APPLICATIONS OF MODERN CRYPTOGRAPHY

Technologies, applications and choices

TABLE OF CONTENTS

MANAGEMENT SUMMARY

MANAGEMENT SUMMARY

Cryptography is at the heart of a vast range of daily activities, such as electronic commerce, bankcard payments and electronic building access to name a few. It is one of the cornerstones of Internet security.

It is of key importance for modern, connected organisations to have an understanding of cryptography and its applications. When used appropriately, cryptography can play an important role in securing online services. But incorrect applications of the technology can lead to a false sense of security and can ultimately lead to the loss or disclosure of sensitive data.

This white paper aims to provide an introduction to cryptography and an overview of its real-world applications. The paper is aimed at system integration engineers, consultants, system administrators and IT decision makers.

The paper starts with a short history of cryptography and the introduction of general cryptographic principles. This is followed by a comprehensive overview of the current state-of-the-art regarding cryptography. The focus then shifts to applications of cryptography, followed by an explanation of several generically applicable procedures and policies. The paper concludes with recommendations that should be taken into consideration when using or purchasing products or systems that rely on cryptography.

INTRODUCTION

1

1 INTRODUCTION

1.1 Purpose

Cryptography is at the heart of a vast range of daily activities, such as electronic commerce, bankcard payments and electronic building access to name a few. It is one of the cornerstones of Internet security.

It is of key importance for modern, connected organisations to have an understanding of cryptography and its applications. When used appropriately, cryptography can play an important role in securing online services. But incorrect applications of the technology can lead to a false sense of security and can ultimately lead to the loss or disclosure of sensitive data.

This white paper aims to provide an introduction to cryptography and an overview of its real-world applications. To achieve this, the following subjects are addressed in this white paper:

- A high-level introduction to cryptography
- A discussion of cryptographic techniques and technologies
- An overview of applications in which cryptography plays an important role
- An introduction to policies and procedures that can play a role when cryptography is used
- A set of product selection criteria for products that rely on cryptography
- A short guide to further reading for in-depth information related to cryptography

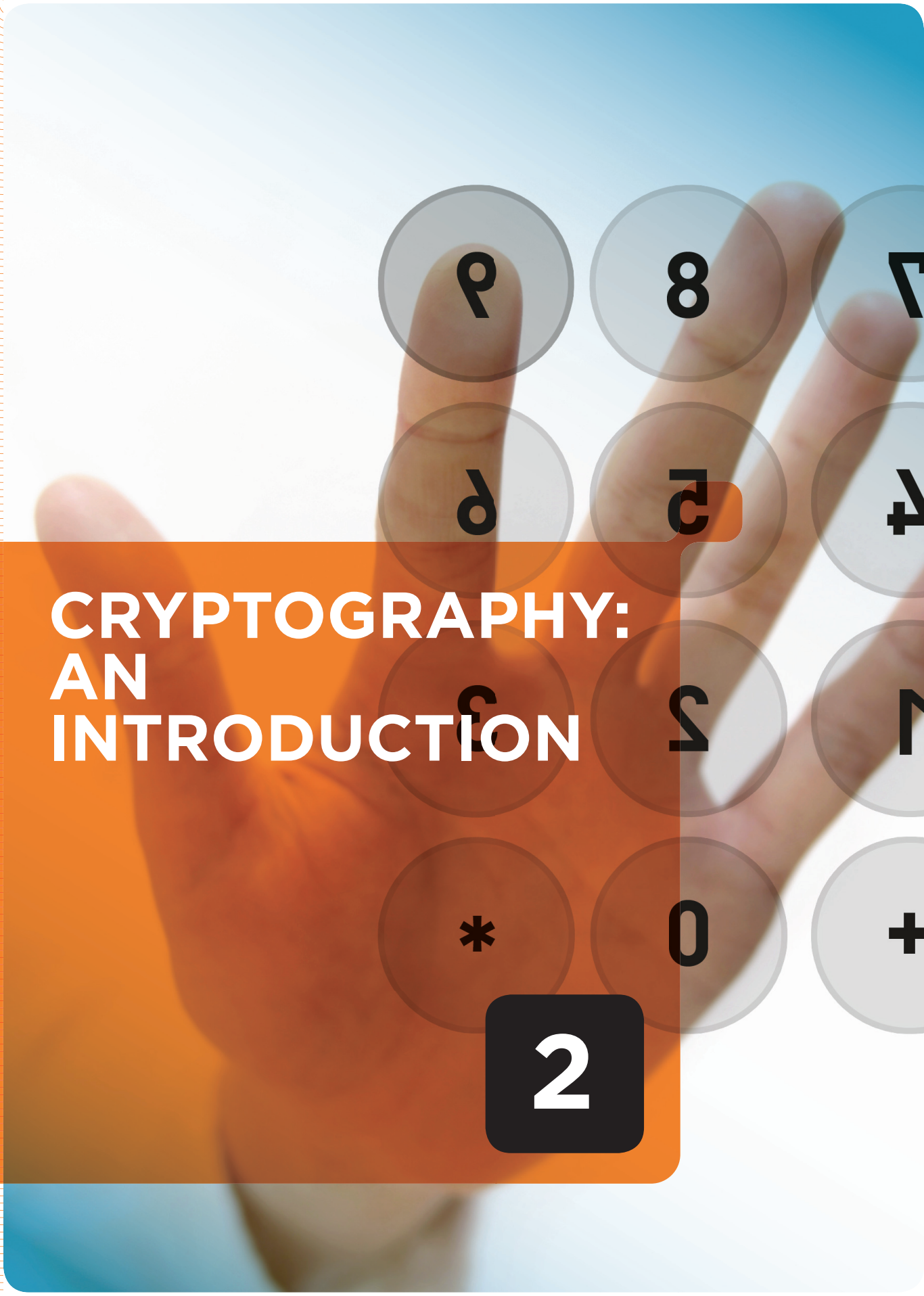
1.2 Required knowledge and intended audience

Readers of this document are assumed to have a working knowledge of basic system administration and Internet technologies. No prior knowledge of cryptography or academic mathematics is required (the intention is for this document to be an accessible source of practical information on cryptography, as such, cryptographic theory will not be explained in detail).

The intended audience of this document includes system integration engineers, consultants, system administrators and IT decision makers.

1.3 Reading guide

- Chapter 2 provides an introduction to cryptography. It starts by giving a definition of cryptography, then goes on to discuss the historical perspective of cryptography and ends with a brief description of basic cryptographic principles.
- Chapter 3 describes the state-of-the art of cryptography. It provides an overview of the cryptographic algorithms that are currently in use.
- Chapter 4 discusses a set of common applications and details how these applications rely on cryptography for their operation.
- Chapter 5 describes common policies and procedures that relate to cryptography (such as for instance key backup and escrow).
- Chapter 6 contains a set of guidelines for decision making when choosing products that provide cryptographic functionality or rely on cryptography. The emphasis is on judging the quality of the cryptography used in products.
- Chapter 7 draws general conclusions and contains an afterword.



CRYPTOGRAPHY: AN INTRODUCTION

2

2 CRYPTOGRAPHY: AN INTRODUCTION

2.1 What is cryptography?

The Compact Oxford English Dictionary [6] defines cryptography as:

cryptography
- **noun** the art of writing or solving codes

This is a very concise description of cryptography that does not do justice to the wide range of applications that cryptography has in the modern world. A more accurate definition would be:

'Cryptography is the art of concealing information from eavesdroppers by means of a secret that is only known to the communicating parties'

But even this description does not cover all applications of cryptography satisfactorily as will become evident in this paper.

In modern times, cryptography is almost always used to refer to electronic scrambling of data, but in a historical context cryptography refers to using written secret codes. This chapter contains a brief history of cryptography and an abstract introduction to cryptographic principles.

2.2 Historical perspective

2.2.1 Ancient and mediaeval times

Cryptography is known to have been in use from the earliest onset of civilisation. Early examples of cryptographic techniques date back as far as ancient Egypt and Mesopotamia [5]. A well-known example of an early cryptographic technique is the so-called *'Caesar Cipher'* [8]. This cipher is based on the substitution of letters in the alphabet by letters further on in the alphabet; in order to make this work, the alphabet is treated as cyclic, i.e. the letter following 'Z' is 'A'. Figure 1 shows the *Caesar Cipher* with a shift of 2:

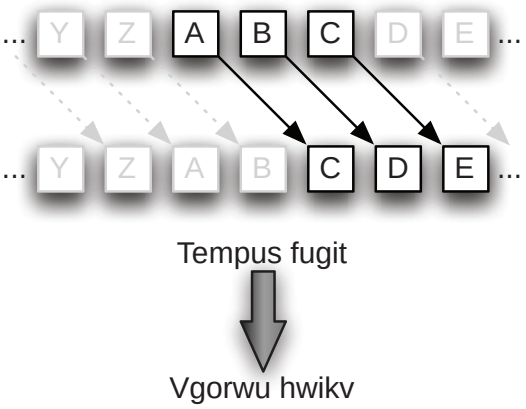


Figure 1 - Caesar Cipher with a shift of 2 to the right

The transformation of the Latin sentence *'Tempus fugit'* (time flies) is shown. The *Caesar Cipher* was apparently used by Julius Caesar to communicate with his generals in the field. The secret that both parties share in this case to communicate securely is the shift that was used to create the *cipher* text from the *plain text.* It is fairly obvious that the *Caesar Cipher* is not very secure, as it is trivial to determine the shift that was used by brute force (i.e. trying all possible 26 shifts to see if this yields intelligible text).

The Caesar Cipher is more generally known as a simple substitution cipher; it uses a fixed way to substitute one character for another. Even if the substitution does not rely on shifts (like in the Caesar cipher) but instead uses a more or less random substitution table, it is still very easy to break by brute force. A common method used relies on counting the number of occurrences of letters in the cipher text. Since every letter is substituted by exactly one other letter every time, the number of occurrences of a letter in the plain text is preserved in the cipher text. Certain letters occur much more frequently in most languages than others, making it possible to guess – based on the occurrence frequency – certain substitutions (for instance: the letter 'e' occurs more frequently in English than other letters).

Mediaeval times saw the advent of slightly more complex ciphers called *'polyalphabetic ciphers'.* These ciphers are based on an algorithm that uses different substitution methods for different parts of the message, which means that it becomes much harder to recover the *plain text* if one does not know the substitution method that was applied. They also hide character frequencies, making it impossible for attackers to rely on counting the number of occurrences of a character.

A common principle that holds for historical ciphers is that they usually rely on what is called *'security by obscurity'.* This means that the security of the cipher relies on the algorithm remaining a secret.

2.2.2 From secret algorithms to secret keys

By the end of the 19th century it became generally accepted that relying on keeping a cryptographic algorithm secret to preserve its security was bad practice. This was explicitly stated in Kerckhoffs' Principle [9], which states that a *cryptosystem* should be secure even if every aspect of the system, except the key, is publicly known.

The modern perspective on this is even stronger: it is now generally accepted that cryptosystems become more secure by making the underlying algorithm publicly available to be studied. Such study is called *cryptanalysis*[1]. Any weaknesses that are discovered are invariably published in peer-reviewed journals and lead both to the users of algorithms being warned about potential weaknesses, giving them the opportunity to switch to other systems and to general improvements in the science of cryptography.

2.2.3 World War II

Cryptography played an important role in World War II. In the years preceding the war, both the Allies as well as the Germans were investing heavily in building cryptosystems. Many of these systems were based on a combination of mechanical and electronic systems. The most well known example is the Enigma [7], [11]. An example of an Enigma machine is shown in Figure 2.

Figure 2 - A three-wheel Enigma machine

The Enigma machine implements a *symmetric key algorithm* (i.e. the same key is used to both encrypt as well as decrypt the message, for more information see §3.1). The Enigma machine is electromechanical and uses rotating wheels (rotors) in its operation. The initial setting of the *rotors* is used together with a daily code and wire settings to determine the key that is used. All branches of the German armed forces used the Enigma (albeit in different implementations).

The most well known application of Enigma machines was by the German U-boot fleet in the Battle of the Atlantic [12]. An important aspect of this battle was the Allied effort to crack the Enigma cipher. The British performed cryptanalysis on encoded Enigma messages at the Bletchley Park[2] facility near Milton Keynes. One of the key players in this effort was Alan Turing. He was instrumental in devising the first electronic computers that could be used to crack the Enigma code. Much of the groundwork that would allow the development of modern computers was done at Bletchley Park in an effort to break Enigma.

2.2.4 The digital era

With the advent of modern digital computers information security made several major leaps. On the one hand, cryptographic algorithms could be made – and became – much more complex. In the past, algorithms were chiefly based on substitution schemes and could only work with written text; nowadays algorithms are based on mathematical principles and can be used to encrypt any input data. On the other hand, the science of *cryptanalysis* also made some major leaps and algorithms that were previously thought to be unbreakable are now considered trivial to break.

Arguably the most important invention in modern cryptography is asymmetric cryptography. *Asymmetric cryptography* allows sharing information publicly (a public key) that can be used to encrypt data that can then only be decrypted by someone owning the corresponding secret key (the private key). More information about *asymmetric cryptography* can be found in §3.2.

Finally, recent years have seen the development of quantum cryptography, more information about which can be found in §3.4.

1 Cryptanalysis is the study of breaking encryption algorithms, see [13]

2 Bletchley Park has a visitor's centre, see http://www.bletchleypark.org.uk/

2.2.5 Further reading

An excellent overview of the history of cryptography with lots of photographs and detailed explanation can be found at the online Museum of Cryptography: http://www.cryptomuseum.com/index.htm.

2.3 Cryptographic principles

Most cryptographic algorithms and cryptosystems share common principles. This section aims to introduce and briefly explain some of these principles.

2.3.1 Definition of a secure cryptographic algorithm

One of the main goals of cryptography is to guarantee the secrecy of data. In order to achieve this, a cryptographic algorithm should be 'secure'. This raises the question what a secure algorithm is. Claude Shannon, the father of information theory, coined the term *perfect secrecy* (see [115]). He defined this as follows:

'Perfect Secrecy is defined by requiring of a system that after a cryptogram is intercepted by the enemy the a posteriori probabilities of this cryptogram representing various messages be identically the same as the a priori probabilities of the same messages before the interception.'

Though this may sound complicated, when paraphrased it is actually rather simple: perfect secrecy means that even if the encrypted message is observed, this should in no way increase the likelihood of uncovering the plain text.

2.3.2 A system's security should not rely on the system itself remaining a secret

This is a rephrasing of Kerckhoffs' Principle (introduced in §2.2.2)[3]. In effect, his was the first modern perspective on cryptography; this principle still applies to the cryptographic algorithms in use today.

A phrase that is often used by the security community in this context is *'security by obscurity'* – that the security of a system is based on the workings of the system remaining secret. The general consensus of the security community is that relying on security by obscurity is a very bad practice. Time and again relying on *security by obscurity* has been proven to fail, either because someone was compelled to reveal the obscured design or because it was reverse engineered. Once this is done, cryptanalysis can be performed on the system and since no peer-reviewed research was performed in the past, weaknesses are invariably discovered (see also §2.2.2).

A poignant example of a system being compromised because it relied on *security by obscurity* is the hacking of the Dutch public transport chip card (OV-chipkaart) by security researchers. The MIFARE Classic cryptographic algorithm used in this card was secret. Researchers, however, managed to reverse engineer the algorithm [10] and devise an efficient attack on the system that could easily be used to break the security of the chip [69].

Figure 3 - An 'OV-chipkaart' public transportation pass

2.3.3 Mathematics

All modern cryptographic algorithms rely on mathematical principles. The most important of these is that the maths that the algorithm relies on should be based on a problem that is very hard to solve by brute force (i.e. without knowing the secret information).

A second principle, derived from this, is that a good cryptographic algorithm should be efficient in use (i.e. not consume excessive CPU time and/or memory) but should be inefficient (requiring excessive or infeasible amounts of CPU time) to break.

2.3.4 Random number generation

Modern cryptographic algorithms depend on random data generation, for instance for key material. It is therefore very important to use a good random number generator.

Most random number generators used on computers today are what are called *'pseudo random number*

3 Another well-known reformulation of this principle is Shannon's maxim: 'the enemy knows the system', see [14]

4 The term 'brute force' is often used in conjunction with breaking cryptographic algorithms to indicate determining the key by trying all possible values; the nature of an algorithm should be such that a brute force attack should be computationally infeasible (i.e. impossible to carry out in an acceptable amount of time)

generators' (pseudo RNGs) [16]. A pseudo RNG is based on a mathematical formula that produces output that should approximate truly random data as best as possible. A pseudo RNG always takes some input data as a starting point (called a *seed*) from which it then starts generating data. All pseudo RNGs are predictable; given a certain seed they will always produce the same output sequence. Furthermore, they have a periodicity (i.e. after a certain number of cycles they will start reproducing the same output data). A special subclass of pseudo RNGs is used in *cryptography called cryptographically secure pseudo random number generators*. These are pseudo RNGs that have been designed according to certain principles that make them suitable for use in cryptographic applications. Examples of these include the pseudo RNG used in Microsoft CryptoAPI (part of the Windows operating system) and the Yarrow algorithm (included in Mac OS X and FreeBSD).

Because of their inherent predictability it is of key importance to provide sufficient seeding material to pseudo RNGs. The seeding data should in turn be 'as random as possible'. To achieve this, many implementations use physical events, such as the user moving the mouse or typing on the keyboard. In general, the following principle holds: the more truely random data is input into a pseudo RNG, the more secure the output becomes.

Hardware random number generators also exist. The problem with these usually is that they are relatively slow (when compared to software pseudo RNGs). It is not uncommon to use the output of a (slower) hardware RNG to seed a software pseudo RNG.

Summarising: the security of modern cryptographic systems heavily depends on the quality of the random data that is used in their operation.

In addition to this, randomness plays another role in cryptography. An important property of a good cryptographic algorithm is that its encrypted output data should – to all intents and purposes – be indistinguishable from random data (i.e. should have no content that can be leveraged to derive any information on the original input data from the encrypted output)[5]. If some of the structure of the plain text is preserved in the cipher text then this can provide attackers a foothold that they can use to break the code (recall the letter occurrence counting attack on simple substitution ciphers described in §2.2.1).

2.3.5 Bits of key strength

Cryptographic algorithms invariably use keys. The size of these keys is usually expressed in bits (e.g. 128 bits or 1024 bits). It can be very hard to compare the cryptographic strength of different cryptographic algorithms. To overcome this problem, the concept of "bits of key strength" is often used. This number expresses the number of different keys that would have to be tried to break an algorithm by brute force.

For instance: the 3DES algorithm (discussed in §3.1.3) has 168-bit keys but due to certain properties of the algorithm, a 168-bit key only delivers 112 bits of key strength. This means that in theory there are 2^{112} different keys; breaking a key by brute force requires trying at least 2^{111} different keys (the fact that this does not require trying 2^{112} keys is due to the so-called birthday paradox[6]).

Effectively, for every extra bit of key strength, the effort required to break an algorithm by brute force doubles.

The number of bits of key strength for an algorithm may decrease over time if successful attacks are discovered that weaken the algorithm (note: attacks on openly published algorithms rarely fully break the algorithm; most attacks only make it easier than before to recover a secret key). This has happened for most commonly used algorithms.

2.3.6 Alice, Bob and Eve

In many places in this document three fictional characters are encountered, Alice, Bob and Eve. Alice and Bob are the fictional characters favoured by cryptographers to describe two communicating parties. The third character – Eve – often represents an attacker. It should be noted that Alice, Bob and Eve do not necessarily represent real world people; they can also represent computer systems (servers) or entire organisations.

5 A side-effect of this is that encrypted data cannot be compressed by – for instance – tools like 'ZIP', for more information see [15]

6 http://en.wikipedia.org/wiki/Birthday_problem

STATE OF THE ART

3

3 STATE OF THE ART

3.1 Symmetric cryptography

3.1.1 General principles

Symmetric cryptography is the most widely used form of cryptography. It can be used to secure communication by two or more parties and relies on a secret that is shared between the parties. Figure 4 shows symmetric cryptography in diagrammatic form:

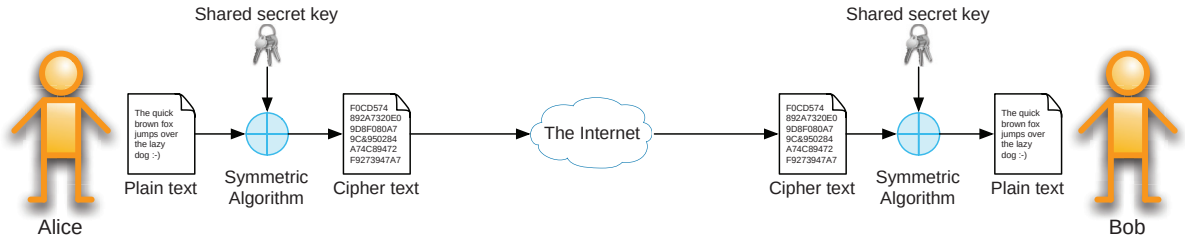


Figure 4 - Schematic overview of a symmetric key algorithm

The diagram shows Alice and Bob[7] who want to communicate securely over the Internet. To achieve this using a symmetric key algorithm they need to have a shared secret key indicated at the top of the diagram. They must have shared this key beforehand securely and *out-of-band*[8]. Note that this may be hard in the real world if Alice and Bob are – for instance – hundreds of kilometers apart (depending on the level of secrecy required, they cannot rely on things like telephones, faxes or the postal service).

Alice takes the plain text, depicted on the left and uses this together with the shared secret key as input for the symmetric key algorithm (represented by the blue plus-sign). The output of the symmetric key algorithm is the cipher text. As already mentioned in §2.3.4, the output of a good algorithm is indistinguishable from random data and can thus be transmitted safely over the Internet without running the risk of the data being intercepted and disclosed.

Bob receives the cipher text and inputs it – together with the shared secret key – into the symmetric key algorithm. The output of the symmetric key algorithm will be the plain text that Alice input on the other side of the communication.

The example above demonstrates why a symmetric key algorithm is called 'symmetric'; the same key is used in both the encryption as well as the decryption operation.

Given a good symmetric key algorithm the security depends on the shared secret key remaining secret. Once the key has been disclosed, any eavesdropper can decrypt the message traffic between the communicating parties.

Symmetric key algorithms exist in two variants:

- *Block ciphers* – these operate on groups of bits called blocks; a block cipher typically has a fixed block size that is a multiple of 8 bits (common block sizes are 64 bits and 128 bits)
- *Stream ciphers* – these operate on single bits of data

The security of block ciphers depends on the ability of the algorithm to create an output that is indistinguishable from truly random data and on the randomness of the key.

Because of the nature of block ciphers, it is almost always necessary to pad[9] data to a multiple of the block size. Furthermore, several modes of operation exist for block ciphers. The two most common ones are called *'Electronic Codebook'* (ECB) mode and *'Cipher Block Chaining'* (CBC) mode [21]. Of these two modes, CBC-mode is significantly more secure. In this mode, the current cipher block in an encryption is used as input for the encryption of the next block, thus creating a chain of cipher blocks (hence the name). The result of this is that macro structures in well-organised data (such as images) are effectively hidden ([21] shows an excellent example of this property).

Stream ciphers work in a completely different way. They are based on a perfectly concealing algorithm , the one-time pad (see §3.1.2). They work by generating a pseudo random bit stream called the

7 Alice and Bob are the fictional characters favoured by cryptographers to describe two communicating parties. A third character – Eve – often represents an attacker.

8 To communicate out-of-band means to exchange information through a different channel then the one normally used for communication. For example: if normal communication goes over the Internet (e.g. e-mail) than a suitable out-of-band channel could be by telephone or by mobile text message.

