lower level. Obviously this is not very practical so the model allows the ability to:

- Temporarily downgrade a subject's level from f_s to f_c where $f_c < f_s$.
- Identify a set of subjects which are allowed to violate the no-write down policy. These are called *trusted* subjects.

If the three rules are satisfied then the state of the model is called *secure*. Furthermore the *basic security theorem* states that if you start with a secure state and all transitions are secure then every subsequent state will also be secure.

However, it is possible, without breaking any of the rules, to:

- downgrade all subjects to the lowest level
- downgrade all objects to the lowest level
- give all subjects permission to perform any operation on any object by completely filling in the access control matrix.

[1] This means that the security levels can be listed in an hierarchical order, although some may be at the same level. $L_A > L_B = L_C > L_D = L_E \ldots$

Now all subjects and all objects are at the lowest security level and all subjects have access to all objects. None of the rules of the model have been broken so this is still 'secure' as designed by Bell-LaPadula. Should this state be regarded as secure?[2]

This has only been a brief overview of the Bell-LaPadula model. For further details, and for details on the other security models named above, see Chapter 4 of *Computer Security* by Gollmann.

3.4.2 Unix – access control in practice

Finally we will look at an actual control access model and see what happens in real life rather than in theory. We will be looking at how Unix deals with access control.

Unix users and superusers

In Unix every user has an identifier, their user name, and each user belongs to one or more groups.

Every Unix system has one user with special privileges. This *superuser* has user ID 0 and usually the user name *root*. The root account is used by the operating system for essential tasks like login, recording the audit log, or accessing I/O devices. Almost all security checks are turned off for the superuser. The root account is required to perform certain tasks such as installing certain software. The system manager who holds the root password should never use the root account as their personal account but should change to user root as and when necessary to perform a superuser task.

Unix security measures

Users are authenticated by user name and password. Passwords are encrypted using a one-way function which is based on the DES algorithm and run 25 times. The encrypted passwords were traditionally stored in the `/etc/password` file. However, most modern Unix systems use either *password shadowing* or NIS and much of the Unix password data is stored elsewhere. Password shadowing means that the encrypted password field of `/etc/password` is replaced with a special token and the encrypted password is stored in a separate file (or files) which is not readable by normal system users. Networked systems may also use NIS (Network Information System) which allows many machines on a network to share configuration information, including password data. On a machine with NIS there will be a very short `/etc/password` entry and the real password file is elsewhere. Note that NIS is not designed to promote system security.

Entries in the password file have the following form:

$accountname : encryptedpassword : UID : groupID : IDstring : homedirectory : loginshell$

and so may look something like this:

```
RS : ru78Pjey :  92 :  4 :  Shipsey,R : /usr/RS : /bin/sh
```

[2]The designers of Bell-LaPadula say yes because if such a state is not required then it should not be implemented.

When changing their password, users must supply the old password first to guard against another person changing the password. The new password must be entered twice in order to confirm that it has been typed correctly. The actual characters of the password are never shown on the screen but appear as * or • characters instead.

Passwords may be salted if required. Controls can be set so that weak passwords are rejected. Password expiry dates can also be set, together with enforced rules on the re-use of old passwords. Root login can be restricted to specially nominated terminals only.

Unix access control

Subjects in Unix are users. Each user belongs to at least one group, their *principal group*. They may also belong to other groups. If a user belongs to more than one group then they will have a designated principal group. For example, a user may belong to the groups **Staff** and **Project1** where **Staff** is their principal group.

Objects in Unix are files. Unix thinks of all resources as files. Each file belongs to a user and a group, and has a set of permissions associated with it. These permissions are from the set {read, write, execute}. Three different sets of access rights are defined for the file:

- one is for the file's owner (usually the user who created the file),
- one is for the file's group (usually the principal group of the file's owner but sometimes the directory group if this has been set using Set GroupID[3]),
- and the last is for all other users (i.e., those users who are neither the owner nor in the file's group).

Each file in Unix is really a pair consisting of the filename and the **i-node** number of the file. The i-node number contains a lot of information about the file including:

- where the file is stored
- the length of the file
- the last time the file was read
- the last time the file was written
- the last time the i-node was read
- the last time the i-node was written
- the owner – generally the UID of the user who created the file
- a group – the GID of the group that the file belongs to
- 12 mode bits which encode a set of access rights.

Nine of the twelve mode bits are used to encode access rights.[4] These access bits can be thought of as the access control matrix entry. They are divided into three groups of three bits which represent the owner, group and other users respectively. For each of these three groups, the three bits are r w x representing read access, write access

[3]Suppose a group of users are working together on a project. They want to share all the files that they write concerning this project, but not their other unrelated files. In this case, they may use Set GroupID on the directories containing their project files. Now the GID of the files in these directories will be the GroupID rather than the principal group of the file's owner.

[4]One of the three remaining bits is the Set GroupID bit which can be turned on or off as mentioned above.

and execute access respectively. Each of the three bits can be turned on or off – a dash indicates that the permission is not granted.

For example, the 9 bits r w - r - - r - - indicates that the owner has read and write access but not permission to execute the file, whereas the group members and all other users have read access only. The 9 bits r w x r - - - - - indicates that the owner has read, write and execute access, whereas the group members have read access only and other users have no access to the file.

If a user wants to access a file, the permission bits are checked in the following order. Firstly, if the user's UID shows that they are the owner of the file, then the first three permission bits decide whether or not the user is allowed access (and what kind of access i.e. read, write or execute). If the user is not the owner of the file but any of their GIDs matches the file's GID, then the permission bits for group decide whether or not the user is allowed access. If the user is neither the owner nor a member of the group which owns the file, then the last three permission bits decide whether or not this *other* user is allowed access.

There are a number of rules in Unix which define how the access control bits are set initially. They can be changed using the chmod (change mode bits) command. Only the UID that is the owner of the file can execute a chmod command for that file (except for the root user who can do whatever they like)[5] Some systems also have a command to change the owner of a file.

Learning activity

If you have access to a Unix system find your own entry in /etc/password and check the permission setting on your files and directories.

Create a subdirectory in your home directory and put a file helloworld.txt with a short message into this subdirectory. Now set the permission bits on the subdirectory so that the owner has execute access (- - x - - - - - -). Try to perform the following tasks:

- make the subdirectory the current directory using cd
- list the subdirectory
- display the contents of helloworld.txt
- create a copy of helloworld.txt in the subdirectory.

Repeat the tasks but give yourself first read access and then write access to subdirectory containing helloworld.txt. From the results of these experiments, define exactly what read access, write access and execute access permit.

3.5 Summary

In this chapter we have looked at access control and have seen how important it is to ensure that access controls are properly designed and maintained. We have learnt

[5]Note that it is possible for the permission bits to be set so that the owner of a file has less access rights than other users. For example if the permission bits are r - - r w x r w x then everyone will be able to read, write and execute the file except for the file's owner who will only be able to read it. This is true even through the owner may be a member of the file's group because only the UID permission bits will be checked if the owner tries to access the file.

what is meant by the terms *subjects, objects, permissions, modes* when talking about access control. We have considered different ways of stating and illustrating access control models including access control lists, matrices and graphs. We have looked at protection rings which are suitable for modelling access control when the security levels are strictly hierarchical. We have looked at the Bell-LaPadula model for access control and seen that the no-read up and no-write down policies required to enforce security can cause practical problems. Finally, we have looked at how Unix achieves access control in practice.

3.6 Learning outcomes

After reading this chapter and the recommended reading, you should be able to:

- understand how important it is to get access control right when designing security systems
- identify *subjects, objects, permissions, modes, operations* in access control scenarios
- interpret and write access control lists or matrices
- use groups appropriately to simplify access control
- draw and interpret graphs illustrating access control
- describe a situation when it is appropriate to use a protection ring to model access control
- know the difference between *mandatory access control* MAC and *discretionary access control* DAC
- discuss the Bell-LaPadula security model and how it gets around the restrictions and problems that its no-read-up and no-write-down policies bring about
- describe how Unix implements access control in practice.

3.7 Sample examination questions

Question 1

In a University department, there is an administrator (Mrs A), three lecturers (Dr B, Dr C and Dr E) and a Head of Department (Prof H). Three documents concerning the department are salary.doc, timetable.doc and marks.doc.

The Head of Department has permission to read all of the documents and has write access to salary.doc. The administrator has read and write access to timetable.doc and read access to salary.doc. The lecturers each have read access to timetable.doc and write and read access to marks.doc.

Information regarding the permissions each staff member has regarding the documents is to be stored in an access control matrix indexed by subjects and objects.

a) List the *subjects, objects* and *operations* in this scenario.

[3]

b) Explain how you could use a group to simplify the access control matrix.

[1]

c) Represent the given information in an access control matrix.

[4]

Question 2

Following is a protocol used for the preparation and marking of examination papers.

- Candidates are entered for the examination.
- Lead examiner and second examiner are appointed by the University examinations office.
- Lead examiner writes the examination paper and solutions and posts these to the second examiner.
- Second examiner checks the examination paper and solutions, marks corrections on these, and posts them back to the lead examiner.
- Lead examiner makes corrections as appropriate and posts final copies of examination paper and solutions to the University examinations office.
- University examinations office prints the examination paper and distributes it by post to the examination centres.
- Candidates sit the examination at their appointed examination centre and then leave their scripts and examination paper at the examination centre.
- Examination scripts are distributed, half each, to the lead examiner and second examiner for first marking.
- Lead examiner and second examiner exchange first marked scripts for second marking.
- Scripts are given a final mark after second marking and returned to the University examinations office.
- Final marks are entered on the University database by the University examinations office.

a) The *subjects* in this protocol are the candidates, the lead examiner, the second examiner, the University examinations office and the examination centres. One of the *objects* in the protocol is the examination paper. Identify **two** other objects.

[2]

b) Suggest suitable permissions from the list {write, read, alter, limited access read} for each of the subjects on each of the objects. Your answer may be in the form of an access control table if you wish.

[6]

c) Draw a *protection ring* to illustrate the access control rights of the subjects with regards to the examination paper.

[3]

d) Which of the access control permissions are enforced by the protocol, and which are dependent upon the trustworthiness of the subjects?

[7]

e) As it stands there are no integrity checks on either the examination papers or the final marks awarded. Suggest steps that could be added to the protocol to ensure that integrity is maintained.

[7]

Chapter 4

Encryption

4.1 Introduction

In this chapter, we will discuss encryption and see how encryption works using some simple examples that can be done by hand. Commercial encryption is done by computer these days and we will be looking at some particular cryptosystems in the following chapters. In this chapter we introduce the terminology necessary for the discussion and definition of cryptosystems.

Supplementary reading

Chapters 6 and 7 of *Secrets and Lies*, by Schneier gives a good introduction to cryptography without going into any details.
Chapter 1 of *Applied Cryptography* (also by Schneier) includes definitions of the terminology.
Chapter 2 of *Network Security Essentials* by Stallings gives details on symmetric key cryptosystems and includes a table of average time required for an exhaustive key search. Differential cryptanalysis is discussed in Chapter 12.
Chapters 1 to 5 of *Cryptography: A Very Short Introduction* by Piper and Murphy gives lots of details about historical ciphers, such as the substitution cipher, and how to break them.

After studying this chapter and the related reading, you should be able to:

- describe what encryption aims to achieve
- describe how perfect secrecy can be achieved using the one-time pad and discuss why this method is not used for all encryption
- describe how Caesar's cipher and substitution ciphers work and explain that these ciphers are not secure for use in the modern world of computers
- demonstrate familiarity with the terms *plaintext, ciphertext, encryption algorithm, decryption algorithm, encryption key, decryption key, message alphabet, ciphertext alphabet, keyspace, trivial key, block, blocksize* and *padding*, with regards to cryptography
- understand that defining a cryptosystem means giving details on all of the above
- describe the difference between a cryptographer and a cryptanalyst
- distinguish between the different kinds of attacks that a cryptanalyst might make on a cryptosystem, depending on what information they have available
- describe the properties that a good cryptosystem should incorporate to make it both secure and functional.

4.2 The history of encryption

Encryption is the process of transforming a *plaintext* message (a message that can be read) into an unreadable encrypted form called a *ciphertext* message.[1] The intention of encryption, is that if the encrypted message is intercepted, then the interceptor will not be able to interpret the ciphertext.

Messages have been encrypted for many years. Mary Queen of Scots used a substitution cipher (see section 4.4) to write secret letters in the sixteenth century while she was imprisoned in the tower of London. These secret letters contained her plans to escape and assassinate Queen Elizabeth. As they were using encryption, Mary and her allies openly discussed their plans in these letters. Unfortunately for them, the letters were intercepted and decrypted and their contents led to the execution of Mary!

Long before Mary Queen of Scots, Julius Caesar used a simple substitution cipher. We will study the Caesar cipher in section 4.4.1. Even before this time people were sending secret messages. The ancient Greek historian Herodotus tells a story about events preceding the Persian Wars at the beginning of the fifth century BC. In this story Histiaeus had to get a message to Aristagoras telling him to start a rebellion. The roads between them were closely watched so how could a message be sent safely? Histiaeus shaved the head of his most trusted slave, pricked the message out on his skull, and waited for the hair to grow back again. When the hair was grown, the slave travelled to Aristagoras with instructions to tell him to shave his hair off and look at his head. A good way of concealing a message, but not very efficient and quite painful for the slave!

In more recent history, the Enigma machine was used in the Second World War for the encryption and decryption of secret messages. The Enigma machine is a combination of mechanical and electrical systems. It creates a substitution cipher, but the scrambling action of the rotors means that the same letters are encrypted differently with consecutive uses. For example A,A may encrypt to G,C and at a later part of the message A,A may encrypt to Y,P. The Enigma code was broken but this was due to user error, for example using the same key for two consecutive messages, rather than an underlying weakness in the system. There are many good books and websites about the Enigma machine which make for good reading but are beyond the scope of this subject.

Simple substitution ciphers are discussed in the following sections as they give a nice demonstration of the art of encryption. However, note that such methods for encryption are not secure and are no longer used except for fun. These days mathematical algorithms are used to encrypt and decrypt messages using computers.

4.3 Perfect secrecy – the one-time pad

The aim of anyone encrypting a message is to ensure that no-one viewing the resulting ciphertext will be able to decrypt or make any sense of the ciphertext, apart

[1] Encrypting and decrypting may also be referred to as coding and decoding messages. This is not the correct terminology. Coding means getting the message into a particular format which may be a prerequisite of encryption but does not actually generate a ciphertext. For example, a message written in characters a, b, c...z may be coded into numbers 1, 2, 3...26 before encryption.

from the genuine receiver who will know how to decrypt the ciphertext and retrieve the original message. This is hard to achieve although not impossible. The *one-time pad* is a method of encryption that offers perfect secrecy.

Suppose Alice wants to send Bob a message using the one-time pad. Alice generates a stream of random binary digits (a list of 0s and 1s occurring at random) which is as long as the message. She makes a copy of this stream of digits and gives it to Bob. It is important that no-one else gets a copy of this digit stream so to achieve perfect secrecy Alice should personally give it to Bob. When Alice wants to send her message to Bob (this may be some time later when Alice cannot physically meet Bob) she codes it into a stream of binary digits. Then for each binary digit in the message she XORs [2] it to the binary digit in the random digit stream at the corresponding position.

Message	0 1 1 1 0 0 1 0 1 0 0 1 1...
Random Stream	1 0 0 1 1 0 0 0 1 1 0 1 0...
Ciphertext	1 1 1 0 1 0 1 0 0 1 0 0 1...

Alice sends Bob the ciphertext. Bob uses his copy of the random stream to retrieve the original message. The inverse of XOR is also XOR, so all Bob has to do is XOR the ciphertext with the random stream and he will recover the message.

Learning activity

Using the example above, recover the original message by XORing the ciphertext with the random stream.

The *inverse* of a mathematical operation is the operation that has the reverse effect. For example, the inverse of addition is subtraction, and the inverse of multiplication is division. Try some examples to show that the inverse of XOR is XOR (i.e. show that if $a \oplus b = c$ then $c \oplus b = a$ and $a \oplus c = b$, where a, b, c are binary streams all of the same length).

This method is known as the one-time pad because each digit in the random stream is used just once and the sequence is never repeated. The random stream must be as long as, or longer than, the original message (encoded into binary digits) so that there is no need to loop around to the beginning of the random stream. Each binary digit used for the encryption is therefore truly random and unpredictable.

The random binary stream is only used once and is discarded after use – hence the name *one-time pad*. An interceptor has no way of knowing whether the next bit in the random stream is a 0 or 1 and so can only take a 50/50 guess. This means that he can only make a 50/50 guess as to whether the next bit in the message is a 0 or a 1. These are the same odds as simply guessing the bits in the message – this is called *perfect secrecy*.

The disadvantage of the one-time pad is that it is enormously costly to operate and difficult to organise. Alice and Bob both have to have copies of the same random stream and this has to be sent and stored with extreme care. Therefore the one-time pad would only be used in extremely critical situations.

[2] XOR is addition mod 2. The mathematical symbol for XOR is $\oplus$. For example, $1 \oplus 0 = 1, 1 \oplus 1 = 0$.

Learning activity

Suppose Alice and Bob represent two governments who wish to communicate with each other using the one-time pad method during a time of crisis. Alice knows that she will have to send Bob a message of vital importance after a meeting takes place next week. Alice generates a random binary stream (the one-time pad) and sends a copy of it, secured in a locked case with an armed guard, to Bob. After the meeting, Alice encrypts her message to Bob using the one-time pad and sends the resulting ciphertext to Bob electronically.

An aide suggests to Alice that instead of using the one-time pad, she simply sends Bob the message in a locked case with an armed guard. Alice refuses saying that the one-time pad method is more secure. Why is this?

4.4 Substitution ciphers

One of the most basic techniques for encrypting a message is to use a *substitution cipher*. This is when each letter in the plaintext message, is substituted for a different letter to make the ciphertext message. The *key* that Alice and Bob (the sender and receiver) must share in order to encrypt and decrypt the messages is a table of letters showing the substitution alphabet, or in the case of a simple alphabet rotation (Caesar's cipher) it is the number of positions that the alphabet has been rotated by.

4.4.1 Caesar's cipher

The Caesar cipher is a particular type of substitution cipher. It is so called because Caesar is thought to have used this kind of encryption. The alphabet is shifted by a certain number of positions, k, where k is the *key* and (assuming we are using the English alphabet) will be a value between 1 and 25.

For example, if $k = 5$ then every letter in the plaintext message will be replaced with the letter five positions along. Thus A becomes F, B becomes G and so on. For the letters at the end of the alphabet, we *wrap around* to the beginning of the alphabet so that Y becomes D, for example.

message alphabet A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
ciphertext alphabet F G H I J K L M N O P Q R S T U V W X Y Z A B C D E

Using $k = 5$, the message THIS IS CRYPTOGRAPHY
becomes YMNX NX HWDUYTLWFUMD.

To decrypt the ciphertext message, the alphabet is shifted in the reverse direction by k places, so that A becomes V, B becomes W and so on.

Learning activity

Using the key $k = 5$ decrypt the message IJHWDUYNTS NX JFXD.

There are only 25 different keys possible for a Caesar cipher. This makes it very easy for an interceptor who gets hold of the ciphertext to decrypt it. They can simply try each of the 25 keys (different rotations of the alphabet) until they find the one that makes the message meaningful.

4.4.2 Random substitution cipher

A random substitution cipher is more secure than a Caesar cipher. This time each letter in the plaintext alphabet is replaced by a random letter in the ciphertext alphabet. The key is a table of letter substitutions. An example of a substitution cipher key is given below.

message alphabet A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
ciphertext alphabet J C G K S A I P B W L T F O Q H N D R E M Z Y U V X

Using this substitution key, the message MEET ME AT SIX becomes FSSE FS JE RBZ.

Learning activity

Using the substitution key above, decrypt the message B GJO'E B'O CMRV.

How many keys?

When making a substitution cipher key, we can choose any of 26 letters to replace A, then any of 25 letters to replace B, and any of 24 letters to replace C, and so on. Therefore the number of different possible permutations (re-rearrangements) of the alphabet is equal to $26 \times 25 \times 24 \times \ldots \times 2 \times 1 = 26! \approx 4 \times 10^{26}$. This is the number of different keys that can be used when implementing a substitution cipher.

> The set of possible keys that can be used to implement an encryption algorithm is called the ***keyspace***. In general, the larger the key space, the harder it is to decrypt the ciphertext without knowledge of the key.

The keyspace often includes *trivial* keys, for example in the Caesar cipher the key $K = 0$ which maps $A \rightarrow A, B \rightarrow B, \ldots Z \rightarrow Z$ is trivial because the ciphertext would be exactly the same as the plaintext if this key were used. Instructions not to use trivial keys, and details of which keys are trivial, should be included in the description of an encryption algorithm.

Breaking a substitution cipher

A substitution cipher has many more possible keys than a Caesar cipher, but it is often easy to break a substitution cipher by using statistical analysis and knowledge of the language that the plaintext message is written in.

In English, certain letters occur a lot more frequently than others. E is most common letter followed by T, A, O, I, N, S, H and R. Given a long enough piece of ciphertext,

it is possible to analyse which letter occurs the most frequently and infer that this is probably the letter substituted for E and so on.

Furthermore, the *redundancy* of the English language means that certain letter combinations do not occur and others occur frequently. For example, a letter on its own is most likely to be A or I. Pairs of letters might be IS, IN, AN, ON and so on. Scanning the ciphertext and applying some logic and common sense it is often possible to decrypt the message by working out some key letter substitutions and guessing the rest. This method will reveal the plaintext message although it might not reveal the entire key as some letters might not appear in the ciphertext. For example, suppose we have intercepted part of a message which reads S GK SZ G .

We would guess that S and G must represent either A and I, although we don't know which is which. First trying S=A and G=I we have:

ciphertext S GK SZ G
message A I A I

Now in English there is no two letter word beginning with I that would normally follow A, so it is more likely that S=I and G=A. Trying this combination we have:

ciphertext S GK SZ G
message I A I A

Common two letter words beginning with A are AM, AN, AS, AT. Following after I the most likely is AM, so next we try the substitution K=M.

ciphertext S GK SZ G
message I AM I A

Common two letter words beginning with I are IF, IN, IS, IT. To make sense of the decryption so far, it is most likely that the word is IN so now we guess that Z=N.

The part of the message that we have intercepted reads I AM IN A and we know some parts of the substitution key.

message alphabet A I M N
ciphertext alphabet G S K Z

On its own, this part of the message and substitution key is not much use, but if we had more of the message, we could use the parts of the key that we now know in order to try and decrypt the rest of the message.

Learning activity

Suppose you intercept the rest of the message in the example above. The complete ciphertext is:

S GK SZ G UGAO SZ KSVGZ. KOOL KO WONO GL FOHOZ GZB S YSVV WGZB RHON LWO URKTXLON UWST.

Use statistical analysis and the redundancy of the English language in order to recover the original message.

4.4.3 Improving security

There are many ways in which a substitution cipher can be made more secure (i.e. harder for a cryptanalyst to break). See Chapter 3 of *Cryptography: A very short introduction* for more details on these ciphers.

We will consider one method of improving security here because of its importance in all types of encryption.

4.4.4 Blocking

Consider the ciphertext G QT Q TQZ. If this is a message written in the English language then it most likely decrypts to I AM A MAN. No other meaningful English sentence fits this pattern. We can use language redundancy to decrypt this message without knowing the key.

We can make decrypting the ciphertext a harder task if we include *space* as a character in the substitution key. For example if *space* encrypts to R then the ciphertext becomes GRQTRQRTQZ. This is harder to decipher, but we could still use a frequency test to see which letters are occurring most often and make some deductions about the message from the results.[3]

We can decrease the efficacy of statistical tests by using a technique called *blocking*.

In our previous examples, we have always encrypted the plaintext character by character. In effect we have been using a *blocksize* of one with each block being one of the 27 different characters from the *message alphabet*. If we increase the blocksize to two we will increase the number of possible blocks from 27 to $27^2 = 729$. For each possible different block AA, AB, AC, AD, ... AZ in the message alphabet there is a corresponding block (also of size two) in the ciphertext alphabet. For example, the first part of the substitution key might look like this. The *space* character is represented by △.

message alphabet	AA	AB	AC	AD	AE	AF	AG	AH	AI
ciphertext alphabet	JW	CF	GR	KE	SD	WG	IJ	PP	BV
message alphabet	AJ	AK	AL	AM	AN	AO	AP	AQ	AR
ciphertext alphabet	WE	LR	TO	FN	OV	QW	HI	NB	DE
message alphabet	AS	AT	AU	AV	AW	AX	AY	AZ	A△
ciphertext alphabet	RQ	EN	MT	ZC	YW	UN	VT	XL	TE

To encrypt the message I AM A MAN we first block the message characters into blocks of size two: $\underbrace{I\triangle}\ \underbrace{AM}\ \underbrace{\triangle A}\ \underbrace{MA}\ \underbrace{N\triangle}$

We look up each of these blocks in the substitution key and replace the plaintext block with the corresponding ciphertext block. The entire substitution key is not given here, but let us suppose that I△ encrypts to BR, △A encrypts to HT, MA encrypts to UL and N△ encrypts to JJ. Then the ciphertext corresponding to the message
I AM A MAN is BRFNHTUKJJ.

[3] In general, frequency analysis and other statistical techniques are most likely to succeed for longer messages. If a message is short then there may not be enough repetition of letters to make an analysis of the frequencies meaningful. For example, the ciphertext TRNKYO gives nothing away about the plaintext LEMONS.

The three As in the original message have been encrypted to F,T and J respectively and so do not give any indication to the cryptanalyst that the letter A occurs three times in the message.

> In general, the larger the block size, the harder it is to decrypt the ciphertext without knowledge of the key. This is because the size of the *message alphabet* and *ciphertext alphabet* increases as the block size increases.

If we increased the blocksize to three, we would have a message alphabet of size $27^3 = 19683$. This would further improve the security of the cryptosystem, but the substitution table would have to have 19683 entries which is not practical. Instead of simple substitution, other methods of encryption are generally used to transform a plaintext block from the message alphabet into a ciphertext block from the ciphertext alphabet. This is generally based on an encryption algorithm which relies on some mathematical function seeded with an initial encryption key which is known to both the sender and receiver of the cryptogram. Block sizes for block ciphers (see Chapter 5) which are in commercial use today are typically 128 bits giving a message alphabet (and ciphertext alphabet) of size 2^{128}.

Note that in all the examples we have looked at so far, the message alphabet and the ciphertext alphabet have been the same. This is often the case although not always so. However, the size of the message alphabet cannot be larger than the size of the ciphertext alphabet.

Padding

Suppose we have a blocksize of five characters and a message of length 17 characters such as THIS IS A MESSAGE.

We would group the characters of the message into blocks of length five as follows:

$$\underbrace{THIS\triangle}_{block1}\ \underbrace{IS\triangle A\triangle}_{block2}\ \underbrace{MESSA}_{block3}\ \underbrace{GE}_{block4}$$

The fourth block is too short as it only has two characters instead of five. We have to pad the last block with three padding characters to make it of length five. For example, we could add three X characters to the message before encryption so that the length of the message is divisible by the blocksize. Now the fourth block is GEXXX.

Giving the block size, method of blocking and how to pad blocks if necessary, is an important part of any encryption algorithm. Some algorithms also include sending the original length of the message so that padding characters can be stripped away by the receiver after decryption.

4.5 Definitions

Here is a recap of the important definitions that we have covered so far in this chapter.

- The *sender* A (Alice) wants to send a message m, called the ***plaintext*** to a

receiver B (Bob).

- Alice encrypts the message using an ***encryption key*** k and an ***encryption algorithm*** f to produce a ***ciphertext*** $c = f_k(m)$.
- To decrypt the ciphertext c, Bob uses his ***decryption key*** k', and a ***decryption algorithm*** g to recover the original message $m = g_{k'}(c)$.
- The characters of the message come from a ***message alphabet*** **M**, and the ciphertext will consist of characters from the ***ciphertext alphabet*** **C**. The two alphabets M and C may be the same, but they could be different.
- The ***keyspace K*** is the set of all possible encryption keys. The keyspace may include ***trivial keys*** even though these should not be used when implementing the encryption algorithm.
- The message may be grouped into ***blocks*** of a specified length, called the ***blocksize*** before encryption. Details should be given on how to block the message, including instructions on how to ***pad*** the message if it does not fill an entire set of blocks.
- The enemy in the cryptosystem is the person who intercepts the message and tries to gain unauthorised information. Such a person is called a ***cryptanalyst*** and is often given the name Charles in discussions regarding cryptosystems.
- A ***cryptographer*** is someone who studies all aspects of cryptosystems.
- A ***cryptosystem*** refers to all the aspects of a particular encryption system, and so includes information about the message and ciphertext alphabets, the encryption and decryption algorithms, the method of blocking, and the allowable keys. A property of the encryption process must be that Bob retrieves the original message. Thus for a particular pair of keys k, k' it must be true that for every m in M, if $c = f_k(m)$ then $m = g_{k'}(c)$. Equivalently we can say that for all m in M, $g_{k'}(f_k(m)) = m$. This property binds the keys k and k'.

4.5.1 Types of encryption algorithm

The encryption algorithms that we have considered so far are all ***symmetric key*** encryption algorithms. The decryption key is either the same as, or can easily be derived from, the encryption key. Knowledge of one key gives knowledge of the other. In order to keep the encryption secure, both keys need to be kept secret. We will study some of the symmetric key encryption algorithms used today in Chapter 5.

We will also look at ***asymmetric key*** encryption and decryption algorithms, also known as ***public key*** algorithms in Chapters 7, 8 and 9. In these algorithms, knowledge of the encryption key reveals no information about the decryption key. Therefore the encryption key does not need to be kept secret; anyone can use it to send a message to the holder of the corresponding decryption key. It is vital that the decryption key is kept a secret. The encryption and decryption keys are also known as the ***public*** and ***private*** keys respectively.

4.6 Attacking a cryptosystem

What might a cryptanalyst try to do if he intercepted a message encrypted using an encryption algorithm? Suppose that cryptanalyst Charles intercepts a message from Alice to Bob. He might try to do any of the following:

1. Determine the message without determining the key.
2. Determine the decryption key so that he can decrypt the message (and any other messages that he intercepts which have been encrypted using the same key).
3. Determine the encryption key so that he can determine the decryption key (in the case of a symmetric key algorithm).
4. Determine the encryption key so that he can masquerade as (pretend to be) Alice and send false messages encrypted using the encryption key.

When considering the security of a cryptosystem, it is best to assume that Charles, the enemy, has knowledge of the encryption and decryption algorithms, the key space and alphabets and the method of blocking. In fact, we assume that Charles knows everything about the cryptosystem except for the keys used. We also assume that Charles has adequate computing power, finance and expertise on his side.