9 In computing, padding refers to adding additional data to create a block of data of a certain size. Cryptographic padding commonly contains information on the amount of padding applied so the padding can be removed upon decryption.

key stream from a starting key. This key stream is then applied to the bit stream that needs to be encrypted using a simple exclusive-or (XOR) operation[11]. This means that encryption and decryption are the same operation. Stream ciphers can be implemented in hardware extremely efficiently. And because they can operate very efficiently on streams of data of variable and unknown length (contrary to block ciphers which usually require padding) stream ciphers are used in applications such as wireless communication. An example of this is the A5/1 algorithm used for GSM telephony.

The big disadvantage of stream ciphers is that their security is heavily dependent on the quality of the pseudo random number generator that generates the key stream. A number of stream ciphers (including – for instance – RC4, which was used for SSL and Wi-Fi WEP encryption) have been broken because of this.

3.1.2 One-Time Pads (OTP)

The one-time pad is the only perfectly concealing cryptographic algorithm (i.e. it completely hides the plain text and offers no chance of recovering the plain text by brute force without trying all possible pads of the same length as the cipher text which is usually completely infeasible). The mechanism is based on using a truly random set of data that is exactly the same length as the plain text that has to be encrypted. The random data used to encrypt may only be used once (hence the 'one-time' in the name). Traditionally this random information was stored on pieces of paper (hence the word 'pad' in the name).

The one-time pad data is often used to encrypt the plain text using a simple binary exclusive-or (XOR)[11] operation; this yields the cipher text. The advantage of using the exclusive-or operation is that it is a native instruction on almost all computer systems and operates very efficiently.

The receiving party uses the same one-time pad data to perform the XOR operation on the cipher text, which will yield the plain text.

The workings of one-time pads are depicted in Figure 5 below:

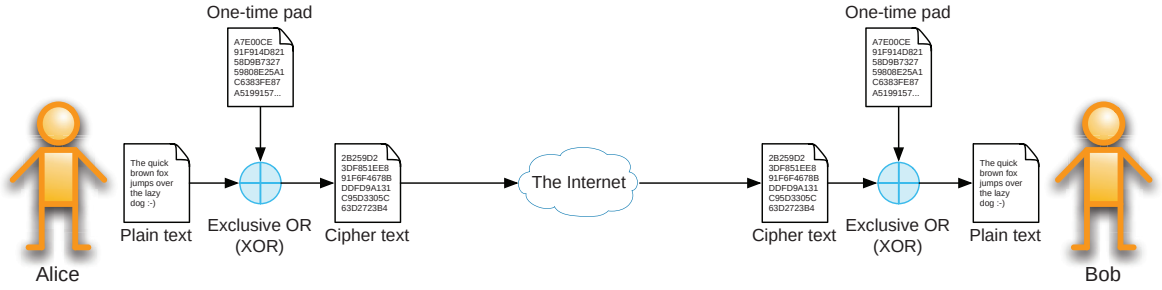


Figure 5 - The one-time pad system

Although one-time pads should in theory be completely immune to cryptanalysis they are not commonly used anymore because of several reasons:

- There have been many incidents in the past where one-time pads were re-used more than once, resulting in them being broken
- One-time pads are problematic to distribute since they need to remain absolutely secret and since they have to be the same size as the plain text that is to be encrypted
- They are only secure if truly random data is used to create them (why this is an issue is described in §2.3.4)
- They are subject to malleability issues, i.e.: flipping a bit in the cipher text always results in flipping of that same bit in the plain text. This is – for instance – and issue for messages of which the general structure may be known (such as bank transactions); an example is given in [116].

In the past, one-time pads were used in all sorts of applications, notably by spies. Figure 6 shows a picture of a one-time pad printed on a silk handkerchief

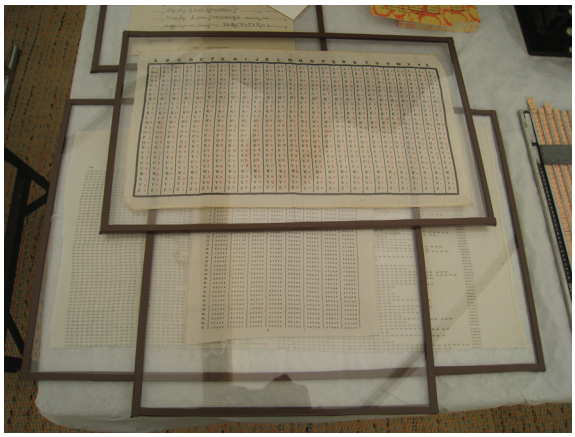


Figure 6 – Historic one-time pads[12]

Because of the drawbacks described, it is not recommended to use one-time pads or systems claiming to be based only on one-time pads.

3.1.3 Data Encryption Standard (DES)

The Data Encryption Standard (DES) was developed by IBM and the NSA[13] in the United States during the 1970s. It was standardised in 1976 by the National

Bureau of Standards[14] and was designed to protect sensitive non-classified information used by the US Government and by enterprises [17].

DES is also commonly referred to as the Data Encryption Algorithm (DEA).

The original DES algorithm uses 64-bit keys, but because some of the key bits are used for key integrity checks only 56 bits are actually used for the encryption/decryption of data. This variant of the DES algorithm is now obsolete and should no longer be used. It is considered to be insecure for most applications. The standard has been withdrawn by NIST and a 1999 joint effort between the Electronic Frontier Foundation and distributed.net demonstrated that it is possible to break a DES key through brute force in just over 22 hours (given Moore's Law, this can be done much faster by now).

A modern derivative of the original DES algorithm is the Triple DES algorithm (also called TDES, TDEA or DES-EDE) [18]. This algorithm is based on using three DES keys consecutively in encrypt, decrypt and encrypt mode[15, 16]. Triple DES works with either 112-bit keys (where the first DES key is also used as the third key, called 2TDEA) or 168-bit keys (with three separate DES keys, called 3TDEA).

Because of certain properties of the Triple DES algorithm the effective key-strength of a 168-bit key is 112 bits and the effective key-strength of a 112-bit key is 80 bits. Because of this, NIST recommends that 2TDEA should no longer be used after 2010. 3TDEA, however, is considered to be secure through 2030 [19].

Both DES and Triple DES are block ciphers. For (Triple) DES the cipher block size is 64 bits (8 bytes). DES and Triple DES are extensively used although use of the former is declining (because of the weakness mentioned above). The algorithm also has certain properties that make it possible to very efficiently implement in hardware [20]. It is therefore commonly used in embedded systems.

3.1.4 Advanced Encryption Standard (AES)

In early 1997 NIST announced that they were looking for a successor to DES and they solicited input from the cryptographic community. Because of the amount of interest this sparked, NIST decided to issue a call for new algorithms in the fall of 1997 and to hold a competition to decide on the best candidate for the standard.

Submissions ran well into 1998 and in all 15 algorithms were entered into the competition. After a public debate and two conferences organised by NIST in which the competing algorithms were analysed both for security as well as for performance a shortlist of 5 candidates was selected. Another round of cryptanalysis followed which ended with another conference in April 2000.

On October 2nd 2000 NIST announced that the Rijndael algorithm – designed by Belgian cryptographers Joan Daemen and Vincent Rijmen – had won the competition and would become AES.

The AES algorithm is a block cipher with a block size of 128 bits (16 bytes). It supports key lengths of 128, 192 and 256 bits.

AES has been thoroughly screened by the cryptographic community and no significant attacks have been found to date. NIST currently believes AES to be secure beyond 2030 [19]. Contrary to its predecessor DES – which was specifically designed for sensitive but not for secret information – AES has been approved for use in encrypting official material marked 'SECRET' with 128-, 192- and 256-bit keys and for use in encrypting official material marked 'TOP SECRET' with 192- and 256-bit keys by the United States' Committee on National Security Systems (CNSS) [22].

3.1.5 Other algorithms

The focus of this chapter is on the most commonly used symmetric key algorithms, (Triple) DES and AES. There are – of course – many other symmetric key algorithms available. The rule-of-thumb for the use of cryptographic algorithms is to follow the masses; i.e. it makes sense to use the most common algorithms since these are likely to be the best researched ones. To recommend this without mentioning a couple of other well-known algorithms, however, would not do these justice. Note that it is impossible to provide an exhaustive list; some algorithms are bound to be missing in the list below.

10 The fact that they are based on a perfectly concealing algorithm by no means implies that stream ciphers are by default secure.

11 The XOR operation is a bitwise operation that is reversible, i.e. if a bit is XOR-ed once with another bit and the result of this operation is XOR-ed again with the same other bit then the result is the original input bit, see also http://en.wikipedia.org/wiki/Exclusive_or

12 Source: http://www.fredandre.fr/images/BletchleyER2009/coton.JPG permission to reproduce the file kindly granted by Mr. Frederic André

13 National Security Agency, http://www.nsa.gov, the US signals intelligence office

14 Now known as the National Institute of Standards and Technology, NIST, http://www.nist.gov

15 Hence the acronym DES-EDE

16 For encryption; for decryption this is reversed, i.e. decrypt, encrypt, decrypt

- IDEA (The International Data Encryption Algorithm) – this algorithm is used in the commercial PGP program for data encryption (see [23])
- RC4 – this algorithm was used extensively in the past, for instance in SSL and for Wi-Fi encryption (in the WEP protocol). The WEP protocol was compromised because of issues in the RC4 algorithm; use of the RC4 algorithm is therefore no longer recommended (see [24])
- Blowfish – this algorithm was designed by Bruce Schneier. It is a popular algorithm in both the open source community as well as among independent software vendors[17] (see [25]). Schneier recommends that newer applications use Twofish instead (see below)
- Twofish – this algorithm was also designed by Bruce Schneier. It was one of the candidates for AES and is also popular among the open source community and independent software vendors[18] (see [26])
- The Tiny Encryption Algorithm (TEA) – this is a very fast and simple algorithm that has been optimised for 32-bit processors (see [27]).

If in doubt about the quality of an algorithm not listed above and used in a product it is prudent to look the algorithm up on the Internet and to attempt to find independent (i.e. not published by the algorithm's inventor or manufacturer) information about its security. If it is hard to find information assume that it is not suitable.

3.1.6 Summary

Symmetric key algorithms are a fast way to securely encrypt data using a shared secret. The two most commonly used algorithms to date are Triple DES and AES.

The table below shows the number of bits of key strength (see §2.3.5) offered by the most common algorithms described above:

Algorithm	Bits of key strength
2-key Triple DES	80 bits
3-key Triple DES	112 bits
128-bit AES	128 bits
192-bit AES	192 bits
256-bit AES	256 bits

Table 1 - Key strengths for symmetric key algorithms

Based on recommendations by NIST, the following guidelines should be taken into account when choosing one of these algorithms for use in an application:

Timeframe	Acceptable algorithms
Through 2010 (min. 80 bits of key strength)	2-key Triple DES 3-key Triple DES AES-128 AES-192 AES-256
Through 2030 (min. 112 bits of key strength)	3-key Triple DES AES-128 AES-192 AES-256
Beyond 2030 (min. 128 bits of key strength)	AES-128 AES-192 AES-256

Table 2 - Symmetric key algorithm recommendations[19]

3.2 Asymmetric cryptography

3.2.1 General principles

Asymmetric cryptography is also known as *public-key cryptography.* In fact, this refers to the most important property of asymmetric key algorithms. They always have two keys, a *public key* that can be freely shared over unprotected channels and a *private key* that needs to be kept secret. Together, these two keys form what is known as a *key pair.*

The two most important applications of asymmetric cryptography are encrypted communication without the need for a pre-shared secret and digital signatures for authentication and non-repudiation.

Assume that Bob wants to send a message to Alice. They have no previously shared secret. Instead, Bob sends Alice his public key. It is important that Alice then checks that this is indeed Bob's public key. Alice needs to do this over a trusted channel, for instance by telephone. She calls Bob and asks Bob to confirm some attributes of his public key[20] that allow her to establish the authenticity of the key.

The reason for this confirmation step is precisely because Alice and Bob have no previously shared secret. If an attacker Eve can falsify a message, claiming to be from Bob containing a public key

17 A list of applications using Blowfish can be found here: http://www.schneier.com/blowfish-products.html

18 A list of applications using Twofish can be found here: http://www.schneier.com/twofish-products.html

19 This is an excerpt of Table 4 from [19]

20 This is usually what is known as a fingerprint – a hash code computed over the public key. The PGP desktop application, for instance, can represent this fingerprint as a collection of dictionary words that can be read to a party with which one is communicating.

(that belongs to Eve) and Alice would not perform this verification then she could falsely assume to have Bob's public key. If she then sends a confidential message and encrypts it using this key, Eve could read it instead of the intended recipient, Bob.

The verification step is thus intended to establish trust in the public key and it should always be performed the first time a key is exchanged between communicating parties. Apart from one-to-one verification there are also mechanisms that involve a *web of trust* or *a trusted third party* (TTP) to achieve this. More information on this can be found in §3.2.9 and §3.2.10.

Figure 7 shows how Bob sends Alice his public key (1) and how Alice verifies the authenticity of Bob's public key over the phone (2). Bob uses a mechanism that is included in some applications that represents a fingerprint of the public key using dictionary words.

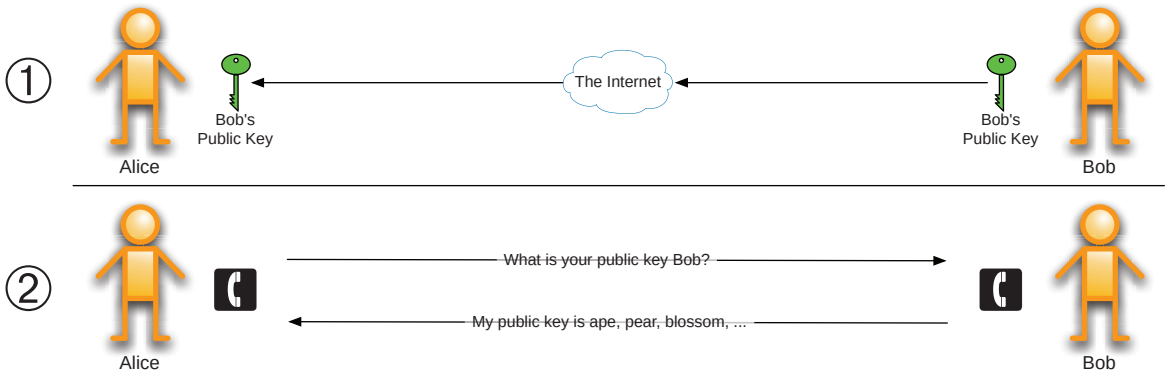


Figure 7 - Public key exchange between Alice and Bob

Once trust has been established Alice can send Bob a message securely. She does this by encrypting the message using a symmetric key algorithm with a random key and then encrypting the secret key using Bob's public key. The reason why a symmetric key is used to perform the actual encryption of the transmitted data is that most asymmetric key algorithms are unsuitable for encrypting large amounts of data. She can then send the encrypted message and the encrypted key to Bob. Only Bob can decrypt the secret key using his private key and can then decrypt the message using the decrypted secret key. Figure 8 shows Alice sending the encrypted secret key to Bob.

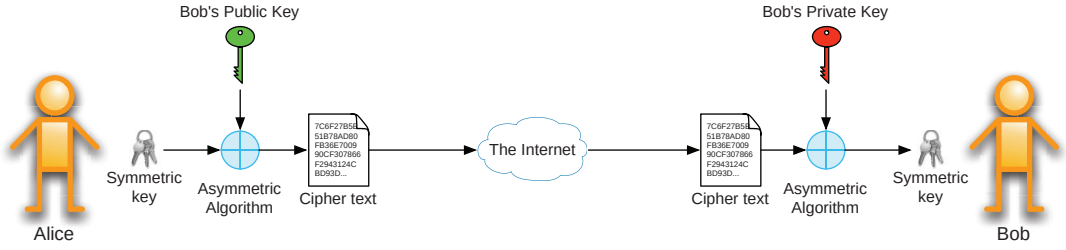


Figure 8 - Alice sends Bob a secret key for encrypted communication

Another important application of asymmetric cryptography is (message) authentication. It is possible to authenticate an entity based on what is called *'proof-of-possession'*. The entity proves that it has a certain private key belonging to a known public key, thus proving its identity. An example of this is shown in Figure 9. This figure shows Bob sending a challenge to Alice (1). Alice can then prove possession of her private key by encrypting this challenge using her private key. Bob can decrypt the cipher text using her public key and compare the challenge to what he sent thus verifying that Alice indeed possesses the private key belonging to the Alice's public key that Bob already has (2).

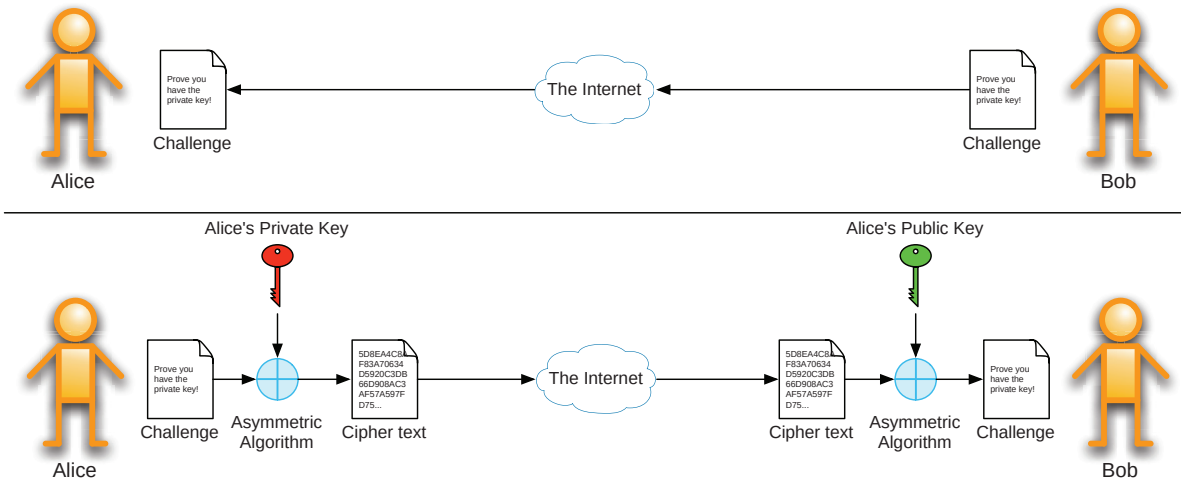


Figure 9 - Authentication based on asymmetric cryptography

For security, asymmetric key algorithms all rely on the mathematical principle that some mathematical operations are computationally easy in one direction but hard to reverse.

Asymmetric key algorithms are a relatively new development (in the history of cryptography). They were discovered independently during the 1970s by James Ellis, Clifford Cocks and Malcolm Williamson at GCHQ[21] and by Whitfield Diffie and Martin Hellman [28].

3.2.2 RSA

The RSA algorithm was developed by Ron Rivest, Adi Shamir and Leonard Adleman while they were at MIT [29]. The name of the algorithm is derived from the first letters of their respective last names.

The security of the RSA algorithm depends on the difficulty of factoring very large numbers.

RSA is the most widely used asymmetric key algorithm today. Over the past 33 years extensive cryptanalysis has revealed a number of weaknesses in the algorithm that have all been mitigated successfully. RSA therefore remains a secure algorithm given that the correct threat mitigations are implemented. As computing power has improved over the years, certain RSA key lengths have now been deprecated as they are no longer thought to be secure enough. Currently, the minimal recommended key length is 1024 bits, but it is advisable to use larger keys for applications that will be used for a number of years, for instance 2048-bit keys.

One more note on RSA key lengths: it is very common to only use powers of 2 for RSA key lengths (such as 512, 1024 and 2048). This is, however, not strictly required. In fact, any key length can be used, although many implementations have some limitations (i.e. they only accept multiples of 8, 16, 32 or 64 bits).

3.2.3 DSA

The Digital Signature Algorithm (DSA) was developed by the United States government. It was designed by the National Security Agency (NSA) and the standard is published by the National Institute for Standards and Technology (NIST) as Federal Information Processing Standard (FIPS) 186. The current revision of the standard is version 3 (see [34]).

21 GCHQ (Government Communications Headquarters) is the UK signal intelligence office

DSA is based on the same principles as Diffie and Hellman's original key exchange algorithm and it is in fact derived from the ElGamal encryption system (see §3.2.8), which in turn is derived from Diffie and Hellman's algorithm. It differs from RSA in that DSA only supports digital signature schemes. It cannot be used to encrypt data.

Many people are mistrustful of DSA because it was developed by the NSA. This is one of the reasons why DSA use is not as widespread as RSA use. Another reason why DSA is used less frequently is that it has certain unfavourable properties regarding computational complexity. Specifically, signature validation is a relatively costly operation in terms of the required number of CPU cycles. This is different for RSA where validation is an order of magnitude faster than signature generation.

A benefit of DSA is that it produces much smaller signatures than RSA, thus making DSA a somewhat more preferable candidate for memory-constrained devices.

A recent development is the Elliptic Curve Digital Signature Algorithm (ECDSA). More information about ECDSA can be found in §3.2.6.

3.2.4 Elliptic Curve Cryptography

Elliptic Curve Cryptography (ECC) is based on a special class of mathematical structures known as elliptic curves.

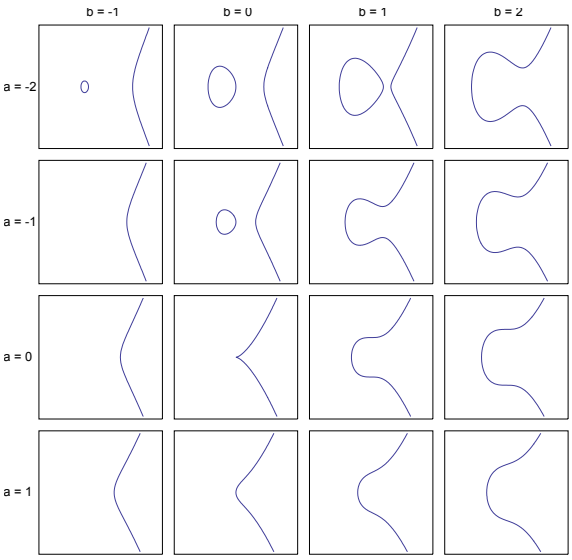


Figure 10 – Various curves characterised by parameters a and b

An elliptic curve is characterised by a relatively simple mathematical equation of a particular form (involving only elementary arithmetic operators and two parameters *a* and *b*), producing curves such as shown in Figure 10.

As with other cryptographic building blocks (such as RSA) the key feature of the mathematics behind ECC is that certain operations (repeated multiplication) can be computed relatively easily while the reverse operations (finding the multiplicants given their product) is computationally difficult.

Elliptic Curve Cryptography forms a public key cryptosystem, in which Diffie-Hellman Key Exchange (see §3.2.8), signing and encryption can be implemented. An interesting feature of ECC, when compared to RSA is that the number of key bits can remain relatively low, as witnessed by Table 3 below (taken from [87]).

Symmetric key size (bits)	RSA/DH key size (bits)	ECC key size (bits)
80	1024	160
112	2048	224
128	3072	256
192	7680	384
256	15360	521

Table 3 - Comparison of ECC key sizes with other algorithms

Another positive aspect of ECC is that computation costs are relatively low when compared to other signature and key exchange options. These features make ECC ideal to be implemented in small devices such as smart cards and mobile gadgets.

The NSA has indicated that they plan to move to ECC based public key cryptography. In Suite B[22] a list of recommended cryptographic primitives is suggested, including some ECC algorithms.