The type of attack that Charles can launch depends on what other knowledge or ability to manipulate the messages he has. Types of attack are called:

- **Ciphertext only** – Charles has only the ciphertext. He has no idea what the message is about.
- **Known message** – Charles knows what the message, or part of the message, says and he has the corresponding ciphertext. For example, Charles might know that the message begins 'Dear Bob'. He will use this knowledge to try and decipher the rest of the message, or to find the encryption/decryption key.
- **Probable known message** – Similar to the above but Charles may not be sure that he knows the message. For example, he may think that the message starts 'Dear Bob', or he may know that the message concerns shares in a particular company and so will include the name of that company, but he might not know whereabouts in the message the name occurs.
- **Chosen message** – Charles generates a message and he persuades Alice to encrypt it. Then armed with the full chosen message and ciphertext generated by Alice, Charles tries to find the encryption key.
- **Chosen ciphertext** – As above, but Charles generates some ciphertext and persuades Bob to decrypt it for him.
- **Chosen message and ciphertext** – Charles is able to choose both messages for encryption and decryption. He has full access to all plaintext and ciphertext messages. He is trying to find the encryption and decryption keys.

4.6.1 Methods of attack

The cryptanalyst has various strategies that he can use to attack the cryptosystem. Following are brief explanations of three of them.

Try all possible decryption keys

The cryptanalyst might try to decrypt part of the ciphertext using all possible decryption keys until he finds a match with the plaintext (in the case of a known message attack) or a probable match with the plaintext (in the case of a probable known message or ciphertext only attack).

The success of this method will depend on:

- the size of the keyspace
- the redundancy in the message (if the message is not known).

How long it will take for the cryptanalyst to recover the key using this method will depend on:

- the number of keys in the keyspace
- the length of time it takes to investigate each key.

Analyse the ciphertext statistically

As we saw in section 4.4.2 it may be possible for a cryptanalyst to deduce which characters are replaced by which by doing a frequency count on the letters in the ciphertext. Redundancy in the English language may give away the rest of the message.

Differential cryptanalysis

In the real world, instead of persuading Alice and Bob to encrypt and decrypt messages for him while he is a performing a chosen message and ciphertext attack, Charles would probably have developed a *black box* that he can use to encrypt as many messages as he wants. He would change a small part of the message each time and then study the change in the corresponding ciphertext. This is known as **Differential cryptanalysis** and has been used to break several symmetric cryptosystems including weak versions of DES.

Learning activity

Alice is using a substitution cipher to send a message to Bob. The message consists of upper and lower case letters, digits and the space character. She replaces each character in the message with another character. Alice encrypts the message character by character. She has a table telling her which character is to be replaced by which.

1. What is the message alphabet?
2. What is the ciphertext alphabet?
3. What is the keyspace?
4. Are there any trivial keys? If so what are they?
5. What is the size of the message alphabet?
6. What is the size of the keyspace?

4.7 Properties of a good cryptosystem

In order to make it both secure and functional, a good cryptosystem should have the following properties:

- a large alphabet M – to make it hard to do statistical analysis
- a large keyspace – to make it hard to do an exhaustive keyspace search
- high speed of execution – for high message throughput
- the same (or similar) algorithm for encryption and decryption – to reduce development costs and prevent bottlenecks occurring when many messages are being encrypted and decrypted.

4.8 Summary

In this chapter we have looked at the general theory of encryption and have defined some important terms that will be used in the next chapters when we look at some particular cryptosystems in more detail.

4.9 Learning outcomes

After studying this chapter and the related reading, you should be able to:

- describe what encryption aims to achieve
- describe how perfect secrecy can be achieved using the one-time pad and discuss why this method is not used for all encryption
- describe how Caesar's cipher and substitution ciphers work and explain that these ciphers are not secure for use in the modern world of computers
- demonstrate familiarity with the terms *plaintext, ciphertext, encryption algorithm, decryption algorithm, encryption key, decryption key, message alphabet, ciphertext alphabet, keyspace, trivial key, block, blocksize* and *padding*, with regards to cryptography
- understand that defining a cryptosystem means giving details on all of the above
- describe the difference between a cryptographer and a cryptanalyst
- distinguish between the different kinds of attacks that a cryptanalyst might make on a cryptosystem, depending on what information they have available
- describe the properties that a good cryptosystem should incorporate to make it both secure and functional.

4.10 Sample examination questions

Question 1

a) In a symmetric cryptosystem, what do the terms *blocksize* and *keyspace* mean?

[2]

b) A symmetric key cryptosystem should have a large alphabet and a large keyspace. Explain why each of these properties is important.

[4]

c) One type of attack on a cryptosystem that a cryptanalyst might use is called a *known message attack*. Name and describe three other types of attack.

[6]

Chapter 5

Symmetric key cryptosystems

5.1 Introduction

In this chapter, we will look in more detail at the method of encryption which uses symmetric keys (i.e. the encryptor and decryptor both share the same key). The Caesar cipher and substitution ciphers are simple examples of symmetric key cryptosystems, but they are not practical for use in today's world of technology. We will look at the classic cryptosystem DES and the new standard AES as well as other symmetric schemes including Twofish, IDEA and RC6.

Supplementary reading

Chapter 2 of *Network Security Essentials*, by William Stallings gives details on symmetric encryption including details on the Feistel Cipher Structure used by many block ciphers. It also gives details on DES and AES as well as an overview on other block cipher algorithms.

Chapters 4 and 5 of *Practical Cryptography*, by Niels Ferguson and Bruce Schneier discuss different block ciphers and the different ways in which they may be implemented in practice.

Several websites are referenced throughout this chapter and it is recommended that you follow up these references to learn as much as possible about the NIST competition for a new block cipher standard (AES) and the shortlisted algorithms.

Cryptonomicon is a novel by Neal Stephenson [ISBN 0099410672] which makes use of the Solitaire block cipher designed by Bruce Schneier. You may enjoy the novel, although this is not required reading. The algorithms for Solitaire can be found in an appendix at the back of the book or on the website `www.schneier.com/solitaire.html`

After studying this chapter and the recommended reading, you should be able to:

- explain what is meant when a cryptosystem is called a symmetric key cryptosystem
- describe the difference between a block cipher and a stream cipher
- discuss why it is important for a block cipher to have a large blocksize and a large keyspace and understand what is meant by these term
- list the blocksize and keysize for the block ciphers DES, 3DES and Rindjael
- describe how the DES block cipher works using a Feistel structure and how it may be implemented as 3DES in order to increase the keysize
- describe how the Rijndael (AES) block cipher works,
- name at least three other block ciphers which are currently in commercial use.

5.1.1 Symmetric key cryptosystems

We have seen in the previous chapter that *encryption* and *decryption algorithms* are used to encrypt and decrypt messages. A *key* is used which may be shared between the parties encrypting and decrypting messages.[1] This is known as a symmetric key. The decryption key is either the same as, or can easily be derived from, the encryption key. Both Caesar's cipher and the substitution cipher are examples of symmetric key encryption schemes. However, they are trivial to break and are not used for commercial cryptography.

Symmetric key cryptosystems which are used today include Rijndael (also known as the AES), 3DES, Idea, Blowfish and RC6. We will study Rijndael in some detail, and also see how the symmetric scheme DES can be made more secure by implementing a version known as Triple DES or 3DES.

5.2 Block ciphers and stream ciphers

Symmetric encryption cryptosystems are also known as *block ciphers* or *stream ciphers*. As the names suggest, a block cipher encrypts data block by block, whereas a stream cipher encrypts data in a continuous stream. All of the symmetric schemes which we will discuss can be used in different modes of operation (see section 5.2.3) which means that they can act as block ciphers or stream ciphers depending on how they are implemented.

5.2.1 Stream ciphers

The one-time pad (see section 4.3) is a kind of stream cipher. The plaintext is enciphered bit by bit by adding the *keystream* to the plaintext. Although it is very secure, the problem with the one-time pad is that since the keystream is random, it cannot be generated simultaneously by both the sender and receiver. A more practical stream cipher uses a short key to generate a long keystream.

A simple method for generating a keystream is to start with a binary key of length n bits, and generate the next bit of the sequence by XORing (addition modulo 2) the first and last bit of the previous n bits. Depending on the initial key, it is possible to generate a sequence which does not repeat until a keystream of length $2^n - 1$ bits has been produced.

For example, if the initial key is the 4-bit string 1111 we can generate the 15-bit keystream:

$$\underbrace{\mathbf{1111}01011001000}_{non-repeating\ \ keystream\ \ of\ \ 15\ \ bits} \qquad 1111$$

This is an example of a *Linear Feedback Shift Register* or LFSR. In general, if the initial state (initial key of length L bits) of the LFSR is $s_{L-1}, s_{L-2}, \ldots, s_1, s_0$ then the each bit of the output $s_j = (c_1 s_{j-1} \oplus c_2 s_{j-2} \oplus \ldots \oplus c_L s_{j-L})$ for $j \geq L$.

[1] Encryption schemes where the decryption key is different from, and cannot be derived from, the encryption key are known as asymmetric schemes. We will study these in Chapters 7, 8 and 9.

Learning activity

What are the key streams produced by XORing the first and last bits of the previous four bits if the initial 4-bit key is:

1. 1011
2. 0110
3. 0000?

A stream cipher produces ciphertext (C) by XORing the plaintext (P) with the keystream (K). Thus for the i^{th} bit in any message:

$$C_i = P_i \oplus K_i$$

This means that the ciphertext is decrypted by XORing the ciphertext with the keystream:

$$P_i = C_i \oplus K_i$$

It also means that the keystream can be recovered if the plaintext and ciphertext are known:

$$K_i = C_i \oplus P_i$$

Therefore if a cryptanalyst Charles knows a section of plaintext and the corresponding ciphertext, he can easily recover the keystream for that section.

The security for a stream cipher therefore depends on the design of the keystream generator. The keystream must be unpredictable. Designing a good keystream generator is difficult and requires advanced mathematics. However, there are many applications for stream ciphers because of their speed of use, ease of implementation and the fact that one bit of corrupt ciphertext does not impact on the rest of the message.

Designing a stream cipher

Linear Feedback Shift Registers are widely used in keystream generators because they are well suited for use in hardware implementations, produce sequences with large periods (i.e. sequences with up to $2^n - 1$ bits with no repetitions) and have good statistical properties. However, they are also easy to predict and so should not be used on their own as a keystream generator.

Instead LFSRs should be used in combination with each other. One technique for combining the output of n LFSRs in a non-linear manner, is to use a *combining function* f which takes as input the n outputs from the LFSRs, and outputs the keystream. This is illustrated in Figure 5.1.

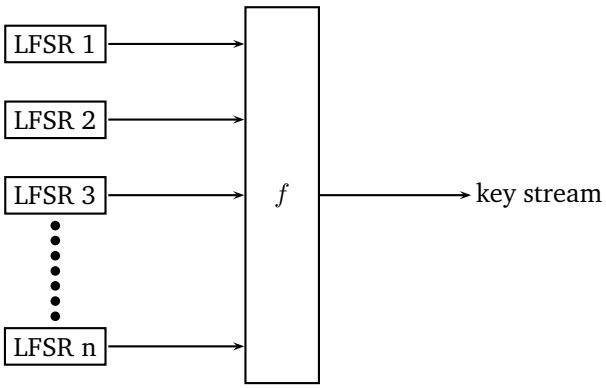


Figure 5.1: LFSRs combined to produce a key stream using function f

The function f must satisfy several criteria in order to be able to withstand cryptographic attacks. Students who are interested should see section 6.3 of *The Handbook of Applied Cryptography*.[2]

5.2.2 Block ciphers

For a block cipher, the plaintext bit-string is divided into blocks of a given size, called the *blocksize*, and the encryption algorithm acts on that block to produce a ciphertext block which is usually the same size. The algorithm uses a key which is chosen from a *keyspace* and which has a particular *key size*.

Generally the greater the blocksize the greater the security, but there is a compromise on the speed of the encryption and decryption. Until recently a blocksize of 64 bits was thought to be large enough, but modern algorithms use a blocksize of 128, 192 or 256 bits. The blocksize is directly related to the alphabet size. If the blocksize is 64 bits then there are 2^{64} possible different plaintext blocks. Modern computers can use statistical analysis on an alphabet of 2^{64} blocks.

The *keysize* is also important. If the keysize is too small then the keyspace can be searched using an exhaustive searching technique in order to find the key.

So, a well designed block cipher should satisfy each of the following properties.

1. A large blocksize (and therefore alphabet) to prevent statistical analysis attacks.
2. A large keysize (and therefore keyspace) to prevent exhaustive search attacks.
3. The *diffusion* property (which means that a small change in the plaintext should produce an unpredictable change in the ciphertext). This is to prevent a chosen message and ciphertext attack called *differential analysis*.
4. The *confusion* property (which means that if a key is 'nearly correct' it should not give any indication of this fact). This will make exhaustive key searching much harder.

[2]Not all stream ciphers make use of LFSRs. Notable exceptions are SEAL and RC6.

5. The *completeness* property (which means that every bit of the ciphertext should depend on every bit of the key). This prevents a *divide and conquer* attack whereby the cryptanalyst tries to determine part of the key independently of other parts.

5.2.3 Block cipher modes

As we have seen, a block cipher encrypts a block of plaintext using a symmetric key. If the plaintext is longer than the size of one block, then the block cipher must be used in a particular *mode* in order to encrypt the entire message. We will have a brief look at the different modes that can be used. In the following, P_i represents plaintext block number i, C_i represents the corresponding ciphertext block and K represents the symmetric key.

Electronic codebook mode (ECB)

The simplest method is to break the plaintext up into blocks of the correct size and apply the encryption algorithm with the encryption key to each block individually. This is called *electronic codebook mode* or ECB.

$$C_i = Encrypt(K, P_i)$$

This mode of use is generally not recommended because if blocks are repeated and encrypted using the same key, the ciphertext will also be repeated. An attacker will be able to see this.

Cipher block chaining mode (CBC)

The problems of ECB mode can be overcome by *randomizing* the plaintext using the previous ciphertext block. This is known as cipher block chaining or CBC and is the most widely used block cipher mode. Each plaintext block is XORed with the previous ciphertext block before encryption. This means that two identical plaintext blocks will be transformed into non-identical blocks before the encryption. An attacker will not be able to tell that the plaintext blocks were identical.

$$C_i = Encrypt(K, P_i \oplus C_{i-1})$$

We still have the problem of deciding on a value for C_0 (i.e. what value should be XORed to the first plaintext block?). This value is known as the *initialization vector* or IV. The IV may be chosen in one of several ways.

- **Fixed IV** – We could always use the same IV. However, this is not recommended because in this case if the first block of different messages is the same ('Dear Bob' for example) then the corresponding ciphertext block will also be the same leading to similar problems as with ECB.
- **Counter IV** – We could use a counter to set the IV. Using IV=0, then IV=1 and so on. However, this is also not recommended as the difference in the IV is only 1 bit and so this is not very secure.

- **Random IV** – Using a random IV is arguably more secure than using a fixed or counter IV. However, now we have two problems. The first is how do we generate a random number; and the second is how do we securely transmit the random IV between the sending and receiving parties?
- **Nonce-generated IV** – This is often the solution to the IV problem. A *nonce* is a number used once. Each message can be assigned a number using a counter. This is the nonce. Instead of using the nonce as the IV, the nonce is encrypted using the block cipher in ECB mode to generate the IV. The block cipher is now used in CBC mode with C_0 equal to the nonce generated IV.

Output feedback mode (OFB)

Used in output feedback mode (OFB) the message itself is never used as input to the block cipher. Instead, the block cipher is used to generate a pseudorandom stream of bytes called the keystream. The keystream is then XORed with the plaintext to generate the ciphertext. This mode of operation uses a block cipher as a stream cipher.

We have both the symmetric key K, and an initialisation vector IV. The next part of the keystream is generated by encrypting the previous block of keystream.

$$\begin{aligned} K_0 &= IV \\ K_i &= Encrypt(K, K_{i-1}) \\ C_i &= P_i \oplus K_i \end{aligned}$$

It is very important that the value of IV is never reused because this would produce an identical keystream. It is also important to check that the keystream is not producing a repeating block. This would lead to each block being encrypted using the same key which is not a secure encryption scheme.

In general, block ciphers can be used to provide confidentiality, data integrity, user authentication, or as the keystream generator for a stream cipher. In the next sections we will look at the algorithms of some symmetric key cryptosystems.

5.3 DES and Triple DES

The Data Encryption Standard (DES) is a classic symmetric block cipher algorithm. It was developed in the 1970s as a US government standard. It has a blocksize of 64 bits and a keysize of 56 bits.

DES uses a *Feistel* cipher structure which means that for every *round* of encryption, the block is divided into two halves L_i and R_i. Each half of the block is processed and then used to provide input to the other half for the next round of the encryption. The encryption key K is split into pieces and a piece K_i is also used as input for the next round.

In DES, the encryption key K is used to generate 16 subkeys $K_1, K_2, K_3, \ldots, K_{16}$, each of length 48 bits. The following is then performed 16 times. After the 16^{th}

round has been completed, the two halves L_{16} and R_{16} are concatenated to form the ciphertext.

1. The block is split into two halves L_i and R_i.
2. The left half of the output at the next round is the right half of the previous round ($L_{i+1} = R_i$). The new right half is the XOR of the left half and the value of $F(K_i, R_i)$ where F is a function.[3]

Decryption is the reverse of encryption since $R_i = L_{i+1}$ and $L_i = R_{i+1} \oplus F(K_i, R_i)$.

The diagram in Figure 5.2 shows one round of a Feistel cipher structure as used in DES.

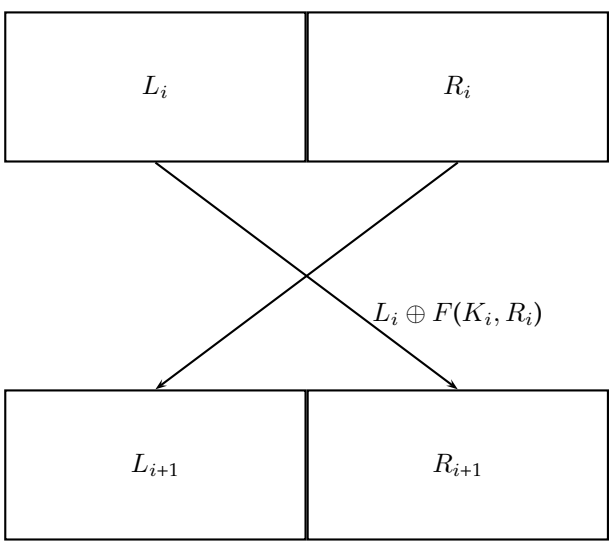


Figure 5.2: DES Feistel structure

Strengths and weaknesses of the DES algorithm

DES is a tried and tested algorithm. It has been in use since the 1970s and has been subjected to a great deal of cryptanalysis. Even if you have both the plaintext and the corresponding ciphertext, it seems difficult to find the key. Altering one bit of the plaintext block, alters about half of the bits of the ciphertext block. The functions used are a mixture of different mathematical structures with no apparent shortcuts.

However, although DES was secure when it was designed in 1977, the key size of 56 bits is now considered to be too small. Cryptanalysis techniques such as linear and differential cryptanalysis can be used to improve the search time for the key.

5.3.1 Triple DES

One way to improve the security of DES is to increase the key size. This is done by using a variation called Triple DES or 3DES. Three DES keys K_1, K_2 and K_3 are used

[3] Details of the function F are omitted as they are very complicated and are not examinable.

which means that the keysize is tripled and the size of the keyspace is exponentially increased because it now contains 2^{168} keys. This is large enough to defeat exhaustive key search attacks with current technology. The blocksize of 64 bits remains unchanged.

If Alice and Bob share DES keys K_1, K_2 and K_3, and Alice wants to send a message, M, to Bob using 3DES, she takes the following steps:

1. Alice encrypts M using DES key K_1 to obtain C.
2. Alice decrypts C using DES key K_2 to obtain M'.
3. Alice encrypts M' using DES key K_3 to obtain C'.
4. Alice sends the ciphertext C' to Bob.

In order to decrypt the ciphertext, Bob performs the reverse steps:

1. Bob decrypts C' using DES key K_3 to obtain M'.
2. Bob encrypts M' using DES key K_2 to obtain C.
3. Bob decrypts C using DES key K_1 to obtain M.

Learning activity

Alice and Bob have designed a very simple stream cipher with the following encryption protocol:

1. The message is padded at the right-most end with 0s so that its length is divisible by eight.
2. The padded message is split into blocks of 8 bits.
3. A further block of 8 bits is appended at the right-most end of the message which contains the length of the original message. (It is assumed that the messages are of length less than 2^8 bits so that the length will fit into one 8-bit block.)
4. Alice and Bob agree on a symmetric key of 8 bits.
5. Starting with the left-most block:
   (a) The key is XORed with the message block to obtain the ciphertext block.
   (b) The message block and the key are then added together and the eight most significant bits form the key for the next block.

   Step 5) is repeated until all of the message blocks have been encrypted.

For example, the 28 bit message 1101001111001101100010101101 is padded with four zeros and length byte 00011100. Using the key 01100110, the message is encrypted to produce the ciphertext 10110101010100010011111001001111101010 11.

Message	11010011	11001101	10001010	11010000	00011100
Key	01100110	10011100	10110100	10011111	10110111
Ciphertext	10110101	01010001	00111110	01001111	10101011

Write the corresponding decryption protocol and demonstrate it by decrypting the ciphertext 11001100011110000111011001001001 0 using the key 01100110.

How secure do you think Alice and Bob's symmetric key cryptosystem is?

5.4 Advanced Encryption Standard (AES)

When DES (Data Encryption Standard) was designed in 1977, it was intended to be secure for the next 15 years. In 1997 NIST (National Institute of Standards and Technology) issued a call for proposals for a new *advanced encryption standard* or AES to replace DES as the standard. NIST specified that the new cryptosystem must be a symmetric block cipher with a block length of 128 bits and support key lengths of 128,192 and 256 bits.

The proposals submitted were evaluated for security, computational efficiency, memory requirements, software sustainability and flexibility. Five ciphers were short listed, and in 2000, the winner was announced as Rijndael which was designed by two Belgian cryptographers, Joan Daemen and Vincent Rijmen. Rijndael has now been adopted as the new standard and may be referred to as the AES. A European project called NESSIE (New European Standards for Security, Integrity and Encryption) also recommended Rijndael in the block cipher category.

We will look at the Rijndael algorithm in some detail. We will also have a brief look at some of the other algorithms as they are each useful for different purposes. See the paper *Report on the Development of the Advanced Encryption Standard (AES)* at http://csrc.nist.gov/archive/aes/index.html, for further details on the AES competition and criteria used to reach a decision in choosing Rijndael as the winner.

5.5 Rijndael

Rijndael uses a blocksize of 128 bits and a keysize of 128, 192 or 256 bits. We will describe the 128-bit algorithm.

The input to the encryption and decryption algorithms is a single 128-bit block. This block is copied into the *state array* which is a square matrix of bytes and which is modified at each stage of the encryption or decryption. After the final stage, the state array is copied to an output matrix. The 128-bit key is also depicted as a square matrix of bytes. The key is expanded into an array of key schedule words each of four bytes and the total key schedule is 44 words for the 128-bit key.

The Rijndael algorithm does not use a Feistel structure. Instead it processes the entire data block in parallel during each round using substitutions and permutations. The key that is provided as input is expanded into an array of 44 32-bit words w_i. Four distinct words (which is $32 \times 4 = 128$ bits) serve as a round key for each round of the algorithm. Four different operations are used. These are:

- **Substitution bytes** – a look-up table referred to as an S-box is used to perform a byte-by-byte substitution of the block.
- **Shift rows** – this is a simple permutation performed row by row.
- **Mix columns** – this is a substitution algorithm that alters each byte in a column using a function which takes as input all of the bytes in the column.
- **Add round key** – this is a simple bit-wise XOR of the current block with a portion of the expanded key.

For both encryption and decryption, the algorithm starts with an *Add round key* operation, followed by nine rounds which include all four operations, followed by a tenth round of three operations.

Only the Add round key operation makes use of the key. This is why this operation must be performed at the start and end of the algorithm. If any other round were performed at the beginning or end of the algorithm, it would be reversible without knowledge of the key and so would be of no additional security value. However, the Add round key operation by itself would not be secure. The other three operations scramble all the bits but do not make use of the key. Putting all of the operations together makes Rijndael both highly efficient and secure.

All of the operations are easily reversible. The decryption algorithm makes use of the expanded key in the reverse order to recover the plaintext. The decryption algorithm is not identical to the encryption algorithm as a consequence of the particular structure of Rijndael.

Strengths and weaknesses of Rijndael

Rijndael is very flexible and can be implemented using different block sizes and keysizes depending on the application. It is also very efficient and requires little memory. This means that it can be implemented in software as well as hardware. However, because it is a relatively new cryptosystem, it has not been studied in as much detail as other older block ciphers such as DES. It is possible to think of the data in Rijndael as being polynomials with co-efficients of either 0 or 1. This means that, unlike DES, it is possible to write a Rijndael encryption as an algebraic formula. This mathematical structure might, or might not, prove to be a weakness – if someone could solve the algebraic formula then the cryptosystem will be broken. Only time and further study will tell how secure Rijndael is in the future.

5.5.1 Some other symmetric cryptosystems

As mentioned above, five symmetric cryptosystems were shortlisted to be the new AES. These were MARS, RC6, Twofish, Rijndael and Serpent. We will have a brief look at some of these algorithms. Further details can be found in Stallings *Network Security Essentials* or by doing an Internet search.

Twofish

Twofish was developed by Bruce Schneier, the independent cryptographer and author of *Practical Cryptography* and *Secrets and Lies*. Twofish is a 128-bit block cipher that accepts a variable length key of up to 256 bits. It is a 16 round Feistel network (similar to DES) but the S-boxes are key-dependant (unlike DES). Twofish is similar to the Blowfish algorithm, also developed by Schneier (see Stallings for details of Blowfish). Twofish can be implemented in hardware or on a smartcard. It has been designed to permit a wide variety of trade-offs between speed, software size, key setup time, gate count and memory.

See the paper *Twofish: A 128-bit Block Cipher* available at`www.schneier.com/paper-twofish-paper.pdf` for further details.

RC6

The RC6 block cipher was designed by Ron Rivest (who was also one of the inventors of the RSA public key cryptosystem which we will discuss in Chapter 8), Matt Robshaw, Ray Sidney and Yiqun Lisa Yin from RSA Laboratories. The RC6 algorithm is an evolutionary improvement over the RC5 algorithm developed by the same people.

RC6 is designed to be very simple to allow direct analysis. Such analysis has not been able to break the cipher and so implies that it is secure. It is suitable for use in hardware or software such as smartcards as it has a low memory requirement, and is very fast. It is also very flexible allowing for variable word lengths, number of rounds and key lengths.

RC6 is used in a number of products from RSA Data Security Inc.

See the paper *The RC6 Block Cipher* available at ftp://ftp.rsasecurity.com/pub/rsalabs/rc6/rc6v11.pdf for further details.

IDEA

The IDEA (International Data Encryption Algorithm) block cipher was not on the AES shortlist. However, it was one of the first proposed 128-bit key replacements for DES and has thus undergone considerable scrutiny and appears to be highly resistant to cryptanalysis. IDEA is used in PGP and in a number of commercial products.

IDEA does not use substitution boxes (S-boxes) for the round function. Instead it relies on three different mathematical functions, namely XOR, binary addition of 16-bit integers, and binary multiplication of 16-bit integers. These functions are combined in such a way as to produce a complex transformation that is very difficult to analyse and hence very difficult to cryptanalyse.

IDEA has eight rounds, with six subkeys being generated for each round. The subkey generation algorithm uses a complex series of circular shifts.

Learning activity

In Neal Stephenson's novel *Cryptonomicon* the character Enoch Root describes the Solitaire cryptosystem (it is called *Pontifex* in the book to disguise the fact that it uses a pack of cards). This is a symmetric cryptosystem which is designed to be low-tech (it can be implemented by hand using a deck of cards) but high-security. The cryptosystem was designed by the independent cryptographer Bruce Schneier, who we have already mentioned several times.

The encryption and decryption algorithms for Solitaire are given at the end of the book *Cryptonomicon* and on the website `www.schneier.com/solitaire.html`

Read the description of Solitaire (either in the book or on the Internet) and then have a go at encrypting and decrypting a message using the Solitaire algorithms. The best way to see how a cryptosystem works is to try it out for yourself.

5.6 Summary

In this chapter we have looked at a type of symmetric key cipher known as a block cipher. We have seen that a block cipher can be used to encrypt a plaintext message block by block, or as the key stream generator for a stream cipher which encrypts plaintext bit by bit in a continuous stream. We have looked at the classic block cipher DES and we have seen how to make DES more secure by tripling the keysize and therefore exponentially increasing the size of the keyspace by implementing 3DES. We have discussed the NIST call for a new block cipher and looked at the winning cipher Rijndael, which is also known as the AES. We have also briefly discussed some other block ciphers, namely IDEA, RC6, Twofish and Solitaire. These ciphers use a mixture of mathematical functions and substitution boxes (or S-boxes) over a number of rounds to encrypt a plaintext message block by block.

5.7 Learning outcomes

After studying this chapter and the recommended reading, you should be able to:

- explain what is meant when a cryptosystem is called a symmetric key cryptosystem
- describe the difference between a block cipher and a stream cipher
- discuss why it is important for a block cipher to have a large blocksize and a large keyspace and understand what is meant by these term
- list the blocksize and keysize for the block ciphers DES, 3DES and Rindjael
- describe how the DES block cipher works using a Feistel structure and how it may be implemented as 3DES in order to increase the keysize
- describe how the Rijndael (AES) block cipher works,
- name at least three other block ciphers which are currently in commercial use.

5.8 Sample examination questions

Question 1

a) In a public key cryptosystem, you do not need to keep the encryption key secret. Why is it essential to keep the encryption key secret in a symmetric key cryptosystem?

[1]

b) In a symmetric key cryptosystem, what do the terms *blocksize* and *keysize* mean? Why is it important for a symmetric cryptosystem to have a large blocksize and a large keysize?

[4]

c) What is the main difference in the way in which data is encrypted by a stream cipher and a block cipher?

[2]

d) Why are stream ciphers used in implementations where a lot of interference might be expected such as mobile telephone communications?

[2]

e) A block cipher should have the properties of *diffusion*, *confusion* and *completeness*. Explain what is meant by each of these properties and why they are essential for a secure block cipher.

[9]

f) Triple DES and Rijndael are two widely used symmetric key cryptosystems. State one advantage and one disadvantage of Rijndael when compared to Triple DES.

[2]

Question 2

A commercially available symmetric key cryptosystem called *XYZ* has blocks of size b-bits and keys of size k-bits.

a) Why does the value of b need to be large? With modern algorithms what would you estimate to be the smallest value of b that could guarantee security?

[2]

b) How could *XYZ* be modified to a new symmetric key cryptosystem called *Double-XYZ* so that the blocksize is still b-bits but the keysize is $2k$-bits? Explain the suggested encryption and decryption algorithms for *Double-XYZ* in terms of *XYZ*.

[6]

c) It takes 10 seconds to encrypt one block using *XYZ*, 0.01ms to transmit one block of ciphertext using *XYZ*, and 100 hours to do an exhaustive key search for the *XYZ* key. Approximately how long would it take to perform the following tasks (give explanations for your answers):

   i. To encrypt one block using *Double-XYZ*?

   ii. To transmit one block of ciphertext using *Double-XYZ*?

   iii. To do an exhaustive key search for a *Double-XYZ* key?

[9]

Chapter 6

Hash functions

6.1 Introduction

In this chapter, we will look at the uses and properties of hash functions in cryptography. We will study the SHA family of hash algorithms in some detail.

Supplementary reading

Section 3.2 of *Network Security Essentials* by Stallings discusses hash functions and SHA.
The NIST website *http://csrc.nist.gov/publications/* gives details and examples of all of the SHA algorithms.

After studying this chapter and the related reading, you should be able to:

- describe what is meant by a hash function
- state the four properties that make a hash function cryptographically strong, and describe why each of these properties is important
- demonstrate an understanding of the SHA-512 hash algorithm.

6.2 Hash functions

A hash function, h, takes data of arbitrary size as input and returns a value in a fixed range. If you compute the *hash* of the same data at a different time, you should always get the same answer – if not then the data has been modified. The hash of data x is sometimes called the *message digest* or *fingerprint* of x.

A hash function may be used simply to shorten and store data. For example, we might represent a person by the initials of their first and last names. This is effectively a hash function taking as input the person's full name. We are taking data (the full name) which is of arbitrary length and hashing it to obtain a hash of exactly two characters. If we hash the same name at a different date, we will obtain the same two characters. Such a hash function is easy to compute, but it is also easy to find *collisions* (i.e. two names which hash to the same value); and *preimages* (i.e. names which would hash to a given hash value). Hash functions are regularly used in cryptographic protocols including the generation of digital signatures (see section 7.3) and the X.509 certification protocol (see section 10.4.1). Hash functions which are used in cryptography need to be *cryptographically strong* and must satisfy certain properties.

6.2.1 Properties of a cryptographically strong hash function

A hash function h acts on data x and returns a value $h(x)$. In order to be considered cryptographically strong, the hash function should have these four essential properties.

1. Given x it is easy to compute $h(x)$.
2. The input x can be of arbitrary length.
3. Given a value y, it should be hard to find an x such that $y = h(x)$. This is also known as finding a *preimage*. By 'hard' we mean that it should not be possible to find a preimage using an exhaustive search within a feasible amount of time.
4. It is hard to find two different values x_1 and x_2 such that $h(x_1) = h(x_2)$. This is also known as finding a *collision*.[1] 'Hard' in this instance means that you should not be able to find a collision within a feasible amount of time (but note that the birthday paradox means that you would expect to only have to search the square-root of the possible values to find a collision).

These properties are all very important. We will consider the reasons behind each:

1. The hash function must be easy to compute in order to make computations fast and efficient. Cryptographic protocols may require numerous hashes to be computed. If the hash function is not efficient this may cause a bottleneck with subsequent steps in the protocol being held up.
2. We might require any message to be hashed and we cannot predetermine the lengths of the messages. Therefore the hash function must be able to process messages of any length.
3. This property ensures integrity, meaning that the message x cannot be replaced by another message with the same hash value.
4. This property ensures non-repudiation. If a cryptanalyst could find two messages x_1 and x_2 such that $h(x_1) = h(x_2)$ then they could potentially send message x_1 with a signature based on the hash of x_1 and later claim that the message was actually x_2 since the signatures for both messages would be the same.

Note that since the hash function can take input of arbitrary length and outputs a value in a fixed range it is necessarily a many-to-one function. This means that there will exist different values of x which have the same value of $h(x)$. In order to satisfy properties three and four above, the number of possible hash outcomes must therefore be large enough to prevent an exhaustive search revealing different values of x with the same hash value. The hash function is said to be *collision resistant* if it is hard to find two different values of x with the same hash value.

6.2.2 Hash functions as one-way functions

Hash functions which are cryptographically strong can be used as one-way functions (see section 2.4.1) because they cannot be reversed. Windows software typically uses a SHA hash function to create a hash of a user password. This hash is stored in

[1] A surprising statistical fact about collisions is known as the *birthday paradox*. The probability that at least two people in a room of 23 people have the same birthday is greater than a half. Considering that there are 326 possible birthdays this is not intuitive. In hash functions too, collisions occur much more frequently than you would intuitively expect. Generally, if there are 2^n possible hash functions, you would only have to look at $2^{\frac{n}{2}}$ before finding a collision.

the password file rather than the actual password. This has led to tables of SHA hash values being pre-computed and stored to produce incredibly fast password cracking algorithms. These algorithms use *rainbow tables* to store the hash values and these tables can be searched in a matter of seconds to find the corresponding password. These techniques are very useful in the case of users retrieving their lost passwords, but are obviously not such good news if they are operated by cryptanalysts who are trying to hack into a password protected computer system.

Learning activity

Do an Internet search for more details on rainbow table password cracking techniques and search times.

The discrete logarithm problem (see section 9.2) is a one-way function which satisfies property three of a cryptographically strong hash function because it is very hard to find x such that $y = h(x)$ given y. However, if numbers of arbitrary value are used as input to the function, it is relatively easy to find collisions (i.e. different values of x with the same output value of y).

6.3 The Secure Hash Algorithm (SHA)

There are two families of hash functions in the public domain. These are *MD* and *SHA*. The algorithm *MD5* was used until recently, but has now been broken and is no longer considered to be cryptographically secure.

There are five *SHA* (Secure Hash Algorithm) algorithms called *SHA, SHA-1, SHA-256, SHA-384* and *SHA-512*. The original SHA algorithm has been broken. Its successor SHA-1 is under attack and is no longer recommended for cryptographic use. NIST (National Institute of Standards and Technology) recommends that all cryptographic protocols use SHA-256 in place of SHA-1 by the year 2010.

The SHA hash algorithms are iterative, one-way functions that can process a message to produce a condensed representation called a *message digest*. A message digest can be used in a cryptographic protocol to ensure the integrity of a message, since any change to the message will, with a very high probability, result in a change to the message digest.

The algorithms can be described in two stages, namely pre-processing and hash computation. Pre-processing involves padding the message, parsing the padded message into m-bit blocks and setting initialisation values. The hash computation generates a message schedule from the padded message and uses that schedule, along with the functions, constants and word operations to iteratively generate a series of hash values. The final hash value generated by the hash computation is used to determine the message digest.

The algorithms provide *diffusion* which means that a small change in the message produces a big change in the message digest. This is a desirable property for a cryptographic hash function to prevent a cryptanalyst from using 'nearly correct' hash values to find collisions or preimages. This is achieved by the expansion of each original message block into eighty words.

The different SHA algorithms differ most significantly in the number of bits of

security that are provided for the data being hashed. For the later versions this is directly related to the length of the message digest. For example, the message digest size for SHA-256 is 256 bits whereas the message digest size for SHA-512 is 512 bits. Note that SHA-1 does not, of course, generate a message digest of 1-bit!

We will look at the version, SHA-512 in more detail. Further details and full specification of all of the SHA algorithms can be found at`http://csrc.nist.gov`.

6.3.1 SHA-512

The SHA-512 algorithm uses a message schedule of eighty 64-bit words, eight working variables of 64 bits each and a hash value of eight 64-bit words. The final result is a 512-bit message digest.

The words of the message schedule are labelled $W_0, W_1, \ldots, W_{79}$. The eight variables are labelled a, b, c, d, e, f, g and h. The words of the hash value are labelled $H_0^{(i)}, H_1^{(i)}, \ldots, H_7^{(i)}$. At the start of the algorithm, these hold the initial values, H^0 and they are replaced by each successive intermediate hash value (i.e. they are updated during each iteration of the algorithm). SHA-512 also uses two temporary words T_1 and T_2.

SHA-512 preprocessing

First the message M is padded as follows:

Suppose the length of the message M is L bits. The bit 1 is appended to the end of the message, followed by k zero bits, where k is the smallest non-negative solution to the equation:

$$L + 1 + k \equiv 896 \bmod 1024$$

The 128-bit block that is equal to the number L expressed in binary is appended to the end of the message.

For example, the message **abc** is written in ASCII as 01100001 01100010 01100011. This message has length $8 \times 3 = 24$ bits so $L = 24$. The value of k is $896 - 24 - 1 = 871$ so we pad the message with a 1 bit followed by 871 0 bits, followed by a 128-bit binary representation of the number 24.

$$\underbrace{01100001}_{\mathbf{a}}\ \underbrace{01100010}_{\mathbf{b}}\ \underbrace{01100011}_{\mathbf{c}}\ 1\ \overbrace{00\ldots00}^{871 bits}\ \overbrace{00\ldots0\underbrace{11000}_{L=24}}^{128 bits}$$

The length of the message is now a multiple of 1024. It is parsed into blocks each of length 1024 bits, $M^{(1)}, M^{(2)}, \ldots, M^{(N)}$.

The hash values $H_0^{(i)}, H_1^{(i)}, \ldots, H_7^{(i)}$ are given the following initial values. These are 64-bit words, but are given here in hex notation. These values are included here for completeness, but please note that you do not need to remember or quote these values.

$$H_0^{(0)} = 6a09e667f3bcc908$$
$$H_1^{(0)} = bb67ae8584caa73b$$
$$H_2^{(0)} = 3c6ef372fe94f82b$$
$$H_3^{(0)} = a54ff53a5f1d36f1$$
$$H_4^{(0)} = 510e527fade682d1$$
$$H_5^{(0)} = 9b05688c2b3e6c1f$$
$$H_6^{(0)} = 1f83d9abfb41bd6b$$
$$H_7^{(0)} = 5be0cd19137e2179$$

SHA-512 hash computation

SHA-512 uses six logical functions which involve shifts, bitstring operations (and,or) and modular arithmetic mod2^{64}. Each of these functions operates on a 64-bit word and outputs a new 64-bit word.

Each message block $M^{(1)}, M^{(2)}, \ldots, M^{(N)}$ is processed in order using the following steps:

For i = 1 to N:

1. Prepare the message schedule $(W_0, W_1, \ldots, W_{79})$ by expanding the 1024-bit message block $M^{(i)}$ into 80 64-bit words using the following algorithm.

$$W_t = \begin{cases} M_t^i & 0 \leq t \leq 15 \\ \sigma_1^{512}(W_{t-2}) + W_{t-7} + \sigma_0^{512}(W_{t-15}) + W_{t-16} & 16 \leq t \leq 79 \end{cases}$$

   σ_1^{512} and σ_0^{512} are functions involving rotations and shifts of the input parameter word. (Please note that details of the expansion algorithm will not be examinable.)

2. Initialise the eight working variables a, b, c, d, e, f, g and h with the $(i-1)^{st}$ hash value:

$$a = H_0^{(i-1)}, b = H_1^{(i-1)}, \ldots, h = H_7^{(i-1)}$$

3. The values of a, b, c, d, e, f, g and h are operated on using the six logical functions and with input of the eighty W values and temporary variables T_1 and T_2. Details of the functions are omitted and are not examinable. Just as an example, the function T_1 is given below. Ch is a function which combines three binary parameters using bitwise AND, XOR and complement. K_t^{512} is the first sixty-four bits of the the fractional part of the cube root of the t^{th} prime number.

$$T_1 = h + \Sigma_1^{512}(e) + Ch(e, f, g) + K_t^{512} + W_t$$

4. The intermediate hash values $H^{(i)}$ are computed as follows:

$$
\begin{aligned}
H_0^{(i)} &= a + H_0^{(i-1)} \\
H_1^{(i)} &= b + H_1^{(i-1)} \\
H_2^{(i)} &= c + H_2^{(i-1)} \\
H_3^{(i)} &= d + H_3^{(i-1)} \\
H_4^{(i)} &= e + H_4^{(i-1)} \\
H_5^{(i)} &= f + H_5^{(i-1)} \\
H_6^{(i)} &= g + H_6^{(i-1)} \\
H_7^{(i)} &= h + H_7^{(i-1)}
\end{aligned}
$$

After repeating steps one to four a total of N times all the blocks of the message M have been processed. The values of $H_0^{(N)}$, $H_1^{(N)}, \ldots, H_7^{(N)}$ are concatenated to produce the 512-bit message digest.

Statistical and experimental tests show that the SHA-512 algorithm has the required properties for a cryptographically strong hash function. However at the time of writing, no proofs can be given that the SHA algorithms satisfy properties 3 and 4, and it is now known that SHA-1 is vulnerable.

6.4 Summary

In this chapter we have looked at hash functions which are used to produce a message digest. We have studied the four properties that are necessary in order for a hash function to be considered cryptographically strong. We have looked in some detail at the SHA-512 algorithm and have seen how a message is padded, the pre-computation steps and iterative hash computation of this algorithm.

6.5 Learning outcomes

After studying this chapter and the related reading, you should be able to:

- describe what is meant by a hash function
- state the four properties that make a hash function cryptographically strong, and describe why each of these properties is important
- demonstrate an understanding of the SHA-512 hash algorithm.

6.6 Sample examination questions

Question 1

a) Name four essential properties of a cryptographic hash function explaining for each property why it is important.

[8]

b) Which of the four properties does the SHA-512 hash algorithm satisfy?

[4]

c) Explain how a cryptographically strong hash function can act as a one-way function and how it can be used to protect a password word file. Why might it be a problem to use a well-known hash function in this way?

[13]

Chapter 7

Asymmetric cryptosystems

7.1 Introduction

In this chapter, we will look at the idea behind public key or asymmetric key cryptosystems. Such cryptosystems differ from symmetric schemes in that each user requires a public key which is available to all other users, and a private key which is never transmitted but is kept a secret by its holder. We will start to use modular arithmetic which is very important in asymmetric cryptosystems, and we will briefly discuss computational complexity applied to the algorithms that are used in public key cryptography.

Supplementary reading

Practical cryptography, by Niels Ferguson and Bruce Schneier covers many aspects of public key cryptosystem including details on how to find prime numbers and random numbers, and modular arithmetic.
Pfleeger, *Stallings* and *Gollmann* all contain sections or chapters on asymmetric cryptography.

After studying this chapter and the related reading, you should be able to:

- discuss the concept of a public key cryptosystem
- describe how a public key cryptosystem can be used as the basis for a digital signature protocol
- apply the algorithm for exponentiation and modular exponentiation using small numbers
- understand why the computational complexity of the algorithm for exponentiation is $O(b^3)$,
- understand why computational complexity plays an important part in cryptography.

7.2 Public key cryptosystems

We have so far only considered symmetric key cryptosystems where a pair of encryption and decryption keys are shared between two corresponding parties. Until quite recently these were the only kind of cryptosystem in existence. However, in the late 1970s a new kind of cryptosystem was devised by Whit Diffie and Martin Hellman. Their paper, entitled *New Directions in Cryptography*, and published in 1976, had an enormous impact on cryptography and its subsequent development.

The basic idea of a public key cryptosystem is that each person using the cryptosystem has a private key known only to themselves. The corresponding public key is made available to everyone else. The keys are devised so that knowledge of the public key reveals nothing about the private key. If Alice wants to send Bob a message using this system, she would look up Bob's public key in a key directory, encrypt the message for Bob using his public key and then send the resulting ciphertext to Bob. Bob can then use his private key to decrypt the message. Without this private key the message is undecipherable.

The public keys are stored openly, but there is a requirement that all public keys are authorised as being genuinely associated with their declared owner. For example, Alice must be sure that when she encrypts a message for Bob using a public key, that she is in fact using Bob's key and not a key which actually belongs to Charles. If she accidentally used Charles's public key, perhaps due to malicious behaviour by Charles, then the ciphertext would be meaningless to Bob and only Charles would be able to decrypt the message.

In a public key cryptosystem, not only the public keys, but also the encryption algorithm, are public. Therefore an attacker is in the position of trying to decrypt a ciphertext armed with the knowledge of the public key used to encrypt it and the method that was used to encrypt it. It may seem impossible to believe that even in these circumstances, it is hard for the attacker to deduce the message. Of course, the encryption algorithm and keys must be carefully chosen to ensure that the cryptosystem is secure and that no information about the decryption (private) key is leaked.

This is a difficult concept to understand. If Charles knows the public key and encryption algorithm, why can't he simply replicate the encryption process to find the corresponding ciphertext and hence find the message? The following non-mathematical example is often used to help explain the situation.

Suppose you are in a locked room containing nothing but a telephone directory consisting of millions of numbers indexed by names in alphabetical order. If someone asks you to find the telephone number of a particular person, that is an easy task. You simply look up the name and find the corresponding number. This is the equivalent of encrypting a message given the public key. However, if someone asks you to find the name of the person who has a particular number, this is a much harder task. You know what you have to do, look though all of the numbers until you find a match and hence read the corresponding name. This is not a difficult task in itself, but would be incredibly time-consuming. Most people would give up and beg to be let out of the room long before finding the correct number and corresponding name. This is the equivalent of trying to find the message given the ciphertext and public key.

In order for the security of a public key cryptosystem to be ensured, the parameters used must be extremely large. For example, prime numbers in the range of 200 to 600 decimal digits are required to defend against exhaustive search attacks for the period of secrecy required. Working with such large numbers takes some time, even for the most sophisticated modern computers. Therefore implementations of public key cryptosystems tend to be much slower than symmetric key cryptosystems. Public key cryptosystems are generally used to exchange a key to be used in a symmetric key cryptosystem, or to generate a *digital signature*.

7.3 Digital signatures

When you sign your name on a document, your signature is binding and proves that you agree to whatever it is that you have signed. For example, if you sign a cheque then you are agreeing to pay a certain amount of money to a specified person or company. However, if you send money or important information digitally, over the Internet for example, it is not possible to provide a hand written signature. Digital signatures are used instead to prove:

- who the message originates from
- that the message has not been corrupted or altered in any way.

Therefore a digital signature provides a means of both identification and authentication.

Digital signatures are calculated using the private key of the signer, and decrypted using the public key of the signer. Since only the genuine holder of the private key has access to that key, only he or she can create a signature which will decrypt correctly using the corresponding public key.

For example, suppose Bob wants to send a signed message to Alice. We will assume that the message is not secret and does not need to be encrypted. However, Alice wants to be assured (and wants to be able to prove if necessary) that it was Bob who sent the message.

Bob uses his private key to encrypt the message. This is the signature.

$$signature = \mathsf{encrypt}_{Bob\_private}(message)$$

Bob sends Alice both the unencrypted message and the signature.

Alice uses Bob's public key to decrypt the signature. She checks whether the result, *message'*, is equal to the original message.

$$message' = \mathsf{decrypt}_{Bob\_public}(signature) \stackrel{?}{=} message$$

If the decrypted signature is the same as the message then Alice knows that the message is genuinely from Bob, since only Bob knows his private key which has been used to create the signature. She is also assured that the message has not been altered or corrupted in any way since Bob signed it because the message itself is used in the creation of the signature.

7.3.1 Using hash functions in digital signatures

When a cryptosystem is used, the message has to be in a certain format before the encryption algorithm can be applied. The message is generally converted into a number, and then split into *blocks*, each of a maximum size. A long message may consist of many blocks, each of which are encrypted (and then decrypted separately).

To save time and computational power, a *hash* or condensed version of the message is usually produced before the signature is created. See Chapter 6 for details on cryptographically strong hash functions.

Instead of signing (by encrypting) the whole message, just the hash of the message is signed. The advantages of this are:

- Speed – the message digest is generally of length less than one block so can be signed using just one encryption. If the whole message has to be signed it must be broken into blocks and each block encrypted separately.
- Confidentiality – in order for the signature to be verified, the signer must send the receiver both the unsigned message and the signature. If the message is confidential the sender would not want to send the message unencrypted. However they could send the hash of the message unencrypted since the hash does not leak any information about the message.

To produce and authenticate a digital signature using a hash function, Alice and Bob take the following steps. We will assume again that Bob is sending a signed, unencrypted message to Alice.

1. Bob creates his message m.
2. Bob uses a cryptographically secure hash function to create a hash of the message $h(m)$.
3. Bob uses his private key to encrypt the message hash. This is the signature:

$$signature = \mathsf{encrypt}_{Bob\_private}(h(m))$$

4. Bob sends the pair $(m, signature)$ to Alice.

To decrypt the signature and confirm the message as genuine, Alice takes the following steps:

5. Alice decrypts the signature using Bob's public key to obtain s'.

$$s' = \mathsf{decrypt}_{Bob\_public}(signature)$$

6. Alice uses the hash function to create a hash of the message $h(m)$.
7. Alice checks whether or not $h(m)$ is equal to s'. If $h(m) = s'$ then the message is verified as genuinely from Bob and uncorrupted. However, if $h(m) \neq s'$ then the message is rejected.

The problems used to create public key cryptosystem are mathematical problems. They are *one-way functions* which we have already seen in use in Chapter 2 for protecting a password file. We will look at two different public key cryptosystems, called *RSA* and *El Gamal* in detail in the next two chapters. In the rest of this chapter, we will introduce some necessary mathematics, in particular *modular arithmetic* which is used in both of the public key cryptosystems that we will study. We will also briefly consider the *computational complexity* of some basic algorithms.

7.4 Modular arithmetic

Both RSA and El Gamal use *modular arithmetic*. This is when, instead of using the whole field of integers, we work in a field of integers $\mathrm{mod}\, m$. The answer to a

calculation is always in the range 0 to $m-1$ where m is some positive integer called the ***modulus***.

To calculate the value of n mod m you subtract as many multiples of m as possible from n until you are left with an answer in the range 0 to $m-1$. In other words, any number n can be written in the form $n = km + r$ where r is some integer between 0 and $m-1$ (inclusive). The result of n mod m is r. (i.e. the remainder when as many multiples of m as possible are subtracted from n). If n is a negative number then you add as many multiples of m as necessary to get an answer in the range 0 to $m-1$.

Modular arithmetic may seem complicated but we use it all the time without thinking about it. One example of this is when using a 12 hour clock. If we are told that a meeting is happening at 16:00 hours, we subtract 12 to work out that the meeting is at 4 o'clock in the afternoon. If I go to bed at 23:00 hours and get up at 07:00 hours then how long have I spent in bed? The answer is found by 07:00 – 23:00= –16:00=+8:00 hours. In these examples we are working mod12.

Here are some more mathematical examples:

$$\begin{aligned} 25 \bmod 7 &= 4 \\ 18 \bmod 12 &= 6 \\ 573 \bmod 2 &= 1 \\ 15 \bmod 20 &= 15 \\ -15 \bmod 20 &= 5 \\ 81 \bmod 3 &= 0 \\ -18 \bmod 5 &= 2 \end{aligned}$$

Congruent numbers

Two numbers r and s are said to be ***congruent mod m*** if $r \bmod m = s \bmod m$. In this case we write $r \equiv s$.

If r and s are congruent mod m, then the difference between r and s will be a multiple of m. Equivalently, if the difference between r and s is a multiple of m then r and s must be congruent mod m.

For example, $4 \equiv 9 \equiv 14 \equiv 19 \equiv -1 \equiv -6 \equiv -21 \bmod 5$.

When we are using modular arithmetic, it does not matter which equivalent numbers are used during the calculation – the final answer will be the same. The most efficient method is to apply the mod function at every stage of a calculation in order to keep the numbers involved in the calculation as small as possible.

For example, $(39 \times 15) \bmod 11 = 585 \bmod 11 = 2$ but we would be better off calculating:

$$(39 \times 15) \bmod 11 = (6 \times 4) \bmod 11 = 24 \bmod 11 = 2$$

7.4.1 Exponentiation

Exponentiation means *raising to the power*, for example calculating 2^8 or x^{17}. This is an important part of both the RSA and El Gamal algorithms. Most scientific calculators have an exponentiation button that allows you to work out x^n for small values of x and n. However, when the numbers get large, calculators cannot give an accurate answer. Furthermore, when we are using modular arithmetic, calculators cannot provide the answer. We therefore need to develop an algorithm for exponentiation.

Suppose we want to calculate x^8 where x is any positive integer.

Method 1

The first method is to multiply x by itself 8 times.

$x^8 = x \times x \times x \times x \times x \times x \times x \times x$

The total number of multiplications used in this method is eight. This is OK, but not efficient. Imagine if we were raising x to the power 72, then we would have to do 72 multiplications using this method.

Method 2

Another approach is to realise that $x^8 = (x^4)^2 = ((x^2)^2)^2$.

So we need to calculate x^2 which is one multiplication, then square the result $x^2 \times x^2$ to find x^4 which is another multiplication, and finally we square this result $x^4 \times x^4 = x^8$ to find the answer. Altogether we have used three multiplications instead of eight.

Now suppose that we want to compute x^9. We can compute x^8 using three multiplications as above and then multiply the result by x to find x^9 a total of four multiplications. This is because $x^9 = (x^8)(x) = ((x^2)^2)^2)(x)$.

What about x^{13}? Instead of making 13 multiplications, we can compute $x^{13} = x^8 \times x^4 \times x$. So to calculate x^{13} we need to compute x^2, then $x^4 = (x^2)^2$ then $x^8 = (x^4)^2$ and then multiply together x^8, x^4 and x.

A numerical example with $x = 2$ is as follows:

$$\begin{aligned} x &= 2 \\ x^2 &= 2^2 = 4 \\ x^4 &= (x^2)^2 = 4^2 = 16 \\ x^8 &= (x^4)^2 = 16^2 = 256 \\ x^{13} &= x^8 \times x^4 \times x = 256 \times 16 \times 2 = 8192 \end{aligned}$$

How do we know which powers of x to calculate?

This method of calculating only certain powers of x by squaring and then multiplying together the required powers to get the final answer is much more efficient than simply calculating every power of x up to the required answer. But how do we know which powers of x to compute and which ones to multiply together?

To work it out, write the power in binary e.g. 13 = 1101_2, then calculate *squares* for each digit in the binary representation starting with x under the rightmost bit. Finally multiply together the squares that have a 1 in the binary representation.

write 13 in binary:	1	1	0	1	
compute squares for each binary digit:	x^8	x^4	x^2	x	
multiply together the squares under a 1:	x^8	x^4		x	$=x^{13}$

7.4.2 Fast algorithm for exponentiation

The following algorithm can be used to compute x^n for any value of x and positive integer n. It is a formulation of Method 2 described above.

To compute x^n:

Initialise $y = 1, u = x$
Repeat
    if n mod 2 = 1 then $y = y \times u$
    $n = n\ div\ 2$
    if $n \neq 1$ then $u = u \times u$
Until $n = 0$
Output y

You may be expected to use this algorithm in the examination. As an example, we will use the algorithm to calculate 2^{25}. The steps are shown in Table 7.1.

The result is $2^{25} = y = 33554432$.

y	u	n	explanation
1	2	25	Initialise the values $y = 1, u = x = 2, n = 25$
2	4	12	$n = 25, 25$ mod 2 = 1 so $y \rightarrow y \times u$, $u \rightarrow u^2$ then $n \rightarrow 25$ div 2 = 12
2	16	6	$n = 12, 12$ mod 2 = 0, $u \rightarrow u^2$, then $n \rightarrow 12$ div 2 = 6
2	256	3	$n = 6, 6$ mod 2 = 0, $u \rightarrow u^2$, then $n \rightarrow 6$ div 2 = 3
512	65536	1	$n = 3, 3$ mod 2 = 1 so $y \rightarrow y \times u$, $u \rightarrow u^2$, then $n \rightarrow 3$ div 2 = 1
33554432	65536	0	$n = 1, 1$ mod 2 = 1 so $y \rightarrow y \times u$, then $n \rightarrow 1$ div 2 = 0 so we stop.

Table 7.1: Computing 2^{25} using the algorithm for fast exponentiation

7.4.3 Fast algorithm for modular exponentiation

Now suppose that we want to calculate 2^{25} mod 211. The algorithm for exponentiation given above still applies, but this time we *take mods* at each stage.

To compute x^n mod m:

Initialise $y = 1, u = x$ mod m
Repeat
    if n mod 2 = 1 then $y = y \times u$ mod m
    $n = n\ div\ 2$
    if $n \neq 1$ then $u = u \times u$ mod m
Until $n = 0$
Output y

You may be expected to use this algorithm in the examination. As an example we will calculate 2^{25} mod 211. The steps are shown in Table 7.2. The result is 2^{25} mod 211 = 157.

y	u	n	explanation
1	2	25	Initialise the values $y = 1, u = x$ mod 211 = 2, n = 25
2	4	12	
2	16	6	
2	45	3	$u \rightarrow u^2$ mod 211 = 256 mod 211 = 45
90	126	1	$y \rightarrow y \times u$, $u \rightarrow 45^2$ mod 211 = 126
157	126	0	$y \rightarrow y \times u$ mod 211 = 90 $\times$ 126 mod 211 = 157, then $n \rightarrow 1$ div 2 = 0 so we stop.

Table 7.2: Computing 2^{25} mod 211 using the algorithm for modular exponentiation

Note that we could calculate 2^{25} = 33554432 and then compute the modular answer 33554432 mod 211 = 157 but this is not efficient as we are working with much larger numbers than necessary. Furthermore, in the case of bigger numbers, the intermediate steps get too large for a normal calculator if mods are not taken at each step.

Learning activity

Use the algorithm for modular exponentiation to calculate:

1. 3^{23} mod 17
2. 5^{45} mod 2311

7.4.4 Modular inverses

The *modular inverse* of a mod p is the number b, $1 \leq b \leq p-1$ such that $a \times b$ mod $p = 1$. This is denoted by a^{-1} mod p.

For example, the modular inverse of 2 mod 17 is equal to 9 because $2 \times 9 = 18 = 1$ mod 17. Since modular multiplication (like ordinary multiplication) is commutative (i.e. it doesn't matter which order you have the numbers in) the modular inverse of 9 mod 17 = 2.

If p is a prime modulus, then all numbers between 1 and $p-1$ will have a modular inverse mod$p$. The only numbers which do not have a modular inverse mod$p$ are

multiples of p. If the modulus is a composite number n then all numbers which are co-prime with n (i.e. do not have any factors in common with n) will have an inverse $\bmod n$.

Table 7.3 gives all of the modular inverses for the prime 17.

$a \bmod 17$	1	2	3	4	5	6	7	8	
$a^{-1} \bmod 17$	1	9	6	13	7	3	5	15	
$a \bmod 17$	9	10	11	12	13	14	15	16	
$a^{-1} \bmod 17$	2	12	14	10	4	11	8	16	

Table 7.3: Modular inverses for the prime 17

Learning activity

Make a table of modular inverses for the prime numbers 19 and 23.

7.4.5 Euclid's algorithm

For small numbers, it is easy to work out a modular inverse using trial and error. However, this will not work for larger numbers.

An efficient method to find a modular inverse is called *Euclid's Algorithm*. This algorithm involves only multiplications and divisions and so is of computational complexity $O(b^2)$ (see section 7.5.1 below). The code for this algorithm and an example is given below. You will not be required to find a modular inverse other than a trivial one such as $5^{-1} \bmod 19 = 4$, in the examination. However, you may have to use Euclid's algorithm or an alternative method to find a less trivial inverse in order to complete a coursework assignment.

Code for Euclid's algorithm

The following code applies the extended Euclid algorithm to find the inverse of $b \bmod a$:

```
static long FindInverse(long a, long b)
{
        long store = a;
        long temp;
        long q;
        int sign = 1;
        long r = 1;
        long s = 0;
        while(b! = 0)
        {
                q = a/b;
                temp = r;
                r = temp * q + r;
                s = temp;
                temp = b;
                b = a - q * temp;
                a = temp;
                sign = -sign;
        }
        long answer = (r - (sign * s))%store;
        return(answer);
}
```

Given a number n which is co-prime to modulus m, Euclid's algorithm works by finding the integers a and b such that $1 = an - bm$. This means that $an = 1 + bm$ and hence a is the inverse of n mod m.