3.2.5 ECDH Key exchange

The elliptic Diffie-Hellman (ECDH) key exchange protocol allows Alice and Bob to exchange a (symmetric) key. The protocol is a variant of standard Diffie-Hellman (DH) key exchange in which Alice and Bob, after some computation using their own private key and each other's public key, arrive at the same shared key. The protocol prevents eavesdroppers on the wire from computing the same key but does not establish authentication, i.e. it cannot prevent a man-in-the-middle attacker to pretend to both parties to be the other party and establish shared keys with both parties. Protocols based on DH which, also establish authentication, such as MQV (Menezes, Qu, Vanstone) do exist, however, and have an ECC variant.

22 See http://www.nsa.gov/ia/programs/suiteb_cryptography/index.shtml

3.2.6 ECDSA signing

Elliptic Curve digital signature algorithm (ECDSA) is a variant of the DSA algorithm (§3.2.3) Alice can sign messages and Bob can verify them using Alice's public key. Like the DSA, ECDSA has been standardised by NIST in FIPS 186-3 [34].

3.2.7 Intellectual property problems with ECC

Some companies, notably Certicom (which has been purchased by RIM, makers of the Blackberry smartphone), claim intellectual property on some of ECC related aspects such as the ECMQV protocol. There are claims that this is one of the main factors limiting adoption of ECC. In [87], published by the NSA, words with similar meaning can be found. The NSA has purchased a license from Certicom allowing US and allied nations to use some of the claimed technology under certain conditions.

There are, however, academic groups that strive to publish elliptic curves that are free of royalties and can thus be used freely[23].

3.2.8 Other algorithms

Apart from RSA and DSA – which are the most commonly used algorithms – there are several other algorithms worth mentioning:

- The ElGamal algorithm (see [35]); this algorithm is used in the open source community and up until recently was the default algorithm in the GNU Privacy Guard, an open source alternative to PGP (PGP is described in §4.9.3)
- The original Diffie-Hellman algorithm (see [36]); this algorithm was created specifically to enable key exchange based on public key cryptography. It does not implement sender or receiver authentication, so before a key is exchanged the authenticity of the peer should be established using other mechanisms. Diffie-Hellman (or DH for short) key exchange is extensively used in various Internet protocols (for example in the IKE protocol that is part of IPsec)

3.2.9 Web of trust

§3.2.1 discussed the need to exchange public keys between communicating parties and the need to establish trust in these keys by verifying their authenticity out-of-band. As was already mentioned in that section, this trust model does not scale very well if many users and or parties are involved.

For example: there are two researchers, Bob who works for Beta University in Belgium and Alice who works for Alpha University in Austria. Bob and Alice have never met in real life but their respective bosses have set up a joint project in the medical research field. Bob and Alice have been introduced to each other by e-mail and they have been co-operating remotely by sending each other messages. Now though, Bob needs to send Alice some confidential patient records. He could exchange public keys with Alice, but he would still need to validate the authenticity of her public key. Although he could do this over the phone, he doesn't actually know what her voice sounds like. And since the patient data is very sensitive, Bob does not want to do this. So the only way he has of really making sure that he is talking to the real Alice is by either meeting her in person or by having someone he trusts vouch for her.

Now remember that their bosses set up the research project. Let's assume that their bosses have met in person and trust each other. And let's also assume that Bob and Alice each trust their bosses. This trust can be leveraged to allow Bob and Alice to communicate securely. To do this, Alice's boss, Dave checks (verifies the authenticity of) and digitally signs Alice's public key[24]. In turn, Bob's boss Jennifer checks and digitally signs Dave's public key. And because Bob trusts Jennifer (he has verified the authenticity of her public key), Jennifer trusts Dave and Dave trusts Alice, Bob can now trust Alice's public key. Figure 11 shows this graphically:

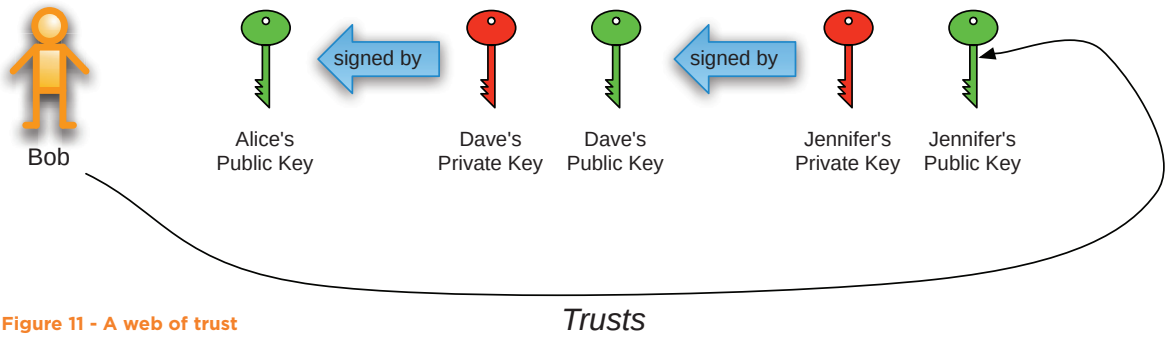


Figure 11 - A web of trust

23 This – for instance – includes work done at the Eindhoven Institute for the Protection of Systems and Information (http://www.win.tue.nl/eipsi/) and work done by Daniel Bernstein (http://cr.yp.to)

24 More information on digital signatures can be found in §4.7

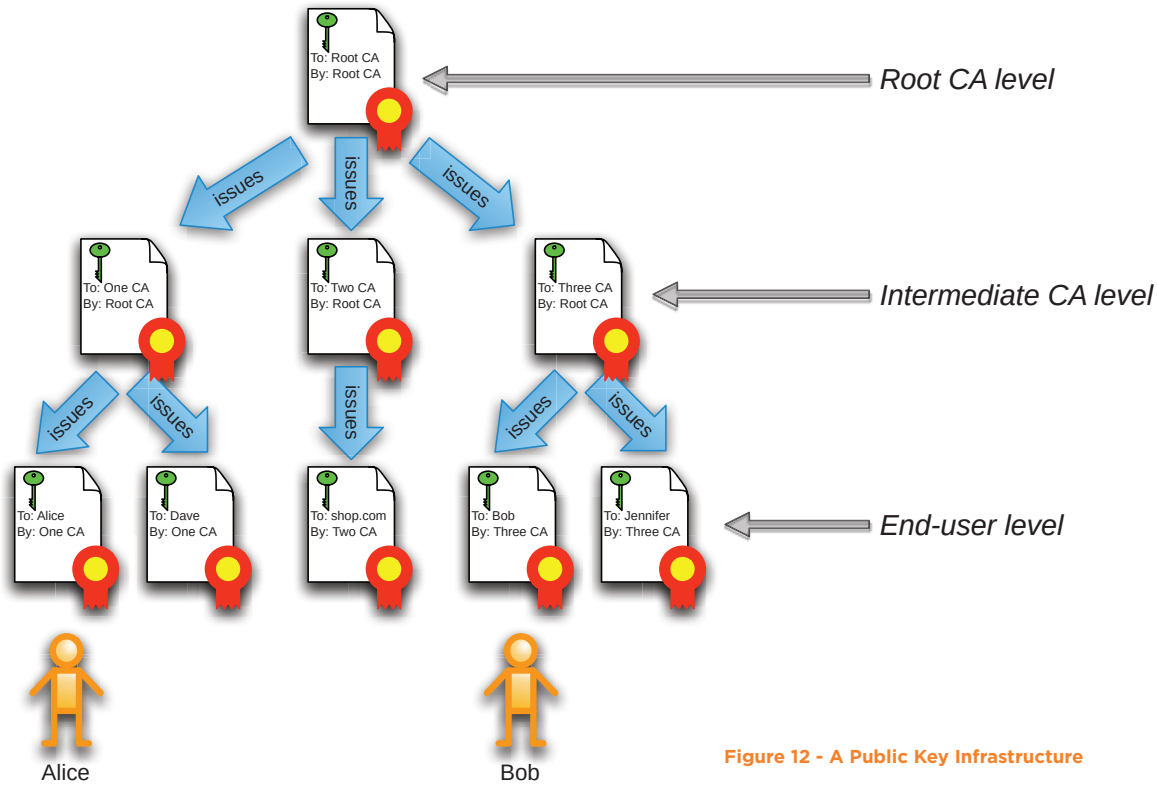


Figure 12 - A Public Key Infrastructure

In this set-up, trust is not established directly but instead goes through an intermediary. The level of confidentiality is made higher by not just relying on the signatures of one trusted person on a new key but by checking signatures made by more than one trusted individual (this is common practice).
This way of establishing trust is called a web of trust.

3.2.10 Public Key Infrastructure (PKI)

Although a web of trust is a vast improvement over one-to-one key verification, it still does not scale very well in certain situations, for instance when:

- Trust needs to be established in unknown servers, such as an electronic banking service
- Large organisations need their employees to be able to communicate securely

The web of trust still relies on interpersonal relations and many one-to-one verifications. There is another model, however, that does not. This model is called a *Public Key Infrastructure* (PKI). In a PKI, there is no longer a need for interpersonal verification. Instead, the establishment of trust is outsourced to a *Trusted Third Party* (TTP).

The TTP issues what are called certificates (or X.509[25] certificates) to users. These certificates state their credentials (e.g. name, e-mail address, etc.) and their public key and are digitally signed by the TTP. In this context, the TTP is often called a *Certificate Authority* (CA) or *Certificate Service Provider* (CSP). Now, a user who wants to establish whether he can trust a certificate holder only has to trust the TTP and has to verify the signature on a certificate against the TTP's public key to establish whether or not he can trust the holder of the certificate.

Figure 12 shows a common set-up of a PKI. It shows the top-level TTP, often called a Root CA. This Root CA expresses trust in Intermediate CAs by signing their certificates. In turn, the Intermediate CAs issue certificates to end users. The example shows Alice and Dave who have received certificates from "One CA" which in turn has received its certificate from the Root CA. It also shows a certificate issued to a web server called "shop.com", this certificate was issued by "Two CA". Finally, it shows certificates issued to Bob and Jennifer by "Three CA".

Now, if – for instance – Alice wants to send Bob a message, all she needs is his certificate and she only needs to trust the Root CA in order to establish the authenticity of Bob's public key.

This raises the question of how the Root CA can be trusted. The answer to this is simple: that trust needs to be established through an out-of-band channel. This is most commonly achieved by the fact that many Root CA certificates are included in modern browsers and operating systems. This takes away the need for users to establish trust relationships with individual Root CAs. The downside is that trust is not immediately established by the user but rather outsourced to browser vendors. It is debatable (and in fact a hot topic) whether or not this is truely safe.

25 X.509 is a standard published by the International Telecommunications Union (ITU); it describes a syntax for digital certificates

Obviously, if a person or a service receives a certificate from another entity, they will need to check the validity of the certificate. This usually means checking:

- if the digital signature on the certificate is correct by validating it using the CAs public key (which is included in the CA certificate);
- the validity of the certificate – certificates always have a limited validity and they include fields stating when they were issued and until when they are valid;
- if the certificate has not been revoked – CAs usually offer the possibility to revoke certificates (for instance in case the accompanying private key was lost or compromised) and they maintain a list of revoked certificates
- and most importantly: if the certificate has been issued to the right holder (i.e. if the subject of the certificate is correct)

Of course, Certificate Authorities need to prove their trustworthiness in order for this model to work. They do this by implementing policies, such as a *Certificate Practice Statement* (CPS). Such a CPS states under what conditions certificates are issued. There are varying levels of verification, from a cursory check whether the e-mail address of the party to which the certificate is to be issued exists to complete face-to-face and passport checks in a notary's office for individual certificates that require a high level of assurance. More information on CPSes can be found in §5.2.

An interesting hybrid between a web-of-trust and a full-blown PKI is CAcert.org[26]; this is a community initiative for establishing a PKI. Participants can gain 'assurance points' by having their identity verified by other participants (the web of trust part). Having more assurance points allows users to get progressively more privileged certificates issued by CAcert (the PKI part) that have a stronger tie to their identity.

3.2.11 Summary

Asymmetric key algorithms allow two or more parties to communicate securely without having to establish a shared secret beforehand. It is important to establish trust in the public key of a party one communicates with. This can be done by a one-to-one verification but is made much easier by relying on a web of trust or on a PKI.

The table below shows the number of bits of key strength (see §2.3.5) offered by the most common algorithms described above:

Algorithm	Bits of key strength
RSA 1024-bit	≥ 80 bits
RSA 2048-bit	≥ 112 bits
RSA 3072-bit	≥ 128 bits
DSA 1024-bit	≥ 80 bits
DSA 2048-bit	≥ 112 bits
DSA 3072-bit	≥ 128 bits

Table 4 - Key strengths for asymmetric key algorithms

Based on recommendations by NIST, the following guidelines should be taken into account when choosing either one of these algorithms for use in an application

Timeframe	Acceptable algorithms
Through 2010 (min. 80 bits of key strength)	RSA with keys ≥ 1024 bits DSA with keys ≥ 1024 bits
Through 2030 (min. 112 bits of key strength)	RSA with keys ≥ 2048 bits DSA with keys ≥ 2048 bits
Beyond 2030 (min. 128 bits of key strength)	RSA with keys ≥ 3072 bits DSA with keys ≥ 3072 bits

Table 5 - Symmetric key algorithm recommendations[27]

3.3 Secure hash algorithms

Hash functions are generally used throughout computer science to transform arbitrary large chunks of data into numbers of a fixed range. Traditionally hash functions were used as a structuring mechanism for storing information in data stores to allow quick lookup by first computing the hash value of the information and then using it as an index to access the information.

This is analogous to the quick lookup of information by searching for a lemma in a dictionary. An alphabetically sorted dictionary may be further structured into sections, indexed using the first letter of the words in that section. If easy access to the correct section is facilitated with tabs (as in the picture) then a reader can quickly lookup the word by searching the relevant section only.

26 http://www.cacert.org

27 This is an excerpt of Table 4 from [19]

Figure 13 - A tabbed dictionary

Another use of hash functions is in error detection when sending information over an unreliable channel. The sender can send the hashed value of a large message to the recipient who can then compute the hash himself and verify that it is the same. If sender and recipient compute a different value then something went wrong with the message in transit. Adding a hash value to a message is commonly referred to as checksum, thumbprint/fingerprint, or message digest.

The general characteristic of a hash function is that only few input values map to the same hashed value. Or, put differently, that the chance of a collision of two hash values, given different inputs, is low. As the input domain is typically much larger than the output range, collisions cannot be avoided, of course, but it should not be possible to steer collisions. This property is sometimes referred to as collision-resistance.
Secure hash functions have another characteristic. Given a hash value computed using a secure hash function, it should be infeasible to produce an input that hashes to the same hash value. This property is sometimes referred to as one-wayness (or pre-image-resistance).

An example of the use of secure hash functions is allowing a computer system to check password validity without actually storing the passwords themselves on disk. By storing hashed versions of the passwords instead, all the system has to do to check an entered password is to hash that password and compare the result to the stored hash value. Since hash functions are collision-resistant the odds of getting access to the system with the wrong password are extremely low. Since secure hash functions are assumed to be one-way functions, trustworthy system operators with access to the password file will not learn the user's password[28].

3.3.1 MD5

The MD5 secure hash algorithm, devised by Ron Rivest, is a strengthened version of its predecessor MD4 (see [64]). MD5 is described in [65]. It takes as input any string of bytes and produces a 16-byte output. MD5 has been in use in real security systems for quite some time yet has been shown to be vulnerable to attacks by Wang and Yu in 2005. They show that MD5 is not collision-resistant. A practical demonstration of this is given by Stevens, Lenstra en De Weger [84] in the form of a so-called Nostradamus attack: In 2007 they published a MD5 hash value of a document describing their prediction of the November 2008 US presidential election. After the election they were able to show a PDF file whose hash value was indeed the value they published one year ahead of the elections. In fact, they produced twelve PDF files announcing different candidates as winner which all hash to the same MD5 hash value.

3.3.2 Secure Hash Algorithm (SHA)

SHA, the secure hash algorithm, was published by NIST in FIPS 180-1. Its successor, SHA-1, fixes the original algorithm after the NSA found an attack and it is used in many real world applications. It is described in an update 3 to FIPS 180 [66]. SHA-1 takes any string of bytes as input and produces a 20-byte output. Like MD5 SHA-1 is also based on MD4. In 2005 Wang, Yin and Yu [85] discovered weaknesses in SHA-1, which may potentially break collision-resistance. SHA-2, published by NIST in 2001, is a family of hash functions producing 224-, 256-, 384-, and 512-bit outputs. Since SHA-2 is a relatively new family of algorithms (and therefore not tested by the cryptography community) and since SHA-2 is based on the same principles as SHA-1, new efforts are underway to design a successor: SHA-3.

28 System operators with access to the password file can, however, mount an offline dictionary attack on a copy of the password file. Often some random noise (the salt) is added to the input before hashing and stored alongside the hashed values so that the dictionary attack can only be mounted per password and not per word in the dictionary. If salts are not used, a hacker could prepare hashed tables of popular passwords prior to getting access to the password file; special data-structures known as rainbow tables make looking up pre-hashed passwords exceedingly fast. In any event, the password file should be readable to as few people as possible.

3.3.3 RIPEMD-160

RIPEMD-160 (RIPE stands for RACE Integrity Primitves Evaluation, MD stands for message digest) is another hash algorithm with a 160 bit (20 byte) output. The algorithm was designed by Hans Dobbertin, Antoon Bosselaers and Bart Preneel of Leuven University in 1996 (see [86]), i.e. it originates from the cryptographic research community as opposed to the SHA family of algorithms which were all published by the NSA up till now. RIPEMD-160 is the successor of RIPEMD for which collisions were reported in 2004. Different variants of RIPEMD-160 with various size outputs, RIPEMD-128, -256, 320, are also available. RIPEMD-160 appears to be less popular with implementers than SHA-1 (a possible reason for this is given in [86]: RIPEMD-160 is approximately 2 times slower than SHA-1).

3.3.4 Other algorithms and future standards

It turns out that designing secure hash functions is very difficult. Contrary to cryptographic ciphers, hash functions have received relatively little attention from the cryptographic research community. This is changing. NIST has organised a competition in which cryptographers propose candidates for a new standard hash algorithm: SHA-3. A winner will be elected in 2012. In the mean time, the SHA-2 algorithms are likely the best option for implementing a hash function, at least these are recommended by Ferguson, Schneier, and Kohno in [67].

3.4 Quantum cryptography

3.4.1 Introduction

Strictly speaking, quantum cryptography is not a cryptographic algorithm per se. Instead, it is a method that uses quantum mechanical principles to enable two communicating parties to securely establish a shared secret. Thus, quantum cryptography is a misnomer; the correct term is *Quantum Key Distribution* (QKD).

Quantum Key Distribution relies on one of two quantum mechanical principles:

- Measuring – quantum theory states that measuring the properties of a particle will influence the particle's quantum state. This principle can be used to detect eavesdroppers as will be explained below.
- Entanglement – the quantum states of two or more particles can be linked together in such a way that they share a single quantum state. Quantum theory states that measuring one particle in an entanglement influences all other particles in the entanglement. This principle can also be used to detect eavesdropping.

3.4.2 Quantum Key Distribution in practice

There are several published methods for Quantum Key Distribution, some of which are still theoretical or only available in research laboratories. The most popular scheme, for which commercial solutions are available is the protocol devised by Charles Bennett and Giles Brassard in 1984 [57], usually referred to as BB84.

The basic principle of this protocol relies on using different polarisations of light. It relies on using two non-orthogonal ways of polarising light that is sent from the sender (Alice) to the receiver (Bob). There are three commonly used polarisation schemes called bases: the rectilinear basis where light is polarised in the vertical (0˚) or horizontal (90˚) direction, the diagonal basis where light is polarised in either 45˚ or 135˚ direction or the circular basis where properties of left- and right-handedness are used. Quantum theory holds that it is not possible to measure in two non-orthogonal directions. It is therefore only possible to measure in one basis at a time. This last property is what is used to establish the security of the scheme.

Each basis has two directions that encode for the value 0 or 1 respectively and two bases are used in the protocol.

If Alice now wants to establish a shared secret with Bob[29], she first generates a random one-time pad (see §3.1.2). She then decides which two bases she is going to use to transmit the data in and communicates this to Bob. Then Alice transmits the shared secret bit-by-bit to Bob, making sure to randomly select a basis for each bit that is transmitted. She keeps track of which basis she used to transmit each bit.

On the receiving end, Bob does the same thing: he randomly selects a basis to measure in and receives the message bit-by-bit, making sure he notes the basis he used to receive the bit. Now if Bob chooses the right basis he will correctly measure the bit that was transmitted. If, however, he chooses the non-orthogonal other basis there is no guarantee what he will receive.

Once the complete message is transmitted, Alice sends Bob here list of bases. This can be done in the clear since this information has no relation to the secret that was transmitted. Bob then compares the list of receiving bases he used against Alice's list. Each received bit for which Bob's receiving basis matches Alice's transmission basis is kept, bits with mismatched bases are discarded. This means that – statistically speaking – approximately 50% of the message is received correctly by Bob. Bob now sends his list of bases to Alice who repeats the process, comparing Bob's list to hers. The end result is that Alice and Bob each know which data the other can reliably know about. This data is now their shared secret one-time pad.

29 Note that QKD offers Alice no way to establish that she is really talking to Bob. It is therefore important to also include an authentication step in the process.

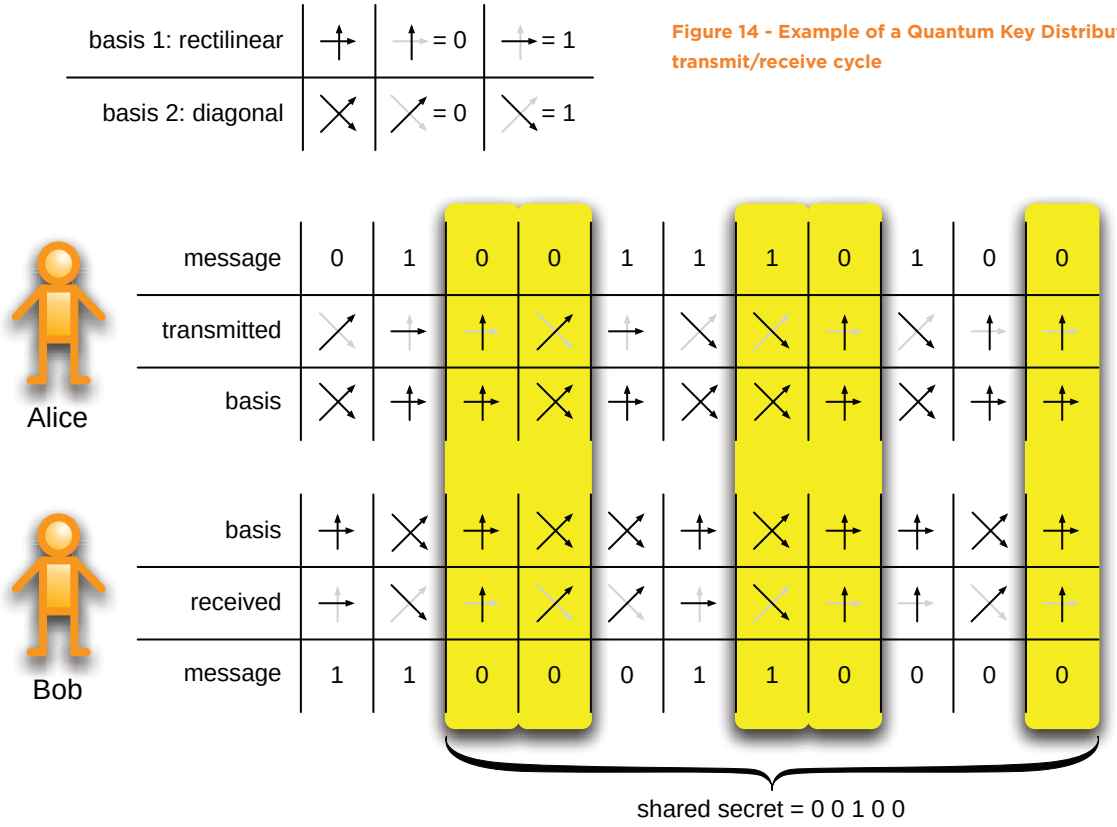


Figure 14 - Example of a Quantum Key Distribution transmit/receive cycle

Figure 14 shows an example transmission between Alice and Bob in the rectilinear and diagonal bases.