When working by hand, it is best to work iteratively first dividing the modulus by n and then n by the remainder, and so on until the remainder is equal to one. Then re-arrange the final equation so that it is in the form $1 = \ldots$ and step by step substitute in the values from the preceding equations until we have $1 = an - bm$. It is easier to see how this works by following an example.

We will demonstrate the algorithm by calculating the inverse of 223 mod 660:

$$660 = 223 \times 2 + 214 \quad (7.1)$$
$$223 = 214 \times 1 + 9 \quad (7.2)$$
$$214 = 9 \times 23 + 7 \quad (7.3)$$
$$9 = 7 \times 1 + 2 \quad (7.4)$$
$$7 = 2 \times 3 + 1 \quad (7.5)$$

See how in the equations above, the number on the right-hand-side of the = becomes the number on the left-hand-side of the next equation. The remainder in each equation becomes the divisor in the next. We keep going like this until we have remainder 1 in equation 7.5.

Now we re-arrange equation 7.5 to make 1 the subject. It is easier to use brackets rather than $\times$ symbols.

$$1 = 7 - 3(2) \quad (7.6)$$

From equation 7.4 we have $2 = 9 - (7 \times 1)$ or $2 = 9 - 7$. We substitute this into equation 7.6 and then simplify as follows:

$$1 = 7 - 3(9 - 7) \quad (7.7)$$
$$1 = 4(7) - 3(9) \quad (7.8)$$

From equation 7.3 we have $7 = 214 - 9(23)$. Substituting this into equation 7.8 and simplifying gives:

$$1 = 4(214 - 9(23)) - 3(9) \quad (7.9)$$
$$1 = 4(214) - 95(9) \quad (7.10)$$

From equation 7.2 we have $9 = 223 - 214$. Substituting this into equation 7.10 and simplifying gives:

$$1 = 4(214) - 95(223 - 214) \quad (7.11)$$
$$1 = 99(214) - 95(223) \quad (7.12)$$

Lastly from equation 7.1 we have $214 = 660 - 2(223)$ and we substitute this into equation 7.12:

$$1 = 99(660 - 2(223)) - 95(223) \quad (7.13)$$
$$1 = 99(660) - 293(223) \quad (7.14)$$

From equation 7.14, we can see that the inverse of 223 mod 660 is equal to $-293 \bmod 660 = -293 + 660 = 367$.

To check the calculation, make sure that $223 \times 367 - 1$ is divisible by 660.

Learning activity

Use Euclid's algorithm to find the inverse of:

1. 19 mod 562
2. 300 mod 847

There is an alternative method for finding a modular inverse. This uses exponentiation and so is $O(b^3)$ and therefore not as efficient as Euclid's algorithm.

Instead of computing $x^{-1} \bmod p$ directly, we can calculate $x^{-1} = x^{p-2} \bmod p$.

This is because $(x \times x^{-1}) \bmod p = 1$ by definition of modular inverse.
And $x^{p-1} = 1$ by Fermat's Little Theorem (see section 8.2.3).

Hence $x^{p-1} = (x \times x^{-1}) \bmod p$.
Re-arranging this equation we have $\frac{x^{p-1}}{x} = x^{-1} \bmod p$.
And $\frac{x^{p-1}}{x} = x^{p-2} \bmod p$.
Hence $x^{-1} = x^{p-2} \bmod p$.

7.5 Computational complexity

Suppose that there are several algorithms which can all be used to perform the same task. We need some way to judge the efficiency of the algorithms compared with each other. We can break the algorithms down into the basic steps, and then count how many steps there are in each. This is called the *computational complexity* of the algorithm.

The computational complexity of an algorithm is given in terms of the size of the problem. For example, XORing two five bit binary numbers will take five basic steps, but XORing two 100 bit binary numbers will take 100 basic steps. The algorithm used to XOR is the same in both cases and its complexity is given in terms of the size of the binary numbers.

The *size* of a number n is defined to be the number of binary bits needed to write n in base 2. We use $b(n)$ to denote the size of n.

For example:

$b(5) = b(101_2) = 3$
$b(20) = b(10100_2) = 5$
$b(2^{12}) = b(4096) = b(1000000000000_2) = 13$

Learning activity

One way to compute $b(n)$, the binary size of a decimal number n, is to find the smallest power of 2 which is bigger than n. If $2^k > n \geq 2^{k-1}$ then $b(n) = k$. For example, $2^{10} > 1,000 > 2^9$ so $b(1,000) = 10$.

1. Find $b(127), b(128), b(129), b(10^6), b(10^9)$.
2. It is too big for your calculator, but use your previous answers to find $b(10^{12})$.
3. What is $b(0)$?

7.5.1 Computational complexity of basic algorithms

If we XOR two binary numbers each of size b then we take b steps. We say that the operation XOR has computational complexity of ***order b*** which is written $O(b)$.

If we double the size of an $O(b)$ operation, then we double the number of steps required and thus the time needed. Computational complexity is measured independently of the implementation. The amount of time that is required to perform b basic steps will vary from computer to computer. The important thing to note is that for an algorithm with complexity $O(b)$ multiplying the size of the problem by X will have the effect of multiplying the time required by X as well.

The computational complexity of adding two binary numbers is also $O(b)$.

However, the computational complexity of classical multiplication of two binary numbers is $O(b^2)$. This means that if you double the number of bits in the numbers to be multiplied, then the time required quadruples. This is because if it takes b^2 steps to multiply two b-bit numbers, it will take $(2b)^2 = 4b^2$ steps to multiply two 2b-bit numbers.[1]

Dividing x by y is another $O(b^2)$ operation, where b is the size of y.

Learning activity

1. Using a particular computer package, it takes $3\mu s$ to add two 200 bit numbers. How long does it take the same computer package to add two 1,000 bit numbers?
2. Using the same computer package, it takes $80\mu s$ to multiply two 200 bit numbers. How long does it take to multiply:
   (a) two 1,000 bit numbers?
   (b) two 100 bit numbers?

7.5.2 Complexity of the algorithm for exponentiation

Recall the algorithm for exponentiation:

To compute x^n:

Initialise $y = 1, u = x$
Repeat
    if n mod 2 = 1 then $y = y \times u$
    $n = ndiv2$
    if $n \neq 1$ then $u = u \times u$
Until $n = 0$
Output y

If the size of the power n is b, then the number of multiplications will be at least b because that is the number of successive squaring multiplications required. We also need to do up to $b - 1$ other multiplications depending on how many 1s there are in the binary representation of n. Thus the total number of multiplications required is between b and $2b$.

We have already seen that the order of complexity of multiplying two numbers together (using a classical multiplication algorithm) is $O(b^2)$. We need to do between b and $2b$ multiplications, so the number of steps we do in total will be between b^3 and $2b^3$. We ignore constant values when talking about complexity, so we say that this algorithm has order $O(b^3)$.

[1] Note that although the classical method of multiplication is $O(b^2)$ more efficient algorithms do exist. The FFT (Fast Fourier Transforms) method of multiplication has complexity of less than $O(b^2)$ and the Karatsuba multiplication method which uses a divide and conquer technique is $O(b^{1.58})$. You do not need to know these algorithms for multiplication work, but you should be aware of their existence.

Complexity of the algorithm for modular exponentiation

The computational complexity of calculating a mod is $O(b^2)$ because taking a mod is equivalent to making a division and a subtraction. Therefore, the computational complexity of performing a modular multiplication is $O(b^2)$. The algorithm for modular exponentiation is essentially the same as the algorithm for exponentiation above, except that at each step we perform a modular multiplication instead of a multiplication. Since both of these operations are $O(b^2)$ (assuming that we are using the classical multiplication algorithm) the overall computational complexity of the algorithm does not change. Thus the complexity of computing x^n mod m is $O(b^3)$ where b is the size of n.

Why is complexity important when studying cryptography?

Many public key cryptosystems make use of this binary method of modular exponentiation. There are ways of refining the method saving between 10 per cent to 30 per cent of the time required to calculate a modular exponentiation. Since both the RSA and El Gamal public key cryptosystem require many modular exponentiations to be calculated, implementing algorithms which are as efficient as possible is obviously extremely important.

On the other hand, we rely on the fact that there is no efficient algorithm that can be used to factorise a composite number or solve a discrete logarithm problem. These are the mathematical problems that form the one-way functions used in the RSA and El Gamal public key cryptosystems respectively. If someone came up with an algorithm that could efficiently factorise large numbers then the security of RSA would be destroyed.

7.6 Summary

In this chapter we have introduced the idea of a public key cryptosystem where the owner of the keys keeps the private key a secret but shares the public key with all other users. We have seen how a public key cryptosystem can be used as the basis for a digital signature protocol, with the private key being used to create the signature. We have also studied the algorithm for modular exponentiation which is used in the RSA and El Gamal public key cryptosystems, and have used this algorithm to perform exponentiations with relatively small numbers. We have seen that the computational complexity of the binary method for exponentiation is $O(b^3)$ and explained why computational complexity, although not studied in detail in this subject, plays an important part in cryptography.

7.7 Learning outcomes

After studying this chapter and the related reading, you should be able to:

- discuss the concept of a public key cryptosystem
- describe how a public key cryptosystem can be used as the basis for a digital signature protocol

- apply the algorithm for exponentiation and modular exponentiation using small numbers
- understand why the computational complexity of the algorithm for exponentiation is $O(b^3)$,
- understand why computational complexity plays an important part in cryptography.

7.8 Sample examination questions

Question 1

a) i. What is the computational complexity of modular multiplication, that is the computation of $(x \times y)$ mod m where x, y and m all have size b?

[2]

ii. In a particular implementation, it takes $20\mu s$ to perform a modular multiplication when $b = 100$. Approximately how long would it take to do a modular multiplication when $b = 300$?

[4]

b) i. Describe the algorithm for *modular exponentiation*, that is the computation of x^n mod m. Your description may be given as pseudo-code if you wish.

[6]

ii. Show that, if n has size b, there are at most $2b$ multiplication steps in the algorithm, and hence deduce its order of computational complexity.

[5]

iii. Using the algorithm, and showing all your working, compute 6^{11} mod 13.

[7]

Chapter 8

RSA

8.1 Introduction

In this chapter, we will look at the RSA public key cryptosystem. This cryptosystem is named after its three inventors Ron Rivest, Adi Shamir and Leonard Adleman. Since the algorithm was first published in 1977, RSA has managed to withstand years of intensive cryptanalysis. The one-way problem at the heart of RSA is the factorisation problem – it is easy to multiply together two prime numbers, but given the result it is very hard to find which prime numbers were multiplied together. We will study the factorisation problem, the key generation, encryption and decryption algorithms for RSA.

Supplementary reading

Most books on cryptography cover RSA as it is a classic well-known public key cryptosystem.

In particular, see section 19.3 of *Applied Cryptography* by Schneier, or Chapter 13 of *Practical Cryptography* by Ferguson and Schneier, or section 8.2 of *The Handbook of Applied Cryptography* by Menezes, van Oorschot and Vanstone.

Cryptography: a very short introduction by Piper and Murphy also contains some details on RSA and gives the 617 digit number that could earn you $200,000 if you can factorise it!

An Internet search will provide more articles and papers including details of the RSA Challenge. In particular, see the web pages of RSA, Certicom and Hewlett Packard.

After studying this chapter and the additional reading, you should be able to:

- recall the key mathematical facts that make RSA easy to implement but hard to break
- give the key generation, encryption and decryption algorithms for RSA
- understand the importance of choosing suitable parameters (large primes) when implementing RSA in a commercial setting so that the cryptosystem cannot be broken by factorisation of the public key
- use the RSA encryption and decryption algorithms (with the modular exponentiation algorithm if necessary) to demonstrate the RSA cryptosystem using small prime parameters.

8.2 The factorisation problem

8.2.1 Prime numbers

An integer (whole number) is said to be ***prime*** if it has exactly two factors, namely one and itself. The first prime numbers are 2, 3, 5, 7, 11, 13, 17, 19, 23,... An

Internet search will find many sites that give lists of prime numbers.[1]

An integer greater than one that is not prime is called ***composite***. Thus a composite number can be written as a product of its prime factors. For example, 15 is a composite number and can be written as $15 = 3 \times 5$ where 3 and 5 are both prime numbers.

8.2.2 Factorisation

There is an efficient algorithm which can be used to determine whether or not a given integer is prime or composite. The algorithm is based on Fermat's Little Theorem which is given in section 8.2.3.[2]

Thus given an integer, n, we can easily check whether n is prime or composite. If n is composite then we know that we can write $n = p \times q$ for some integers p and q both greater than 1. However, finding these integers (factors of n) is a very hard problem known as the *factorisation problem*.The fastest factorisation algorithm at the moment is called the *number field sieve*, but even this algorithm is not all that efficient. To find the factors of a composite number n which is the product of two large primes and has about 640 binary bits (approximately 200 decimal digits) is at present an impossible task – even if you could use all of the computing power in the world! The difficulty of solving the factorisation problem is the basis of the security of the RSA public key cryptosystem.

The following box contains a summary of the important facts discussed so far:

Given two integers p and q it is easy to multiply them together to obtain $n = p \times q$.

Given an integer n it is easy to determine whether n is prime or composite.

Given a composite number n is it very hard to find the factors of n (i.e. to find p and q such that $p \times q = n$).

Learning activity

Write a computer program that multiplies together two integers giving as the result a third integer. Next write a computer program that factorises a given integer giving as the result two integers that multiply together to produce the input number.

Use your programs to test the speed with which you can multiply and factorise. You will find that as the numbers start to increase in size, factorisation takes an exponentially long time.

Can you find the two numbers that multiply together to give 7534534523?

How about 553907349221053088878733995771?

[1] There are infinitely many prime numbers. The largest prime number known so far has 7816230 digits. There are big money prizes for finding unknown primes so the hunt for the next prime continues!

[2] Do an Internet search to find a prime number checker. There are many free prime number checkers available and they take only a few seconds to check small (up to 10 digit) integers.

8.2.3 Fermat's Little Theorem

If p is a prime number and a is any integer between 1 and $p-1$ ($1 < a < p-1$), then

$$\boxed{a^{p-1} \bmod p = 1}$$

This is not true in general (i.e. $a^{n-1} \bmod n \neq 1$ in most cases if n is not a prime).[3] This is why we can use Fermat's Little Theorem as the basis for a primality checker. If we want to check whether or not a number n is prime, we choose an integer a which is between 1 and $n-1$ and calculate $a^{n-1} \bmod p$. If the answer is not equal to 1 then n is not prime. On the other hand, if the answer is equal to 1 then n is probably prime. To be sure, we can choose another value of a and repeat the test. If the answer is 1 again then we become more sure that n is a prime number. The number of trials performed (different values of a tested) depends on how crucial it is that we do not get a false positive result (i.e. declare that n is a prime when in fact it is composite). Most composite numbers will fail on the first test but some take two or three. Most prime number generators perform between five and 10 tests with different values of a. Note that this test can only prove that a number is composite. It cannot prove that a number is prime – only that it is probably prime.

Learning activity

Use the fast exponentiation method given in 7.4.3 to compute $2^{46} \bmod 47$ and $2^{50} \bmod 51$. Hence deduce whether the numbers 47 and 51 are prime or composite.

8.2.4 Using Fermat's Little Theorem to solve a problem

In order to help explain the mathematics of RSA, we will first look at a similar mathematical problem that *can* be solved.

Suppose I have:

- a prime number p
- a number m between 2 and $p-1$
- a number e between 2 and $p-1$

and I compute $c = m^e \bmod p$.

For example, if $p = 11$, $m = 5$ and $e = 7$ then $c = 5^7 \bmod 11 = 3$.

The problem is to find m given the values of c, e and p.

We can solve this problem by taking the following steps:

First we find a number d such that $e \times d \bmod (p-1) = 1$.

In our example, it is not hard to see that $d = 3$ because $7 \times 3 = 21 = 1 \bmod 10$. In general, it is quite easy to find the value of d using Euclid's algorithm (see 7.4.5).

[3] A few numbers called *Carmichael numbers* pass the Fermat's little theorem test for all values of a even though they are composite. These numbers are also known as *pseudo-primes*. They are very rare. The first Carmichael number is 561.

Next we compute $m = c^d \bmod p$.

So in our example, $m = 3^3 \bmod 11 = 27 \bmod 11 = 5$.

Why does it work?

This method of solution works because of Fermat's Little Theorem.

First we find d such that $e \times d \bmod (p-1) = 1$.

This means that $e \times d \bmod p = 1$, or equivalently $ed - 1 = k(p-1)$ for some integer value of k.

Re-arranging this we have $ed = k(p-1) + 1$.

Next we computed $c^d \bmod p$.

This is equivalent to:

$$\begin{aligned} c^d &= (m^e)^d \bmod p \\ &= m^{(ed)} \bmod p \\ &= m^{(k(p-1)+1)} \bmod p \\ &= m^{k(p-1)} \times m^1 \bmod p \\ &= (m^{p-1})^k \times m \bmod p \\ &= 1^k \times m \bmod p \\ &= m \end{aligned}$$

The term $(m^{p-1})^k = 1 \bmod p$ because by Fermat's Little Theorem $m^{p-1} = 1 \bmod p$ for all values of m.

However, this only works because p is a prime number. If the modulus is not a prime number then this method does not work since in general $a^{n-1} \neq 1 \bmod n$ if n is not prime.

We could make the method work for a composite number only if we can find a number r such that $a^r = 1 \bmod n$. If a and n are co-prime (i.e. they have no common factors) then there will be such a number r and there is a way to find it. However, in order to do so, you must be able to factorise n.

Finding r

In order to find r such that $a^r = 1 \bmod n$, you have to be able to factorise n into all of its prime factors.

If $n = p \times q$ where p and q are prime numbers, then

$$\boxed{r = (p-1)(q-1)}$$

8.2.5 Mathematical summary

Here is a summary of the mathematics we have covered in this chapter. You will not be asked in the examination to explain this mathematics but you should know the following facts. The word *easy* is used to mean that there is an efficient algorithm available to perform *easy* steps. Likewise, *hard* means that there is no efficient algorithm available to perform *hard* steps.

1. It is easy to determine whether a large number is prime or composite.
2. It is easy to compute the product of two large prime numbers $n = p \times q$.
3. Setting $r = (p-1)(q-1)$ we have $m^r = 1 \bmod n$ for all m co-prime with n.
4. Given e (co-prime with r), it is easy to determine d such that $(e \times d) = 1 \bmod r$.
5. It is easy to compute $m^e \bmod n$.
6. If $c = m^e \bmod n$ then $m = c^d \bmod n$ and it is easy to compute $c^d \bmod n$ if you know d.
7. It is only possible to find d if you can find r and you can only find r if you can factorise n.
8. Factorising n is hard.

The mathematics above is the basis for the RSA public key cryptosystem. The holder of the public key knows p and q and therefore he or she can compute r and hence find the private key which is d and thus decrypt the ciphertext c by computing $m = c^d \bmod n$. No-one else knows p or q so they cannot find r or d and so they cannot recover m.

There is no known way to recover m which is not equivalent to factorising n.

8.3 RSA

We are finally in a position to give the key generation, encryption and decryption algorithms for the RSA public key cryptosystem. We will assume that Alice wants to send a message to Bob. First Bob must generate his public/private key pair.

8.3.1 RSA – key generation

1. Bob uses a cryptographically strong random number generator to generate two large primes p and q.[4]
2. He computes $n = p \times q$.
3. He computes $r = (p-1) \times (q-1)$.
4. Bob chooses a random number e which is inbetween 1 and r ($1 < e < r$) and which has no factor in common with r.
5. He computes the ***private key*** d by solving the equation $(e \times d) = 1 \bmod r$.
6. He carefully disposes of the values of p, q and r.
7. Bob keeps d private, and publishes the value of the pair (e, n). This is his ***public key***.

[4]Current recommendations for the size of n typically range from 640 to 2,048 bits depending on the level of security required. This is approximately 200 to 617 decimal digits. There is a prize of $200,000 for the first person who successfully manages to complete the *RSA challenge* and factorise a given 617 digit number.

8.3.2 RSA – encryption

If Alice wishes to send Bob a message m she takes the following steps:

1. Alice looks up Bob's public key (e, n) in a public key directory.
2. Alice breaks the message m up into blocks $m_1, m_2, ...$ so that each block of the message is a positive value less than n, (i.e. $0 \leq m_i < n$).[5]
3. For each message block m_i, Alice computes the ciphertext $c_i = m_i^e \bmod n$ and sends the value of c_i to Bob.

8.3.3 RSA – decryption

Bob receives Alice's encrypted message $c = c_1, c_2, ...$ He decrypts the message using his private key d. For each ciphertext block c_i Bob computes $m_i = c_i^d \bmod n$.

8.3.4 RSA – an example

Here is an example of the RSA scheme. The prime numbers used are small so that it is easier to see what is happening at each stage.

Key generation

Bob performs the following steps:

1. $p = 7$, $q = 11$
2. $n = p \times q = 7\times = 77$
3. $r = (p-1) \times (q-1) = 6 \times 10 = 60$
4. $e = 13$ (chosen at random)
5. $d = 37$ (computed using Euclid's algorithm – see page 79)
6. Private key $d = 37$, public key $(e, n) = (13, 77)$

Encryption

Alice wants to send Bob the message $m = 5$.

1. Alice looks up Bob's public key $(n, e) = (77, 13)$.
2. There is no need to break this message into blocks because the message value is smaller than the value of n.
3. Alice computes the ciphertext $c = m^e \bmod n = 5^{13} \bmod 77 = 26$ and sends this ciphertext $c = 26$ to Bob.

[5] If $m = 0$ or $m = 1$ then $c = m$. Do you think this could happen in practice?

Decryption

Bob receives Alice's ciphertext $c = 26$ and decrypts it by computing $m = c^d \bmod n = 26^{37} \bmod 77 = 5$ which is the original message.

Note that the algorithm for modular exponentiation can be used to compute $26^{37} \bmod 77$ as follows:

y	u	n
1	26	37
26	60	18
26	58	9
45	53	4
45	37	2
45	60	1
5		

Learning activity

1. Bob is generating his RSA keys. He randomly chooses primes $p = 11$ and $q = 13$. What are the values of n and r?
2. Bob randomly chooses the public key $e = 7$. Show that his private key has the value $d = 103$.
3. Alice wants to send Bob the message $m = 4$. What is the corresponding ciphertext?
4. Bob receives the ciphertext $c = 29$. Use the algorithm for modular exponentiation to find the corresponding message.

8.4 Summary

In this chapter we have studied the mathematics behind the RSA cryptosystem. We have seen how Fermat's Little Theorem can be applied with a non-prime modulus to make a secure public key cryptosystem with the difficulty of factorising a large (200+ decimal digits) composite number as the basis of its security. We have seen the RSA algorithms for key generation, encryption and decryption, and we have used these algorithms to show examples of RSA in action using small prime values.

8.5 Learning outcomes

After studying this chapter and the additional reading, you should be able to:

- recall the key mathematical facts that make RSA easy to implement but hard to break
- give the key generation, encryption and decryption algorithms for RSA
- understand the importance of choosing suitable parameters (large primes) when implementing RSA in a commercial setting so that the cryptosystem cannot be broken by factorisation of the public key
- use the RSA encryption and decryption algorithms (with the modular exponentiation algorithm if necessary) to demonstrate the RSA cryptosystem using small prime parameters.

8.6 Sample examination questions

Question 1

a) Describe the RSA (Rivest, Shamir and Adleman) public key cryptosystem. Your answer should include:

i. the generation of the public and private keys

ii. the encryption algorithm

iii. the decryption algorithm

iv. the basis for the security of RSA.

[12]

b) Bob has public RSA key ($n = 65, e = 5$). Show that Bob's private key is ($d = 29$).

[6]

c) Alice wants to send Bob the message $m = 11$. She encrypts the message using Bob's public key. What is the value of the ciphertext that Alice sends to Bob?

[3]

d) David has also sent an encrypted message to Bob. The ciphertext value that Bob receives from David is $c = 19$. Showing all your working, use Bob's key to decrypt this ciphertext and recover the value of David's message.

[5]

Chapter 9

El Gamal

9.1 Introduction

In this chapter, we will look at another public key cryptosystem which is called El Gamal after its inventor, Tahar Elgamal. The security of the El Gamal cryptosystem is based upon the difficulty of solving the *discrete log problem*. Before looking at El Gamal, we will study a key exchange protocol called *Diffie-Hellman*.The paper *New Directions in Cryptography* by Whit Diffie and Martin Hellman, published in 1976, describes how two people can derive the same secret key without ever having to actually send the key, or any information that could lead to an interceptor being able to retrieve the secret key. This is the basis of the Diffie-Hellman key exchange protocol and was the first example of public key cryptography. The security of Diffie-Hellman is also based upon the difficulty of solving a discrete logarithm problem.

Supplementary reading

As with RSA, most books on cryptography will include some details on El Gamal and Diffie-Hellman. However, none of the recommended textbooks do so in any great detail. Students who wish to know more about the schemes and their uses should do an internet search. The paper *New Directions in Cryptography* by Whit Diffie and Martin Hellman can be found at
`http://www.cs.jhu.edu/$\sim$rubin/courses/sp03/papers/diffie.hellman.pdf`.

After studying this chapter and the additional reading, you should be able to:

- state the discrete logarithm problem and understand that it is a one-way problem
- understand what is meant by saying that g is a *generator* for prime number p
- describe the Diffie-Hellman key exchange protocol and demonstrate the protocol using small numbers
- give the key generation, encryption and decryption algorithms for the El Gamal public key cryptosystem and demonstrate the protocol using small numbers.

9.2 The discrete logarithm problem

Both the El Gamal public key cryptosystem and the Diffie-Hellman key exchange protocol are based upon the difficulty of solving the *discrete logarithm problem* (DLP) which is stated as follows:

Given a prime number p and values g and y, find x such that $y = g^x \bmod p$

For a small value of p it is easy to solve a DLP using trial and error or exhaustive search.

For example, given $p = 11$, $g = 2$ and $y = 9$ we can try different values of x until we find the value that makes $2^x \bmod 11 = 9$ as in Table 9.1.

x	$2^x \bmod 11$
1	2
2	4
3	8
4	5
5	10
6	9

Table 9.1: Solving a small DLP

From Table 9.1 we can see that $2^6 \bmod 11 = 9$ so the solution to the DLP is $x = 6$.

However, for large values of p, for example when p has 100 decimal digits or more, it is not possible to solve a DLP using current technology. The best algorithm currently in use for solving discrete logarithm problems is called the *index calculus method* but even this method would not be able to solve a discrete logarithm problem (with parameters p, g and y chosen to be cryptographically strong) within our life time.

The discrete logarithm problem is an example of a *one-way function* because given p, g and x, it is easy and fast to compute $y = g^x \bmod p$. However, given p, g and y it is very hard to compute x such that $y = g^x \bmod p$.

9.2.1 Finding a generator g

When using the discrete logarithm problem in a cryptographic protocol, it is important that the value of g chosen is a *generator* for the prime. This means that $g^n \bmod p$ is equal to a different value for every different value of n between 1 and $p - 1$. For example, Table 9.2 shows that 2 is not a generator for prime 17 because $2^8 \bmod 17 = 2^{16} \bmod 17 = 1$. However, 3 is a generator for prime 17 because every value of n between 1 and 16 gives a different value for $3^n \bmod 17$.

Learning activity

Show that 2 and 3 are both generators for prime 19 but that 5 is not a generator for prime 19.

9.3 The Diffie-Hellman key exchange protocol

The Diffie-Hellman key exchange protocol allows two people to use secret random values and yet each generate the same symmetric key without ever transmitting the value of the key. We will discuss other key exchange algorithms in the next chapter, but will briefly consider Diffie-Hellman here because it is a very good example of how the discrete logarithm problem is used in cryptography.

n	$2^n \bmod 17$	$3^n \bmod 17$
1	2	3
2	4	9
3	8	10
4	16	13
5	15	5
6	13	15
7	9	11
8	1	16
9	2	14
10	4	8
11	8	7
12	16	4
13	15	12
14	13	2
15	9	6
16	1	1

Table 9.2: Finding a generator for prime 17

Suppose Alice and Bob need to agree on a key to use in a symmetric key cryptosystem. They choose a large prime number p and a generator g. The values of p and g are not secret and can be openly transmitted and shared between Alice and Bob.

Then Alice and Bob simultaneously take the following steps:

	Alice		Bob
1.	Generates a random number a between 2 and $p-2$.	1.	Generates a random number b between 2 and $p-2$.
2.	Computes $x = g^a \bmod p$.	2.	Computes $y = g^b \bmod p$.
3.	Sends x to Bob.	3.	Sends y to Alice.
4.	Receives y from Bob.	4.	Receives x from Alice.
5.	Computes $k = y^a \bmod p$.	5.	Computes $k = x^b \bmod p$.

Now Alice and Bob both know the value of the same secret key k although they have never transmitted the values of a, b or k.

Diffie-Hellman – an example

Suppose Alice and Bob have agreed to use prime $p = 17$ and generator $g = 3$. They take the following steps to obtain a shared secret key.

	Alice		Bob
1.	Generates random number $a = 6$.	1.	Generates random number $b = 11$.
2.	Computes $x = 3^6 \bmod 17 = 15$.	2.	Computes $y = 3^{11} \bmod 17 = 7$.
3.	Sends $x = 15$ to Bob.	3.	Sends $y = 7$ to Alice.
4.	Receives $y = 7$ from Bob.	4.	Receives $x = 15$ from Alice.
5.	Computes $k = 7^6 \bmod 17 = 9$.	5.	Computes $k = 15^{11} \bmod 17 = 9$.

The protocol works because Alice has computed:

$$\begin{aligned} k &= y^a \bmod p \\ &= (g^b)^a \bmod p \\ &= g^{ba} \bmod p \\ &= g^{ab} \bmod p \end{aligned}$$

and Bob has computed:

$$\begin{aligned} k &= x^b \bmod p \\ &= (g^a)^b \bmod p \\ &= g^{ab} \bmod p \end{aligned}$$

If a cryptanalyst Charles is eavesdropping on the transmissions between Alice and Bob, then we must assume that Charles knows the values of p, g, x and y. However, he cannot find the values of either a or b without solving a discrete logarithm problem. It is believed to be just as hard to find k given the values of p g, x and y as it is so solve a discrete logarithm problem.

Learning activity

Alice and Bob want to use the Diffie-Hellman key exchange protocol to exchange a secret key. They agree to use prime number $p = 23$ and generator $g = 5$. Alice chooses secret number $a = 9$ and Bob chooses secret number $b = 6$. What is the value of their shared secret key?

Although it is necessary for a cryptanalyst to solve a discrete logarithm problem in order to find the secret key k or the values of a or b, the Diffie-Hellman key exchange protocol is very vulnerable to a *man-in-the-middle* attack and therefore it should not be used (in the form described above) to exchange a secret key for cryptographic purposes.

9.3.1 Diffie-Hellman and the man-in-the-middle attack

Suppose Charles can intercept all communications between Alice and Bob who are using the Diffie-Hellman key exchange protocol as described above to establish a

shared secret key. Then Charles could corrupt the protocol and get the secret key himself by performing a man-in-the-middle attack.

The simplest form of attack is for Charles to intercept both x and y and change their values to equal 1 before sending them on to Bob and Alice respectively. This means that whatever the values of a and b, the value of the secret key k will be equal to 1. To counter this attack, Alice and Bob should always ensure that the values of y and x which they receive are not equal to 1.

A more clever attack is as follows and leads to Charles sharing a secret key with Alice and a second secret key with Bob. Alice and Bob think that they are communicating with each other when in fact they are communicating with Charles.

Alice	Charles	Bob
Chooses a and computes $x = g^a \bmod p$.	Chooses c and computes $z = g^c \bmod p$.	Chooses b and computes $y = g^b \bmod p$.
Sends x to Bob.	Intercepts x, replaces it with z and sends it to Bob. Intercepts y, replaces it with z and sends it to Alice.	Sends y to Alice.
Receives z from Charles (but thinks it is y from Bob).		Receives z from Charles (but thinks it is x from Alice).
Computes $k_A = z^a \bmod p$.	Computes $k_A = x^c \bmod p$. Computes $k_B = y^c \bmod p$.	Computes $k_B = z^b \bmod p$.

Now Charles shares key k_A with Alice and key k_B with Bob. If Alice and Bob do not realise that Charles has modified their messages to each other then they will each believe that they are communicating with, and sharing a secret key with, each other.

If Alice wants to send an encrypted message to Bob, she will encrypt the message using key k_A. Charles can intercept this ciphertext from Alice and decrypt it using key k_A. He can then modify the message, encrypt it using key k_B and send the resulting ciphertext to Bob. Bob will decrypt this using key k_B and think that the message has come from Alice.

9.4 El Gamal

Now that we have seen how the discrete logarithm problem can be used to provide the security in a cryptographic scheme, we are ready to describe the key generation, encryption and decryption protocols for the El Gamal public key cryptosystem.

9.4.1 El Gamal – key generation

Alice generates public and private El Gamal keys as follows:

1. Alice chooses a large random prime number p. (p should be approximately 100 decimal digits or more to ensure security with current technology.)
2. She finds a generator g for the prime p.

3. She picks a random number a which is between 2 and $p-2$. This will be Alice's private key.[1]
4. She computes $y = g^a \bmod p$.

Alice publishes the public key (p, g, y) and keeps the private key (p, g, a) a secret.

9.4.2 El Gamal – encryption

If Bob wants to send Alice a message encrypted using the El Gamal cryptosystem he will first look up Alice's public key (p, g, y). Next Bob must break the message, m, up into blocks $m_1, m_2, ...$ with each message block m_i having a value which is less than p. For each message block m_i Bob performs the following steps:

1. He generates a random number k which is between 2 and $p-2$.
2. He computes r, x and c where:

$$r = g^k \bmod p$$
$$x = y^k \bmod p$$
$$c = (m_i \times x)$$

3. Bob sends Alice the values (r, c) and carefully discards the values of k and x.

9.4.3 El Gamal – decryption

Alice receives the ciphertext pair (r, c) from Bob. She decrypts the ciphertext using her private key (p, g, a) as follows:

1. She computes $x = r^a \bmod p$.
2. She finds the inverse of $x \bmod p = x^{-1}$.
3. She computes $m_i = (c \times x^{-1}) \bmod p$.

The decryption protocol works because Alice can use her private key a to recover the value of x without knowing the value of k. This is because
$r^a \bmod p = (g^k)^a \bmod p = (g^a)^k \bmod p = y^k \bmod p = x$.

Only Alice knows her private key a and only Bob knows the random key k hence only Alice is able to recover the value of x once k has been discarded, and so only Alice can decrypt the ciphertext.

A cryptanalyst might know the values of p, g, y, r and c, but they would have to solve the discrete logarithm problem of finding a such that $y = g^a \bmod p$ in order to find the private key; or solve the discrete logarithm problem of finding k such that $r = g^k \bmod p$ in order to decrypt the ciphertext without knowing the private key.

[1] See Fermat's Little Theorem (section 8.2.3) to see why $a = p - 1$ is not acceptable.

9.4.4 El Gamal – an example

We will use prime $p = 53$ and generator $g = 2$. The secret key $a = 10$. The public key y is computed by calculating $y = 2^{10} \bmod 53 = 17$.

Hence the keys are:

$$\text{public key: } (p = 53, g = 2, y = 17)$$
$$\text{private key: } (p = 53, g = 2, a = 10)$$

We will encrypt the message $m = 39$.

First we choose a random number $k = 42$, and then we compute r, x and c as follows:

$$r = g^k \bmod p = 2^{42} \bmod 53 = 25$$
$$x = y^k \bmod p = 17^{42} \bmod 53 = 16$$
$$c = (m \times x) \bmod p = 39 \times 16 \bmod 53 = 41$$

Note that the exponentiations $2^{42} \bmod 53$ and $17^{42} \bmod 53$ are too large to do on a calculator so we use the fast modular exponentiation algorithm given on page 77 to calculate the values of r and x.

We send the ciphertext ($r = 25, c = 41$) and discard the values of k and x.

To decrypt the ciphertext, the receiver uses the private key (p, g, a) and computes:

$$x = r^a \bmod p = 25^{10} \bmod 53 = 16$$

Next it is necessary to find the inverse of 16 mod 53. This is equal to 10 because $10 \times 16 = 160 = 1 + (3 \times 53) = 1 \bmod 53$.

Hence $m = (c \times x^{-1}) \bmod p = (41 \times 10) \bmod p = 39$.

Learning activity

Given the El Gamal keys:

$$\text{public } (p = 53, g = 2, y = 17)$$
$$\text{private } (p = 53, g = 2, a = 10)$$

1. Use the public key and random value $k = 6$ to encrypt the message $m = 25$.
2. Use the private key to decrypt the ciphertext ($r = 6, c = 33$).

9.5 Comparison of RSA and El Gamal

The RSA and El Gamal public key encryption schemes are both widely used in modern day cryptographic applications. Public key cryptography is generally much

slower and therefore more expensive to use than symmetric key cryptosystems. Public key cryptosystems are therefore generally used to exchange secret keys for use in symmetric schemes, or to generate digital signatures.

As we have seen RSA is based upon the difficulty of factorising a large number, and El Gamal is based upon the difficulty of solving a discrete logarithm problem. If in future years, someone developed an algorithm that could be used to factorise large numbers easily, RSA would no longer be secure but El Gamal could still be used. Conversely, if an algorithm for solving the discrete logarithm problem was invented, then El Gamal would no longer be secure, but RSA could still be used.

RSA has an advantage over El Gamal in that an RSA ciphertext is a single value c which is roughly the same size as the message m. For El Gamal, the ciphertext is a pair (r, c), and so is approximately twice the size of the original message.

For RSA, the encryption and decryption algorithms are the same (modular exponentiation). This means that encryption and decryption takes the same amount of time and a cryptanalyst may not be able to tell whether messages are being encrypted or decrypted. The same technological developments will be applicable to both the encryption and decryption algorithms. On the other hand, the encryption and decryption algorithms for El Gamal are different, although they both take approximately the same amount of time to perform.

Since the encryption and decryption algorithms are the same for RSA, it is very straightforward to produce a digital signature using RSA. The message (or more likely a hash of the message) is simply encrypted using the signer's private key, and then decrypted using the signer's public key. There is a digital signature scheme called DSS which is based upon El Gamal, but it is not quite so straightforward due to the fact that the encryption and decryption algorithms are not the same for El Gamal.

Until September 2000, RSA was covered by a US patent and therefore was not available until recently to use for free. El Gamal has never been patented and so has always been available for free use.

The prime and generator used in El Gamal keys can be shared between a group of people without compromising the security. Furthermore El Gamal can be used in a commutative manner, which means that if Alice and Bob have El Gamal keys with the same p and g, then the result of Alice encrypting a message, followed by Bob encrypting the resulting ciphertext, is the same as Bob encrypting the message, followed by Alice encrypting the resulting ciphertext.[2] This means that El Gamal can be used to implement games such as mental poker which can be played securely over the Internet with no player being able to cheat. It is not possible to use RSA in this way because the modulus n cannot be shared.

A further advantage of the El Gamal cryptosystem, is that it can be generalised to a cryptosystem based on the *discrete logarithm problem for elliptic curves*. This appears to be even harder to solve and so the prime number used can be smaller and therefore the encryption process is faster. Elliptic curve cryptography is beyond the scope of this subject, but students who are interested should refer to *Elliptic Curves in Cryptography* by Blake, Seroussi and Smart [ISBN 0-521-65374-6] or do an internet search for further information.

[2]This is analogous to Alice and Bob each putting their own padlocks onto a briefcase. It does not matter which order the padlocks are put onto the briefcase, and it does not matter in which order they are removed.

9.6 Summary

In this chapter we have studied the discrete logarithm problem, and seen how the Diffie-Hellman protocol uses the discrete logarithm problem to enable users to share a secret key without having to transmit the key. We have looked at the El Gamal public key cryptosystem, the security of which is also based upon the difficulty of solving a discrete logarithm problem. We have implemented both the Diffie-Hellman and El Gamal algorithms using small prime parameters.

9.7 Learning outcomes

After studying this chapter and the additional reading, you should be able to:

- state the discrete logarithm problem and understand that it is a one-way problem
- understand what is meant by saying that g is a *generator* for prime number p
- describe the Diffie-Hellman key exchange protocol and demonstrate the protocol using small numbers
- give the key generation, encryption and decryption algorithms for the El Gamal public key cryptosystem and demonstrate the protocol using small numbers.

9.8 Sample examination questions

Question 1

a) State the discrete logarithm problem.

[2]

b) Describe how the discrete logarithm problem is used in the El Gamal public key cryptosystem. Your description should include the generation of keys, the encryption algorithm, the decryption algorithm and a justification for the security of the system.

[10]

c) What are the advantages and disadvantages of the El Gamal public key cryptosystem when compared with the RSA public key cryptosystem?

[4]

d) Describe how the discrete logarithm problem is used in the Diffie-Hellman key exchange protocol.

[4]

e) Showing all your working, illustrate the Diffie-Hellman protocol by determining the key in the case where the prime $p = 17$, the generator $g = 3$, Alice's random number is 5 and Bob's random number is 10.

[6]

Chapter 10

Key management

10.1 Introduction

In this chapter, we will consider some of the problems that arise when generating, storing, transmitting and using keys in cryptosystems. Public key cryptosystems and symmetric key cryptosystems each have their own problems. We will see how some of these problems can be addressed.

Supplementary reading

Network Security, by William Stallings covers Key Distribution, X.509 and the web of trust.

Practical Cryptography by Niels Ferguson and Bruce Schneier contains a discussion on the practical aspects of key management. This book talks about the Kerberos key management system which is based upon the Needham-Schroeder protocol.

Cryptography: a very short introduction by Fred Piper and Sean Murphy contains an easy-to-read chapter about key management.

10.2 Key management

We have seen in the previous chapters that users of a symmetric key cryptosystem need to share encryption and decryption keys; whereas users of an asymmetric key cryptosystem generate public keys which are available to all others users and private keys which are never shared.

In a symmetric cryptosystem, the encryption algorithm and decryption algorithm are known publicly; so the security of the cryptosystem is embodied in the values of the keys. This type of cryptosystem is called *symmetric* because the encryption and decryption keys are the same, or can be derived easily from each other. The keys must be kept secret. If either party reveals their key then the cryptosystem is compromised.

One advantage of public key cryptosystems over symmetric key cryptosystems, is that only the private key needs to be kept secret, and the owner of the key never needs to transmit or share their private key with anyone else. Therefore, the private key owner does not need to trust anyone else not to reveal their key. However, there are other issues of trust involved when using a public key cryptosystem which we will discuss in section 10.4.

10.2.1 Number of keys

A second advantage of public key cryptosystems when compared with symmetric key cryptosystems is that the private key holder only needs to store one key (their own private key). The corresponding public key is available for all other users and is stored in a public key directory.

In comparison, the numbers of keys each person requires for a symmetric key cryptosystem grows as the number of users grows.

If two people, Alice and Bob, want to communicate with each other using a symmetric key cryptosystem, they need to share one key, K_{AB}, between them.

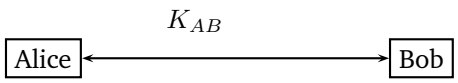


Figure 10.1: Two users and one shared key

If a third person, Carol, also wants to communicate with Bob and Alice, then she must share a key, K_{AC}, with Alice and a key, K_{BC} with Bob.

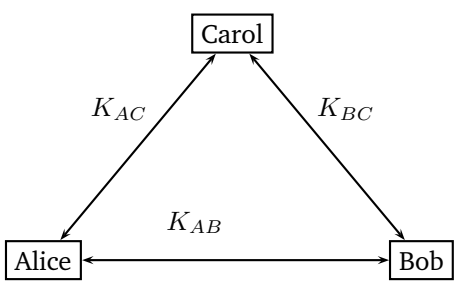


Figure 10.2: Three users and two shared keys

If ten people want to communicate then each would need to store nine keys, and there are $\frac{(10\times 9)}{2} = 45$ keys altogether. This is illustrated in Figure 10.3.

In general, if n people are using a symmetric key cryptosystem to communicate then the total number of keys required is:

$$\frac{n \times (n-1)}{2}$$

Learning activity

Suppose one hundred people want to communicate with each other using a symmetric key cryptosystem. How many keys does each person have to store? How many keys are there altogether?

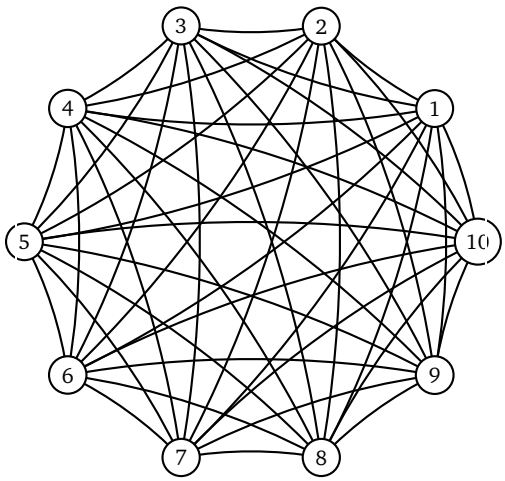


Figure 10.3: Ten users and 45 shared keys

10.2.2 Symmetric key management issues

With so many keys required for a symmetric key cryptosystem, there are a number of key management issues to consider.

- Key generation – how and by whom are the keys generated?
- Key storage – how are the keys stored?
- Key distribution – how are the keys delivered safely to the appropriate users?
- Key replacement – how often should the keys be replaced?

Suppose Alice and Bob want to communicate using a symmetric key cryptosystem. Two possible solutions to the issues raised above are as follows.

Cryptosystem users generate their own keys

Alice may generate the initial encryption key using a cryptographically strong random number generator. She then determines the corresponding decryption key and sends it to Bob using some *covert channel* where security is guaranteed – for example, she might physically deliver the key to Bob.

Alice and Bob store their keys on their encryption and decryption devices which are physically secured. They use their keys to encrypt and decrypt messages which are transmitted over an insecure channel. (i.e. the channel may be susceptible to eavesdroppers but since the messages are encrypted this is not a security threat so long as the keys are not compromised).

At the end of the key's life, Alice generates another encryption key using her random number generator and determines Bob's corresponding decryption key. Alice now has two options. She could either send Bob the new key, encrypted using the old key,

over the insecure communication channel; or, she can deliver the key to Bob using the same covert channel as she used to deliver the first key.

This first method, encrypting the new key using the old key, is called *key chaining*. Although this method is very efficient, it is not very secure, because if a cryptanalyst discovers one key, he will be able to decrypt the next key and all subsequent keys used. This somewhat defeats the point of giving keys a short session life.

The second method is obviously more secure, but also more time-consuming and costly. However, there may be a covert channel in place which can be used on a regular basis to exchange keys. For example, a bank might use an armoured van to transfer money and new cryptosystem keys to each of its branches every morning.

Using a trusted third party (TTP)

Instead of Alice generating her own cryptosystem keys, she may rely on a trusted third party to do this for her and deliver the keys securely to herself and Bob. The Needham-Schroeder protocol is a key exchange protocol which uses a TTP. New session keys are generated each time the protocol is invoked (see section 10.3.2).

Random number generators

There are sources of truly random data such as tossing an unbiased coin, but it is not practical to use such random sources in a computer program. The alternative is to use *pseudorandom numbers* which are generated by a pseudorandom number generator or PNG. The PNG takes a *seed* as input and then generates a number using a function f. If the seed is x_0 then the sequence of pseudorandom numbers generated is the sequence $x_1, x_2, x_3, \ldots$, where $f(x_n) = x_{n+1}$ for all $n > 0$. This means that if an attacker finds part of the sequence, he or she may be able to predict the next part. It is therefore essential that if you are using such a function to generate keys, that the function be kept secret.

Most computers and calculators have an inbuilt PNG. Usually these PNGs are designed to produce output that looks random and they are suitable for use in sampling methods and other branches of mathematics. However, they should not be used in a cryptographic protocol. A PNG is called *cryptographically strong* if an attacker cannot predict the next number, however much previous data output he or she sees.

10.3 Key exchange protocols

10.3.1 Using asymmetric keys to exchange symmetric keys

Suppose that Alice and Bob both have public and private keys to use in an asymmetric cryptosystem such as RSA or El Gamal. They could simply use this asymmetric cryptosystem to securely encrypt and transmit all of their messages, but this would be costly and time-consuming. Instead they can use the asymmetric cryptosystem to encrypt and transmit a key, K_{AB}, for use with a symmetric key cryptosystem as follows:

- Alice generates a symmetric key K_{AB} using a cryptographically strong PNG.
- Alice looks up Bob's public key and encrypts K_{AB}.

$$c = encrypt_{Bpub}(K_{AB})$$

- Alice sends the ciphertext c to Bob who decrypts it using his private key to recover K_{AB}.

$$K_{AB} = decrypt_{Bpriv}(c)$$

- Now Alice and Bob both know the value of K_{AB} and they can use this as the key for a symmetric key cryptosystem.

Diffie-Hellman key exchange protocol

We studied the Diffie-Hellman key exchange protocol in section 9.3. This protocol can be used to exchange a shared secret key, although care must be taken to ensure that the protocol is not compromised by a man-in-the-middle attack.

10.3.2 Needham-Schroeder protocol

The Needham-Schroeder protocol is another protocol for exchanging keys between Alice and Bob. This time they only use symmetric key cryptography. But they need to use a trusted third party (TTP) or server (S). Alice and the server share a secret key K_{AS}. Bob and the server share a secret key K_{BS}. Alice and Bob want to establish a shared key K_{AB} so that Alice can send Bob an encrypted message. They communicate with each other and the server as follows:

1. Alice sends the server the names of Alice and Bob to request that a session key be generated.
2. The server sends to Alice the following three items all encrypted using K_{AS}:
   (a) The name of Bob.
   (b) A session key for Alice and Bob to share.
   (c) The name of Alice and the session key both encrypted using K_{BS}.
3. Alice uses her key K_{AS} to decrypt the three items sent to her in step 2. Alice now knows the session key K_{AB}.
4. Alice sends Bob the value of 2c) which is the name of Alice and the session key K_{AB} encrypted with K_{BS}.
5. Bob decrypts the name of Alice and the session key using his key K_{BS}. Now Bob knows the session key K_{AB} which he uses to communicate with Alice.

The Needham-Schroeder protocol may be explained symbolically as follows:

1. $A \longrightarrow S : A, B$
2. $S \longrightarrow A : e_{KAS}(B, K_{AB}, e_{KBS}(A, K_{AB}))$
3. Alice decrypts to get $B, K_{AB}, e_{KBS}(A, K_{AB})$
4. $A \longrightarrow B : e_{KBS}(A, K_{AB})$
5. Bob decrypts to get A, K_{AB}

A cryptanalyst, Charles, who is eavesdropping on the communications between Alice, Bob and the Server cannot determine the keys since they are encrypted.

Charles can masquerade as Alice to the Server but he will not be able to decrypt the ciphertext encrypted with K_{AS} so this will not help him. He could prevent the messages sent in steps 1 and 2 from reaching their destinations, but this will alert Alice because she will be expecting a response from the Server. Charles cannot masquerade as the Server to Alice because he does not know the key K_{AS}.

However it may be possible, with the protocol as above, for Charles to prevent the message sent from Alice to Bob reaching its destination. To counter this, the protocol adds an extra step whereby Bob uses the key K_{AB} to send Alice a message signifying that he has received the key.

There is also the possibility that Charles can fake the response in step 2 by sending Alice a replay from a former run of the protocol. To avoid this, each step involves a *nonce*, N. A nonce is a *number used once* and includes some extra information such as a time-stamp and a unique identifier such as a random number.

Including the use of nonces, the full Needham-Schroeder protocol is as follows.

1. $A \longrightarrow S : A, B, N_A$
2. $S \longrightarrow A : e_{KAS}(B, N_A, K_{AB}, e_{KBS}(A, K_{AB}))$
3. Alice decrypts to get $B, N_A, K_{AB}, e_{KBS}(A, K_{AB})$
4. $A \longrightarrow B : e_{KBS}(A, K_{AB})$
5. Bob decrypts to get A, K_{AB}
6. $B \longrightarrow A : e_{KAB}(N_B)$
7. $A \longrightarrow B : e_{KAB}(N_B - 1)$

The final two steps in this protocol are a *hand shake* which reassures Alice and Bob that they are communicating with each other as expected.

Since a new session key is generated each time Alice invokes the Needham-Schroeder protocol, there is no need for her to store a large set of keys for use with all the different cryptosystem users. She need only store the key K_{AS} which she shares with the Server.

10.4 Trusting public keys

We have seen that a public key may be used to encrypt a session key for use with a symmetric key cryptosystem. This solves the problem of securely transmitting symmetric keys. However, the use of public keys also poses a security question. Namely, if Alice uses Bob's public key to encrypt session key K_{AB}, how can Alice be sure that she is genuinely using Bob's public key? What happens if Charles has placed his own public key in the public key directory under Bob's name?

In general, when using a public key cryptosystem for any purpose, there has to be a reliable way of ensuring that identities are correctly and incorruptibly linked with cryptographic keys.

One way to solve this is to use *certificates* and another method is to use a *web of trust*.

10.4.1 Certificates

A certificate consists of a public key together with an identification of the key holder. The certificate is issued by a trusted third party (TTP) called a *certification agency* or CA. The certification agency may be a government agency or financial institution for example.

The certification agency guarantees the link between the key holder and the public key by digitally signing a document which contains the user name, the public key, the name of the CA, the expiry date of the certificate and perhaps some other information such as access rights.

Below is the standard protocol for issuing a public key certificate. This is known as the **X.509 standard** and is used in most network security applications including IP security.

- Bob generates a document containing his relevant information and presents himself with this document at the CA.
- The CA confirms Bob's identity.
- The CA hashes the document using a cryptographically secure hash algorithm, and encrypts the resulting message digest using their own private key.
- The encrypted message digest is the certificate and it is published together with the unencrypted document including the public key in a public key directory.

If Alice wants to communicate with Bob she looks up his public key document and certificate. She takes the following steps to ensure that the public key is genuine and belongs to Bob.

- Alice uses the public key of the CA to decrypt the certificate.
- She uses the same hash algorithm as the CA to hash the document.
- She checks to see whether the hash of the document is exactly equal to the decrypted certificate. If these two are the same then Alice knows that she can safely communicate with Bob using the public key because the CA has verified his identity.

10.4.2 Web of trust

One problem with using the X.509 protocol to certify keys is that there must be a trusted third party involved. This means that not only do the cryptosystem users have to find such a trusted third party, but also that they will have to pay for their services. Furthermore, if the trusted third party turns out to be corrupt, then all keys are compromised. An alternative approach to key certification is to let users of the cryptosystem certify each others' keys. This method is called the *Web of trust* and is used to allow PGP users to verify keys.

Suppose Alice wants to use the PGP (Pretty Good Privacy) email security protocol (see Chapter 11). She will have to build up a *public key-ring* containing the public keys of other users. Alice's key-ring may contain a public key attributed to Bob, but really belonging to Charles. Maybe Alice got the key from a bulletin board that was used by Bob to post his public key, but which has been compromised by Charles.

There are a number of possible approaches for minimising the risk that a key-ring contains false public keys.

1. Alice can physically get Bob's public key from Bob. This is a very secure method but has obvious physical limitations.
2. Alice can verify the key by telephone. If Alice knows Bob's voice then she can get him to dictate the public key, or a hash of it, over the phone.
3. Alice could get the key from someone else whom she trusts. If Alice knows (and trusts) David, and David knows Bob's public key then he can vouch for it with Alice. To indicate this David signs Bob's public key with his own private key.
4. Alice can obtain Bob's public key from a trusted certification agency.

PGP associates a level of trust with each public key as follows:

- When Alice inserts a new public key onto her public key-ring, she can specify whether this user is *unknown, untrusted, marginally trusted* or *completely trusted*.
- The public key may have one or more signatures[1] attached to it. A *legitimacy value* is given to the key depending on who has signed it, and how much Alice trusts these signatories. For example, a key might need to be signed by one completely trusted person, or two marginally trusted people, or ten untrusted people before it is given a legitimacy value that shows that Alice trusts the key.

The diagram in Figure 10.4 shows a public key-ring and illustrates how signature trust and key legitimacy are related. Each node represents a key, and the diagram shows the different levels of trust associated with each key. The owner of the key-ring is 'You' and the top node is the public key of the key-ring owner. Thanks are due to Phil Zimmermann – the creator of PGP – who gave me permission to use this diagram.

In this example, the owner has specified that he always trusts users D, E, F and L, to sign other keys. The owner partially trusts users A and B to sign other keys. The shading of the nodes indicates the level of trust assigned by this key-ring owner. A dot in the middle of a key shows that the key is considered to be legitimate.

The tree structure indicates which keys other users have signed. An arrow pointing from one key to another shows that the first key has been signed by the second. For example, key H has been signed by key A. An arrow leads to a question mark if a key has been signed by someone whose own key is not in the key ring, indicating that the signatory is unknown to the key-ring owner. For example, key R has been signed by N and also by two unknown signatories.

The key-ring owner has signed nearly all of the keys that he fully or partially trusts. In practice, most users will sign the keys of other users that they trust.

We assume that two partially trusted signatures are sufficient to certify a key. Hence the key of user H is deemed legitimate by PGP because it has been signed by A and B, both of whom are partially trusted.

A key may be determined to be legitimate, but its owner may not be trusted to sign other keys. For example, N's key is legitimate because it is signed by E who is trusted, but N is not trusted to sign other keys. The key-ring owner trusts that he has the correct public key for N but does not actually place any trust in N. Therefore R's key is not considered to be legitimate, even though it has been signed by N.

[1] Every pubic key will have at least one signature because every user should sign their own key.

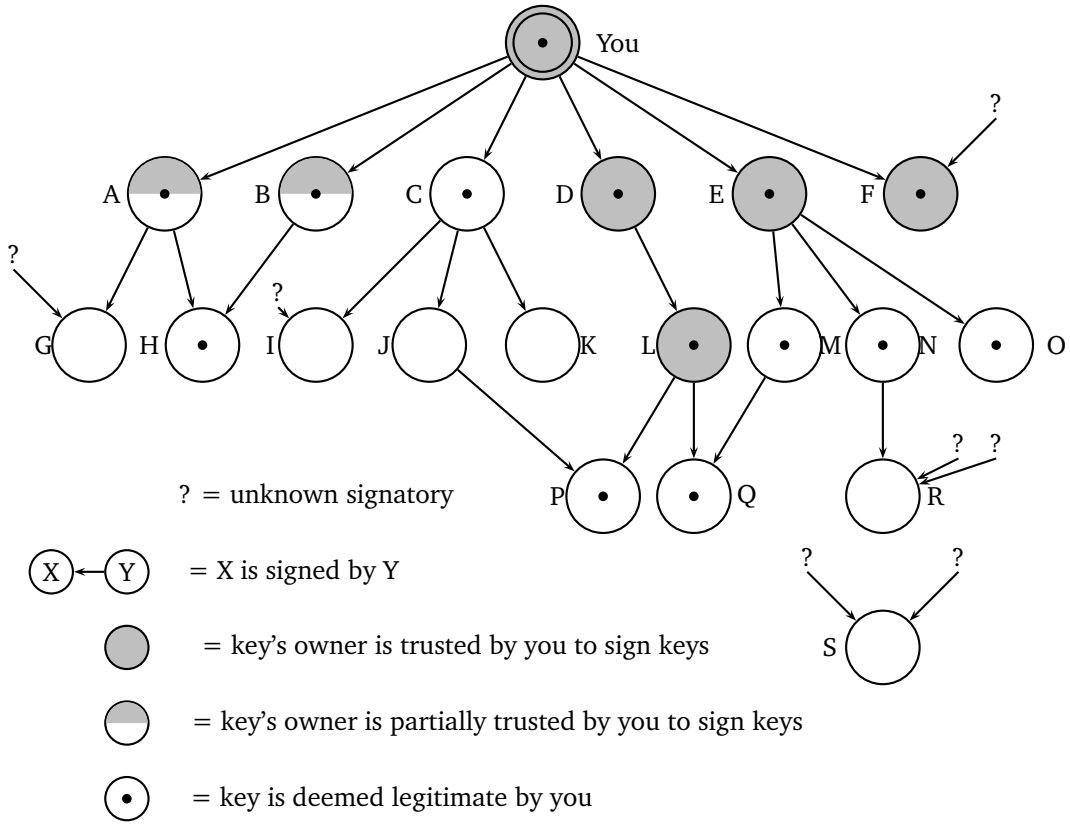


Figure 10.4: PGP trust model example

Node S is detached from the tree with two unknown signatures. Such a key may have been acquired from a key server. PGP cannot assume that this key is legitimate unless the key-ring owner decides that it is and either signs the key, or declares that he is willing to trust one of key S's signatories.

Learning activity

The diagram in Figure 10.5 shows a public key-ring. An arrow from one key to another indicates that the second key has been signed by the owner of the first key.