The security of Quantum Key Distribution using the scheme described above lies in the fact that an attacker Eve who intercepts the message using a man-in-the-middle attack has to guess the transmission basis whilst the message is being transmitted. Just like Bob, she will guess the right basis 50% of the time. Because she is using a man-in-the-middle attack, she will retransmit whatever she believes she has received. This means that Bob receives whatever Eve thinks she has measured. Bob again randomly selects the receiving bases. This means that he will receive approximately half the message retransmitted by Eve correctly.

Bob has now – statistically speaking – received 25% of the message correctly. Remember that Bob does not know what Alice originally transmitted so there is no way for Bob to determine the additional transmission errors introduced by Eve's attack.

So how do Alice and Bob find out if there was an eavesdropper? In theory, once Alice and Bob have exchanged the bases they used, they know exactly which part of the data was received in the correct base by Bob, so they should have exactly the same shared secret. However, because Eve interfered, Bob has only received approximately 25% of the data in the correct base instead of the expected 50%. Now, to determine if an eavesdropper interfered in the transmission there is only one solution: Alice and Bob have to sacrifice part of the secret (based on their mutual understanding of the selected bases) to determine if errors have been introduced. To do this, Alice and Bob randomly select part of the secret, compare it and then discard this part of the secret (since it is revealed by their exchange). They can now detect any errors that were introduced by Eve eavesdropping and retransmitting. To determine with near 100% certainty that an attack took place they have to compare (and thus sacrifice) 72 bits of the secret.

3.4.3 Attacks on Quantum Key Distribution

As with any new cryptographic technique a lot of efforts have been made to break the security of the technology. These efforts show that Quantum Key Distribution is certainly not invulnerable to attacks. The method described above in §3.4.2 relies on data bits being transmitted using a single photon. In practice, it is currently impossible to guarantee this, which leads to an attack called a *photon splitting attack* (see [58]). Another attack deals with the physics behind Quantum Key Distribution (see [60]). Finally, Nature has recently published an online article about 'blinding' Quantum Key Distribution systems with bright laser pulses (see [108]). This last attack was successful against both commercially available Quantum Key Distribution systems[30].

Some further delving into publications on the security of Quantum Key Distribution and counter-publications by vendors showing mitigation of such problems gives the impression that QKD is far from mature technology. It is extremely hard for a layperson to determine whether a QKD implementation can be considered secure or not. The inherent risk in this is that QKD is currently being touted as a panacea of security woes – or to put it differently: it is sexy. It would be very wise to delve deeper into the pros and cons of QKD before making a purchasing decision (other reasons for this are very eloquently stated in [61]).

3.4.4 Applications of Quantum Key Distribution

There are two commercial vendors offering QKD products:

- ID Quantique
- MagiQ

Both offer products that can be used for network link encryption (VPN) purposes.

3.5 The effects of quantum computing on cryptography

3.5.1 Introduction

There has been a lot of speculation about and research into the effects of quantum computing on cryptography. The problem is that quantum computing introduces a completely new computing paradigm. This has an effect on certain cryptographic techniques that rely on the complexity of certain mathematical problems. When quantum computing comes into play, these problems may suddenly no longer be complex. It goes beyond the scope of this paper to explain quantum computing, more information on that subject can be found in [31]. This section deals with what researchers currently think the impact of quantum computing will be on cryptography.

3.5.2 Effect on symmetric key algorithms

Mathematical researchers Charles Bennett, Ethan Bernstein, Gilles Brassard and Umesh Vazirani proved in a 1996 paper that quantum computing effectively halves the number of bits of key strength for any symmetric key algorithm [32]. They have shown breaking a key with n bits of strength cannot be done faster than performing 2n/2 operations of the cryptographic algorithm used. Thus if quantum computing becomes a reality, symmetric key algorithms can be protected against this by simply doubling the key size. This will, of course, have an impact on the performance of the symmetric key algorithm.

3.5.3 Effect on asymmetric key algorithms

The effect of quantum computing on asymmetric key algorithms such as RSA, DSA, ECC and Diffie-Hellman is thought to be quite serious. Researchers believe that sufficiently powerful quantum computers can compromise the security of asymmetric key algorithms in the same time it requires to use them in day-to-day use. For example: it is currently believed that a quantum computer capable of breaking 2048-bit RSA could be available in 20 to 30 years (source: [32]).

3.5.4 Protecting against quantum attacks

There is an ongoing effort to devise new algorithms that are more resilient to quantum attacks. Currently, progress is being made both in the field of elliptic curve cryptography as well as with new types of public key algorithms. More information can be found in [33] and [59].

30 A series of photographs of the setup used to perform the attack can be found here: http://www.iet.ntnu.no/groups/optics/qcr/hacking-commercial-quantum-cryptography-2010/

APPLICATIONS

4

4 APPLICATIONS

4.1 Introduction

Cryptography plays an important role in many IT applications. This chapter provides an overview of commonly used applications that rely on cryptography and aims to explain how cryptography is used in these applications. Where appropriate, advice is offered for implementers and decision makers on what to pay attention to when choosing products with cryptographic functionality. More general advice on judging the quality of cryptographic products is given in chapter 6.

Note that cryptography is by no means and can never be a complete solution for all security woes.
A secure system has many different elements of which cryptography is only one. Bruce Schneier explains this very well in his book Secrets and Lies [2].

4.2 Strong Authentication

Authentication of users plays an important role in the identity management process. Authentication is the process by which claimed identities of users are verified (to some degree of certainty). Other processes, such as authorisation can only take place *after* authentication has been successfully completed. Many different methods are used for authentication, ranging from simple user name/password combinations to advanced biometric pattern recognition. These methods all have their own advantages and disadvantages and carry with them varying degrees of certainty that a claimed identity indeed checks out.

4.2.1 Multi-factor

Combining different factors is the most obvious way to strengthen the authentication process. Typical authentication factors include:

- Something the user has, e.g. a hardware token of some kind
- Something the user knows, e.g. a password or PIN code
- Something the user is, e.g. biometric features such as a fingerprint or the iris[31]

Common wisdom dictates that a combination involving at least two of these three categories works best. Note that the key strength of keys derived from "Something the user knows" can never be sufficiently large enough to amount to "strong" cryptography. For low entropy passwords (e.g. PIN codes) usually a layered approach is taken in which the PIN code can only be used a limited number of times before the authentication method locks and can only be unlocked using a key of higher entropy (e.g. the PUK code).

Even more categories can be added to this list of factors above, for instance behavioural characteristics of the user ("something the user can do") such as the ability to type on a keyboard following a certain rhythm or typical location or time based patterns that are observed using sensors in personal devices carried by the user. Such mechanisms do not involve cryptography and are outside the scope of this white paper.

4.2.2 Multi-channel

By providing a separate extra channel to the user, authentication can be achieved by sending one-time codes to the user for signing on to the service. Cryptographically speaking this is equivalent to a one-time pad (§3.1.2) under the assumption that the extra channel cannot be eavesdropped on by an attacker. In practice a TAN[32] list with codes is distributed to the user in advance before the user signs on to the service, or codes are sent during the authentication process over a different channel such as SMS. Recent doubts over the confidentiality of the latter channel (the GSM A5 algorithm weaknesses observed by Nohl et al., see §4.7.3) make SMS-OTP somewhat less secure, although the advantages of two separate channels (an attacker has to control both) may make other attack vectors more attractive for attackers.
Some security tokens, used for instance in e-Banking services, create an end-to-end secure channel with the verifying entity (the back end systems at the bank) using the desktop computer of the user by connecting to it via USB or Bluetooth. An example of a prototype is the ZTIC token developed in IBM's Zürich research laboratory (see [102]).

4.2.3 Tamperproof hardware

A major problem in any authentication system that uses cryptography at the user's site is where to safely store the key. In an attacker model where the attacker resides on the network this is not so much of a problem: the hard disk of the user's desktop computer can hold the key in that case. This situation changes if the user's desktop computer is shared among several users, which cannot be trusted or in an attacker model which allows an attacker to install malware on the user's desktop computer. In such cases special cryptographic hardware such as PKI tokens or smart cards may be used to hold the user's private key (see also §5.1.4). The private key never leaves the cryptographic hardware but is used during a challenge-response protocol by the hardware itself. If the hardware is tamper proof it can even be used in an attacker model where the user is not trusted.

31 Note that there is a lot of discussion about the reliability of biometrics; it is highly inadvisable to create a system that only relies on biometrics as a second factor. It is better to use biometrics as a third factor next to "something the user knows".

32 Transaction Authentication Number, a one-time password (OTP)

4.2.4 One time password generating tokens

Password generators form a class of hardware tokens that use cryptography to generate session passwords (sometimes called one-time-password, or OTP) that can be recognised by the verifying party as valid and cannot be guessed by an attacker. Internally such a token has a clock whose value is hashed and encrypted using a key shared with the verifying party. The verifying party has a clock that is synchronised with the token's clock.

Usually the session password is shown to the user on a display from which the user types it into a password entry field on the webpage of the service provider. An example is shown in Figure 15.

Figure 15 - An OTP token

All major authentication token manufacturers (RSA, Verisign, VASCO, Todos) make such tokens.

Yubikey is an OTP token, which plugs into a free USB slot and emulates a USB keyboard. It can type the generated session password into the password entry field and thus saves the user the trouble of having to type the password himself (which means that a higher entropy password can be used).

OTP tokens are a popular solution for securing authentication to VPNs.

4.2.5 Challenge-response tokens

In a challenge-response based authentication scenario each user is given a token (for an example see Figure 16) that holds a cryptographic key. Possession of the token (or, actually, the embedded key) is proved using a challenge-response protocol (see §3.2). For an attacker without access to the key guessing such a key is completely infeasible.

Challenge response tokens typically require the user to type the key into the token, which has a keypad and a display. The response is shown by the token on its display and the user needs to type this value into the webpage of the service provider.

All major authentication token manufacturers sell such tokens.

Figure 16 - A challenge/response token

These kinds of tokens are very popular among banks.

4.3 SSL/TLS

4.3.1 History of the protocol

The Secure Sockets Layer (SSL) or – as it is now known – Transport Layer Security (TLS) is more of a building block rather than a stand-alone application. The reason it is discussed here is that it is the foundation on which many networked applications rely for secure communications.

SSL was originally developed by Netscape Communications. The first publicly available version was version 2.0, which was released in 1995. Because of a number of security flaws, it was quickly followed by version 3.0 in 1996 [37]. Work on SSL was subsequently continued by the Internet Engineering Task-Force (IETF) and in 1999 the first IETF standard, RFC 2246, was published. By then, the name of the protocol had been changed to TLS. TLS version 1.0 is not directly compatible with SSL 3.0 but can downgrade a connection to provide backward compatibility. The current version of TLS – version 1.2 – is defined in RFC 5246 [38]. It includes support for modern cipher suites such as AES.

4.3.2 How the protocol works

TLS is a client/server protocol. The protocol consists of two phases: a negotiation phase and an application phase. A high-level representation of the negotiation phase is shown in Figure 17 below:

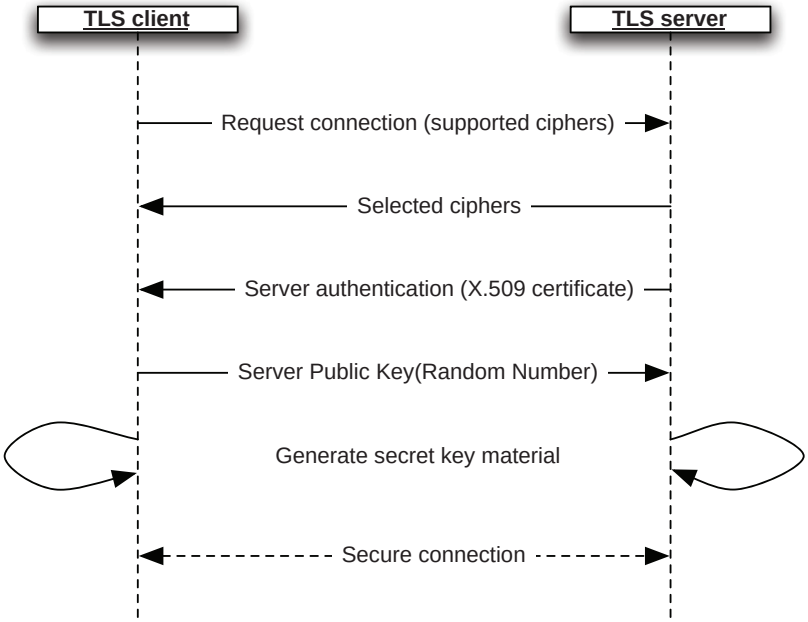


Figure 17 - TLS negotiation

The figure shows how the client initiates the connection, telling the server which cryptographic algorithms it supports. The server then selects the strongest algorithm it supports and returns this information to the client. It also sends a server authentication based on the server's X.509 certificate (see §3.2.10). The client checks the server's credentials and sends the server some random data, which it encrypts using the server's public key. Then, both client and server use this random data to derive key material to use as a session key to encrypt/decrypt data exchanged on the communication channel between the client and the server.

During the application phase, all data is transmitted in encrypted form between client and server using the established session key.

In its basic configuration, the protocol only authenticates the server to the client. Optionally, the protocol can also support client authentication. In this case the client needs its own X.509 certificate to authenticate itself to the server. This setup is called *mutual authentication.*

TLS uses asymmetric key algorithms (RSA, DSA, Diffie-Hellman) to exchange session keys and to authenticate server and – if applicable – client and uses symmetric key algorithms to perform the actual link encryption.

4.3.3 Integration into applications

TLS has been designed for easy integration into applications. Its programming interface is very similar to the common network programming interface (called the sockets interface). Figure 18 shows what application programmers have to do to leverage the security that TLS can provide.

All they have to do is add some extra code to their application that can negotiate a session. From then on, they can use an interface that is virtually identical to the sockets interface to transmit and receive data over the secure channel. The top-half of the figure shows the situation before adding TLS support, the bottom half shows the integration with TLS. The application talks to the TLS layer, which establishes a secure link across an unprotected link to the TLS layer on the other side.

It is even possible to add TLS to legacy applications by introducing a so-called *"bump-in-the-wire"* application. Such an application sits between the legacy application and the Internet and acts as a transparent proxy. It accepts incoming TLS connections and sets up a corresponding local connection to the legacy application. From the point of view of a user, he or she is connecting to a TLS-secured application whereas the legacy application never needs to be aware of this. An example of such a bump-in-the-wire is stunnel[33].

33 http://www.stunnel.org

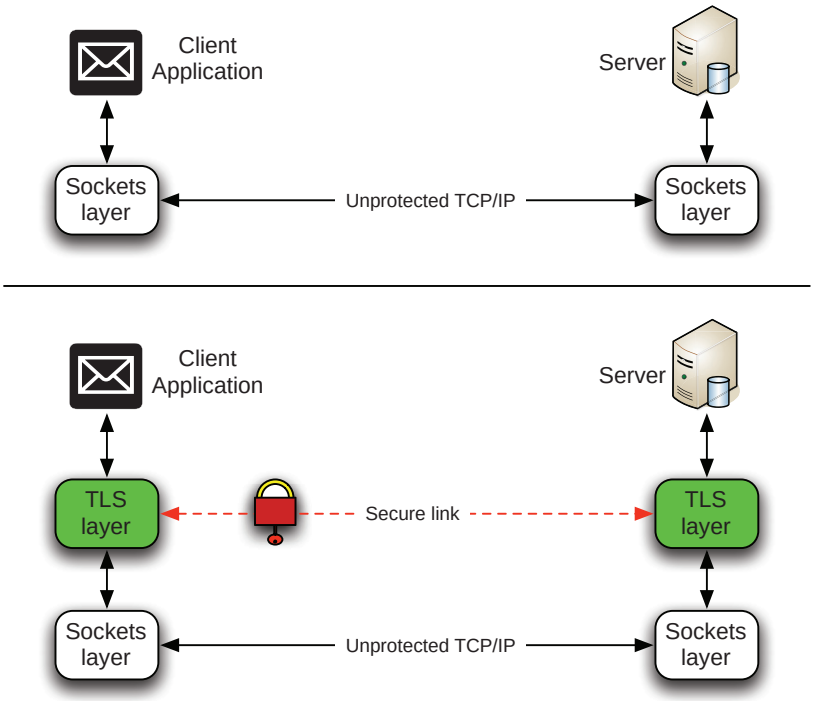


Figure 18 - Adding TLS to an application

4.3.4 Common applications of TLS

The most frequently used application of TLS is HTTPS. HTTPS is the secure version of the HTTP protocol used to transfer web pages. It is – for instance – used to access e-Banking sites, to secure online credit card transactions and to protect exchanges of password data when logging in to online services.

Another common application of TLS is securing previously insecure protocols by adding SSL in the same way as shown in Figure 18. Examples of this include securing of the SMTP protocol, used to send e-mail messages, the POP3 and IMAP protocols, used to remotely access e-mail boxes and the FTP protocol, used to transmit files across a network. It is a common convention to add an "S" to the names of these protocols to indicate that they use SSL/TLS (e.g. POP3S, IMAPS, FTPS, ...).

Note that TLS only secures data in transit; it does not protect the data when it is stored at either end-point. Attackers will always go for the weakest link; thus, if the data is protected in transit, they will try to access it at the end-point of the link where it is usually not encrypted (otherwise the user would not be able to read – for instance – his or her e-mail). Common attacks include 'man-in-the-browser'[34] attacks.

4.3.5 TLS implementations

There are a number of commonly used open source TLS implementations: OpenSSL[35], GnuTLS[36] and NSS[37] (included in the Firefox web browser). Microsoft provides its own implementation for Windows, the Secure Channel[38] package.

4.3.6 Pitfalls and weaknesses of TLS

TLS is one of the most widely used security protocols. It is actively developed and maintained but it is also under constant scrutiny of both security researchers as well as hackers.

Over the years a number of weaknesses have been identified in the TLS protocol. These issues include:

- SSL cipher downgrade attacks – Older versions of the SSL protocol (v2 and below) are vulnerable to a so-called cipher downgrade attack. This means that the SSL server can be tricked into thinking that the client only supports low-grade security (e.g. 40-bit RC4). Servers should therefore be configured not to support SSL v2 and below.
- TLS renegotiation attacks – This is the most recent issue and was identified by Marsh Ray; this attack affects sites that use client authentication (PKI certificate based mutual authentication, see §4.3.2), for a description of the attack see [107].

Fortunately, so far all weaknesses have been successfully mitigated by modifications to the protocol. Consequently, the TLS protocol remains a good and secure way to add transport layer encryption to applications as long as the software that is used has been patched adequately. The use of TLS in applications is strongly encouraged over the development of home grown solutions.

34 For a definition see http://en.wikipedia.org/wiki/Man_in_the_Browser

35 http://www.openssl.org

36 http://www.gnutls.org

37 http://www.mozilla.org/projects/security/pki/nss/

38 http://msdn.microsoft.com/en-us/library/aa380123(v=VS.85).aspx

One major drawback of TLS is its reliance on X.509 certificates for server authentication. There are frequent errors with these server certificates, some examples of these errors are:

- Expired certificates
- Invalid hostnames (for the server) in certificates
- Untrusted certificate authorities (CAs, see §3.2.10 or broken certificate trust chains (this often occurs when home made self-signed certificates are used)

Figure 19 - Example of a TLS certificate warning in Firefox

The problem is that these errors invariably result in some sort of warning or pop-up being presented to the user (an example is shown in Figure 19).

These warnings are ambiguous for the user at best (have a look – for instance – at the advice under "What Should I Do?" in Figure 19 and closely consider what this means for a user who visits a web shop for the first time). Unfortunately, TLS configuration mistakes are not uncommon, which leads to users tending to ignore these warnings. Research shows that more than half of all users (see [39]) ignore SSL popup warnings despite improvements in the way browser show these warnings to the user.

It is therefore very important when deploying SSL/TLS to configure it correctly. The more infrequent warnings are the more likely users are to take them seriously.

To prevent users from having a false sense of security it is also important to configure a web server in such a way that it does not support older, flawed implementations such as SSL v2 and in such a way that it rejects weak cryptographic algorithms (i.e. only allows strong algorithms to be used). Hardly any user will check if the cryptographic algorithm used for the connection is secure, they simply rely on the little padlock symbol in their browser to tell them if the connection is safe.

It is good practice to test a configured web server using a test service such as SSLLabs[39] or Networking4all's "isdezesiteveilig" website[40]. A test to check whether a web server does not accept weak ciphers was added to the latter service after some public discussion with Dutch security researcher Eric Verheul[41]. Likewise test services for testing client applications are available[42].

39 See https://www.ssllabs.com/

40 See http://www.isdezesiteveilig.nl/ (only in Dutch)

41 See http://www.cs.ru.nl/E.Verheul/SSLTEST.htm (only in Dutch)

42 E.g, Microsofts SSL/TLS interop test server at http://tls.woodgrovebank.com/.

4.3.7 Advice when selecting TLS(-based) products

The most important advice is to select (products that are based on) well-known implementations of SSL/TLS. For a list of these, see §4.3.5. One should be wary of homebrewn solutions, since it is more likely that these have been less rigorously tested for weaknesses and since these often lack a large community supporting the development and fixing of any weaknesses that are discovered.

4.4 Secure Shell

4.4.1 Introduction

The Secure Shell or SSH protocol was introduced in 1995 by Tatu Ylönen of Helsinki University. Its main design goal is to be a secure replacement for protocols like TELNET and rsh which had up until then been in use to remotely access server systems, for instance for system administration purposes.

The original implementer of SSH started his own company fairly quickly after it became clear that SSH turned out to be popular. This company, SSH Communications Security, developed and marketed the SSH system. Although the original SSH system had been released as freeware, the version released by the company became increasingly closed in nature.

For this reason, open source developer Björn Grönvall decided to implement an open version of the SSH system. This project was quickly adopted by the OpenBSD project and eventually led to the release of OpenSSH. OpenSSH quickly became the standard implementation shipped with most UNIX- and BSD-like systems (such as Linux, Mac OS X and Solaris).

4.4.2 Security of SSH

Serious security flaws were uncovered in the first version of the SSH protocol, dubbed SSH-1, in 1998. Subsequent attempts to patch these flaws resulted in the introduction of a new vulnerability into most implementations that allowed attackers to execute arbitrary code on the system they were accessing.

This and other issues led to the development of a second version of the protocol (dubbed SSH-2) by the IETF[43]. SSH-2 is not backward compatible with SSH-1 (although some implementations do support both protocol versions). Nowadays, almost all implementations use the SSH-2 protocol.

In 2008 a flaw was also detected in SSH-2. This flaw allows the recovery of 32 bits of plaintext from an arbitrary encrypted block of data. Although the impact of the attack is high, the probability of it succeeding is very low. Furthermore, it is easy to mitigate this attack by configuring the SSH server not to use certain cipher modes[44].

Finally, SSH is vulnerable to keystroke timing attacks. Because of the way SSH transmits data when used in interactive mode (i.e. a user in a shell session), information about the way a user types is revealed. Song, Wagner and Tian [117] show that this information can be leveraged to speed up brute force password attacks by a factor of 50.