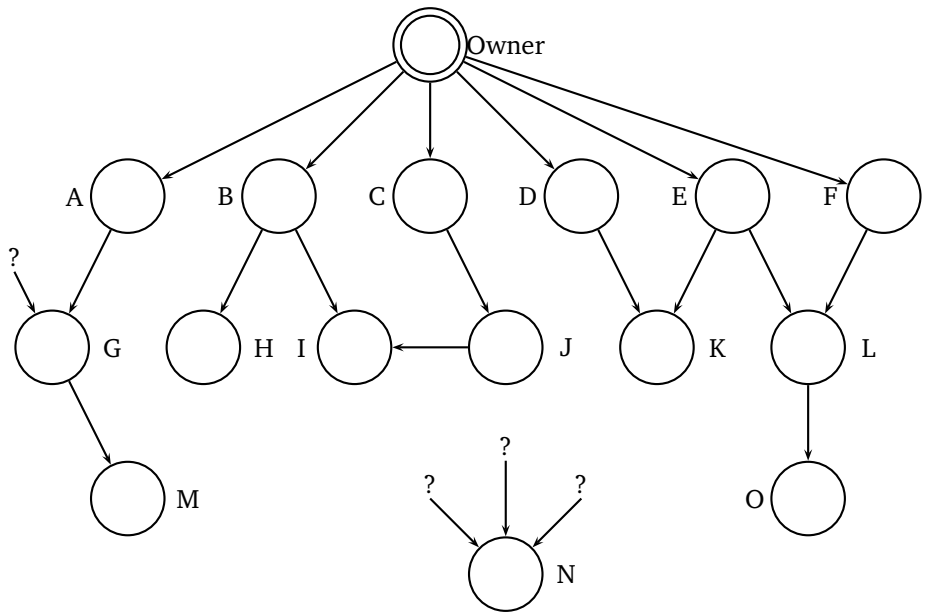


Figure 10.5: An instance of a public key-ring

1. The key-ring owner fully trusts users A,C and D and partially trusts users B,E and F. Show the level of trust associated with each key by fully shading and partially shading the fully trusted and partially trusted key respectively.
2. A key must be signed by one fully trusted user or two partially trusted users to be declared legitimate. Mark the keys that are considered legitimate.

There are websites which try to find a path between two given PGP keys. Try the wotsap site (Web of Trust Statistics and Pathfinder) at
`http://www.lysator.liu.se/~jc/wotsap/`

10.5 Key escrow

Key escrow is a bit like leaving a spare house key with a neighbour in case you lose your key. It allows a secret key to be recovered in the case of an emergency. A key escrow protocol allows two or more people to each hold a piece of a key. Any one piece of the key is of no use and reveals no information about the key, but when enough key pieces are joined together, the key can be reassembled.

A ***t of n key escrow protocol*** is when the key is split into n pieces. Any t of the n pieces ($1 \leq t \leq n$) can be used to recover the key. Any set of less than t key pieces should not reveal any information about the key.

10.5.1 2 of 2 key escrow protocol

In a 2 of 2 key escrow protocol, the key is split into two pieces, both of which are needed to recover the original key. The protocol is very simple and uses the binary XOR function to hide the original key.

- Assume that the key K is a b bit binary number. $K = k_1k_2k_3 \ldots k_b$.
- The first key piece X_1 is a b-bit string chosen at random.
- The second key piece X_2 is computed by XORing K and X_1. So $X_2 = K \oplus X_1$.
- The key K is recovered by XORing the two key pieces together because $K = X_1 \oplus X_2$.

This works because the inverse of XOR is XOR. Neither X_1 or X_2 reveal any information about the original key K on their own since they are both random bit strings.

For example, suppose the key K is the 8 bit random string 10110101. The first key piece X_1 is a second 8 bit random string 01011010. The second key piece X_2 is computed by XORing together K and X_1 as follows:

$$\begin{array}{rcl} K & : & 10110101 \\ \oplus & & \\ X_1 & : & 01011010 \\ & & ----- \\ X_2 & : & 11101111 \end{array}$$

The key K is recovered by XORing X_1 and X_2:

$$\begin{array}{rcl} X_1 & : & 01011010 \\ \oplus & & \\ X_2 & : & 11101111 \\ & & ----- \\ K & : & 10110101 \end{array}$$

10.5.2 n of n key escrow protocol

The 2 *of* 2 protocol can be generalised to an n of n protocol, where n is any positive integer greater than or equal to 2. This time n key pieces are generated $X_1, X_2, X_3, \ldots, X_n$. All of the n pieces are needed to recover the original key. The first $n - 1$ key pieces are b-bit binary strings chosen at random. The final key piece is computed by XORing the key K with all of the other key pieces:

$$X_n = K \oplus X_1 \oplus X_2 \oplus X_3 \oplus \ldots \oplus X_{n-1}$$

The key K is recovered by XORing all of the key pieces together:

$$K = X_1 \oplus X_2 \oplus X_3 \oplus \ldots \oplus X_{n-1} \oplus X_n$$

Learning activity

In a 3 of 3 key escrow scheme, the three key holders have keys:

$$X_1 = 01101011$$
$$X_2 = 10101111$$
$$X_3 = 00100101$$

What is the value of the key K that can be recovered if the three key holders work together?

An n of n key escrow protocol may not always be that practical. It is intended for use in an emergency, but all of the key holders may not be present when the emergency occurs and hence the key cannot be recovered. Suppose for example that the emergency in question is the death of one of the key holders!

Instead a t of n protocol where t is less than n can be used. We will describe a 2 of 3 protocol which uses simultaneous equations in t unknowns to hide and recover the key. This protocol can be generalised to a 2 of n protocol where n is any integer greater than 2.

10.5.3 2 of 3 key escrow protocol

Three key pieces are generated. Any two of the three pieces can be used to recover the original key. This time we think of the original key K as a decimal integer. We require a prime parameter p which is greater than K. The value of p does not have to be a secret. The basic protocol for the generation of the key pieces is as follows:

1. Alice, the holder of the key K, generates a random number a and three more random numbers x_1, x_2, x_3. These numbers must all be different and must be between 1 and p.
2. Alice computes $k_i = ((a \times x_i) + K) \bmod p$ for $i = 1, 2, 3$.
3. Alice keeps the value of a secret, and gives each of the three key holders a pair (x_i, k_i).

Example

Suppose our secret key K is 11.

We can use prime $p = 19$ and generate four random numbers $a = 14, x_1 = 3, x_2 = 17$, $x_3 = 10$.

Next we compute the key pieces:

$k_1 = ((14 \times 3) + 11) \bmod 19 = 53 \bmod 19 = 15$
$k_2 = ((14 \times 17) + 11) \bmod 19 = 249 \bmod 19 = 2$
$k_3 = ((14 \times 10) + 11) \bmod 19 = 151 \bmod 19 = 18$

We give our three key holders the key pieces $X_1 = (3, 15)$, $X_2 = (17, 2)$ and $X_3 = (10, 18)$ respectively.

Recovering K

Each key holder has a pair (x_i, k_i) and knows that $k_i = ((a \times x_i) + K) \bmod p$. Without knowledge of a, this equation cannot be solved. There are p possible values for a and hence p possible values for K.

However, if two key piece holders work together, they can form two equations in two unknowns which can be solved simultaneously to find the value of K.

Suppose the first two key holders share their information. Then they know:

$$k_1 = ((a \times x_1) + K) \bmod p \tag{10.1}$$
$$k_2 = ((a \times x_2) + K) \bmod p \tag{10.2}$$

Multiplying equation 10.1 by x_2 and equation 10.2 by x_1 gives two more equations:

$$x_2 k_1 = (x_2(a \times x_1) + K) \bmod p \tag{10.3}$$
$$x_1 k_2 = (x_1(a \times x_2) + K) \bmod p \tag{10.4}$$

Now subtracting equation 10.4 from equation 10.3 we have:

$$x_2 k_1 - x_1 k_2 = x_2 K - x_1 K \bmod p \tag{10.5}$$

Re-arranging equation 10.5 gives:

$$K = (x_2 k_1 - x_1 k_2)(x_2 - x_1)^{-1} \bmod p \tag{10.6}$$

Equation 10.6 can be solved to find the value of K.

Example

Recall in our example we gave our three key holders the key pieces $X_1 = (3, 15)$, $X_2 = (17, 2)$ and $X_3 = (10, 18)$ respectively.

Suppose the first two key holders decide to share their information. Then they can form the simultaneous equations:

$$15 = 3a + K \bmod 19$$
$$2 = 17a + K \bmod 19$$

Multiplying the first equation by 17 and the second equation by 3 (all mod 19) gives:

$$8 = 13a + 17K \bmod 19$$
$$6 = 13a + 3K \bmod 19$$

Subtracting the second equation from the first gives:

$$2 = 14K \bmod 19$$

Thus $K = (2 \times 14^{-1}) \bmod 19$. The inverse of 14 mod 19 = 15.

Therefore $K = (2 \times 15) \bmod 19 = 11$.

Learning activity

I use the prime 67 as the modulus to generate three key pieces for a 2 of 3 key escrow scheme. The three key pieces are $(x_1 = 3, k_1 = 27)$, $(x_2 = 11, k_2 = 61)$ and $(x_3 = 17, k_3 = 53)$. What is the value of my key K?

The 2 of 3 key escrow protocol is easily generalised into a 2 of n protocol where n is any integer greater than or equal to 2. Alice simply generates n different random values $x_1, x_2, \ldots, x_n$ and computes n key pieces by calculating $k_i = ((a \times x_i) + K) \bmod p$ for $i = 1, 2, \ldots, n$. As with the 2 of 3 protocol, any two of the key holders can work together in order to recover the key.

The scheme can also be generalised into a t of n for any value of $t > 1$ and any value of $n > t$. Alice chooses $t - 1$ random values $a_1, a_2, \ldots, a_{t-1}$ and she uses these to generate key pieces such that each key holder has an equation in t unknowns (the values of a_i and the value of K). Thus t key holders have to work together to form a set of t simultaneous equations in t unknowns which can be solved to find K.

10.6 Summary

In this chapter we have considered some key management problems that arise when using asymmetric key and symmetric key cryptosystems. We have seen that asymmetric (public/private) key schemes can be used to encrypt and transmit the key to be used in a symmetric scheme. We have looked at the Needham-Schroeder key exchange protocol which requires the use of a trusted third party to generate

session keys for use in a symmetric key cryptosystem. We have also seen that the safe use of public keys requires the encryptor to be certain that they are using the genuine public key of the person for whom they are encrypting the message. Verification of public keys can be done by the use of certificates, as in the X.509 protocol, or by requiring users to build up a public key-ring with keys being assigned a level of trust depending on who else has signed the keys. This method is sometimes called the *web of trust* and is used with the PGP email encryption protocol.

10.7 Learning outcomes

After studying this chapter and the additional reading, you should be able to:

- understand that in order to use a symmetric key cryptosystem there must be some method used to generate and transmit the key shared between the two cryptosystem users
- understand how public key cryptography can be used to encrypt and transmit a shared symmetric key
- describe how the Needham-Schroeder key exchange protocol works and how it prevents a cryptanalyst from discovering the shared key
- understand how the X.509 certification protocol allows a trusted third party to sign keys and how a public key cryptosystem user can obtain a certificate and verify that a key is genuine
- describe how the web of trust scheme works and how public key cryptosystem users can build up their own public key-ring with keys being assigned a level of trust depending on who has signed the keys.

10.8 Sample examination questions

Question 1

a) Describe the Needham-Schroeder protocol – for exchanging a session key between two subjects using symmetric key cryptosystems and a trusted third party.

[6]

b) Alice wishes to send Bob a message and uses the Needham-Schroeder protocol to generate a session key. Charles is a cryptanalyst who has access to all communication traffic between Alice, Bob and the trusted third party. How does the protocol overcome the following problems:

i. Charles determining the session key?

[2]

ii. Charles masquerading as Alice?

[2]

iii. Charles masquerading as Bob?

[2]

iv. Charles preventing Bob from receiving any information from Alice?

[2]

v. Charles replaying a previous key generation?

[2]

c) Explain how Alice could use Bob's public key to encrypt a shared session key. What are the advantages and disadvantages of using public key cryptography in this way when compared to using the Needham-Schroeder protocol?

[9]

Chapter 11

PGP and other Internet protocols

11.1 Introduction

In this chapter, we will consider some security protocols designed specifically to provide Internet security and electronic mail. We will give particular consideration to PGP (Pretty Good Privacy) which provides confidentiality and authentication for electronic mail communications, as well as some further services such as compression and segmentation. PGP uses the Web of trust access model for key management as discussed in Chapter 10. We will also discuss TLS (transport layer security) and SSH (secure shell) which are used to provide authentication for internet clients.

Supplementary reading

Chapter 5 of *Network Security Essentials*, by William Stallings covers PGP, S/MIME. Chapter 7 of the same book covers web security including SSL (the precursor of TLS).

See Phil Zimmermann's web pages (`www.philzimmermann.com`) for a description of PGP written by its creator.

You can download a free implementation of PGP called GNU Gnerd from`http://www.gnupg.org`

After studying this chapter and the related reading, you should be able to:

- understand that PGP and S/MIME are two schemes which provide security services for electronic mail, and that S/MIME uses a trusted third party to act as a certification agency, and that PGP uses the Web of trust model, allowing users to verify and sign each others' public keys
- describe how PGP uses the best cryptographic protocols in combination with each other to provide authentication and encryption services for electronic mail
- describe the steps that are taken by both sender (Alice) and receiver (Bob) when using the PGP scheme for authentication or encryption
- describe how PGP uses radix-64 conversion and segmentation in order to make encrypted messages compatible with electronic mail systems
- describe how ZIP compression works and know where the ZIP and UNZIP steps fit into the PGP protocol
- understand how TLS and SSH make use of the public key infrastructure to provide authentication for web browsers.

11.2 Security for Electronic Mail

Electronic mail (e-mail) requires particular security considerations. There are two main schemes that are especially designed to provide confidentiality and authentication for electronic mail systems. These are PGP and S/MIME.

S/MIME (Secure Multipurpose Internet Mail Extension) uses public key certificates conforming to standard X.509 and signed by a certification agency (see section 10.4.1). In other respects, S/MIME is quite similar to PGP. Broadly speaking, S/MIME is generally used for commercial and organisational use, whereas PGP (which is free to use and free of government agency involvement) is used for personal e-mail security. We will be concentrating on PGP in this subject. Students who are interested in learning further details about S/MIME should look at section 5.2 of *Network Security Essentials* by Stallings.

11.3 PGP

PGP – which stands for *Pretty Good Privacy* – was developed by Phil Zimmerman in 1995. The documentation and source codes are freely available. The package is independent of operating system and processor. PGP does not rely on the 'establishment' and there are no government agencies involved. PGP users sign each others' keys using the Web of trust model described in section 10.4.2. The popularity and use of PGP has grown extensively since 1995 with an estimated 60,000 users in the strongly connected set currently using PGP for their own personal e-mail security.

PGP is not a cryptosystem in itself, but combines the best available cryptographic algorithms to achieve secure e-mail communication. All users are assumed to have their own public/private keys for use with an asymmetric key cryptosystem. Either RSA with RSA digital signatures, or El Gamal with DSA (digital signature algorithm) can be used. A symmetric key cryptosystem such as AES or IDEA is also used, as is a secure hash algorithm such as SHA-256 or SHA-512.

PGP offers the following five services. We will look at how it achieves each of these in turn:

1. Authentication
2. Confidentiality
3. Compression
4. Compatibility
5. Segmentation.

11.3.1 PGP authentication

Recall that authentication means proving you are who you say you are. In terms of e-mail security, authentication means that if Bob receives an e-mail from Alice then he should be able to verify that the e-mail is genuinely from Alice and not from a third party posing as Alice for some reason. The PGP authentication protocol is a digital signature algorithm with hashing.

Assume that Alice has private and public keys A_{priv} and A_{pub} and that she wants to send message m to Bob with a digital signature to verify her identity. She takes the following steps. pk and sk are used to signify whether it is a public key cryptosystem or a symmetric key cryptosystem that is being used for encryption and decryption steps.

1. Alice hashes the message to obtain a message digest $h(m)$.

2. Alice encrypts the hash using her private key A_{priv} to obtain a signature.

$$sig = pk.encrypt_{A_{priv}}(h(m))$$

3. Alice sends Bob the pair (m, sig).
4. On receiving the pair (m, sig) from Alice, Bob decrypts sig using Alice's public key to obtain s.

$$s = pk.decrypt_{A_{pub}}(sig)$$

5. Bob computes the hash of m using the same hash algorithm as Alice to obtain the message digest $h(m)$.
6. If $h(m)$ is equal to s then the message is authenticated and Bob has verified that the message came from Alice.

This works because only Alice has access to her private key A_{priv} and so only she can create the correct signature. If either the message or the signature is altered, the authentication will fail because $SHA(m) \neq s$. It is not possible for anyone to alter the message and change the signature to match because to do that the key A_{priv} is required.

Note that the message m is sent in plaintext and so it is not confidential. In order to achieve secrecy the PGP confidentiality protocol is used.

11.3.2 PGP confidentiality

This time Alice wants to send Bob a confidential message, m. Assume that Bob has private and public keys B_{priv} and B_{pub} and that Alice trusts Bob's public key. Alice takes the following steps to send Bob a confidential e-mail communication.

1. Alice generates a random session key K for a symmetric cryptosystem.
2. Alice encrypts K using Bob's public key B_{pub}.

$$K' = pk.encrypt_{B_{pub}}(K)$$

3. Alice encrypts the message m using the session key K to get ciphertext c.

$$c = sk.encrypt_K(m)$$

4. Alice sends the values (K', c) to Bob.
5. On receipt of (K', c), Bob decrypts K' using his own private key to obtain K.

$$K = pk.decrypt_{B_{priv}}(K')$$

6. Bob uses the session key K to decrypt the ciphertext c and recover the message m.

$$m = sk.decrypt_K(c)$$

In this protocol, public and symmetric key cryptosystems are combined to provide security for the exchange of the session key and then efficiency for the encryption of the message. The session key K is used only to encrypt and decrypt message m and is not stored for any length of time or re-used.

PGP authentication and confidentiality (at the same time)

The previous two protocols provide authentication and confidentiality for e-mail communications. Obviously it is likely that a message which is important enough to encrypt should also be signed and vice versa. The following protocol is a combination of the previous two and allows for an e-mail communication to be both encrypted and signed. The following steps are taken by Alice who wants to send a signed confidential message to Bob.

1. Alice generates a signature for her message as in the authentication protocol:

$$sig = pk.encrypt_{A_{priv}}(h(m))$$

2. Alice generates a random session key K and encrypts the message m and the signature sig using a symmetric cryptosystem to obtain ciphertext c.

$$c = sk.encrypt_K(m, sig)$$

3. Alice encrypts the session key using Bob's public key.

$$K' = pk.encrypt_{B_{pub}}(K)$$

4. Alice sends Bob the values of K' and c.

Learning activity

Write the steps that Bob must take to recover and authenticate message m if Alice has encrypted and signed it using the PGP protocol for authentication and confidentiality as given above. Remember to state at each step which key you are using and which cryptosystem is being employed.

11.3.3 PGP compression

PGP can also compress the message if desired. The compression algorithm used is ZIP and the corresponding decompression algorithm is UNZIP. The ZIP algorithm works by replacing repetitions in the text by a short code. Following is a silly sentence which demonstrates how this works.

T$\underbrace{he\ brown\ fox}$ $\underbrace{jump}$ed $\underbrace{over\ the}$ brown foxy jumping frog which was over there.

The under-braces show text which is later repeated.

The sentence is replaced by the code:

The brown fox jumped over t1, 25, 13y14, 41, 4ing frog which was 21, 64, 8re.

Each set of three numbers indicates the start point of the original text, the start point of the text repetition, and the number of characters that are repeated. Bearing in mind that each character is actually eight binary bits this makes a compression rate of 1.24, meaning that a message of 124 bits is compressed to a message of 100 bits.

To send a signed, encrypted, compressed message to Bob, Alice takes the following steps.

1. Alice signs the original message m as before:

$$sig = pk.encrypt_{A_{priv}}(h(m))$$

2. Alice compresses the original message using the ZIP algorithm $M = ZIP(m)$.
3. Alice generates a session key, K, and uses it to encrypt the compressed message and the signature.

$$c = sk.encrypt_K(M, sig)$$

4. Alice encrypts the session key using Bob's public key to obtain K'.
5. Alice sends Bob the pair (K', c).
6. On receiving (K', c) from Alice, Bob decrypts K' using his own private key to obtain K.

$$K = pk.decrypt_{B_{priv}}(K')$$

7. Bob decrypts the ciphertext c using the session key K to obtain M and sig.

$$(M, sig) = sk.decrypt_K(c)$$

8. Bob decompresses M to obtain the original message m.

$$m = UNZIP(M)$$

9. Bob now has the message m. In order to authenticate it he uses Alice's public key A_{pub} to decrypt the signature and hashes the message m. If the two results match then the message is authenticated.

$$h(m) \stackrel{?}{=} pk.decrypt_{A_{pub}}(sig)$$

Note that the compression is applied after Alice has signed the message but before the encryption. This strengthens the security of the encryption because the ZIP algorithm reduces the message redundancy and therefore makes it harder for a cryptanalyst to decipher using linear analysis and other statistical techniques.

11.3.4 E-mail compatibility

Many electronic mail systems can only transmit blocks of ASCII text. This can cause a problem when sending encrypted data since ciphertext blocks might not correspond to ASCII characters. To overcome this problem, PGP uses *radix-64 conversion*.

Radix-64 conversion

Suppose the message to be securely transmitted via e-mail has been converted into binary using ASCII coding and encrypted to give a ciphertext stream of binary bits. Radix-64 conversion maps arbitrary binary into printable characters as follows.

1. The binary input is split into blocks of 24 bits (3 bytes).
2. Each 24-bit block is then split into four smaller blocks each of 6-bits.
3. Each 6-bit block will then have a (decimal) value between 0 and $2^6 - 1 = 63$. This value is encoded into a printable character using Table 11.1.

6 bit value	Character encoding	6 bit value	Character encoding	6 bit value	Character encoding	6 bit value	Character encoding
0	A	16	Q	32	g	48	w
1	B	17	R	33	h	49	x
2	C	18	S	34	i	50	y
3	D	19	T	35	j	51	z
4	E	20	U	36	k	52	0
5	F	21	V	37	l	53	1
6	G	22	W	38	m	54	2
7	H	23	X	39	n	55	3
8	I	24	Y	40	o	56	4
9	J	25	Z	41	p	57	5
10	K	26	a	42	q	58	6
11	L	27	b	43	r	59	7
12	M	28	c	44	s	60	8
13	N	29	d	45	t	61	9
14	O	30	e	46	u	62	10
15	P	31	f	47	v	63	/
						(pad)	=

Table 11.1: Radix-64 conversion characters

For example, suppose the encrypted message is CAT.[1] The ASCII codes for C A T are 67, 65, 84 in decimal so we have the 24-bit binary string:

010000011 01000001 01010100

These 24-bits are split into four 6-bit blocks:

010000 110100 000101 010100

Converting these binary blocks into decimal we have 16, 52, 5, 20 which are converted using the table above into the characters Q 0 F U.

We have a message expansion of 33 per cent, but this can be countered by using ZIP compression. The message should be compressed, then encrypted and finally converted using radix-64.

Learning activity

1. Compute the radix-64 conversion of the message 'Dog'.
2. What does the radix-64 message 'Y2F0' actually represent?

PGP segmentation

Another constraint of e-mail is that there is usually a maximum message length. PGP automatically blocks an encrypted message into segments of an appropriate length

[1] Obviously this could be sent without the need for radix-64 conversion as the characters are all ASCII compatible, but using these characters helps to demonstrate the message expansion caused by using radix-64.

before transmission. On receipt, the segments must be reassembled before the decryption process.

11.3.5 PGP key issues

Key generation

Recall that a new session key is required each time a message is encrypted. PGP generates these keys using pseudorandom numbers. True random numbers are also required for the generation of public and private keys. In PGP these are generated using the timing of key strokes and user input key patterns. See Chapter 5 of Network Security Essentials by Stallings for further details.

Key identifiers

PGP allows users to have more than one public/private key pair. The purpose of this is to increase security and to ease the key changeover period when a key reaches its expiry date. When receiving a PGP message, how does the receiver know which set of public/private keys they should be using to decrypt the message and/or signature?

In the case of encryption, Alice has used one of Bob's public keys to encrypt the session key. This key is not secret, so Alice can send Bob the public key she has used along with the ciphertext. She actually sends a key identifier consisting of the least significant 64 bits of the key rather than the entire public key.

If Alice uses her private key to sign a message, she obviously cannot send the private key she has used to Bob. Instead she sends a key identifier for the corresponding public key.

So altogether a PGP message might consist of:

- A message component which is the actual data to be transmitted + a file-name + a time-stamp.
- A signature component which consists of a time-stamp + hash of the message and time-stamp + first part of the message (so the receiver can check that they are decrypting correctly) + Key identifier of sender's public key.
- The session key component consisting of the encrypted session key + key identifier for the recipient's public key.

11.4 TLS and SSL

TLS (Transport Layer Security) is the successor of SSL (Secure Sockets Layer) and is the security protocol used by web browsers to connect securely to web browsers. SSL was implemented by Netscape and became the de facto standard until TLS, which varies only slightly from SSL, came into use in 1999. TLS is now the official version.

TLS uses public key infrastructure and certificates issued by a trusted third party, the *certification agency* or CA. When a TLS client wants to make contact with a server, a *handshake* is performed which consists of several steps. If any of these steps fail,

then the handshake fails and the connection between the client and the server is not created.

- The handshake is initiated by the client connecting to a TLS-enabled server requesting a secure connection. The client presents the server with a list of supported encryption and hash algorithms.
- The server picks the strongest encryption function and hash function that it also supports and tells the client which algorithms to use.
- The server sends the client a digital certificate which contains the name of the server, the trusted CA and the server's public key.
- The client should contact the CA and verify that the certificate is authentic before proceeding.
- The client generates a random number to use as a session key, and encrypts this key using the server's public key. The client sends this encrypted key to the server who decrypts it using the corresponding private key.
- Now both client and server have a copy of the session key, and they use this to encrypt and decrypt material until the session is closed.

11.5 SSH

SSH (secure shell) is a protocol which allows data to be securely exchanged between two computers. SSH uses encryption to provide confidentiality and integrity of data being passed over an insecure network such as the Internet. Like TLS, SSH uses public key cryptography to authenticate the remote computer. Unlike TLS, SSH also allows the remote computer to authenticate the user if necessary.

Learning activity

Do an Internet search to find out more about TLS and SSH. What are these protocols used for? Which cryptographic algorithms do they support? What, if any, are their vulnerabilities?

11.6 Summary

In this chapter we have studied how PGP (pretty good privacy) combines public and symmetric encryption and decryption algorithms, hash functions and compression algorithms to provide authentication, confidentiality, compression and compatibility for electronic mail messages. We have also looked at TLS (transport layer security) which provides authentication for web browsers allowing users to securely connect to an authenticated server, and SSH (secure shell) which additionally allows servers to authenticate clients.

11.7 Learning outcomes

After studying this chapter and the related reading, you should be able to:

- understand that PGP and S/MIME are two schemes which provide security services for electronic mail, and that S/MIME uses a trusted third party to act as a certification agency, and that PGP uses the Web of trust model, allowing users to verify and sign each others' public keys
- describe how PGP uses the best cryptographic protocols in combination with each other to provide authentication and encryption services for electronic mail
- describe the steps that are taken by both sender (Alice) and receiver (Bob) when using the PGP scheme for authentication or encryption
- describe how PGP uses radix-64 conversion and segmentation in order to make encrypted messages compatible with electronic mail systems
- describe how ZIP compression works and know where the ZIP and UNZIP steps fit into the PGP protocol
- understand how TLS and SSH make use of the public key infrastructure to provide authentication for web browsers.

11.8 Sample examination questions

Question 1

a) Pretty Good Privacy (PGP) provides security for electronic mail systems. Describe how PGP simultaneously provides confidentiality, authentication and compression. Give any assumptions that you are making. Explain in words what is happening at each step and why this step is necessary.

[15]

b) Why is it normal for the message to be encrypted after compression?

[2]

c) Explain how PGP allows a user to have more than one public/private key pair.

[4]

Question 2

a) Alice, Bob and Charles are using PGP to communicate. Consider the following scenario:

- Alice writes a message, signs it, then encrypts the message for Bob.
- Bob receives the message and decrypts it. Bob now has a message with Alice's signature on it, which can be kept. The message says 'I love you.'
- Bob does not share Alice's feelings and decides to play an embarrassing trick on her.
- Bob encrypts the message, already signed by Alice, using Charles' public key, and sends this to Charles.

How will Charles interpret the message? Will he be able to tell that he has been misled?

[4]

What message should Alice have sent in order to avoid this embarrassment?

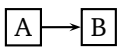
[2]

b) The key management system used by PGP is the Web of trust model where users are allowed to sign each others' public keys. Keys are published together with all of the signatures they have obtained. Alice should only sign Bob's key if she has personally verified his identity and is sure that his key is correct.

i) What are the advantages and disadvantages of the web of trust model compared with using a trusted third party to sign keys?

[3]

The following notation is used to indicate that Alice (A) has signed Bob's (B) key.



ii. Consider the diagrams, Figure 1 and Figure 2, which show two different pathways between keyholders A and E via other keyholders B, C and D. In which case should Alice be more confident of E's identity? Explain your answer.

[5]

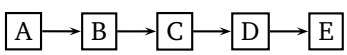


Figure 1

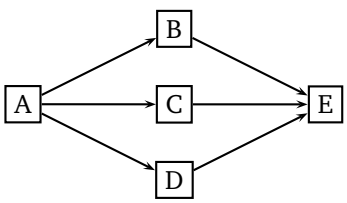


Figure 2

Appendix A

Sample examination paper

Time allowed: 2 hours 15 minutes

Answer any **three** of the following five questions.

Full marks will be awarded for complete answers to a total of three questions. Each question carries 25 marks. The marks for each part of a question are indicated at the end of the part in [ ] brackets.

There are 75 marks available on this paper.

Electronic calculators must not be programmed prior to the examination. Calculators which display graphics, text or algebraic equations are not allowed.

1. The charity *CryptoAid* is run by six committee members. On the committee there is the chairman, treasurer, secretary and three other committee members. The committee meets every three months to discuss how money for the charity should be raised and spent. The committee tries to make decisions by majority vote amongst themselves with no member of the committee having a greater weight than any of the others. *CryptoAid* has a bank account and the treasurer keeps the accounts. The treasurer presents a copy of the accounts to all of the committee members at each meeting. The treasurer, chairman and secretary are all authorised signatories for the bank account. Any one of the signatories can get statements or pay money into the bank account. To withdraw money from the bank account, any two of the signatories must sign a cheque.

   (a) Identify the *subjects* and *objects* in the scenario described above.

   [4]

   (b) Describe a *protection ring* access control structure and explain why a protection ring is not a suitable structure to model the subjects identified in the *CryptoAid* scenario.

   [4]

   (c) At a particular committee meeting, it is decided that a committee member called Frank should become a fourth authorised signatory for the *CryptoAid* bank account. Write a protocol for adding Frank onto the list of authorised signatories held by the bank. The aim of the protocol is to ensure that it is impossible for anyone to become a signatory for the bank account without the proper authorisation and verification.

   [8]

   (d) Now there are four authorised signatories and any two of them have to sign a cheque to withdraw money from the bank account. This is a basic 2 of 4 key escrow scheme. Why is this scheme arguably less secure than a key escrow scheme based on a mathematical formula?

   [1]

   (e) The *CryptoAid* committee want to increase security and so decide to implement a 2 of 4 key escrow scheme based on the solution of simultaneous equations mod a prime number. Explain how the keys for each key holder are generated.

   [4]

   (f) Given that the keys held by the treasurer and chairman are $(x_1 = 7, k_1 = 14, p = 17)$ and $(x_2 = 3, k_2 = 11, p = 17)$ respectively, find the shared key K.

   [4]

2. (a) i. Describe the key generation protocol for the RSA public key cryptosystem.

[7]

ii. Suppose that an algorithm is found that can efficiently factorise a large number. Explain how a cryptanalyst could use this algorithm to break the RSA cryptosystem.

[3]

(b) Below is a simple method for blocking and encoding upper case letters for encryption.

- Group letters into blocks of two starting at the leftmost letter. Add a space character at the rightmost side if necessary to make an even number of characters.
- Encode letters using $A = 1$, $B = 2$, ..., $Z = 26$, space = 0.
- Convert each block of two characters into a single decimal number by multiplying the value of the first character by 27 and then adding the value of the second character.

Thus the word STARS is partially encoded as follows:

characters	S	T	A	R	S	*space*
character values	19	20	1	18		
block value	19(27)+20 =533					

i. Copy and complete the table above. Then using the RSA public keys ($n = 799, e = 3$) encrypt the word STARS.

[5]

ii. Suppose we want to include both lower case and upper case characters in our messages. Suggest changes that have to be made to the key and encoding method that would enable this. (Keep two characters per block.)

[4]

iii. Instead of changing the key, it is suggested that messages are encrypted character by character. Under this scheme, the word stars is encrypted to (39,657,507,490,39). Decrypt the word (39,657,507,490,657) and hence or otherwise explain why encrypting character by character is a bad idea.

[6]

3. The Needham-Schroeder Protocol can be described symbolically as follows:

   A $\rightarrow$ S: A,B

   S $\rightarrow$ A: $e_{K_{AS}}(B, K_{AB}, e_{K_{BS}}(A, K_{AB}))$ A decrypts

   A $\rightarrow$ B: $e_{K_{BS}}(A, K_{AB})$ B decrypts

   (a) Explain each step of the protocol in words.

   [7]

   (b) Adapt the protocol given above so that it prevents another person replaying a message. Explain how the modifications you have made work.

   [6]

   (c) Alice and Bob are using the Needham-Schroeder Protocol to communicate with each other. A cryptanalyst Charles has access to all of the communications between Alice and Bob.
   Explain why this is not a security problem.

   [4]

   Charles now gets hold of Bob's key K_{BS}. What can Charles do now? What should Bob do when he discovers that his key has been compromised?

   [8]

4. (a) Consider the following three mappings where x is a binary number of arbitrary length.

$f(x) = x \oplus$ the current air temperature in degrees Centigrade

$g(x) = x \bmod 2^{64}$ (i.e. the least significant 64 bits of x)

$h(x)$=SHA-1(x)

i. Of the above, which is a cryptographic hash function, which is a hash function, and which is not a proper function?

[3]

ii. Why is the hash function identified in part (i) unsuitable for use in a cryptographic protocol?

[4]

iii. Describe the SHA-1 protocol.

[6]

(b) KeepitSafe Ltd has designed two vaults with electronic locks. The vaults open only after the correct decimal code has been entered. Version A is a low cost model which expects an 8-digit code. After all eight digits have been entered, it will either open or will signal that the code was wrong and remain locked. Version B is a far more secure version which expects a 32-digit code.

i. Explain why some companies may prefer to use version A even though version B is many times more secure.

[4]

To overcome the problems associated with version B, KeepitSafe introduce a modification. Version B2 still expects a 32-digit code but after every four digits, B2 either confirms that the code has been entered correctly so far, or it asks for the previous four digits again.

ii. Compare the security of the three versions A, B and B2. You may assume that a hacker can provide numbers (regardless of length) at a rate of 1,000 per second.

[8]

5. (a) Write a paragraph describing the main differences between public key and symmetric key cryptosystems. Explain the advantages and disadvantages of each.

[6]

(b) Alice, Bob and Charles are using a public key and symmetric key cryptosystem to communicate with each other. Following is the protocol they are using to send a signed, encrypted message.

- Alice wants to send Bob message m.
- Alice generates a random session key K and encrypts the message using the symmetric key cryptosystem with key K to obtain $c = encrypt_K(m)$.
- Alice generates a signature for her message by encrypting c using her private key. Thus $sig = encrypt_{A_{priv}}(c)$.
- Alice uses Bob's public key to encrypt the session key K to obtain $k' = encrypt_{B_{pub}}(K)$.
- Alice sends (c, sig, k') to Bob.

Complete the protocol by writing the steps that Bob must take to read and authenticate the message.

[4]

(c) Consider the following scenario. Alice describes a money making invention in a message m and signs it for Bob as described in the protocol above. Charles intercepts the message.

Show how Charles can replace Alice's signature with his own before sending the message to Bob. How will Bob interpret the message?

[10]

(d) Re-write the protocol given in part b) so that the message is signed first and then encrypted. Does this new protocol prevent Charles from stealing Alice's idea?

[5]

END OF EXAMINATION

Appendix B

Solutions

B.1 Subject guide activity solutions

B.1.1 Chapter 1

Activity 1.2.1

SQL injection is a technique exploiting security vulnerabilities occurring in the database layer of an application. If user input is either incorrectly filtered or not strongly typed, it may be unexpectedly executed. The key disaster in an injection attack (SQL here but this can also occur in HTML or Javascript) in so-called *cross-site scripting* attacks, is treating untrusted user input directly as program text. There are many real world examples of injection attacks. You may not think the cartoon is funny, but the point is that attacks can come from unexpected sources.

Activity 1.3.7

There is no right or wrong answer to these questions, but the activity demonstrates that security issues are not always straight forward. Note that this is a real life example of a situation that occurs quite frequently.

B.1.2 Chapter 2

Activity 2.3.1

The hacker should start with a dictionary search using words of eight characters. First he uses just lower case letters and next he uses the same words but with the first letter of each word capitalised. If this is not successful, he should next try a modified dictionary search using words of seven characters with a digit appended. He could also try a dictionary search but with letters replaced with similar digits such as the number 0 in place of the letter O and so on. The chances are that at least one user of the system has a weak password. It will be faster to check all the users for weak passwords than to perform an exhaustive search for even a single user. Only if all dictionary modifications fail to bring success should the hacker move onto an exhaustive search.

Activity 2.3.2

1. $26^4 = 456976$ possible passwords of length four. $26^8 \approx 2 * 10^{11}$ possible passwords of length eight using lower case letters only.

2. $62^4 = 14776336$ possible alphanumeric passwords of length four. $62^8 \approx 2.2 * 10^{14}$ possible alphanumeric passwords of length eight.

3. (a) There are 26^8 combinations to try. This will take $26^8/10,000 \approx 242$ days. But on average the hacker will only have to try half of the possible combinations which will take approximately 121 days.

   (b) Now there are 62^8 combinations to try and this will take over 692 years. Even given that on average the hacker will only have to search half of the possible combinations, he will not manage to complete this search!

Activity 2.3.4

Answers will vary but the important point is that the user **must** log into the system using their current password **before** being allowed to change the password.

Activity 2.4.3

If a hacker found that two employees had the same password he could assume that these were not random passwords (since it is unlikely that two employees would happen to have the same random password) and he would be likely to succeed in finding the password using an intelligent search. He could either try to find out what the two employees have in common – perhaps they both have a wife named Alison or perhaps they both support the same football club. This might give him a clue to the password. Or, knowing that the password is probably not a random combination he could try a dictionary search with a high probability of success.

B.1.3 Chapter 3

Activity 3.3.2

i)

	O_1	O_2	O_3	O_4	O_5
S_1	√	√			
S_2	√	√	√	√	
S_3	√		√	√	
S_4		√	√	√	√
S_5			√	√	√
S_6			√	√	

ii) S_1:O_1, O_2
S_2:O_1, O_2, O_3, O_4
S_3:O_1, O_3, O_4
S_4:O_2, O_3, O_4, O_5
S_5:O_3, O_4, O_5
S_6:O_3, O_4

B.1.4 Chapter 4

Activity 4.3

Ciphertext 1 1 1 0 1 0 1 0 0 1 0 0 1.....
Random Stream 1 0 0 1 1 0 0 0 1 1 0 1 0.....
Message 0 1 1 1 0 0 1 0 1 0 0 1 1.....

a	b	$c = a \oplus b$	$c \oplus b = a$	$c \oplus a = b$
0	0	0	0	0
0	1	1	0	1
1	0	1	1	0
1	1	0	1	1

Activity 4.3

An attacker may intercept the armed guard, steal the locked case, force it open and retrieve the copy of the one-time pad. On its own this is of no use to the attacker. Bob will inform Alice that the one-time pad has not arrived safely and Alice will dispose of her copy and generate another. On the other hand, if the armed guard was carrying the actual message and was attacked, then the attacker would get hold of the message and confidentiality is destroyed.

Activity 4.4.1

DECRYPTION IS EASY

Activity 4.4.2

I CAN'T I'M BUSY

Activity 4.4.2

I AM IN A CAFE IN MILAN. MEET ME HERE AT SEVEN AND I WILL HAND OVER THE COMPUTER CHIP.

Activity 4.6.1

1. The message alphabet is the set of all characters that can be included in the message. {A,B,C,....,Z,0,1,2,....,9,△}.
2. The ciphertext alphabet is the same as the message alphabet.
3. The keyspace is the set of all possible permutations of the message alphabet.
4. The trivial key is the permutation $A \to A$, $B \to B$,....,$Z \to Z$ etc. Other keys are nearly trivial such as $A \to Z$, $B \to B$, $C \to C$,....,$Y \to Y$, $Z \to A$.
5. The size of the message alphabet is 63 because there are 26 lower case letters, 26 upper case letters, 10 digits and 1 space character.

6. The size of the keyspace is $63! = 63 \times 62 \times 61 \times \ldots \times 2 \times 1 \approx 1.98 \times 10^{87}$ because this is the number of different permutations of the message alphabet. This includes the trivial keys.

B.1.5 Chapter 5

Activity 5.2.1

1. $\underbrace{\mathbf{1011}00100011110}_{keystream\ of\ 15\ bits}$ 1011
2. $\underbrace{\mathbf{0110}01000111101}_{keystream\ of\ 15\ bits}$ 0110
3. **0000** $\underbrace{0000}_{repeating\ 4-bit\ sequence}$

Activity 5.3.1

The decryption protocol is the reverse of the encryption protocol and so can be written as follows:

1. Split the ciphertext into blocks of eight bits.
2. Starting with the left most block:
   (a) XOR the key with the ciphertext block to obtain the message block.
   (b) Add the message block just recovered to the key and the eight most significant bits as the key for the next ciphertext block.
   Repeat step ii) until all the ciphertext blocks have been decrypted.
3. Convert the last eight bits of the decrypted message into decimal to discover the length of the original message.
4. Discard any padding zeros according to the length discovered in step iii).

The ciphertext 11001100011110001110110010010010 is decrypted as follows:

Ciphertext	11001100	01111000	11101100	10010010
Key	01100110	10001000	10111100	10000110
Message	10101010	11110000	01010000	00010100
Next Key	100010000	101111000	100001100	

The last block of the decrypted message is 00010100 which is the decimal number 20. Hence the original message is 20 bits in length. We discard four padding zeros to obtain the original message 10101010111100000101.

Comments on the security of the symmetric cryptosystem

For each block $m_i \oplus k_i = c_i$ and $m_i + k_i = k_{i+1}$. Therefore if an attacker knows m_i and c_i for one message block (for example, in a known plaintext attack or a probably plaintext attack) they can find k_i and all subsequent keys. So the attacker will be able to read the message from block m_i onwards.

The keys generated after the initial key will always start with a 1 bit since we are taking the eight most significant bits. This effectively reduces the size of the keyspace from 2^8 to 2^7.

In blocks two and three of the recovered message above there are two bits different in the plaintext and four bits different in the ciphertext, so there is some diffusion.

A key which is nearly correct will decrypt a message block nearly correct so the cryptosystem does not have the confusion property.

Each bit of the ciphertext depends on only one bit of the key, therefore the cryptosystem does not have the property of completeness.

All in all, this is not a very good symmetric cryptosystem!

B.1.6 Chapter 7

Activity 7.4.3

1. $3^{23} \bmod 17 = 11$

y	u	n
1	3	23
3	9	11
10	13	5
11	16	2
11	1	1
11	1	0

2. $5^{45} \bmod 2311 = 774$

y	u	n
1	5	45
5	25	22
5	625	11
814	66	5
571	2045	2
571	1426	1
774	1426	0

Activity 7.4.4

$a \bmod 19$	1	2	3	4	5	6	7	8	9
$a^{-1} \bmod 19$	1	10	13	5	4	16	11	12	17

$a \bmod 19$	10	11	12	13	14	15	16	17	18
$a^{-1} \bmod 19$	2	7	8	3	15	14	6	9	18

$a \bmod 23$	1	2	3	4	5	6	7	8	9	10	11
$a^{-1} \bmod 23$	1	12	8	6	14	4	10	3	18	7	21

$a \bmod 23$	12	13	14	15	16	17	18	19	20	21	22
$a^{-1} \bmod 23$	2	16	5	20	13	19	9	17	15	11	22

Activity 7.4.5

1. $19 \bmod 562$

$$562 = 19 \times 29 + 11$$

$$\begin{aligned} 19 &= 11 \times 1 + 8 \\ 11 &= 8 \times 1 + 3 \\ 8 &= 3 \times 2 + 2 \\ 3 &= 2 \times 1 + 1 \end{aligned}$$

$$\begin{aligned} 1 &= 3 - 2 \\ 1 &= 3 - (8 - 2(3)) = 3(3) - 8 \\ 1 &= 3(11 - 8) - 8 = 3(11) - 4(8) \\ 1 &= 3(11) - 4(19 - 11) = 7(11) - 4(19) \\ 1 &= 7(562 - 29(19)) - 4(19) = 7(562) - 207(19) \end{aligned}$$

Therefore $19^{-1} \bmod 562 = -207 = 355 \bmod 562$

2. 300 mod 847

$$\begin{aligned} 847 &= 300 \times 2 + 247 \\ 300 &= 247 \times 1 + 53 \\ 247 &= 53 \times 4 + 35 \\ 53 &= 35 \times 1 + 18 \\ 35 &= 18 \times 1 + 17 \\ 18 &= 17 \times 1 + 1 \end{aligned}$$

$$\begin{aligned} 1 &= 18 - 17 \\ 1 &= 18 - (35 - 18) = 2(18) - 35 \\ 1 &= 2(53 - 35) - 35 = 2(53) - 3(35) \\ 1 &= 2(53) - 3(247 - 4(53)) = 14(53) - 3(247) \\ 1 &= 14(300 - 247) - 3(247) = 14(300) - 17(247) \\ 1 &= 14(300) - 17(847 - 2(300)) = 48(300) - 17(847) \end{aligned}$$

Therefore $300^{-1} \bmod 847 = 48$

Activity 7.5

1. $b(127) = 7$, $b(128) = 9$, $b(129) = 9$, $b(10^6) = 20$, $b(10^9) = 30$
2. $b(10^{12}) = 40$
3. $b(0) = 1$

Activity 7.5.1

1. It takes the computer package $3 \times 5 = 15\mu s$ to add two 1,000 bit numbers because the numbers are five times bigger and addition is an $O(b)$ operation.
2. (a) The numbers are five times bigger and multiplication is an $O(b^2)$ operation, so it takes $5^2 = 25$ times as long to multiply the numbers. This is $25 \times 80 = 2,000\mu s$.
   (b) This time the numbers are half the size, so it takes $(0.5)^2 = 0.25$ times as long to multiply the numbers. This is $0.25 \times 80 = 20\mu s$.

B.1.7 Chapter 8

Activity 8.2.2

97397*77359 930329859448403*595388123465657

Activity 8.2.3

Calculate 2^{46} mod 47 using the fast exponentiation method as follows:

y	u	n
1	2	46
1	4	23
4	16	11
17	21	5
28	18	2
28	42	1
1		0

2^{46} mod 47 = 1 hence we can deduce that 47 is (probably) a prime number.

Calculate 2^{50} mod 51 using the fast exponentiation method as follows:

y	u	n
1	2	50
1	4	25
4	16	12
4	1	6
4	1	3
4	1	1
4		0

2^{50} mod 51 = 4 hence 51 is definitely not a prime number.

Activity 8.3.4

1. $n = p \times q = 11 \times 13 = 143$, $r = (p-1) \times (q-1) = 10 \times 12 = 120$.
2. $7 \times 103 = 721 = 1 \bmod 120$ since $721 = (6 \times 120) + 1$.
3. $c = 4^7 \bmod 143 = 16384 \bmod 143 = 82$.
4. $m = 29^{103} \bmod 143 = 68$

y	u	n
1	29	103
29	126	51
79	3	25
94	9	12
94	81	6
118	126	3
68		0

B.1.8 Chapter 9

Activity 9.2.1

n	2^n mod 19	3^n mod 19	5^n mod 19
1	2	3	5
2	4	9	6
3	8	8	11
4	16	5	17
5	13	15	9
6	7	7	7
7	14	2	16
8	9	6	4
9	18	18	1
10	17	16	5
11	15	10	6
12	11	11	11
13	3	14	17
14	6	4	9
15	12	12	7
16	5	17	16
17	10	13	4
18	1	1	1

Note that the first time a 1 occurs in the 2^n mod 19 and 3^n mod 19 columns is in the last row. Thus both 2 and 3 are suitable generators for 19. However there is a 1 when $n = 9$ in the 5^n mod 19 column. Thus 5 is **not** a suitable generator for 19.

Activity 9.3

Alice computes $x = 5^9 \bmod 23 = 11$ and sends this to Bob.
Bob computes $y = 5^6 \bmod 23 = 8$ and sends this to Alice.
Alice computes $k = 8^9 \bmod 23 = 9$.
Bob computes $k = 11^6 \bmod 23 = 9$.

Activity 9.4.4

1. $r = 2^6 \bmod 53 = 11$
   $x = 17^6 \bmod 53 = 44$
   $c = (25 \times 44) \bmod 53 = 40$
   ciphertext is $(r = 11, c = 40)$

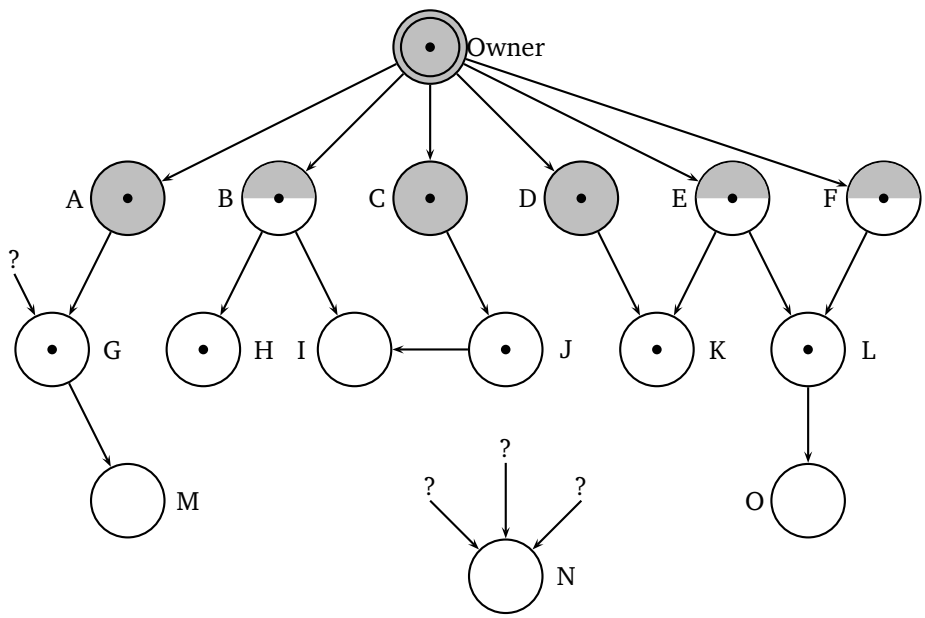
2. $x = 6^{10} \bmod 53 = 13$
   $x^{-1} = 13^{-1} \bmod 53 = 49$
   $m = (33 \times 49) \bmod 53 = 27$

B.1.9 Chapter 10

Activity 10.2.1

Every person will need to store 99 keys. In total there will be $(100 \times 99) \div 2 = 4,950$ keys. (We divide by two because each of the keys is held by two people but only needs to be counted once.)

Activity 10.4.2



Activity 10.5.2

We XOR together the three key pieces X_1, X_2 and X_3 to recover the original key K:

$$\begin{array}{rcl} X_1 & : & 01101011 \\ X_2 & : & 10101111 \\ X_3 & : & 00100101 \\ & \oplus & - - - - - - - - \\ K & : & 11100001 \end{array}$$

Activity 10.5.3

Assuming the first two key holders work together we have simultaneous equations:

$$\begin{aligned} 27 &= 3a + K \bmod 67 \\ 61 &= 11a + K \bmod 67 \end{aligned}$$

Multiplying the first equation by 11 and the second equation by 3 gives:

$$\begin{aligned} 29 &= 33a + 11K \bmod 67 \\ 49 &= 33a + 3K \bmod 67 \end{aligned}$$

Now subtracting the second equation from the first we have:

$$47 = 8K \bmod 67$$

The inverse of 8 mod 67 = 42 because $8 \times 42 = 336 = 1 + (5 \times 67)$. Therefore $K = (47 \times 42) \bmod 67 = 31$.

If you need further practice, try to use another combination of key holders to recover the original key. For example, solve the equations for key holders 2 and 3 working together.

B.1.10 Chapter 11

Activity 11.3.2

In order to recover and authenticate message m Bob must take the following steps:

1. Bob receives the pair (K', c) from Alice.
2. Bob decrypts K' using his own private key to obtain K.

$$K = pk.decrypt_{B_{priv}}(K')$$

3. Bob decrypts the ciphertext c using the session key K to obtain m and sig.

$$(m, sig) = sk.decrypt_K(c)$$

4. Bob now has the message m. In order to authenticate it he uses Alice's public key A_{pub} to decrypt the signature and hashes the message m. If the two results match then the message is authenticated.

$$SHA(m) \stackrel{?}{=} pk.decrypt_{A_{pub}}(sig)$$

Activity 11.3.4

1. Dog → RE9H
2. Y2F0 → cat

B.2 Sample examination questions solutions

B.2.1 Chapter 1

Question 1

a) i. Confidentiality is the prevention of unauthorised disclosure of information.
ii. Integrity is the prevention of unauthorised modification of information.
iii. Availability is the prevention of unauthorised with-holding of information.
iv. Non-repudiation is the prevention of either the sender or receiver denying a transmitted message.
v. Authentication is the verification of a claim.
vi. Access control is the limitation and control of access through identification and authentication.
vii. Accountability is the maintenance of audit trails which indicate unauthorised activities and can be used to trace these activities back to the user.

b) Confidentiality has not been breached since the file is encrypted and the student does not decrypt the material. However, integrity is definitely breached since the computer file is now different to the paper records the University presumably also keeps. Access control has been breached since the student should not have been allowed to access the file in the first place. Hopefully, there will be accountability and the staff will be able to determine that someone has altered the file and who that person was.

Question 2

To prevent burglary of a physical item we could lock it away where no-one can get near it, but locking up a computer does not necessarily mean that no-one can access it. If a physical item is stolen, then we would detect the theft by seeing that the item is missing. However, digital information can be stolen by copying so that the original information is still present. Therefore, we might not detect that there has been a theft. If a physical item is stolen, then one reaction is to replace the item, but if the data is stolen by copying, then the original owner will also still have it. Police might catch a burglar red-handed, but people committing theft of digital information might be nowhere near the scene of the crime.

B.2.2 Chapter 2

Question 1

a) Passwords required are much too complicated. In order to remember them staff will have to use very obvious passwords and will probably write their passwords down. This will make it easy for an attacker to find out a staff password.

b) ■ Staff passwords are of a minimum length of six characters (seven or eight also acceptable).
Explanation: Too short and exhaustive search is possible; too long and staff will not be able to remember their passwords.

- Staff are required to change their passwords every three months (four or six also acceptable).
  Explanation: Too often and staff will not be able to remember which password they are using and so are likely to write them down or simply change a digit on the end of the password to create a new one. Too long and an attacker has longer to try searching techniques.
- Every password contains at least one digit and at least one character.
  Explanation: To increase the size of the alphabet and therefore make exhaustive searches more time consuming.
- Passwords should not be based on dictionary words.
  Explanation: To prevent dictionary attacks.
- The user should be forced to change the default password.
  Explanation: the attacker may try the default password as their first guess.

c) Assuming policy given as solution: size of alphabet=62, number of passwords = 62^6. Try on average half so time taken = $62^6/2 \div 10,000$ = 2840012 mins = 47334 hours = 1972 days = 5.4 years.

Question 2

a) A function $y = f(x)$ from set X to Y is said to be one-way if, given an $x \in X$ it is easy to compute $y = f(x)$. However given a $y \in Y$ it is very hard to find an $x \in X$ such that $y = f(x)$.

b) Instead of storing the users' passwords indexed by user-name, the password file stores an encrypted version of the password indexed by user-name. When a new user joins the system, they enter their user-name and password (x). The computer encrypts the password using the one-way function $f(x)$ and stores $f(x)$ together with the user-name. When the user wants to log onto the system, they enter their user-name and password (x'). The computer computes $f(x')$ and compares the result with the encrypted password stored for the user-name given. If $f(x) = f(x')$ the user is verified and given access to the system.

c) The one-way function used to protect a password file should not be too efficient because a hacker who has gained access to the password file may try to find a password by encrypting a list of possible passwords one by one until he finds a match with one of the encrypted passwords in the file. If each encryption takes a few seconds, this kind of attack will take an extremely long time. For the genuine user, a delay of a few seconds each time they log in is negligible.

B.2.3 Chapter 3

Question 1

a) Subjects: Mrs A, Dr B, Dr C, Dr D, Prof H
Objects: salary.doc, timetable.doc, marks.doc
Operations: read, write

b) Since all three lecturers have the same access control permissions, we can put them into a group called lecturers and have just one row in the access control matrix which represents the group lecturers rather than three seperate rows for the three lecturers.

c)

	salary.doc	timetable.doc	marks.doc
Mrs A	{r}	{r,w}	{}
lecturers	{}	{r}	{r,w}
Prof H	{r,w}	{r}	{r}

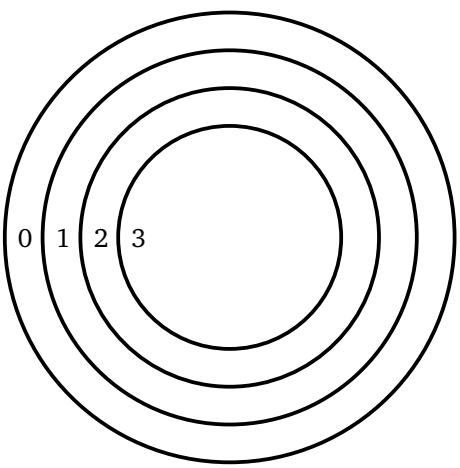
Question 2

a) Further objects are solutions, scripts and marks.

b) Assumptions: write permission does not include read permission; limited access read means that subjects are only allowed to read the object for a limited period. $w = write, r = read, a = alter, l = limitedaccessread$.

	examination paper	solutions	scripts	marks
candidates	{l}	{}	{w}	{}
lead examiner	{w,r}	{w,r}	{r}	{w,r,a}
second examiner	{r,a}	{r,a}	{r}	{w,r,a}
exams office	{r}	{r}	{}	{r}
examination centre	{l}	{}	{}	{}

c)



level 0 = candidates and examination centre
level 1 = exams office
level 2 = second examiner
level 3 = first examiner

d) Candidates cannot do anything but have limited access read to the examination paper. This is enforced by the protocol. The examination centre can only read the examination paper. However, they could open the papers early and tell the candidates what is in the examination. They are trusted not to do so.

The University examinations office could alter the paper before printing and distribution. They could also alter the marks entered on the database. They are trusted not to do so.

The second examiner reports back to the lead examiner so cannot alter the examination paper without the knowledge of the lead examiner. This is enforced by the protocol. However, the second examiner could alter the marks given by the lead examiner when second marking. They could also alter the scripts written by the candidates. They are trusted not to do so.

The lead examiner has permission to make whatever changes s/he wants. However, they are trusted not to alter the first marks of the second examiner or the candidates' scripts.

e) Extra steps that could be added to the protocol include:
- University examinations office sends final version of the examination paper to lead examiner along with examination centres so that s/he can check its integrity.
- Examination papers to be sent to examination centres in sealed tamper proof packages which are only to be opened in the examination room.
- Candidates' scripts to be sealed in tamper proof packages when being sent to (and from) examiners so that scripts cannot be altered or candidate numbers switched.
- Lead and second examiner to meet to agree on final marks so that neither one of them can alter a mark without the others' knowledge.
- A copy of the final marks to be kept by both examiners and then checked against the data input onto the University database to ensure integrity.

B.2.4 Chapter 4

Question 1

a) Blocksize means the size of the block that messages are split into before encryption. Keyspace means the set of all possible keys that can be used when implementing the cryptosystem.

b) A large alphabet is required to prevent statistical analysis of the ciphertext revealing the message. A large keyspace is required to prevent a cryptanalyst finding the message and the key by doing an exhaustive key search.

c) A *ciphertext only attack* is when the cryptanalyst knows only the ciphertext. In this case he might try an exhaustive key search until he finds a message that is meaningful. A *probable known message attack* is when the cryptanalyst knows the ciphertext and guesses part of the message. He starts the attack by trying to find the key for the parts of the message that he guesses. A *chosen message attack* is when the cryptanalyst creates a message of his own choice and then persuades Alice to encrypt the message and give him the ciphertext. A *chosen ciphertext attack* is when the cryptanalyst creates a piece of ciphertext of his own choice and then persuades Bob to decrypt it and give him the corresponding plaintext. A *chosen message and ciphertext attack* is when the cryptanalyst can choose messages both for encryption and decryption. In this case (and for chosen message or chosen ciphertext attacks) he is trying to discover the key that is used rather than the message.

B.2.5 Chapter 5

Question 1

a) It is essential to keep the encryption key secret in a symmetric key cryptosystem because the decryption key is the same as, or can be easily derived from, the encryption key.

b) In a symmetric key cryptosystem, the term *blocksize* means the number of bits in each block that is encrypted, and the term *keysize* means the number of bits in the key. It is important for a symmetric cryptosystem to have a large blocksize to prevent statistical analysis of the ciphertext and a large keysize to prevent exhaustive key search attacks.

c) In a block cipher the plaintext is encrypted block by block, but in a stream cipher the plaintext is encrypted bit-by-bit in a continuous stream.

d) Stream ciphers are used in implementations where a lot of interference might be expected such as mobile telephone communications because if one bit of ciphertext is corrupted this does not have an impact on the rest of the ciphertext.

e) A block cipher should have the property of *diffusion* which means that a small change in the plaintext should produce a large change in the ciphertext. This is to prevent an attack called differential analysis whereby the cryptanalyst chooses plaintext messages which differ by only a small amount each time and conducts statistical experiments on the resulting ciphertext to discover information about the key. A block cipher should have the property of *confusion* which means that a key which is nearly right should not give this information away. This makes it much harder to do a key search attack. And a block cipher should also have the property of *completeness* which means that every bit of the ciphertext should depend on every bit of the key. This prevents a cryptanalyst from performing a divide and conquer attack whereby he tries to find part of the key independently of the rest of the key.

f) Advantages of Rijndael compared with Triple DES are its flexibility in terms of the keysize, number of rounds, and the fact that Rijndael can be implemented in both hardware and software. Disadvantage of Rijndael are that it is relatively new and therefore has not been subjected to the same amount of scrutiny that DES has. Unlike DES, a Rijndael encryption can be written in terms of mathematical equations which means that if these equations could be solved, the Rijndael cryptosystem would be broken.