4.4.3 Authentication mechanisms

SSH is very flexible in the authentication mechanisms it supports. Although the most common method is to rely on username/password, it is also possible to integrate support for one-time password tokens (see also §4.2.4) such as SecurID or Digipass. Furthermore, SSH supports public key authentication[45].

4.4.4 SSH fingerprints

Most users who have used SSH before will be familiar with prompts such as those shown in Figure 20 below:

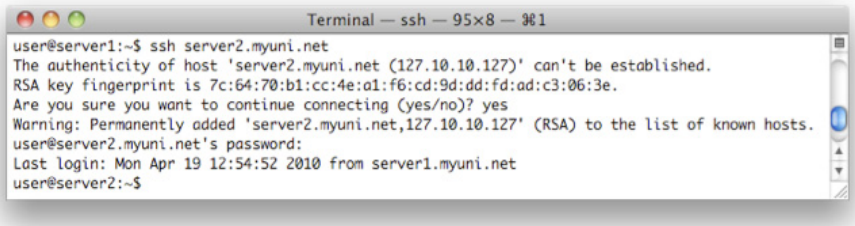


Figure 20 - SSH fingerprint request

43 The SSH-2 protocol is described in [109], [110], [111], [112] and [113]

44 For a description of the attack and mitigation guidelines, see http://www.cpni.gov.uk/docs/vulnerability_advisory_ssh.txt

45 A comprehensive HOWTO on SSH public key authentication can be found here: http://sial.org/howto/openssh/publickey-auth/

These requests are commonly ignored by users; most people will simply answer 'yes' to the query without thinking about what it means.

The fingerprint is actually a way for the server to identify itself to the user (the fingerprint is a secure hash of the server's public key). It is therefore important, especially in high security environments to check this fingerprint before accepting it. The best way to do this is to establish what the fingerprint of a server is by accessing it on the console and looking up the fingerprint[46]. Then, when an SSH session is established for the first time the fingerprint that is reported by the SSH client should be compared to the server's known fingerprint to make sure that the user is dealing with the correct server.

4.4.5 Other applications of SSH

Apart from using it to access a shell on remote server, the SSH protocol can also be used for:

- Secure file copying (SCP and SFTP)
- Secure backup of remote servers (in combination with rsync)
- Setting up secure tunnels or VPNs
- Forwarding windowing sessions of the X Window System

4.5 Network Link Encryption

Network Link Encryption usually refers to a mechanism for encrypting the network traffic on a single physical point-to-point link (for instance on a line between two network switches). It is applied at levels 1 and 2 of the network and is transparent to all the layers above.

The main application for Network Link Encryption is to secure transmission of data over a single physical link that is untrusted. As such, it is normally only applicable in high security environments, such as the military, intelligence services and diplomatic corps. It is important to note that Network Link Encryption does not provide end-to-end encryption on a multi-hop connection. Data needs to be decrypted and re-encrypted for re-transmission on the next link whenever it reaches a switching point in the network. Therefore, Network Link Encryption has very limited applicability and to achieve end-to-end encryption other technologies are used, such as TLS (§4.2) and VPN (§4.4).

Both symmetric key algorithms as well as elliptic curve cryptography are used to perform Network Link Encryption.

4.6 VPN

The term Virtual Private Network (VPN) can refer to many different technologies. The common denominator is that all these technologies provide a way to interconnect multiple distributed entities as if they were sharing a common network. Not all VPN technologies use cryptography, nor do all VPN technologies provide confidentiality. This section will focus on VPN technologies that do provide confidentiality and that rely on cryptography to achieve this.

4.6.1 End-to-end encryption

All VPN technologies that provide confidentiality rely on cryptographic algorithms to encrypt data in transit on a logical link between two endpoints in the virtual network. Figure 21 shows an example of a VPN:

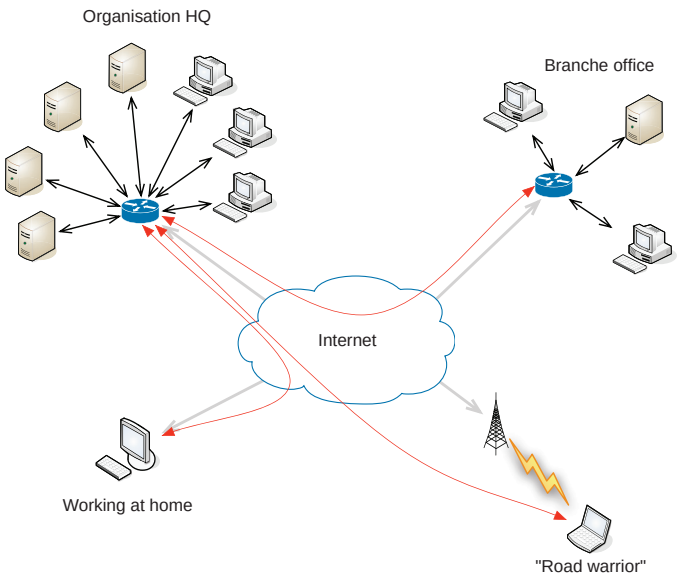


Figure 21 - Example VPN

46 A description of this can for instance be found here: http://www.lysium.de/blog/index.php?/archives/186-How-to-get-ssh-server-fingerprint-information.html

The diagram shows four user (groups) participating in a VPN.

The top left-hand side of the figure shows the Organisation Headquarters. This location has a VPN gateway appliance or application.

The top right-hand side shows a branch office. The branch office router or switch has a VPN connection to the gateway at headquarters. This enables seamless secure communication between computers on one LAN with computers on the other LAN. The users in the branch office are not aware that communication with their headquarters takes place over a VPN connection.

The bottom left-hand side shows a user working from home. If this user wishes to connect securely to his or her office network, the user sets up a VPN connection to the VPN gateway at headquarters. It will then seem to this user as if he or she is directly connected to the office network. The bottom right-hand side shows a mobile user[47]. Just like the home user this user can set up a VPN connection to the office gateway. The only difference is that this connection is routed over mobile connectivity.

In all three cases described above where a VPN connection is set up, the actual data traffic is routed over public Internet infrastructure. To ensure security and privacy of the connection, end-to-end encryption is applied. In effect, the red lines in the diagram show the secure virtual circuits that are established when a VPN connection is set up. The difference with network link encryption (§4.5) is that the secure channel is independent of the physical link. No matter how many intermediate links exist, the connection is guaranteed to be secure from one end to the other.

4.6.2 Mutual authentication

In order to establish trust between the communicating parties participating in a VPN it is necessary to establish each other's identity. This is achieved by performing mutual authentication. Commonly, the VPN server authenticates itself to the connecting VPN client using an X.509 certificate. The client checks the authenticity of the server and then proceeds to authenticate itself to the server.
This can be done in a number of ways:

- The client is an automated system such as a network element (router, switch, ...). In this case, the two most-used options are:
  - Authentication using a pre-shared secret (a password or pre-shared key)
  - Authentication using an X.509 certificate (commonly referred to as "client authentication")

- The client is an end-user system; the user initiates establishing of the VPN connection. In this case, the three most common options are:
  - Authentication using a username/password combination
  - Authentication using a personal X.509 certificate
  - Strong authentication (see §4.2)

4.6.3 IPsec- versus SSL-based VPN

There are two technologies that are used for VPNs today.

- IPsec – Internet Protocol Security (described in more detail in the next section); this protocol has been around since the 1990s and is well established as a VPN solution. Using IPsec for VPN always requires dedicated client software and/or integration in the operating system.
- SSL (see §4.2) – there is a wide variety of SSL-based VPN solutions[48]; these include browser-based on the fly SSL applications (based on Java or .NET technology) and full VPN client/server solutions that use SSL tunnels for communicating rather than IPsec (see also §4.6.6).

4.6.4 Cryptographic algorithms used for VPN

VPN can rely on a multitude of cryptographic algorithms. Broadly speaking, there are two areas where cryptography is applied:

- For link encryption (see §4.6.1); this usually involves using asymmetric key algorithms to establish a session key and uses symmetric key algorithms to perform the actual link encryption
- For end-point authentication (see §4.6.2); this mostly involves asymmetric public key algorithms (X.509 based PKI)

To illustrate this, a high-level explanation of IPsec is given below.

IP Security (IPsec) was developed during the 1990s by the IETF to address one of the shortcomings of the Internet Protocol (IP): IP does not provide authenticity or confidentiality. The most current version of the architecture of IPsec is described in RFC 4301 [40].

IPsec has two consecutive phases with distinct sub-protocols:
- Phase 1: the establishment of a security association (sharing of security parameters between the two end-points)
- Phase 2: communicating securely

Phase 1 usually depends on the Internet Key-Exchange (IKE) protocol [41]; this protocol uses the Diffie-Hellman asymmetric key algorithm (§3.2.8) or the Elliptic Curve Diffie-Hellman key exchange

47 Mobile users are sometimes referred to as "road warriors"
48 Sometimes, even browser-based applications that run over SSL are referred to as being on a VPN

algorithm (§3.2.5) to securely exchange information necessary to establish a secure session between two end-points (this usually includes symmetric key material). It uses either a pre-shared secret or X.509 certificates to authenticate the end-points.

Phase 2 can be operated in either authenticating mode (with only message authentication), or in encrypted mode (with both message authentication as well as confidentiality). Both these modes in turn can operate in tunnelling mode and in transport mode. In tunnelling mode, the entire IP packet is encapsulated before it is transmitted. In transport mode, it is just the IP packet's payload that is encapsulated; the rest of the packet traverses the wire normally. Normally, tunnelling is the mode that is used for creating a VPN. This has several advantages:

- Tunnelling mode does not require IPsec to be integrated fully in the IP implementation of the system
- Tunnelling mode prevents traffic analysis because it hides all the headers of the secured IP packet

From a cryptographic perspective, authenticating mode and encrypting mode are mostly similar. Both rely on the security association established in phase 1. Both also use a secure hash algorithm (§3.3) to guarantee message authenticity and integrity. The only difference is that encrypting mode additionally uses symmetric cryptographic algorithms (§3.1) to provide message confidentiality.

4.6.5 Advice when selecting VPN products

VPN technology is very popular, especially in the enterprise arena. There are countless commercial VPN vendors supplying a large variety of VPN solutions. Choosing the right VPN solution is difficult because it depends on a number of factors that are highly specific to an organisation (such as security policies, network lay-out, etc.). It is therefore only possible to give general advice on selecting VPN products:

- Select a product that integrates well with the existing network infrastructure (many network equipment manufacturers also supply VPN solutions)
- Choose a hardware accelerated[49] (appliance based) solution if the traffic volume is expected to be large, go for software otherwise (which is invariably cheaper)
- Choose a solution with flexible authentication mechanisms; many VPN solutions support integration of strong authentication (§4.2), which can be of great benefit because of the added security over username/password

4.6.6 Open source VPN

There are two well-known fully-fledged open source VPN solutions:

- Openswan[50] – This is an open source IPsec implementation for Linux; it claims to be compatible with a large number of IPsec implementations by other vendors[51]
- OpenVPN[52] – This is an open source VPN solution based on SSL; it is easier to configure than IPsec based VPN solutions. OpenVPN is developed by a commercial vendor but the open source version is available free of charge.

4.6.7 Commercial VPN vendors

Commercial VPN vendors include:

- IPsec VPN: Cisco, Juniper, CheckPoint, Fortinet, SonicWALL, Stonesoft, NETASQ, Watchguard, phion, Astaro, Microsoft
- SSL VPN: AEP Networks, Microsoft, CheckPoint, Array Networks, SonicWALL, F5, Cisco, Juniper, Citrix, NeoAccel, Appgate, PortWise

4.7 Wireless Communication

4.7.1 Why use cryptography for wireless communication?

It is much easier for an attacker to eavesdrop on wireless communication than it is to eavesdrop on wired communication. There is no need for access to infrastructure that is potentially buried several meters beneath the street; instead, all that is required are an antenna and a laptop. The freedom that wireless communication gives to its users also extends to eavesdroppers.

For this reason, it is common to use cryptographic techniques to scramble wireless communications. This section will discuss three applications of cryptography in wireless communication: Wi-Fi, cellular telephony and Bluetooth.

4.7.2 Wi-Fi

Wi-Fi is the common name for wireless local area network (WLAN) technology based on the IEEE 802.11 standard [44]. It is the most widely used standard for wireless networking today. This section will focus on two aspects of the technology in which cryptography plays a role: securing data transmission and end-point authentication.

Figure 22 - Wi-Fi logo[53]

49 Such a solution includes special cryptographic hardware that can process encrypted data faster and (usually) more securely

50 http://www.openswan.org

51 http://wiki.openswan.org/index.php/Openswan/Interoperate

52 http://www.openvpn.net

53 Wi-Fi and the Wi-Fi logo are registered trademarks of the Wi-Fi Alliance; for more information see http://www.wi-fi.org

4.7.2.1 Secure data transmission

As was already stated in §4.7.1, eavesdropping on wireless data communication is trivial. For this reason, the IEEE 802.11 standard explicitly includes mechanisms to secure data communication.

Equipment vendors do not always enable wireless security[54] or have a default password. The main reason for this is that this makes it simple for users to configure their Wi-Fi setup. It is also common for public Wi-Fi hotspots to have wireless security disabled (for obvious reasons). In both cases this leaves users of these networks vulnerable to eavesdropping and all sorts of other attacks. For that reason, users of unencrypted networks are strongly encouraged to use a VPN link or only SSL secured sites over these networks. N.B.: it is becoming more common for hotspot providers to enable wireless security (for instance in hotels). One of the driving forces behind this is legislation. Several EU nations have introduced laws that mandate the use of strong cryptography to secure wireless networks to protect the privacy of their users.

Wireless security comes in several flavours:

- Wired Equivalent Privacy (WEP)
- Wi-Fi Protected Access (with Temporal Key Integrity Protocol) (WPA / WPA-TKIP)
- Wi-Fi Protected Access version 2 (WPA2)

The oldest variant is WEP. WEP is based on the RC4 symmetric key algorithm (see §3.1.5) and uses either 40-bit or 104-bit keys. WEP has serious security issues, both because of weaknesses in the WEP protocol as well as in RC4. It should no longer be used [50].

WPA was developed by the IEEE Computer Society as a response to the problems that surfaced in WEP. It is an extension of the WEP protocol based on what is called the Temporal Key Integrity Protocol. TKIP adds key mixing, sequence counters (to prevent replay attacks) and a message integrity checking mechanism to the WEP protocol. It was designed in such a way that the updated protocol could easily run on existing hardware that supports WEP. This allowed users of WEP devices to keep on using these by making it possible to upgrade their firmware. Despite adding several security features to WEP, WPA-TKIP is vulnerable to some of the same attacks and has reached the end of its designed lifetime. It should therefore no longer be used.

Both WEP and WPA-TKIP have been deprecated in the upcoming version of the 802.11 standard.

WPA2 is the current standard for wireless security on Wi-Fi networks. It uses the AES symmetric key algorithm (see §3.1.4) to secure data communication. WPA2 can be used in two modes: personal mode (using a pre-shared key, often a passphrase) or in enterprise mode (where it uses 802.1x authentication, see §4.7.2.2). Users are encouraged to use WPA2.

4.7.2.2 End-point authentication

To further enhance the security of wireless networks, end-point authentication has been added to the standard with the introduction of WPA. End-point authentication gives network administrators control over who can access the network. There are two modes of end-point authentication:

- Personal mode
- Enterprise mode

Personal mode is based on a pre-shared key. This pre-shared key is (in the case of WPA2) usually represented by a password/passphrase. The actual key that is used – in this case – is based on both the network name (SSID[55]) as well as the password/passphrase. As long as a strong password/passphrase (preferably a randomly generated string of at least 16 characters) is used and not one of the top 1000 SSIDs (see Figure 23), this is currently thought to be secure. In cases where these guidelines are not followed, Personal mode may be vulnerable to dictionary-based password/passphrase + SSID attacks[56].

Enterprise mode is based on IEEE 802.1X authentication. It leverages the Extensible Authentication Protocol (EAP). A typical setup is shown in Figure 24:

54 This is improving, however, many newer appliances force a user to select a security mechanism when the device is powered on for the first time

55 Wireless networks are identified using a Service Set IDentifier (SSID)

56 For instance, the http://www.wpacracker.com service accepts captured packets and test whether the password is in a 135 million word dictionary. For a mere $35 the service will use its 400 CPU cluster to exhaust such a dictionary in under 20 minutes

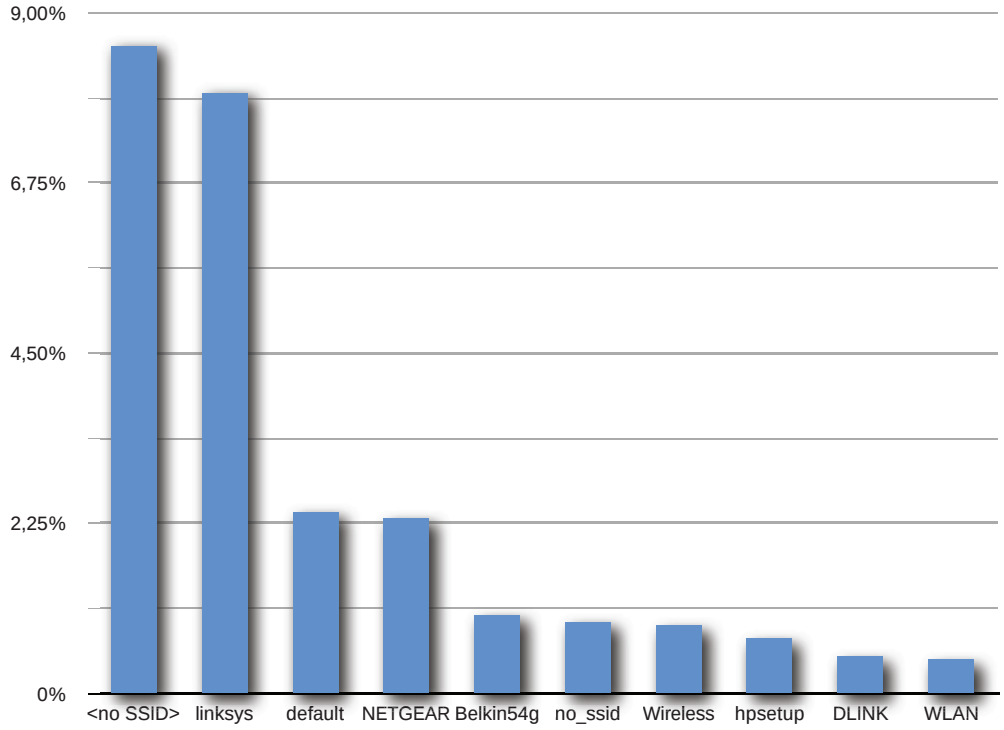


Figure 23 - Top 10 SSIDs[57]

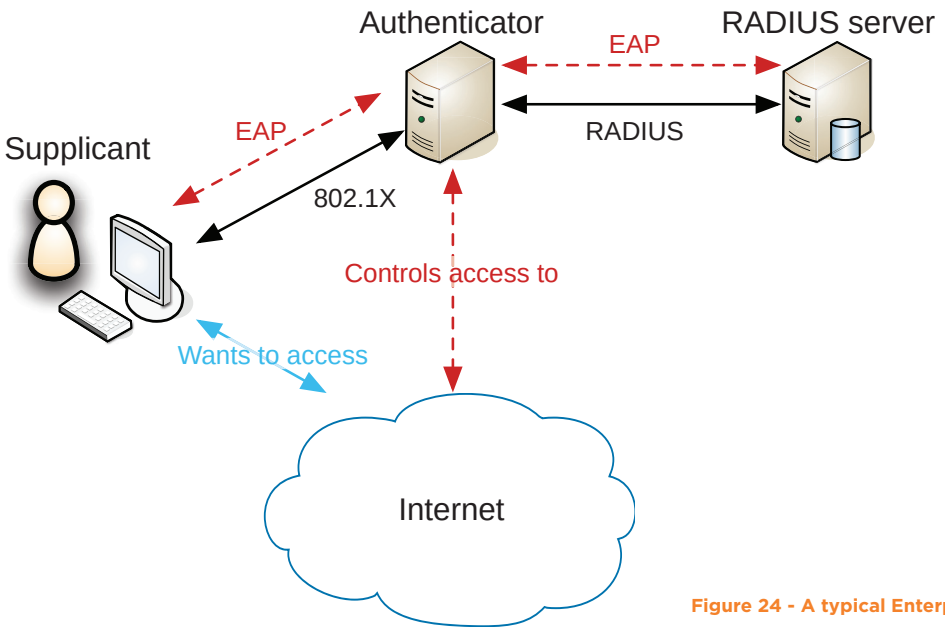


Figure 24 - A typical Enterprise mode setup

57 Source: http://www.wigle.net/gps/gps/main/ssidstats

The user (left-hand side of the diagram) who wants access to the Internet (bottom half of the diagram) runs a piece of software on his or her system called a *supplicant.* This supplicant sets up an 802.1X connection to an *authenticator* on the network. This authenticator controls access to resources such as the Internet. A commonly used system for authentication is RADIUS, a protocol that can be used for centralised authentication and authorisation (see [118]). The authenticator has a RADIUS connection to a RADIUS server. The RADIUS server has user credentials, which it can use to authenticate users.

EAP is now tunnelled (usually through an SSL tunnel) from the supplicant to the RADIUS server to authenticate the user. This can be done, for instance, using username/password or using X.509 certificates. If the RADIUS server can successfully authenticate the user, it will signal this to the authenticator, which then grants the user access to the resource it controls (in this case the Internet).

Cryptography plays a role in several aspects of this setup. EAP is almost always set up to operate over a secure tunnel (for instance an SSL tunnel). Furthermore, Enterprise mode allows mechanisms to be used that provide mutual X.509 based authentication; as a consequence it is even possible to use strong authentication (see §4.2) to access Enterprise mode secured networks.

Enterprise mode is also used for eduroam (see §4.12.2)

4.7.2.3 Advice when selecting Wi-Fi products

When purchasing new Wi-Fi equipment, only choose equipment that supports WPA2; additionally, for eduroam, WPA2 Enterprise is required.

4.7.3 Cellular telephony (GSM) and data traffic (GPRS/UMTS)

The GSM[58] standard has been under development since the 1980s. The development was initiated by the European Conference of Postal and Telecommunications Administrations (CEPT) and the standard is currently maintained by the GSM Association[59].

The base GSM specification defines a cellular telephony network. Voice calls are transmitted as a data stream over the air to and from the cell tower, which in turn dispatches the call over the mobile operator's network.

Cryptography is used to both authenticate subscribers to the network (interestingly enough, the network itself is not authenticated to the subscribers, so there is no mutual authentication) as well as to provide confidentiality for transmitted data by applying encryption. The cryptographic algorithms used in GSM are all identified using an 'A+number' combination (e.g. A5 for encryption)[60]. The actual implementation of the algorithm is then identified using a second number (e.g. A5/1 for the actual encryption algorithm that is most commonly used). Authentication of the subscriber is done using pre-shared keys (which in the case of the subscriber are stored on the Subscriber Identity Module or SIM for short) and a challenge/response system. The mechanism used for authentication, indicated with A3/A8, consists of two parts. The A3 part is responsible for authentication, whereas A8 is responsible for generating an initial key to be used for protecting the communication channel (it is input for the A5 algorithm mentioned below). The actual algorithms that are used to implement A3 and A8 can differ from network to network. The algorithms are usually kept confidential (i.e. a security by obscurity strategy is applied). An early implementation of A3/A8, known as Comp128-1 was leaked[61] and turned out to contain serious vulnerabilities.

Confidentiality is provided by applying a GSM-specific symmetric cryptographic algorithm, which is either A5/1 or A5/2. Both algorithms are stream ciphers that were developed specifically for GSM. A5/1 was developed first and was thought to be quite strong. Recent publications [45], [46], however, show that A5/1 is vulnerable to attacks. A5/2 was developed as a weaker version of A5/1 to be used for export purposes to countries or areas where strong cryptography was not allowed. Cracking A5/2 is trivial and can be done in real-time [47].

Apart from voice connectivity, GSM also provides packet data connectivity in the form of the General Packet Radio Service (GPRS). This service utilises the same infrastructure that is also used for voice data. During the late 1990s development of the third generation (3G) of mobile networks started, and in 2001 the first 3G network was rolled out in Japan. The third generation network is often referred to as the Universal Mobile Telecommunications System (UMTS). The system offers both voice as well as broadband data connectivity.

With the introduction of UMTS, authentication was upgraded to now include mutual authentication of both the subscriber to the network as well as the network to the subscriber. A new cryptographic algorithm for over-the-air encryption was also introduced, the A5/3 algorithm (also called "Kasumi"). A5/3 is claimed to be stronger than A5/1. Ongoing cryptanalysis on the algorithm, however, has already revealed potentially serious flaws in the

58 The name GSM derives from the "Groupe Spéciale Mobile" but is currently also said to mean "Global System for Mobile Communications"

59 http://www.gsm.org/

60 See http://www.gsmworld.com/our-work/programmes-and-initiatives/fraud-and-security/gsm_security_algorithms.htm

61 http://www.gsm-security.net/papers/a3a8.shtml

algorithm [48], [49]. It should be noted, however, that these have not yielded a practical attack yet.

4.7.3.1 Advice for using cellular data services

Given the current state of affairs with respect to the security of the cryptographic algorithms used on cellular networks, it is prudent to assume that communications can be eavesdropped on. To ensure confidentiality of data transmitted over such networks, the use of VPN and/or SSL is highly recommended.

4.7.4 Bluetooth

Bluetooth is a wireless communication system designed to enable wireless data exchanges over short distances[62]. Typical applications are synchronisation of devices like PDAs with desktop systems, wireless modem functionality for mobile phones, wireless headsets and carkits for mobile telephony.

Figure 25 - Bluetooth logo

Cryptography plays a role in two aspects of Bluetooth: authentication (called *pairing* in Bluetooth terminology) and confidentiality.

4.7.4.1 Bluetooth device authentication (pairing)

Most Bluetooth enabled devices require that they be authenticated to another Bluetooth device before they can exchange data. This process is called pairing. There are two pairing methods:

- Legacy pairing – this pairing method applies to all Bluetooth versions up to 2.0
- Secure Simple Pairing – this paring method applies to all Bluetooth versions starting from 2.1 and up; devices with version 2.1 or higher may only utilise legacy pairing to be interoperable with devices that have an older (< 2.1) Bluetooth version

Legacy pairing is probably the most well known form of pairing. Both devices must supply a PIN code to the other device and the pairing only completes successfully if both devices supply the same PIN. The PIN code has a maximum length of 16 characters[63]. Some devices have no means to enter a PIN code (such as wireless headsets for phones), in almost all cases these have a default PIN like 0000 or 1234. Once a pairing is successful, both devices share a secret key that they can use for future authentication. Normally, pairing is only done once, after which both devices then rely on the shared secret to authenticate to each other. There is, however, a feature on some devices that gives the user the option to pair every time devices connect. Cryptographically, legacy pairing relies on a symmetric key algorithm[64]. The security of the legacy pairing mechanism has been researched extensively. It was found to be severely lacking in security, and researchers have proven that it is trivial to break the PIN; it takes a few milliseconds to compromise a 4 digit PIN, less than a second for a 5 digit PIN an less than 10 seconds for a 6 digit PIN [51]. Interestingly, the flaw that allows the PIN to be broken is not in the cryptographic algorithm but rather in the algorithm that is used to establish the shared secret key.

Once the PIN has been uncovered, the whole security of the Bluetooth connection between the two devices is compromised since all subsequent communications relies on it.

There are two ways to lower the risk of the connection being compromised: the first is to minimise the number of pairings (e.g. turn off the "always require pairing" feature mentioned above) and to carry out pairings in a trusted environment (i.e. where it is unlikely that the transmitted data traffic is intercepted), the second is to use (significantly) longer PINs.
Secure Simple Pairing relies on public key cryptography. It is much more secure than legacy pairing and there are currently no known attacks against it.

4.7.4.2 Confidentiality

Bluetooth devices are capable of encrypting communication. A symmetric key algorithm called E0 is used for this. There are several issues with encrypted Bluetooth communication:

- The E0 algorithm has known weaknesses
- The key used for encryption is derived from the shared secret established during pairing; if legacy pairing was used then the quality of this key material is highly questionable (§4.7.4.1)
- The encryption key length is negotiable, and a length of 8 bits is allowed (which is obviously not secure)
- Encryption must be disabled for certain Bluetooth operations; it is not always clear (to the user) when encryption is enabled or disabled

62 Short distance is relative in this respect. So-called 'long range' Bluetooth operates over distances up to 100m and with specialised equipment it is possible to extend the range further.

63 In principle, this is not limited to just numbers; in practice, however, the PIN almost always consists of numbers to enable entry on devices with only a numeric key pad (such as phones)

64 The SAFER+ algorithm that was also a round 1 candidate for AES (see §3.1.4)

4.7.4.3 Other Bluetooth threats

There are many other Bluetooth threats, none of these, however, are based on problems with the cryptography used. An overview is provided in §4.2 of [52].

4.7.4.4 Guidelines for using Bluetooth

NIST has written a comprehensive overview of the security risks in Bluetooth [52]. In general, it is good practice not to rely on the built-in security of a Bluetooth connection. If possible, for instance when using a mobile phone as a wireless modem over Bluetooth, users should add additional security by using technologies like VPN and SSL on their laptop computer.

4.8 Device Encryption

4.8.1 Definition

Device encryption (sometimes also referred to as *endpoint encryption*) is the practice of encrypting the data stored on a device. The intention is to stop unauthorised people accessing the data. Device encryption applies to all sorts of devices:

- Mobile devices such as PDAs or smartphones
- Laptop hard drives
- USB flash drives
- Portable hard drives
- Workstation or server hard drives

Although the use cases may differ from device class to device class, the general principles behind device encryption remain the same. These will be explained first.

4.8.2 Individual file versus full device encryption

There are two approaches for device encryption:

- Encryption of individual files on a device; only certain – sometimes user selected, sometimes predetermined – files are encrypted
- Full device encryption; all data on the background storage (disk or flash memory) is encrypted in full

The advantages of one over the other are not clear-cut. In fact, both techniques are sometimes combined in one setup.

Encryption of individual files offers more fine-grained control over which files are encrypted and allows the use of different credentials (keys, passwords, etc.) for different files. Full device encryption, on the other hand, automatically encrypts all data on a device, taking the need away from the user to worry about which files to encrypt. A drawback of full device encryption is that the system cannot be rebooted remotely, this should be taken into consideration when applying this technology on server systems.

4.8.3 Two-step encryption

It is common practice for a device encryption application to use a two-step method. Figure 26 shows this two-step method diagrammatically. All data on the device is encrypted using a common key (referred to as E_k in the diagram). In turn, E_k is encrypted using several different user-specific keys.

In the diagram, there are three different keys. One of these could belong to one user, the other could belong to another user and the third could belong to the system administrator and could – for instance – be used only when a user has forgotten his or her own key. This is one of the big advantages of the scheme: it provides a secure way for multiple users to access the same encrypted data without knowing each other's secret.

Another advantage is that the user keys can – for instance – be derived from a password or passphrase. It is common to give users the possibility to change their password and some organisations mandate that users must change their password once every so often. If the user key were to be used directly to encrypt the data such a change would require all the data to be re-encrypted. But because the two-step process is used, all that has to be re-encrypted is the data blob containing the actual encryption key.

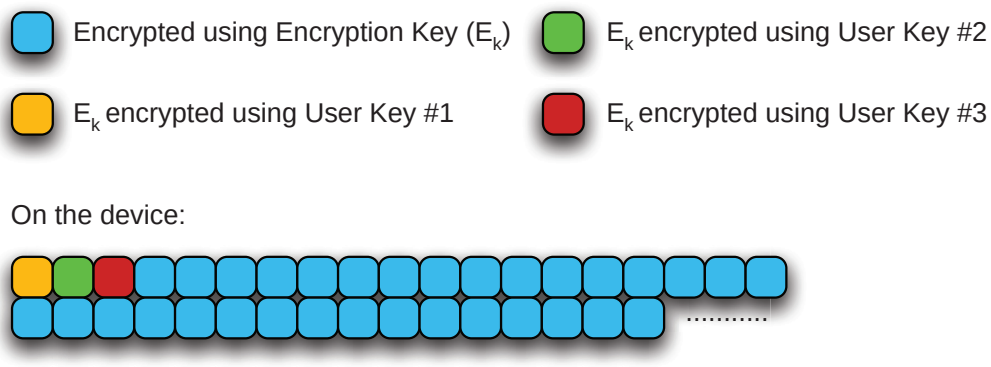


Figure 26 - Two-step encryption process

4.8.4 Key management

One of the biggest risks with device encryption is that a user forgets his or her passphrase or displaces his or her access key. It is therefore important to have some form of key management. It is common for enterprise class device encryption solutions to offer the possibility of having an administrator key. This key can be used to unlock a device and change the user credentials thus recovering access to the encrypted data.

Simpler device encryption solutions may not offer such functionality out-of-the-box. In this case it may be prudent to keep a backup of the secret (often password) used to protect the encrypted data in a secure location such as a safe. Keeping a backup of a key is called key escrow; more information on key escrow can be found in §5.3.

4.8.5 Encrypting swap space

An often overlooked but important issue is the swap space used by operating systems. Modern operating systems rely on something called virtual memory. The bottom line is that the operating system reserves space on disk where it can store data that comes from RAM and is currently not in use (for instance because the associated application is not being used). This reserved disk space is usually referred to as swap space.

A problem with this is that the data in memory can contain potentially sensitive information. It is therefore sometimes necessary to encrypt the data stored in swap space. Most operating systems support some form of encrypted swap space although it is not always trivial to enable this feature. A drawback of using encrypted swap space is that it has an impact on the performance of the system.

4.8.6 Commercial solutions

There are many commercial solutions for device encryption, most of them targeted at specific devices, some of them built-in to operating systems, others come packaged with – for instance – USB flash drives.

Below is a short summary of some popular device encryption products.

4.8.6.1 Disk encryption for desktop OSes

Major operating systems include native support for disk encryption:

- Microsoft includes Bitlocker[65] in the latest versions of its Windows operating system; Bitlocker is a full disk encryption solution
- Apple includes FileVault[66] in Mac OS X since release 10.3; FileVault limits encryption to a user's home folder

There are also many third-party full disk encryption solutions, some examples of vendors are[67]:

- WinMagic
- SafeNet
- CheckPoint Software
- PGP
- McAfee
- Sophos

4.8.6.2 USB flash drive encryption

Many large USB flash drive vendors include some sort of device encryption in their offering. The drawback of these solutions is that they often only work on Microsoft Windows. For instance, in a comparison of 7 secure USB flash drives reviewers found only one model that supports multiple operating systems [54].
An alternative to the offerings by flash drive vendors is to use an open source solution, see §4.8.7.

The European Network and Information Security Agency (ENISA) recommends the use of device encryption on USB flash drives. ENISA has published a comprehensive paper on USB flash drive security that covers all security topics relating to flash drives [55].

4.8.6.3 Mobile device encryption

The number of mobile devices is growing at a staggering rate. With this growth comes a problem: how to protect data on mobile devices. There are commercial device encryption solutions to mitigate this problem. These include solutions from:

- Checkpoint
- McAfee
- GuardianEdge (part of Symantec)
- Mobile Armor
- Credant

Note that it is hard to find products that support newer mobile operating systems. Of the vendors listed above only one claims to support Apple's iOS and none support Android.

65 http://www.microsoft.com/windows/windows-7/features/bitlocker.aspx
66 http://docs.info.apple.com/article.html?path=Mac/10.5/en/8727.html
67 For a more detailed listing, see [53]

4.8.7 Open source solutions

There are quite a few open source device encryption solutions available; there are two that are particularly popular:

4.8.7.1 TrueCrypt

TrueCrypt[68] is a free open source disk encryption solution. It supports all major desktop and server operating systems and includes a GUI for configuration. TrueCrypt can create encrypted volumes that are stored as a single file and can be mounted through the user interface. These can be stored on a disk or – for instance – on portable media such as a USB flash drive. Alternatively it also supports full volume encryption where it encrypts an entire partition or even drive. Finally, it also supports full system encryption of Windows systems with pre-boot authentication
(i.e. the entire disk is encrypted and the system will only boot if the user enters the correct password).

4.8.7.2 Linux disk encryption

There are a number of solutions for disk encryption on Linux operating systems. These include:

- Qryptix
- ecryptfs
- TrueCrypt
- Encfs
- dm-crypt
- CryptoFS

Ecryptfs is of particular interest. It is included in the much-used Red Hat Enterprise Linux distribution. Ecryptfs has an interesting feature in that it allows users to remount parts of the file system as encrypted. When the encrypted variant is not mounted, the files appear in a directory listing but are illegible because they are encrypted. Once the encrypted variant is mounted, the files become legible once more. It supports different modes of user authentication including password and public key authentication.

4.9 Digital Signatures

4.9.1 Introduction

Digital Signatures (sometimes also referred to as Electronic Signatures) are one of the most important applications of asymmetric cryptography. They are used in many different applications, and in many jurisdictions digital signatures can now replace traditional handwritten signatures for certain types of contracts.

There are similarities between handwritten signatures and digital signatures. In both cases, there are two sides to the process: signing and validation. For handwritten signatures, signing commonly is the action of writing a (stylised) version of one's name. In the case of digital signatures, the private key of an asymmetric key-pair is used to generate a cryptographic statement pertaining to a specific document. Figure 27 below shows this process:

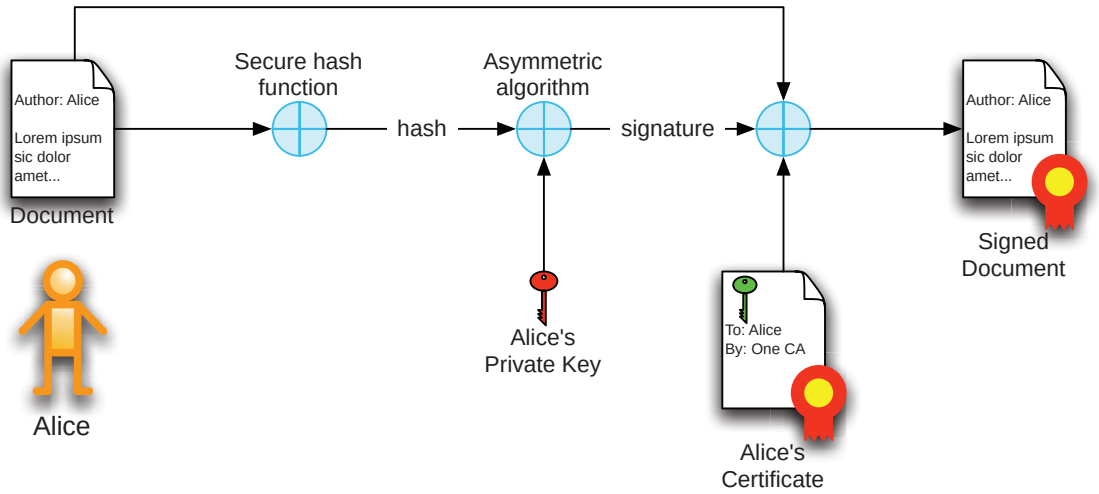


Figure 27 - Producing a digital signature

The diagram shows a user, Alice, who wants to produce a digital signature for a document (this can be any type of document, for instance an e-mail, a word processor file, etc.). To achieve this, she commonly needs two things:

- An asymmetric key-pair for which she owns the private key
- A PKI certificate that proves her identity in relation with the public key (see §3.2.10)

68 http://www.truecrypt.org/

The process now works as follows[69]:

1. Alice inputs here document into the digital signature process
2. A secure hash (a fixed-length unforgeable representation of the document, sometimes also called a *fingerprint*, see §3.3) of the document is produced
3. The private key operation is performed on the hash using Alice's private key; the output of this step is the digital signature
4. Optionally[70,71], the document, the signature and Alice's certificate are all combined into one new signed document so the recipient already has all the data required to verify the signature

At the end of the process, Alice has a digitally signed document that she can share with others.

Now, for handwritten signatures to be of any value they need to be validated. This is commonly done by comparing the signature to a previously made signature in an identification document issued by a trusted third party (in many cases a passport, ID card or driver's licence). The same applies to digital signatures. The validation process of a digital signature is shown in Figure 28.

The diagram shows a user Bob who wants to verify a document that was signed by Alice. The process has the following steps:

1. The signature and (optionally) the signer's certificate are separated from the document
2. The original document is then run through the same secure hash function that was used to produce the original signature, yielding a hash
3. The public key of the signer (commonly taken from the signer's certificate[72], which is usually included in the signed document) is input into the asymmetric key algorithm together with the signature; the output is the hash of the original document as computed by the signer
4. The final step is to compare the independently computed hash from step 2 with the output of step 3; if the hashes match then the signature is valid

Digital signatures have a number of valuable properties (some of which are not applicable to handwritten signatures):

- **Authenticity** – a digital signature proves the identity of the signer (i.e. in the example above it proves to Bob that only Alice could have signed the document)
- **Integrity** – a digital signature proves that a document was not changed after it was signed; changing the document would result in a different secure hash thus invalidating the signature
- **Non-repudiation** – a digital signature can be used such that the signer cannot claim not to have signed the document whilst her private key remains secret; it is common practice to include a time-stamp in a digitally signed document so even if the private key is exposed, previously made signatures may remain valid

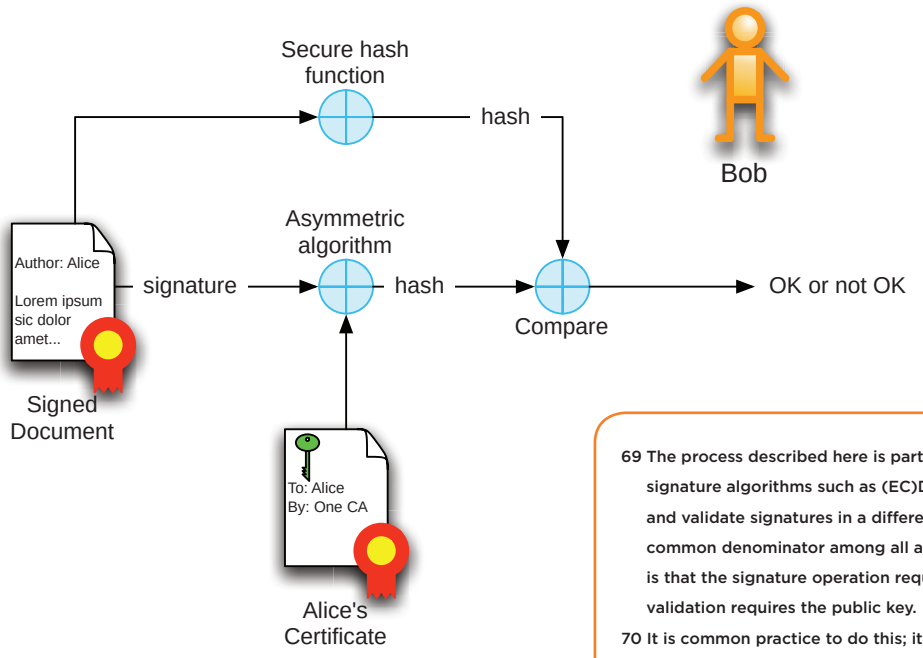


Figure 28 - Validating a digital signature

69 The process described here is particular to RSA; other signature algorithms such as (EC)DSA produce signatures and validate signatures in a different way. The greatest common denominator among all asymmetric key algorithms is that the signature operation requires the private key and validation requires the public key.

70 It is common practice to do this; it depends on the method for trust establishment, though, whether this is an X.509 certificate or, for instance, a PGP public key reference.

71 If the signature is not included in the document it is usually referred to as a detached signature.

72 The verifier, Bob, will also have to verify the validity of the certificate; how this can be done is explained in §3.2.6

Especially the integrity aspect of digital signatures is interesting when compared to a handwritten signature. It is perfectly feasible to change a multi-page document if a signature only appears on the final page (as is often the case). A digital signature, however, always applies to the entire document, thus making it impossible to change the document without invalidating the signature.

4.9.2 Legally binding digital signatures: EU Signature Directive

At the start of the new millennium the European Parliament and the European Council issued a directive concerning a *'community framework for electronic signatures'* [56]. The goal of this directive is to establish a legal framework for digital signatures, such that *'advanced electronic signatures satisfy the legal requirements of a signature in relation to data in electronic form in the same manner as a hand-written signature satisfies those requirements in relation to paper-based data'.*

What this means is that digital signatures must – if they have been created in accordance with the requirements set out in the directive – be treated in the same way – from a legal point of view – as handwritten signatures.

The directive defines high-level guidelines that address (among other things):

- The role of a signatory
- The device on which signatures are created (this should be a so-called Secure Signature Creation Device)
- The role of certificates and trusted third parties

It explicitly does not define specific algorithms or hardware to be used.

EU member states all have their own specific implementations and legislation, all adhering to the signature directive. Because this is the case, signatures made in one country using the methodology appropriate for that country are valid in any other EU member state. Unfortunately, it is a problem that each member state has its own methodology and interpretation of the directive, which makes it very hard to use a signature that was produced using one country's method in another country.

In all member states, legally binding digital signatures require what are referred to as *qualified certificates* to be used. The requirements for these certificates are set out in Annex I of the directive. They are usually supplied by certified Certificate Service Providers. There is a strong bias towards using Certificate Service Providers local to the country because there is usually a strong tie between the production and issuance method of the certificate (e.g. the certificate has to be deployed on a smart card that has to adhere to certain locally defined specifications).

Not just the EU has digital signature legislation; jurisdictions worldwide have adopted various forms of legislation, again, the implementation varies from country to country.

There are – unfortunately – not that many real-life deployments of digital signatures according to the signature directive. Examples include some countries allowing businesses to digitally sign their tax return forms and notaries using digital signatures on digital contracts.

The EU Signature Directive has had a direct impact on the digital signature marketplace because of some of the concepts it introduced (such as the Secure Signature Creation Device concept). The differing interpretations per member state have, however, led to a highly fragmented implementation of the directive. In practice, this means that the directive contributes little to cross-border acceptance of digital signatures. Parties considering deploying digital signatures within their organisation should focus on local legislation and the local implementation of the directive.

4.9.3 E-mail signing

One of the applications of digital signatures is e-mail signing. There are currently two common methods for signing e-mail: S/MIME and PGP.

S/MIME is described in [88] and builds on the MIME standard[73] and on the PKCS #7 standard by RSA [95] for encryption. The use of the PKCS series of standards in S/MIME implies that it employs a PKI based trust model. Certificate authorities (CA) issue standard X.509 certificates to end-users. The certificates (or rather the private keys associated with them)
can then be used to sign e-mail messages with an S/MIME compliant e-mail client. Typically the certificates are RSA based and the same key-pairs used for signing can also be used for encryption as described in §4.10. The recipient of a signed message must have the CA's root certificate in its trust store in order to validate the signature.