Question 2

a) The value of b needs to be large in order to prevent a successful statistical analysis attack on the cryptosystem? With modern algorithms the smallest value of b that could guarantee security is estimated to be 128-bits which is the blocksize required for the symmetric cryptosystems submitted to be the new encryption standard.

b) If Alice wanted to send Bob a message encrypted using *Double-XYZ*, she and Bob would have to share two *XYZ* keys K_1 and K_2. Alice would encrypt the message M using the *XYZ* encryption algorithm with K_1 to get C_1. Then she would encrypt C_1 using the *XYZ* encryption algorithm with K_2 to get C_2. Alice sends C_2 to Bob.

On receiving C_2 from Alice, Bob decrypts C_2 using the *XYZ* decryption algorithm and key K_2 to get C_1. He then decrypts C_1 using the *XYZ* decryption algorithm and key K_1 to get M.

[Note that you could equally well have Alice encrypt and then decrypt, followed by Bob encrypting and then decrypting. It does not affect the security of the algorithm. The important thing is that the steps match up so if Alice encrypts in her first step with K_1 then Bob must decrypt in his last step with K_1.]

c) i. Encrypting one block using *Double-XYZ* requires two *XYZ* encryptions so it will take twice as long i.e. 20 seconds.

ii. Since the blocksize for *Double-XYZ* is the same as the blocksize for *XYZ* it will take the same amount of time to transmit one block of ciphertext using *Double-XYZ* i.e., 0.01ms.

iii. The key space has grown from size 2^k to size $(2^k)^2$. If it takes 100 hours to search through 2^k keys, it will take $\frac{100}{2^k}$ hours to look at each key so it will take approximately $(2^k)^2 \times \frac{100}{2^k} = 100 \times 2^k$ hours to do an exhaustive key search for a *Double-XYZ* key.

[6]

B.2.6 Chapter 6

Question 1

a) The four essential properties for a cryptographically strong hash function h which takes input x to produce output $y = h(x)$ are:

1. Given x it is easy to compute $h(x)$. This is so that computations involving $h(x)$ are fast and efficient to perform.
2. The input x can be of arbitrary length. This is so that any message can be hashed.
3. Given a value y, it should be hard to find an x such that $y = h(x)$. This is to ensure message integrity. A message x cannot be replaced by another with the same hash value.
4. It is hard to find two different values x_1 and x_2 such that $h(x_1) = h(x_2)$. This is to prevent non-repudiation. A user cannot send message x_1 and then later claim to have sent x_2.

b) The SHA-512 hash algorithm satisfies properties 1 and 2. Statistical and experimental tests show that it also satisfies properties 3 and 4 but there is no mathematical proof of this.

c) A cryptographically strong hash function h which takes input x and output $y = h(x)$ satisfies the properties that given a value x it is easy to compute $y = h(x)$, but given y it is hard to find x such that $y = h(x)$. Therefore a cryptographically strong hash function is a one-way function.

Instead of storing unencrypted passwords in a password file, the passwords are hashed and then hash of the password is stored in the password file indexed by username. When a user logs into the system and enters their password, the system hashes the given password and checks to see whether the resulting hash is equal to the hash stored with the given username. If these match then the user is authorised, otherwise the user is not authorised to access the system. If a cryptanalyst gets hold of the password file, they will not be able to read the passwords because only the hashes are stored. However, rainbow tables which store hashes generated by well known hash functions have been created and these can be searched very quickly in order to find the password (or passwords) which generate each hash. This means that passwords stored in this way can be easily retrieved if they are lost by their user. However, it also means that a cryptanalyst attacking the system can also find passwords in a matter of seconds.

B.2.7 Chapter 7

Question 1

a) i. $O(b^2)$

ii. The numbers are three times bigger so it would take $3^2 = 9$ times as long. This is $20 \times 9 = 180\mu s$.

b) i. To compute x^n mod m:

Initialise $y = 1, u = x$ mod m
Repeat
    if n mod 2 = 1 then $y = y \times u$ mod m
    $n = ndiv2$
    if $n \neq 1$ then $u = u \times u$ mod m
Until $n = 0$
Output y

ii. For each digit in the binary representation of n there is one squaring (multiplication) step ($u \rightarrow u^2$) and there may be either one or no other multiplication ($y \rightarrow y \times u$). There are b digits in the binary representation of n and hence there are between b and $2b$ multiplication steps in the algorithm. Each multiplication is of order $O(b^2)$ and hence the entire algorithm is of order $b \times b^2 = O(b^3)$.

iii.

y	u	n
1	6	11
6	10	5
8	9	2
8	3	1
11	3	0

Therefore 6^{11} mod 13 = 11.

B.2.8 Chapter 8

Question 1

a) i. Bob generates public and private RSA keys as follows:

i. Bob generates two large primes p and q.
ii. He computes $n = p \times q$.
iii. He computes $r = (p-1) \times (q-1)$.
iv. Bob chooses a random number e which is in-between 1 and r and which has no factor in common with r.
v. He computes the ***private key*** d by solving the equation $(e \times d) = 1$ mod r.
vi. He carefully disposes of the values of p, q and r.
vii. Bob keeps d private, and publishes the value of the pair (e, n). This is his ***public key***.

ii. If Alice wishes to send Bob a message m she takes the following steps:

i. Alice looks up Bob's public key (e, n) in a public key directory.
ii. Alice breaks the message m up into blocks $m_1, m_2, ...$ so that each block of the message is a positive value less than n, i.e. $0 \leq m_i < n$.
iii. For each message block m_i, Alice computes the ciphertext $c_i = m_i^e$ mod n and sends the value of c_i to Bob.

iii. When Bob receives Alice's encrypted message $c = c_1, c_2, ...$ he takes the following steps to decrypt it. He decrypts the message using his private key d. For each ciphertext block c_i Bob computes $m_i = c_i^d$ mod n.

iv. The security of RSA is based on the difficulty of factorising a large composite number n. In order to compute the private key d, you need to know p and q, the factors of n, but given only the public key (n, e) it is a very hard problem

to find p or q and therefore to find r or d. The prime numbers p and q should be at least 200 decimal digits in length to ensure that the modulus $n = p \times q$ cannot be factorised.

b) Solution 1: Since $n = 65$ is a small number, it is easy to see that the two primes must be $p = 5$ and $q = 13$. Therefore $r = 4 \times 12 = 48$. Check that $e \times d = 1 \bmod r$: $5 \times 29 = 145 = (3 \times 48) + 1 = 1 \bmod 48$. Therefore the private key $d = 29$ is correct.

Solution 2: Choose any value of m and then encrypt it using the public key. Decrypt the corresponding ciphertext using the given private key. If the original message m is recovered then the private key must be correct. For example, with $m = 2$ we have $c = 2^5 \bmod 65 = 32$. To decrypt $c = 32$ we compute $32^{29} \bmod 65$. Using the algorithm for exponentiation as below we see that $m = 2$ as required so $d = 29$ is the correct decryption key.

y	u	n
1	32	29
32	49	14
32	61	7
2	16	3
32	61	1
2		0

c) Alice computes $c = 11^5 \bmod 65 = 161051 \bmod 65 = 46$.

d) Bob computes $m = 19^{29} \bmod 65 = 54$ using the algorithm for modular exponentiation:

y	u	n
1	19	29
19	36	14
19	61	7
54	16	3
19	61	1
54		0

B.2.9 Chapter 9

Question 1

a) The discrete logarithm problem is to find x such that $y = g^x \bmod p$ given the values of p, g and y.

b) **El Gamal key generation**

Alice generates public and private El Gamal keys as follows:

(a) Alice chooses a large random prime number p (p should be approximately 100 decimal digits or more to ensure security with current technology).

(b) She finds a generator g for the prime p.

(c) She picks a random number a which is between 2 and $p - 2$. This will be Alice's private key.

(d) She computes $y = g^a \bmod p$.

Alice publishes the public key (p, g, y) and keeps the private key (p, g, a) a secret.

lnewpage **El Gamal encryption**

If Bob wants to send Alice a message encrypted using the El Gamal cryptosystem he will first look up Alice's public key (p, g, y). Next Bob must break the message, m, up into blocks $m_1, m_2, ...$ with each message block m_i having a value which is less than p. For each message block m_i Bob performs the following steps:

(a) He generates a random number k which is between 2 and $p - 2$.

(b) He computes r, x and c where:

$$r = g^k \bmod p$$
$$x = y^k \bmod p$$
$$c = (m_i \times x)$$

(c) Bob sends Alice the values (r, c) and carefully discards the values of k and x.

El Gamal decryption Alice receives the ciphertext pair (r, c) from Bob. She decrypts the ciphertext using her private key (p, g, a) as follows:

(a) She computes $x = r^a \bmod p$.

(b) She finds the inverse of $x \bmod p = x^{-1}$.

(c) She computes $m_i = (c \times x^{-1}) \bmod p$.

El Gamal security

The decryption protocol works because Alice can use her private key a to recover the value of x without knowing the value of k. This is because $r^a \bmod p = (g^k)^a \bmod p = (g^a)^k \bmod p = y^k \bmod p = x$.

Only Alice knows her private key a and only Bob knows the random key k hence only Alice is able to recover the value of x once k has been discarded, and so only Alice can decrypt the ciphertext.

A cryptanalyst might know the values of p, g, y, r and c, but they would have to solve the discrete logarithm problem of finding a such that $y = g^a \bmod p$ in order to find the private key; or solve the discrete logarithm problem of finding k such that $r = g^k \bmod p$ in order to decrypt the ciphertext without knowing the private key.

So long as the prime and generator are chosen with care and the prime is at least 100 decimal digits in size, there is no known algorithm to solve a discrete logarithm problem.

c) A disadvantage of El Gamal is that the ciphertext is twice as long as the message, whereas for RSA the ciphertext is approximately the same size as the message. The encryption and decryption algorithms are different for El Gamal, whereas they are the same for RSA. This means that it is very easy to use RSA to generate digital signatures. El Gamal has never been patented so has been free to use, whereas RSA had a US patent until 2000. El Gamal can be used commutatively and the p and g values in the keys can be shared. This makes it ideal for use in game protocols such as mental poker. The parameters in RSA keys cannot be shared. El Gamal can be used generalised to a cryptosystem based on the elliptic curve discrete logarithm problem, and this is thought to be even harder to solve.

d) Suppose Alice and Bob need to agree on a key to use in a symmetric key cryptosystem. They choose a large prime number p and a generator g. The values of p and g are not secret and can be openly transmitted and shared between Alice and Bob.

Then Alice and Bob simultaneously take the following steps:

	Alice		Bob
1.	Generates a random number a between 2 and $p-2$.	1.	Generates a random number b between 2 and $p-2$.
2.	Computes $x = g^a \bmod p$.	2.	Computes $y = g^b \bmod p$.
3.	Sends x to Bob.	3.	Sends y to Alice.
4.	Receives y from Bob.	4.	Receives x from Alice.
5.	Computes $k = y^a \bmod p$.	5.	Computes $k = x^b \bmod p$.

Now Alice and Bob both know the value of the same secret key k although they have never transmitted the values of a, b or k. If a cryptanalyst intercepted the values of g, p, x or y, they would have to solve a discrete logarithm problem in order to find a, b or k.

e)

	Alice		Bob
1.	Generates random number $a = 5$	1.	Generates random number $b = 10$
2.	Computes $x = 3^5 \bmod 17 = 5$	2.	Computes $y = 3^{10} \bmod 17 = 8$
3.	Sends $x = 5$ to Bob.	3.	Sends $y = 8$ to Alice.
4.	Receives $y = 8$ from Bob.	4.	Receives $x = 5$ from Alice.
5.	Computes $k = 8^5 \bmod 17 = 9$	5.	Computes $k = 5^{10} \bmod 17 = 9$

B.2.10 Chapter 10

Question 1

a) The Needham-Schroeder protocol for exchanging a session key between Alice and Bob is as follows:

Step 1: Alice sends the server the names of Alice and Bob to request that a session key be generated.

Step 2: The server sends to Alice the following three items all encrypted using K_{AS}:

i. The name of Bob.

ii. A session key for Alice and Bob to share.

iii. The name of Alice and the session key both encrypted using K_{BS}

Step 3: Alice uses her key K_{AS} to decrypt the three items sent to her in Step 2. Alice now knows the session key K_{AB}.

Step 4: Alice sends Bob the value of 2c) which is the name of Alice and the session key K_{AB} encrypted with K_{BS}.

Step 5: Bob decrypts the name of Alice and the session key using his key K_{BS}. Now Bob knows the session key K_{AB} which he uses to communicate with Alice.

b) i. The session key is always encrypted when it is transmitted, either with K_{AS} or with K_{BS} or with both. Since Charles does not have these keys he cannot decrypt the session key.

ii. Charles could masquerade as Alice and send a message to the Server to initiate a request for a session key K_{AB}. However, when he receives a reply from the receiver it will be encrypted with K_{AS} and so it is of no use to Charles.

iii. Charles could intercept the message meant for Bob, but he will not be able to decrypt it because he does not know K_{BS}.

iv. Charles could prevent Bob from receiving any information from Alice, but Alice and Bob should agree to perform a handshake where they exchange the encrypted value of a nonce once they both have their key K_{AB} so that Alice can be sure that she is communicating with Bob and not Charles.

v. Alice should use a nonce as part of her original request to the Server. This nonce value is only used once and includes a time-stamp. If Charles replays this request for a key suspicions will be aroused as the nonce used will not be original.

c) Alice could look up Bob's public key in a public key directory and use this key to encrypt the session key. Bob will be able to use his private key to decrypt the session key. Since only Bob knows his private key, only he will be able to decrypt the session key. The advantage of using this method to transmit a session key is that there is no need for a trusted third party to be involved as in the Needham-Schroeder protocol. However, Alice does need to ensure that the public key she uses does belong to Bob and not to Charles masquerading as Bob.

B.2.11 Chapter 11

Question 1

a) The following steps are taken in order to send an encrypted, signed, compressed message from Alice to Bob using PGP. We assume that both Alice and Bob have agreed on which public key and which symmetric key cryptosystem to use as well as a hash algorithm. Alice and Bob both have public and private keys for use with the public key cryptosystem and have previously signed each others' keys so they each know that the public keys they are using are authentic.

- Alice hashes the message and creates a message signature by encrypting the resulting hash with her private key:

$$sig = pk.encrypt_{A_{priv}}(SHA(m))$$

Explanation: Alice hashes the message before signing it because the hash can be encrypted as one block. She uses her private key to create the signature and only she has access to this key so only Alice can create a genuine signature which will be correctly decrypted with the corresponding public key.

- Alice compresses the original message using the ZIP algorithm $M = ZIP(m)$.
Explanation: The ZIP algorithm compresses the message making it faster to encrypt. It also reduces the redundancy in the message making it harder for a cryptanalyst to perform successful statistical attacks on the ciphertext. This is why the message is compressed before encryption.

- Alice generates a session key, K, and uses it to encrypt the compressed message and the signature.

$$c = sk.encrypt_K(M, sig)$$

Explanation: The session key is a pseudorandom number which is only used for this communication and is then discarded. Alice uses a symmetric cryptosystem to encrypt the message rather than the public key cryptosystem because the symmetric cryptosystem is much faster.

- Alice encrypts the session key using Bob's public key to obtain K'.

$$K' = pk.encrypt_{B_{pub}}(K)$$

Explanation: Alice has to get the session key K to Bob so that he can decrypt the message. She must encrypt the key before sending it to Bob otherwise an interceptor may be able to get hold of the key and decrypt the message. By using Bob's public key, Alice is ensuring that only Bob will be able to decrypt the session key because only Bob knows the value of his private key.

- Alice sends Bob the pair (K', c).
Explanation: Both K' and c are encrypted and give away no information about the message.

- On receiving (K', c) from Alice, Bob decrypts K' using his own private key to obtain K.

$$K = pk.decrypt_{B_{priv}}(K')$$

Explanation: Bob uses his private key to recover the session key. No-one else is able to perform this step.

- Bob decrypts the ciphertext c using the session key K to obtain M and sig.

$$(M, sig) = sk.decrypt_K(c)$$

- Bob decompresses M to obtain the original message m.

$$m = UNZIP(M)$$

- Bob now has the message m. In order to authenticate it he uses Alice's public key A_{pub} to decrypt the signature and hashes the message m. If the two results match then the message is authenticated.

$$SHA(m) \stackrel{?}{=} pk.decrypt_{A_{pub}}(sig)$$

Explanation: If the message m has been altered then the authentication step will fail. Likewise, if the signature has been altered or faked then the authentication step will fail. It is not possible for anyone to create a false message and corresponding signature because they do not have Alice's private key. Therefore this authentication step provides verification both that the message is unaltered and that it is from Alice.

c) PGP allows a user to have more than one pair of private/public key to increase security and aid in the key changeover period when a pair of keys expire. This causes a problem because if Alice encrypts a session key using Bob's public key how does Bob know which private key he should be using to decrypt the session key? Similarly, if Alice uses one of her private keys to sign a message, how does Bob know which is the corresponding public key? In the case of encryption, Alice sends Bob a key identifier which shows him which of his public keys she has used. The key identifier is typically the least significant 64 bits of the key. When creating a digital signature, Alice cannot send Bob the private key that she has used, but in this case, she sends him a key identifier for the corresponding public key.

Question 2

a) Assume that Charles thinks the message is from Alice in the first place (i.e. suppose Bob has used Alice's email account to send Charles the message). Charles will use his private key to decrypt the message. He will thus recover the message and the signature. Charles will decrypt the signature using Alice's public key. This will match the message and so the message will be authenticated as unchanged and coming from Alice. Charles will assume that Alice loves him and he will have no way of knowing that he has been misled.

Alice made the mistake of being ambiguous in her message. If she had written 'Alice loves Bob' then Bob would not have been able to play this trick on her.

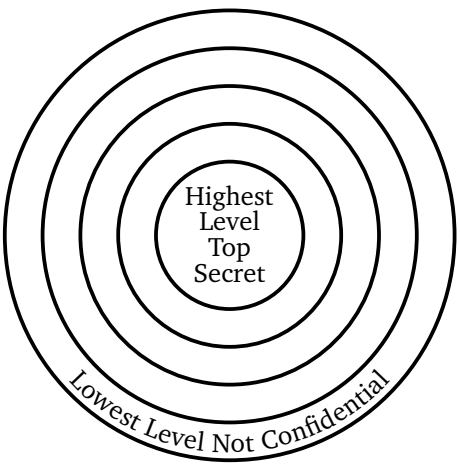
b) i) In the Web of trust model, trust is shared over a large group of people. If there are one or two corrupt people, this will affect the security of the public keys but if a CA is used and turns out to be corrupt then the entire system is insecure. A CA will charge for its services so the Web of trust model is cheaper. However, the users have to spend time and effort getting as many other users to sign their keys as possible.

ii. Alice should be more confident of E's identity in Figure 2 because in this case she personally trusts B, C and D all of whom have signed E's key. If even one of B, C or D has done this properly (i.e. has verified E's identity before signing E's key) then Alice can be sure that E is verified correctly. However, in the case of Figure 1, Alice is having to place trust in C and D, neither of whom she personally knows. If either of these has signed the next key without the correct verification then the key of E cannot be trusted to be genuine.

B.3 Solutions to sample examination paper

1. (a) The *subjects* are the chairman, treasurer, secretary and other committee members. The *objects* are the bank account, the accounts, money and cheques.

   (b) A protection ring is used when access control is hierarchical. There are a number of security levels and each subject is given a level and is allowed to access all objects at that level or lower.



   A protection ring is not suitable in the *CryptoAid* scenario because there is very little hierarchy. All the committee members have equal weight in meetings. Although three of them have the authorisation to sign for money, the accounts are not confidential and all committee members are able to view them.

   (c) There are two main points:

   - Frank has to prove to the bank that he is who he claims to be. He should have to present himself at the bank with his passport of similar identification so that the bank can verify his identity. Also Frank should make his signature in front of a bank witness so that the bank can verify that the signature is genuine. This provides the necessary *verification*.
   - Documents agreeing that Frank is to become a signatory on the *CryptoAid* account must be signed by at least two of the existing signatories. This provides the necessary *authorisation*.

   (d) It is easier to forge a signature than it is to guess a numerical code (assuming that users do not have easy-to-guess codes such as 1234).

   (e) Suppose the main key is K and p is a prime number bigger than K. Generate five distinct random numbers between 2 and $p-1$. Call these a and x_1, x_2, x_3, x_4. For i = 1 to 4 calculate k_i = $((a \times x_i) + K)$ mod p . Give each key-holder H_i a pair (x_i, k_i) and prime p.

   (f)

$$14 = 7a + K \bmod 17 \quad (multiply\ by\ 3) \quad 8 = 21a + 3K \bmod 17$$
$$11 = 3a + K \bmod 17 \quad (multiply\ by\ 7) \quad 9 = 21a + 7K \bmod 17$$

   Subtract 2nd equation from 1st equation to get $1 = 4K \bmod 17$. The inverse of 4 mod 17 = 13 (found by trial and error). Therefore K = 13.

2. (a) i.
   - Bob uses a pseudorandom number generator to generate two large primes p and q.
   - He computes $n = p \times q$.
   - He computes $r = (p-1) \times (q-1)$.
   - He chooses a random number e satisfying $2 \leq e < r$ and such that $gcd(e, r) = 1$.
   - He solves the equation $(e \times d) \bmod r = 1$ to find d.
   - He carefully disposes of the values of p, r and r.
   - He keeps d as his private key and publishes the values of (n, e) as his public key.

   ii. The cryptanalyst knows public key n and e. S/he can use the algorithm to factorise n and hence find p and q. Now s/he can find $r = (p-1)(q-1)$ and then solve $(e \times d) \bmod r = 1$ to find the private key d.

   (b) i.

characters	S	T	A	R	S	*space*
character values	19	20	1	18	19	0
block value	19(27)+20		1(27)+18		19(27)+0	
	=533		=45		=513	

$$m = 533, c = 533^3 \bmod 799 = 148$$
$$m = 45, c = 45^3 \bmod 799 = 39$$
$$m = 513, c = 513^3 \bmod 799 = 265$$

   ii. To include lower case letters we could assign
   $space = 0, A = 1, B = 2, \ldots, Z = 26, a = 27, b = 28, \ldots, z = 52$.
   We calculate the block value by the formula:
   block value = (value of 1st letter $\times$ 53) + value of 2nd letter.
   Now the largest block is 52(53) + 52 = 2808 so we have to increase the size of the key n to make it larger than 2808. We can do this by choosing larger values of p and q. The values of e and d will also have to be changed accordingly.

   iii. Since s t a r s $\leftarrow (39, 657, 507, 490, 39)$ we must have
   $(39, 657, 507, 490, 657) \leftarrow$ s t a r t.
   Each character is always encrypted to the same value. Hence this is now a substitution cipher which is very easy to break using statistical analysis.

3. (a) Step 1: Alice wants to send a message to Bob. She sends the Server her name and Bob's name.
   Step 2: The Server encrypts (Alice's name and a session key K_{AB} for Alice and Bob to use) using K_{BS} which is the key shared by Bob and the Server. Then the Server encrypts (Bob's name, K_{AB} and the encrypted message for Bob) using K_{AS} which is the key shared by Alice and the Server. The Server sends this cryptogram to Alice.
   Step 3: Alice uses K_{AS} to decrypt the message she has received from the Server. Thus she recovers the name of Bob and the key K_{AB}. She sends the message encrypted using K_{BS} by the Server in Step 2 to Bob. Bob decrypts this message using K_{BS} and now he has the name of Alice and the key K_{AB}.

   (b) A $\rightarrow$ S: A,B,N_A

   S $\rightarrow$ A: $e_{K_{AS}}(B, N_A, K_{AB}, e_{K_{BS}}(A, K_{AB}))$ A decrypts

   A $\rightarrow$ B: $e_{K_{BS}}(A, K_{AB})$ B decrypts
   B $\rightarrow$ A: $e_{K_{AB}}(N_B)$

A → B: $e_{K_{AB}}(N_B - 1)$

The value N_A used in the first two steps is a nonce or number used once which is unique and includes a time stamp. This prevents a previous message to initiate a session being replayed. Because N_A is included in the reply from the Server to Alice, Alice can assume that this is a genuine reply from the Server and not a message replay from Charles.

In the final two steps, Bob sends Alice another nonce N_B encrypted using their session key K_{AB} and Alice replies by sending back the encryption of $N_B - 1$. This *handshake* ensures that Alice and Bob are communicating with each other using their key K_{AB} as expected.

(c) Charles knows that Alice and Bob are communicating. However he cannot decrypt the message sent by the Server to Alice because he doesn't know K_{AS}. He cannot decrypt the message sent from Alice to Bob because he doesn't know K_{BS}. Charles cannot fake the handshake between Alice and Bob because he doesn't know K_{AB}. Similarly any messages encrypted using K_{AB} and sent between Alice and Bob are secure.

Charles now gets hold of K_{BS} so he can decrypt the message sent by Alice to Bob. How he has K_{AB}. He can pretend to Alice that he is Bob and can convince Alice by correctly performing the handshake.

Furthermore, Charles can send the Server a request to initiate a session key for any user of the system pretending that he is Bob. Thus Charles can fool all users of the system into communicating with him because they will believe they are talking with Bob.

When Bob discovers that his key has been compromised, he should revoke his key with the Server and get a new K_{BS}. He should also inform all users of the system that his key has been compromised and that previous messages purporting to be from Bob should be verified.

Also, starting from Step 3 of the original protocol Charles can continue to send past messages to users with Bob's name and a session key K_{BA}. So from now on, all users must check that the key K_{BA} sent to them by Bob is a new one and not from a previous communication.

4. (a) i. $f(x)$ is not a proper function.
$g(x)$ is a hash function.
$h(x)$ is a cryptographic hash function.

ii. For $g(x) = x \bmod 2^{64}$ it is very easy to find two values x_1 and x_2 such that $g(x_1) = g(x_2)$. This means that someone could send message x_1 but then later claim to have sent message x_2.
Also, given $y = g(x_3)$ it is very easy to find another value x_4 such that $y = g(x_4)$. This means that someone could change the message x_3 they have received into x_4.

iii. SHA-1 takes binary input of arbitrary length. It is partitioned into blocks of 512 bits and padded so that the last block is of length 448 bits, followed by a 64 bit integer representing the length of the input before padding. Each 512 bit block is broken down into 16×32 bit words $w(0), w(1), \ldots, w(15)$. These are expanded to 80 words $w(0), w(1), \ldots, w(79)$. Initially there are five 32 bit words h_0, h_1, h_2, h_3, h_4 which are given particular values. For each 512 bit block, SHA-1 operates on $w(0), w(1), \ldots, w(79)$ and h_0, h_1, h_2, h_3, h_4 using shifts, bitstring operations and modular arithmetic. The algorithm is a Feistel structure and there are 16 rounds. This algorithm produces new values for h_0, h_1, h_2, h_3, h_4 and these form the initial data for the next 512-bit block. After all the 512 bit blocks have been processed, the final values of

h_0, h_1, h_2, h_3, h_4 are concatenated to form a 160 bit string which is the output of the hash function.

(b) i. It is very hard for a user to correctly remember a 32-digit code. They are very likely to write this code down or use a very obvious code. This negates the security. Also, it is very hard to correctly input 32 digits and genuine users are likely to be refused access because they have not entered the code correctly. Although version A is a lot less secure, it is a lot more user-friendly and so users are more likely to correctly remember and input good strong key codes.

ii. For Version A there are 10^8 possible combinations so it will take the hacker 10^5 seconds to try every code; this is just over one day. For Version B it will take 10^{28} seconds; this is over 3×10^{20} years! For Version B2 there are 10^4 codes in each section to try. This will only take 10 seconds. The hacker has to do this for eight sections. So it will take 80 seconds in total. Therefore Version B is extremely secure (assuming users make proper use of it) but Version B2 is not very secure at all, and is not as secure as Version A.

5. (a) In a symmetric key cryptosystem pairs of users share a key. The disadvantage of this is that somehow the users have to securely get the key to each other. Also the number of keys required grows very quickly with the number of users of the system. The advantage of symmetric key cryptosystems is that they are very fast.

In a public key cryptosystem, a user holds their private key and publishes their public key for all the other system users to use. So each user only has to own one key regardless of how many people are using the system. The disadvantage of public key cryptography is that it is relatively slow and expensive. It is arguably more secure than symmetric key cryptography.

(b)
- Bob receives (c, sig, k') from Alice.
- Bob decrypts k' using his private key to obtain K.
  $K = decrypt_{Bpriv}(k')$.
- Bob decrypts sig using Alice's public key to obtain c'.
  $c' = decrypt_{Apub}(sig)$.
- If $c' = c$ then the signature is genuine, c is authenticated as unaltered and coming from Alice.
- Bob decrypts c using K to obtain the original message.
  $m = decrypt_K(c)$.

(c) Charles intercepts (c, sig, k').

Charles uses Alice's public key to decrypt sig and obtains $c' = c$. Charles cannot decrypt c (or c') because he does not know K and he cannot decrypt k' because he does not know Bob's private key. However, Charles can create a new signature for the message by encrypting c using his own private key.
$newsig = encrypt_{Cpriv}(c)$.

Charles sends Bob $(c, newsig, k')$.

Bob decrypts k' using his private key to obtain K and decrypts $newsig$ using Charles' public key to obtain c'. Since $c = c'$ Bob will assume that the message is genuine and comes from Charles as Charles is the only one who could have signed it using his private key.

Bob decrypts c using K and recovers the plans for the money making invention. Bob assumes that these plans belong to Charles. He has no way of knowing that they originally came from Alice.

(d)
- Alice creates a signature for m using her private key to obtain
  $sig = encrypt_{Apriv}(m)$.

- Alice generates a random key K and uses it with a symmetric key cryptosystem to encrypt the message and signature. $c = encrypt_K(m, sig)$.
- Alice encrypts K using Bob's public key: $k' = encrypt_{Bpub}(K)$.
- Alice sends (c, k') to Bob.

If Charles intercepts the ciphertext (c, k') he will not be able to decrypt k' since he does not have Bob's private key and he will not be able to decrypt c as he does not have K. Therefore this protocol does prevent Charles from stealing Alice's invention.

Index

Notes

Notes

Notes

Comment form

We welcome any comments you may have on the materials which are sent to you as part of your study pack. Such feedback from students helps us in our effort to improve the materials produced for the University of London.
If you have any comments about this guide, either general or specific (including corrections, non-availability of Essential readings, etc.), please take the time to complete and return this form.

Title of this subject guide: .....

.....

Name .....

Address .....

.....

Email .....

Student number .....

For which qualification are you studying? .....

Comments

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

Please continue on additional sheets if necessary.

Date: .....

Please send your completed form (or a photocopy of it) to:
Publishing Manager, Publications Office, University of London, Stewart House, 32 Russell Square, London WC1B 5DN, UK.