PGP (which stands for Pretty Good Privacy) is standardised as OpenPGP in [96]. The original designer of PGP is Phil Zimmerman who started a company around the PGP product in the early 1990s. The trust model of PGP, contrary to that of S/MIME, is based on a web of trust (see §3.2.9), which essentially means that users exchange public key certificates (either bilaterally or by making them available through central key repositories) and

73 MIME, which stands for Multipurpose Internet Mail Extensions, is a standard method for encoding multimedia content (images, audio, ...) in text-based media (originally e-mail messages). It is formalised in these RFCs: [89], [90], [91], [92], [93] and [94]

cross-sign certificates. A hash thumbprint of the PGP public key certificate known as the key-id is used as an identifier to lookup the recipient's key. Sometimes the key-id is printed on a business card. Central key repositories can be queried by services such as the PGP pathfinder[74] using a user's key-id in order to build up trust in a public key that was received. In such a scenario a certificate authority is no longer needed.

S/MIME, because of its PKI requirements is primarily used in the enterprise domain, whereas PGP is primarily used in environments that are less hierarchically organised such as the open source community and academia. E-mail clients such as Microsoft Outlook and Mozilla Thunderbird support S/MIME signing and encryption out-of-the-box. Extensions for PGP (based on GPG) are available for these e-mail clients.

4.9.4 Document signing

Documents can be signed on the file system level using standard utilities such as GPG and OpenSSL. The signature would then be stored alongside the document. Some document formats, however, also support signing natively so that signatures can be embedded in the document and can be displayed and checked inside document viewer applications. Prime examples of such formats are Adobe PDF and Microsoft Office's document format.

The Portable Document Format was developed by Adobe. The format was published as an open standard ISO 32000-1 (see [97]) in 2008 and is the de-facto standard for interchangeable document publishing. A digital signature in a PDF file is shown by Adobe's freely downloadable Acrobat reader as a picture similar to the picture in Figure 29. Users can interact with this object to review additional details such as the certificate chain that assures the authenticity of the author, the validity of the signature at the time of writing (and at the current time), etc. Signatures can also be inserted as invisible elements (still allowing a reader to check their validity, of course, by navigating the document viewer's menu structure).

Figure 29 – A digital signature inside a PDF document as shown by Adobe Acrobat reader

The signature features in the PDF standard are based on the PKCS stack of standards, i.e. typically a PKI is required before users can sign documents. In practice this means that the user should have a certificate and private key that are part of an enterprise PKI or they should get them from a commercial CA, although users can in principle also add individual certificates to the local trust store, thereby creating their own ad-hoc circle of trust. Both RSA and DSA are supported. Adobe Acrobat automatically checks if a certificate has been revoked. For more details see ISO 32000-1 standard [97] and the Adobe document security whitepaper [98].

Microsoft Office has supported embedded signatures since at least Office 2003 [99]. As of Office 2007 signatures are encoded in XMLDSig [100], a general purpose XML signing method standardised by the W3C XML signature working group. Like for the PDF case Microsoft uses the PKCS stack of standards to support X.509 certificates and PKI, which basically means that it uses the trust stores as managed by the operating system. Office 2007 signatures can also be either visible or invisible inside the document. When creating a visible signature, the user can select an image (e.g. of a hand written signature) to represent the digital signature.

Open Office also supports digitally signing documents in much the same way as Microsoft Office does, see [119].

4.9.5 Code signing

Code signing allows an author of a piece of software (in binary form) to show that he or she is authentic. Technically there is little difference between code signing keys and certificates and web server-, document- or e-mail signing keys and certificates. A PKI certificate can contain an attribute indicating to verifiers that the certificate's purpose includes verification of code signing signatures.

A valid signature on a piece of software warrants nothing about the quality, stability, or malignancy of the software. It is merely and assertion that proves that the software originated at the owner of the code signing certificate. As with other certificates the CA will have done a background check on the identity of the software author. While this means that the user of the software will know where to turn to if he or she experiences problems, a-priori it says nothing about the software itself.

In certain cases a valid signature on a software product does prove that the software is well behaved. An example is the Microsoft Windows driver programme. Microsoft will only sign a device driver if it has undergone significant compliance testing in line with their engineering guidelines. This means that the user of such a device driver can have some confidence that the driver will not crash the operating system. A device driver that was not signed by Microsoft will cause Windows to display a warning during installation of the driver.

74 See http://pgp.cs.uu.nl/

Code signing is an enabler of so-called mobile code security as seen in the Java platform. While executing a program, the Java runtime environment loads software components from potentially untrusted sources.

To make sure such software does not misbehave those components run inside a sandbox that isolates them from valuable resources on the host system. Certain components need access to those resources and Java allows this through an elaborate system of policies and permissions in which code signing plays a crucial role.

Similar use of signatures on software applications is seen in the various application stores for mobile devices such as Apple's App Store and the Android Market Place. This also shows how code signing is not all that different from signing of non-executable content such as audio- and video files. Code signing is also seen in areas where a device needs to make sure its internal structure has not been tampered with, such as while booting up the firmware in a set top box or game console. In fact, code signing is one of the key enablers of digital rights management (DRM) and trusted computing, see also §5.1.5.

4.10 Content Encryption

4.10.1 Introduction

Protecting the confidentiality of information is often accomplished by encrypting it. Network level encryption, used in setting up a secure channel, has become a commodity since the advent of the SSL/TLS (§4.2) protocol. Similarly, device encryption (§4.8) is becoming a standard feature on many operating systems. Still, the protection of a secure channel ends as soon as the payload has reached its destination and device encryption only works if the user has exclusive access to the device on which files are stored. In case content is exchanged via file sharing or e-mail another layer of encryption on the content itself is often used to make sure that the content is protected from prying eyes.

As was already mentioned in §4.8.4, when encryption is used it is important to consider keeping a proper backup of key material. This is especially true for mechanisms that are discussed in this section. A common scare is the so-called *disgruntled employee scenario*; this is a scenario in which an employee who has a quarrel with the organisation (for instance because of being made redundant) encrypts all his or her data shortly before leaving the organisation thus rendering it inaccessible. Another problem is if a user loses his or her key. The higher up in the organisation a user is, the bigger this problem usually becomes. When considering deploying one of the technologies discussed in a section it is worthwhile looking at key escrow (§5.3).

This section discusses three mechanisms for content encryption: e-mail encryption, document encryption and archive encryption.

4.10.2 E-mail encryption

Like e-mail signing (§4.9.3), e-mail encryption comes in two flavours. The de-facto standard for the enterprise domain is S/MIME, which uses the PKCS stack of standards to encrypt MIME formatted e-mails. PGP is the other standard and is popular amongst the Internet community at large (although applications of PGP in enterprise settings are common as well).

Both e-mail encryption mechanisms rely on public key cryptography. This means that the content will be encrypted using the public key of the recipient; thus, the sender will need to know the recipient's public key. In the case of S/MIME this means that the sender will need to have the recipient's PKI certificate, in case of PGP the sender requires the sender's PGP public key block. Obviously, sender and receiver also need e-mail clients with encryption support. If the recipient's e-mail client does not have encryption support (of the flavour used by the sender) an encrypted e-mail message will either end up looking like an encoded block of illegible ASCII characters, or like an empty message with a binary attachment.

Most modern e-mail clients support S/MIME out of the box. Setting up PGP can be a little bit tricky on some clients (Outlook is notorious) yet is relatively easy on others (Mozilla's Thunderbird has a plugin called "Enigmail", which uses the open source GNU Privacy Guard, GPG[75]). The commercial PGP desktop solution from PGP Corporation installs a proxy mail server locally on the user's desktop computer, so that configuring trust can be done outside of e-mail clients (which, as a result, need not support PGP natively).

The use of webmail services such as Yahoo mail, Microsoft Live, and GMail makes encryption by the client (now running remotely on a Web server) problematic. A possible solution is a browser plugin (e.g. FireGPG[76], which has, unfortunately been discontinued as of June 2010) which recognises encrypted blocks of data as shown in the browser by the webmail service and encrypts/decrypts them on the fly.

4.10.3 Document encryption

An alternative to having the e-mail client encrypt and decrypt messages is to simply attach an encrypted attachment to an e-mail. Usually the encryption is based on a symmetric key that is derived from a

75 http://www.gnupg.org/

76 Available from http://getfiregpg.org

77 See §3.1.5

78 See §3.3.1

user-supplied password. It follows that this should be a strong password to ensure that the derived key is strong enough to protect the content.

The PDF file format used to support two standard encryption modes: 40-bit and 128-bit, both based on the RC4[77] cipher combined with MD5[78]. As was discussed in chapter 3, neither RC4 nor MD5 are secure anymore. However, the standard also leaves room for other ciphers such as AES. Since Adobe's Acrobat version 7.0 AES with 128-bit key length is supported and version 9.0 extends this support to 256-bit AES. It is therefore advisable to upgrade Acrobat if there are plans to use PDF encryption. Note: the default setting seems to be 128-bit RC4 for backward compatibility; it is therefore important to check that the correct algorithm is used.

Acrobat supports both PKI certificate based encryption as well as password-based encryption.

All major platforms support PDF with encryption; not all readers support AES 256-bit encryption, however. Some available PDF viewers were tested for PDF encryption compatibility. The results are given in Table 6 below:

Application (OS)	Version	Comment
Adobe Acrobat reader (Windows, Mac OS X)	9.3	Supports AES 128-bit and 256-bit encryption
FoxIt Reader (Windows)	4.0	Supports AES 128-bit and 256-bit encryption
Preview (Mac OS X)	5.0	Supports AES 128-bit encryption
Evince (Linux)	2.30	Supports AES 128-bit encryption
Xpdf (Linux)	3.02	Supports AES 128-bit encryption

Table 6 - PDF encryption support in PDF readers

Microsoft's Office suite has supported password-based encryption for (Word) .doc files at least since version '97. The encryption algorithm was initially low security (40-bit) due to export restrictions.

As of Office version 2003 128-bit encryption became available in the binary .doc format, which uses the RC4 algorithm. Users can select 128-bit security (the default was still 40-bit) and can even choose a different cryptographic service provider (CSP). The algorithm seems to be limited to RC4, though. Weaknesses in RC4, especially in the applied setup in which the initialisation vector was reset upon saving a new version of the document, forced Microsoft to reconsider the use of RC4. The weaknesses allow so-called 'password recovery utilities' to compute the password if multiple versions of the document are available.

As of Office version 2007 [106] the default document format is .docx (known as "Office Open XML"). This format supports AES with a 128-bit key as default encryption algorithm. Office 2007 can open protected documents created with earlier versions, yet configuring Office 2007 to use another encryption algorithm has been made harder (i.e. requires an external tool to edit registry settings). Office products of other vendors (listed in Table 7 below) do not all support the new default encryption method.

Application (OS)	Version	Comment
Word	2003 (Windows, with compatability pack)	RC4-128 (.doc) encrypt, decrypt
Word	2007 (Windows)	AES-128 (.docx) encrypt, decrypt
Word	2008 (Mac OS X)	AES-128 encrypt, decrypt
Docs.google.com	2010	No support for encryption
OpenOffice.org	3.2.1	AES-128 (.docx) encrypt, decrypt

Table 7 - Office suite support for encryption/decryption of Word documents

Websites specialising in Office password recovery (such as the RussianPasswordCrackers) deemed the password based encryption in Office 2007 adequate in 2009[79].

79 See http://www.password-crackers.com/blog/?p=16, although recent advancements in password cracking using GPUs is mentioned as a potential new attack vector on the encryption method.

4.10.4 Archive encryption

Another category of possibly encrypted content is found in archive types such as ZIP and RAR. The ZIP format [101] allows encryption based on AES (§3.1.4) and also RC2, RC4, DES (§3.1.3), and Triple DES. Not every client supports this feature, however. Table 8 below lists common ZIP clients and their support for encryption:

Application (OS)	Version	Comment
Windows Explorer	XP SP3	No AES
Windows Explorer	Vista	No AES
WinZip	14.0	AES
WinRAR	3.70	No AES
WinRAR	3.90	AES
7Zip (Windows)	9.09 beta	AES
7Zip (A) (Linux)	4.64	AES
Unzip (Linux)	5.52	No AES

Table 8 - Encryption support in ZIP clients

As with document encryption in §4.10.3, most of these protection mechanisms are based on symmetric cryptography using keys derived from user supplied passwords. The ZIP standard, however, also allows PKI based encryption. Whether concrete products support this, is unknown.

4.11 Physical Access and Contactless Tokens

4.11.1 Introduction

Many people will be familiar with contactless smart cards in some form or other used for building access, wireless payment (for instance in public transport) and in electronic passports. Even though these cards come in various shapes and sizes (they may not even look like a card), they all work in a similar way. Figure 30 shows an example schematic of a contactless smart card:

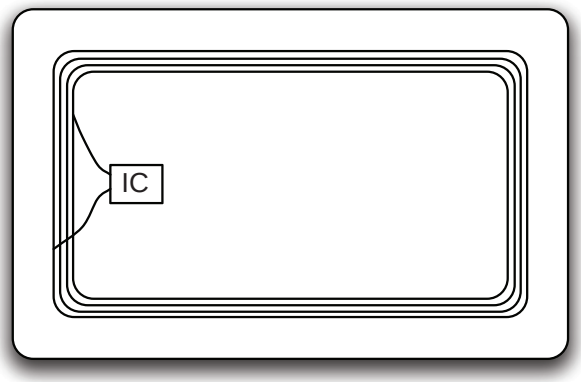


Figure 30 - Sample schematic of a contactless smart card

The diagram shows the antenna embedded in the card as well as the integrated circuit. The antenna doubles as an induction coil and uses the electro-magnetic field emitted by the reader to power the IC.

Cryptography plays a role in two aspects of contactless cards:

- Data on the device can be encrypted using a shared secret between the reader and the device
- The wireless transmissions can be secured using cryptography

4.11.2 MIFARE™

One of the most well-known contactless smart card technologies is MIFARE. According to the manufacturer NXP Semiconductors over 1 billion cards have been sold worldwide[80]. MIFARE was introduced in 1994. MIFARE is based on the international ISO 14443A standard that specifies an interface for contactless smart cards [70].

MIFARE can be seen more as a brand name rather than a single technology; there are many different 'flavours' of MIFARE:

- MIFARE Classic
- MIFARE PRO(-X)
- MIFARE Ultralight
- MIFARE DESFIRE (EV1)
- MIFARE Plus
- MIFARE Ultralight C

All flavours have different characteristics both in functionality as well as in storage capacity and cryptographic capabilities.

The most widely deployed version is the MIFARE Classic. Its popularity is mainly due to its simplicity, ease-of-use and reliability in different operating environments.

The MIFARE Classic is a card with either 1KB or 4KB of EEPROM storage and a security mechanism based on the Crypto-1 algorithm [71]. Crypto-1 is a proprietary symmetric key algorithm that was developed by the manufacturer. Recent crypto-graphic research (see [68], [69], [72], [120]) has shown Crypto-1 to be severely flawed. It is now possible to recover the Crypto-1 keys used on a MIFARE card almost instantaneously, which severely reduces the security of the system.

80 Source: http://www.mifare.net/

Entry to SURFnet's office was previously regulated using MIFARE Classic based tags that fit on a key ring. Tests with the help of researchers from the University of Nijmegen have shown that it is trivial to clone one of these tags and to gain illicit access to SURFnet's office.

New deployments should not use MIFARE Classic. In fact, SURFnet has replaced its access infrastructure with a new system based on well-published secure open algorithms. Owners of current MIFARE Classic deployments should consider replacing the system earlier than planned and should at least contact their system integrator (as advised by NXP[81]).

All other MIFARE flavours support open cryptographic algorithms, such as 3DES, AES and RSA.

4.11.3 Other products

Other commonly used contactless smart card products are:

- HID products – these include the HID Prox system that offers no cryptography-based security at all (cards can be cloned with approximately $20 in equipment) and the HID iCLASS system. The latter uses some form of cryptography (the product documentation [75] states that *"All RF data transmission between the card and reader is encrypted using a secure algorithm"* and *"Access to data stored on the cards is protected by 64 bit diversified keys..."*). It is not clear what cryptographic algorithm is used here; the fact that it uses 64-bit keys does not bode well for the security of the system. The manufacturer also claims that support for DES and Triple DES is also available for additional security (how this is used is not described).

- Legic products – these include the older Legic Prime system (from 1992), which is just as insecure as MIFARE Classic (see [73], [74]) and the Legic Advant system. The latter uses open cryptographic algorithms (either AES 128/256-bit or Triple DES) and one can assume that it is probably secure.

4.11.4 Recommendations

Owners of existing deployments of contactless access or payment systems are advised to review their current set up to check if any products were used that are now known to be insecure. They should discuss upgrade options with their system integrator. For new deployments it is recommended to use modern contactless card solutions that support open, published cryptographic algorithms instead of the deprecated proprietary algorithms as described in the previous paragraphs. Modern, presumably secure systems include:

- MIFARE PRO(-X)
- MIFARE DESFIRE (EV1)
- MIFARE Plus
- MIFARE Ultralight C
- Legic Advant

Note that almost all of these systems offer backward compatibility with their proprietary predecessors. This should in no case be relied upon for new deployments as using the cards in backward compatibility mode effectively means that they will use the insecure and deprecated proprietary algorithms.

4.11.5 Electronic passport

The electronic passport (e-passport or machine readable travel document, MRTD) which has been issued to citizens in many countries over the last couple of years also contains a contactless chip based on ISO14443. It uses cryptography to prove to an inspection system (the customs official at a foreign airport, for instance) that the passport is authentic. A secondary use is to protect the confidentiality of certain information stored in the chip on the passport (such as the highly sensitive biometric features of the holder). The international civil aviation organization (ICAO) operates a PKI in which e-passport issuing countries can sign certificates of other, trusted, countries to allow those countries to retrieve biometric information stored in their citizen's e-passports. The other countries can then verify whether the person presenting the passport is the actual owner of the passport.

4.11.6 Contactless payment cards and NFC

The major credit card companies (Visa, MasterCard) have standardised contactless payment (using contactless chips on credit cards) as an add-on on top of the EMV (Eurocard/MasterCard/Visa) standard used in the contact chips on pay cards which are by now quite common in Europe[82]. The Visa product is known as payWave, while the MasterCard brand is known as PayPass.

Contactless payment systems based on Near Field Communication (NFC) is believed to be the next step. This is usually implemented embedded in a mobile phone in such a way that the SIM can be used for storing user credentials. Despite many pilots in the last couple of years, widespread use of NFC has not yet been seen. NFC uses the same protocols and standards (i.e. based on ISO14443 as contactless credit cards and is compatible with EMV and various tags (such as MIFARE).

81 MIFARE Classic Security Frequently Asked Questions: http://www.mifare.net/security/faq.asp

82 The cryptographic capabilities of embedded chips make skimming attacks (commonly used to copy magnetic stripe information) practically impossible.

4.12 Identity Federations

Having users authenticate at every single service provider they visit can be frustrating for users, especially if a policy is in place that demands that different sets of credentials (such as passwords) are to be used for different services. Implementing and securing the authentication process can also be costly for individual service providers.

A commonly accepted solution nowadays is the formation of *identity federations* that consist of service providers and specialised trusted identity providers. In an identity federation users authenticate towards one of the identity providers as part of the process of signing in to a service provider. Service providers trust that the identity providers perform verification of a claimed identity adequately.

Since federation users will only need to authenticate towards a small set of identity providers, time and effort can be spent on stronger authentication means. In a non-federated world an authentication method based on strong cryptography (§4.2) would not be cost-effective for many services, meaning that the service providers would have to fall back to username/password. In a federated world, however, the cost of a strong authentication solution is shared between many service providers. In this sense, identity federation is an enabler for cryptographic authentication tokens.

The identity providers assert the identity of users to service providers. This is accomplished by having the identity providers store attributes of the user's identity (such as name, address, e-mail, etc.) that are communicated to service providers during sign-on of a user. When an identity attribute carrying a concrete value (such as "the e-mail address of this user is johndoe@mail.com") is communicated from an identity provider to a service provider this is sometimes also referred to as an identity *claim*. The identity provider signs claims using asymmetric cryptography. This enables the service provider to verify the validity of a claim without consulting the identity provider that issued the claim.

For users the primary advantage of identity federations is that they can now use the same set of credentials to prove possession of a claimed identity for all service providers in an identity federation. Another advantage is that users can access multiple service providers during a session without having to enter their credentials multiple times. This combination of features is called *single sign-on* (SSO).

For service providers the primary advantage is that they can *outsource* part of the identity management process to a specialised party and thus share the cost of authentication and attribute management with other service providers. A drawback is that a trust relation has to be established with the identity providers in the federation. Such a relation may require the establishment of an intricate legal and/or organisational framework. Whether this makes sense from a business perspective very much depends on the situation.

It has been generally recognised that the only way to deal with large-scale cross-organisational federations is to adopt open standards. Examples of such standards are SAML, Information Cards, XACML, WS-Federation/WS-Trust, OpenID, and OAuth. All of these standards are web or web-services based.

4.12.1 SAML based federations

The most common identity federation protocol is the security assertion markup language (SAML). SAML is a specification describing, most importantly, the syntax of assertions to be used to exchange identity information within a federation. The assertion language is XML-based. On top of the syntax of assertions, the specification also offers *binding* mechanisms describing how such attributes can be encoded and transported using (web-based) standards. These are called the binding mechanisms. The specification also prescribes protocols involving the exchange of such assertions. SAML is widely used in identity federations such as the federations in higher education and research in Europe operated by NRENs[83].

SAML assertions are cryptographically signed by the identity provider. This means that service providers can check the authenticity and integrity of the assertion. The signatures in SAML assertions enable the different principals in a federation to trust each other, i.e. trust is enabled using cryptography.

4.12.2 eduroam

eduroam is an international application of federated identity management in which users (students and staff) at institutes of higher education can access the wireless network of other institutes that are part of the eduroam federation. Effectively this implements roaming connectivity on top of IEEE 802.1X (§4.7.2.2) networking. eduroam originated as a TERENA (Trans-European Research and Education Networking Association) initiative in 2003.

Figure 31 - The eduroam logo

83 National Research and Education Network

84 The Wikipedia entry at http://en.wikipedia.org/wiki/RADIUS has an impressive list of RFCs associated with the RADIUS standard

eduroam uses the RADIUS[84] set of protocols for authentication and to some degree potentially also authorisation and accounting (see also Figure 24). Roaming eduroam users identify themselves on the guest Wi-Fi network using a so-called realm as part of their login identifier. The realm part of the identifier is used to forward the authentication request to the appropriate RADIUS server of the user's home institution. On successful authentication the user is granted access to the guest wireless network.

The eduroam case demonstrates that identity federation on a very large scale (eduroam is actually a confederation: a federation of federations) is feasible and delivers elegant solutions to real problems (ad-hoc solutions to allow guest access to Wi-Fi networks are usually not very secure).

4.13 DNSSEC

4.13.1 Introduction

The Domain Name System (DNS) is one of the basic building blocks of the Internet. Vulnerabilities in the DNS system can affect the security of the entire Internet. DNS converts logical names for resources on the Internet to IP addresses, making it possible for a user to type a logical name (such as **www.surfnet.nl**) rather than an IP address (such as **194.171.26.203**).

It has been known for a long time that DNS has a number of vulnerabilities in its basic design. However, a recent exploit (the Kaminsky attack, see [62]) has shown how easy it is to abuse these vulnerabilities, leading to a renewed sense of urgency within the Internet community. This Kaminsky attack enables evildoers to falsify information in the DNS; this can be abused to misdirect unsuspecting end users to – for instance – phishing web sites.

To address these vulnerabilities, an extension to DNS has been developed: the Domain Name System Security Extensions (DNSSEC)[85].

DNSSEC rollout on the Internet has taken off since Kaminsky published his attack. A DNSSEC signed version of the root zone (the most important DNS zone on the Internet) was introduced in July 2010 and many top-level domains have deployed DNSSEC or are in the process of doing so.

4.13.2 Signed DNS

The Domain Name System is a hierarchical system by design. Figure 32 shows a part of the DNS tree from the root down as an example.

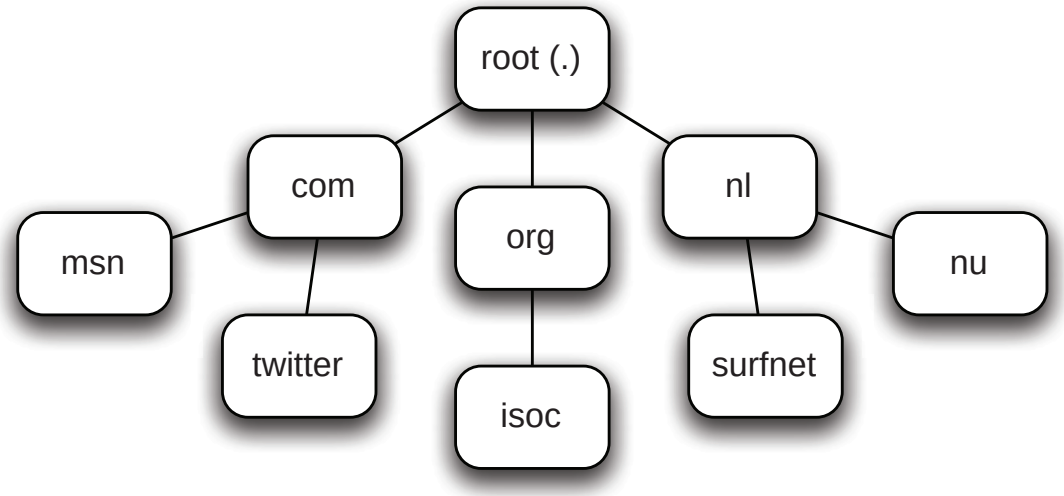


Figure 32 - Part of the DNS tree

85 SURFnet has published a comprehensive white paper on DNSSEC, see [63]

The goal of DNSSEC is to stop attackers from falsifying data by adding authenticity to the DNS. This is achieved using public key cryptography. When DNSSEC is enabled for a DNS zone all the records in that zone are digitally signed. This enables users to check the authenticity of a DNS reply that they receive.

DNS zones are signed using a two-tiered key model:

- There is a *Zone Signing Key* (ZSK) that is used to sign the individual records in a zone; characteristics of this key are:
  - It is relatively small (e.g. 1024 bits RSA)
  - It is used for a short period of time (e.g. one month)
- And there is a *Key Signing Key* (KSK) that is used to sign the ZSK; characteristics of this key are:
  - It is larger (e.g. 2048 bits RSA)
  - It is used for a long period of time (e.g. a year or more)

The reason behind this two-tiered key model is the way trust is established. Since DNS is already a hierarchical system, establishing trust is easy: trust is expressed top-down from the root, through top-level domains all the way to the end-user domain.

This trust is expressed in a so-called *trust chain*. Parent zones express trust in child zones by putting a reference to the child's KSK in their zone (this is called a *Delegation Signer* (DS) record, the reference is in fact a secure hash – see §3.3) of the public key of the child-zone KSK). And since the parent zone is also signed, this establishes trust. This mechanism works all the way up until the signed root zone. The top of the trust chain is called the *trust anchor.*

As an example, Figure 33 shows the trust chain needed to validate a signed answer for **www.surfdnssec.org:**

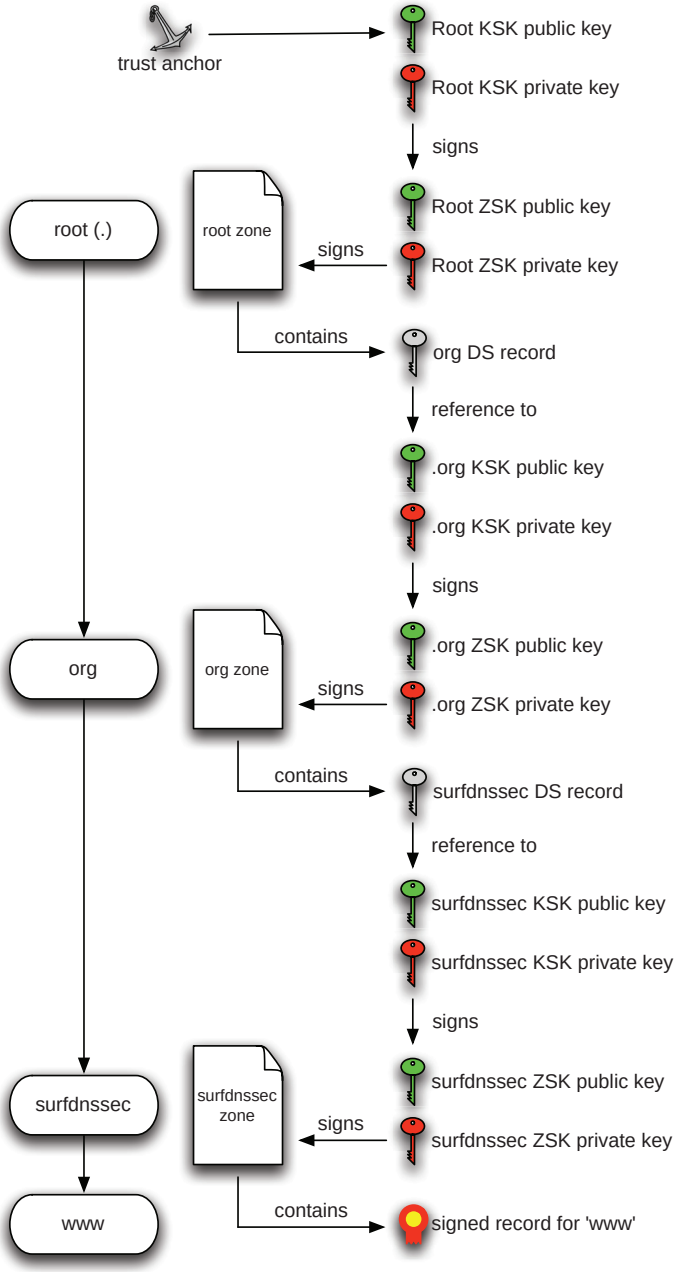


Figure 33 - Example DNSSEC trust chain

What the figure shows is that the only input needed to validate the signed record for **www.surfdnssec.org** is the root trust anchor. All other data can be retrieved from DNS.

4.13.3 Authenticated denial-of-existence

DNS has a denial-of-existence mechanism built-in. If a query is sent for a domain that does not exist, the name server will return an *NXDOMAIN* answer. With the introduction of DNSSEC it became important to add provable denial-of-existence.

Initially, this was implemented using the so-called *NSEC* algorithm. This algorithm proved non-existence by returning a signed statement (NSEC record) that proved that no records exist between two names in the sorted zone (e.g. if a record 'A' exists and the next record is 'D' then a request for 'B' will be answered with a signed statement that says: no record exists between 'A' and 'D'). The problem with this mechanism is that enumerating the zone becomes trivial. This is perceived by some as a security or a privacy risk and is prohibited in certain jurisdictions.

To address this problem, a successor to NSEC called NSEC3 was introduced. Instead of returning a signed statement that says that no record exists between two other records, a secure hash of record names is used. Then a signed statement is issued that says that no record exists between hash('record X1') and hash('record X2'). The hash values are assumed to be sorted in numerical order.

POLICIES AND PROCEDURES

5

5 POLICIES AND PROCEDURES

5.1 Secure storage of key material

5.1.1 Introduction

Cryptographic key material is usually very sensitive information. The name secret key or private key suggests that the data should be closely guarded. The level of protection varies depending on the importance of the key material. There are two dimensions to this:

- The value of the data that is protected using the key material – for instance: is it used to sign important documents or used to protect high value banking transactions? Or is it just used to protect a backup of personal documents?
- The scope of the data that is being protected – for instance: does it protect a large e-commerce site with 100000+ users? Or does it protect a personal bank account?

In the paragraphs below a number of security measures are discussed for securely storing key material. Then, a matrix is presented that gives advice on which security measure to take in different circumstances based on the two dimensions discussed above.

5.1.2 Hardware Security Modules

By far the most ironclad solution for securely storing key material is a Hardware Security Module (HSM). HSMs are designed to securely store and process large amounts of key material i.e. hundreds to thousands of keys. (HSMs are usually expensive devices; they are commonly priced in the thousands of euros.)

In general, HSMs are usually general-purpose cryptographic devices that support both symmetric as well as asymmetric key algorithms. They have integrated special purpose CPUs to handle cryptographic operations securely and they have special secure memory for storing key material. HSMs are commonly also tamper-proof (i.e. they erase their contents when an attempt is made to physically break into them). There are several types of certifications that are applicable for HSMs; more information on these can be found in §6.2.

One of the key features of HSMs is that they can manage the entire life cycle of key material. Keys are generated and used only in the HSM and never leave the device. The HSM ensures that only authorised users have access to the key material and that it is always stored in a secure way.

HSMs are – for instance – used by banks to store master key material used for electronic financial transactions (EFT); they may also be used to protect the key material for high-value domains in DNSSEC signing setups.

There are two common form factors for HSMs. They come either as an expansion card (PCI, PCI Express, etc.) or as a networked appliance in a 19" enclosure. The latter are sometimes referred to as NetHSMs[86]. Other form factors include PCMCIA devices and SCSI based devices.

HSMs are aimed at the enterprise/large organisation market. This is reflected in the features they commonly offer, such as:

- Advanced role separation – giving specific users access to specific keys, authorisation to enable and use the HSM using secure USB tokens, N out of M authentication (where at least N out of M people need to present a token before the device can be activated) etc.
- High-availability – applies only to networked HSMs; this feature allows users to link-up multiple HSMs to improve performance and redundancy. Key material is automatically backed up between participating devices.
- Secure backup – the possibility to create backups of the stored key material in a certified way (e.g. on a special backup token).
- Audit trails – logging that reflects who did what with which key material.

HSMs integrate with applications through standard APIs:

- PKCS #11 – Public Key Cryptography Standard #11 is developed by RSA Laboratories. It is an open standard that specifies a generic application-programming interface for access to cryptographic tokens (this includes HSMs). It is the most commonly used interface and is the dominant interface on UNIX platforms.
- Microsoft CryptoAPI – this is a Windows-specific interface that is mostly used by Microsoft applications and by some[87] other applications on the Windows platform.
- Java Cryptographic Extension (JCE) – this is an extension library for Java that can be used from within Java applications and applets to access cryptographic functions.
- OpenSSL – this is an open source SSL/TLS library that also supports a large number of cryptographic algorithms. It also contains support for a number of HSMs and is – for instance – often used to integrate HSM support into the Apache Web server.

86 Note that this may cause some confusion as netHSM is a specific networked HSM product by Thales (formerly nCipher)

87 PKCS #11 is also commonly used on Windows systems.

When purchasing a HSM, make sure that the APIs are supported by the vendor for integration with applications.

Below is a list of well-known HSM vendors:

- AEP
- IBM
- Oracle (formerly Sun Microsystems)
- SafeNet
- Thales

Additional purchasing information for HSMs can be found in [76].

5.1.3 SSL Accelerators

SSL Accelerators are a special type of HSM. They are designed specifically to accelerate the computationally intensive cryptographic tasks in SSL connections (see §4.2). Commonly, these are the public key operations. In the past, SSL Accelerators were popular for secure web sites that saw high traffic volumes. With the advent of 64-bit computing, however, general purpose CPUs are now more than powerful enough to handle the high numbers of public key operations required to establish SSL sessions on high-volume sites.

5.1.4 Smart cards and USB tokens

Most people will be familiar with smart cards as almost all modern bankcards are smart cards. A smart card is more formally known as an *Integrated Circuit Card* (ICC). Other examples of smart cards include SIM cards as found in mobile phones and contactless smart cards such as those that are used for physical access and micro-payments (see also §4.11). An example is shown below in Figure 34.

Figure 34 - Example of a smart card (SIM)

Smart cards require a card reader to access them. These are available in many different forms for both embedded use as well as for desktop workstations and integrated into laptop computers. In addition to that, there are also so-called USB tokens (for an example, see Figure 35); these are in fact smart card ICs embedded in a USB enclosure and can be used directly by inserting them into a free USB port without requiring a separate reader. More recently general-purpose flash (EEPROM) memory cards have also been extended with a smart card IC. These so-called Secure SD cards include a smart card IC that is embedded in the design and can be accessed over the same interface that is used to access the flash memory.

Figure 35 - Example of a USB token

There are many different kinds of smart cards, ranging from very simple memory cards (that can only store data) to cards that only support symmetric cryptographic algorithms and all the way up to cards that support asymmetric key algorithms and elliptic curve cryptography.

Cryptographic smart cards are usually composed of three components:

- A general purpose processor
- A cryptographic co-processor
- A limited amount of secure EEPROM memory (typically only several kilobytes with larger cards supporting up to 144KB of storage)

The most popular smart cards used in the office environment are PKI smart cards. These cards support asymmetric cryptographic algorithms. Just like HSMs, these cards support the whole life cycle of cryptographic keys; they support on-board key generation and perform cryptographic operations on the card, in fact, keys usually never leave the card. As such, they are a highly secure and portable way of storing and using cryptographic keys.

Access to the key material on the card is mostly protected using a PIN code (although some cards also support biometric identification).

Smart cards are integrated into applications using the same APIs as HSMs (see §5.1.2). In addition to that, the same types of certifications that apply to HSMs also apply to smart cards (see §6.2). The biggest difference between HSMs and smart cards is the number of keys than can be stored on them. Contrary to the usually large capacity of HSMs, smart cards can only store a very limited number of keys (typically some tens of keys) and are thus only suited for personal use.

Popular applications of smart cards include electronic banking, (legally binding) digital signatures (see §4.9) and strong authentication (see §4.2).

Vendors of smart card and USB token products for desktop use include:

- ActivIdentity
- AET
- Gemalto
- Giesecke & Devrient
- Oberthur
- Sagem
- SafeNet

5.1.5 Trusted Platform Module

Most new computer systems (both laptop/desktop as well as server systems) include a so-called *Trusted Platform Module* (TPM). This is a chip that is integrated on the motherboard of the computer and has certain cryptographic capabilities. TPMs were designed by the Trusted Computing Group (TCG) and their capabilities are specified in ISO 11889 [77].

Cryptographically, the capabilities of a TPM are comparable to those of a smart card or simple HSM. The TPM can store a limited number of keys on the chip. By using a master key system, it can also securely store a potentially unlimited number of keys off-chip. These are then loaded onto the chip and decrypted using the master key (that never leaves the TPM) and securely used within the TPM.

TPMs can be used for things like disk encryption (Microsoft's Bitlocker system – which is integrated into certain Windows versions – supports TPMs). They are less suitable for storage of personal keys and certificates since the TPM is tied to the computer it is integrated into (a smart card is much more suited for this since it is portable).

The original reason why TPMs were designed is to facilitate *trusted computing*. Each TPM is equipped with a unique 2048-bit RSA key-pair called the *'endorsement key'* (EK). This key makes it possible to uniquely identify each TPM through a process called 'attestation' (proof-of-possession of the RSA key by means of digitally signing a random challenge). Trusted computing works two ways: it allows manufacturers to ensure that their software is only executed by users with a proper entitlement based on the unique TPM in their machine (*Digital Rights Management*[88], DRM), and it allows for systems only to run software that has been digitally signed to prove its authenticity and integrity.

There is a lot of criticism on trusted computing. Opponents that include the Electronic Frontier Foundation[89] and the Free Software Foundation[90] claim that trusted computing infringes user privacy (since each machine becomes identifiable based on the TPM) and consumer choice (because of DRM). It is still a highly controversial technology and opponents even claim that it could have potentially disastrous effects on basic human freedoms such as free speech.

The Trusted Computing Group on the other hand claims that they have taken steps to safeguard user privacy[91] and that the system they created is based on opt-in: user must choose to use the facilities of the TPM. With version 1.2 of the Trusted Platform Module a new mechanism for attestation based on the Endorsement Key is introduced called *Direct Anonymous Attestation* (DAA). Experts claim that this mechanism provably safeguards user privacy while still providing the required proof-of-possession required for attestation (see [78]).

Note that TPMs do not provide absolute security. Not all cryptographic operations take place in the chip. Disk encryption, for instance, often uses a two-tiered key scheme (as described in §4.8.3). In case of a TPM, this could mean that the top-level key is an asymmetric key that is stored and used securely in the TPM. This key is, however, not the one that is used to perform the actual disk encryption. This is done using a second, symmetric key that is encrypted using the key in the TPM. This second key is used to decrypt the actual data on the disk and is kept in normal memory and processed in the normal CPU. As such, it is still vulnerable to all kinds of attacks.

5.1.6 Soft token

A soft token is often an emulation in software (hence the 'soft' in soft token) of a smart card or HSM and supports the same APIs for access to the token. Contrary to HSMs and smart cards, soft tokens offer no hardware-based protection. Instead, they rely on passphrases or passwords for their security; these are used to derive a symmetric key that is used to encrypt the contents of the soft token. All modern operating systems and browsers support soft tokens. These are usually integrated in the OS or the browser.

Soft tokens offer a convenient way to store cryptographic key material in a way that is compatible with a large number of applications. The two biggest problems with soft tokens are that:

88 Digital Rights Managements is a term for technologies that can be used to regulate access to licensed content. For more information, see http://en.wikipedia.org/wiki/Digital_rights_management

89 See http://www.eff.org/wp/meditations-trusted-computing

90 Read Richard Stallman's essay on what he calls 'treacherous computing', http://www.gnu.org/philosophy/can-you-trust.html

91 See http://www.trustedcomputinggroup.org/media_room/faqs

92 Triggering a core dump of an application that performs cryptographic functions is a tried-and-tested way of uncovering key material

- attackers have access to the stored token and can thus apply brute force attacks to the data – as long as the passphrase or password that is used is strong this does not pose a real threat;
- key material needs to be in memory to use it and it is processed by the normal CPU – it is thus vulnerable to memory snooping[92].

As long as these limitations are kept in mind, soft tokens can offer a moderate to good level of security for many applications.

Examples of soft tokens that may be familiar are:

- The 'My Store' Certificate Store on Windows (used – for instance – by the Internet Explorer web browser)
- The integrated 'Software Security Device' in the Firefox web browser (see Figure 36 below)

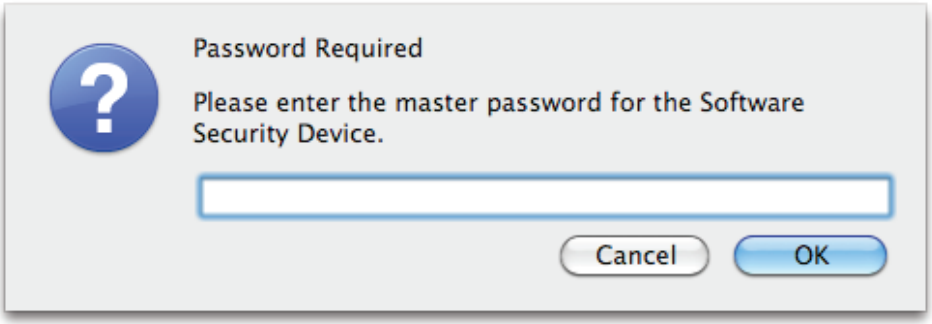


Figure 36 - Login dialog for the integrated soft token in Firefox

5.1.7 When to use which solution

Section 5.1.1 introduced two dimensions that determine which protection mechanisms are most appropriate, the value of data and the scope of data (the number of assets it protects, the impact if it becomes compromised). Figure 37 below shows these two dimensions plotted in a diagram with four quadrants and assigns the protection measures described above to one of these quadrants

To give an impression of which kinds of applications fit in which quadrant, Figure 38 below maps some common applications to the four quadrants:

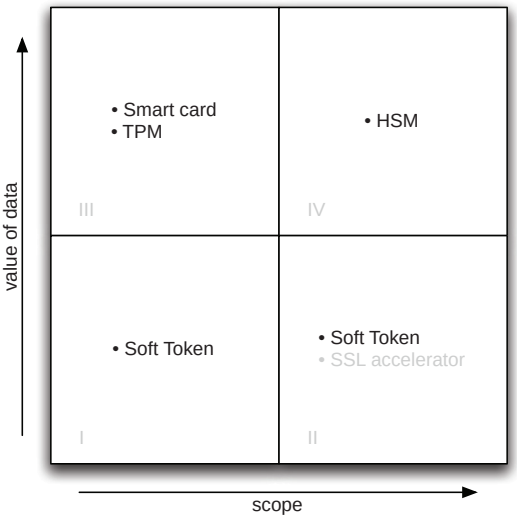


Figure 37 - Protection measures map

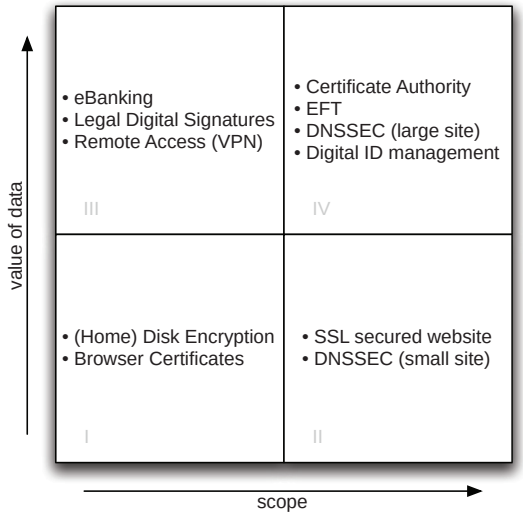


Figure 38 - Applications mapped to quadrants

5.2 Key and certificate practice statements

A certificate practice statement (CPS) is a formal document describing the procedures of a specific PKI certificate authority (CA) (see §3.2.10). A CPS is usually publicly posted by the CA on a website close to where the CA's root certificates can be downloaded. Some of the big commercial CAs for TLS/SSL certificates (Verisign, Thawte, GoDaddy, Comodo) do this. Additionally X.509 certificates contain an attribute with a URL to the CPS document on the Internet. This means that a recipient of a signed message can go and read the CPS to learn what procedures are in place at the CA that issued the certificate.

The contents of a CPS are prescribed in [103] and concrete CPSs typically contain the following sections or chapters:

1. Introduction
2. Publication and Repository
3. Identification and Authentication
4. Certificate Life-Cycle Operational Requirements
5. Facilities, Management, and Operational Controls
6. Technical Security Controls
7. Certificate, CRL, and OCSP Profile
8. Compliance audit
9. Other Business and Legal Matters

A CPS contains both technical provisions (such as 'the root certificate can be identified as follows…' and 'subject certificates should carry attributes…') to organisational procedures (such as 'transport passwords for private keys will be sent in a sealed envelope one day in advance of the key pairs on CD-ROM'). General security controls of the organisation that generates and processes certificates and keys (such as facility, personnel, backup, and audit policies) are also part of a CPS.

5.2.1 Importance of a CPS

Potential customers of a CA, i.e. subjects looking to acquire a certificate, will want to read the CPS to make sure the described procedures warrant a trustworthy protection of their identity, and consequently whether they want to pay the requested amount of money for a certificate. Users deciding whether they should trust a presented certificate by such a subject may also want to read the CA's CPS to determine whether they can trust the presented identity.

Many products (operating systems, web browsers, mobile phones) contain a default list of root certificates (the trust store) of the big commercial CAs. Whether a vendor of such a product decides to include a CA's root certificate depends on the procedures of that CA. The CPS plays a crucial role in the decision making process of vendors. The Mozilla foundation has a policy[93] describing the requirements for inclusion of CA root certificates in its products. Apple[94] and Microsoft[95] have similar policies, which also require a formal audit. For CAs it is important to have their root certificate included in the default trust stores of as many product vendors as possible since very few users of these products alter the default trust stores.

5.3 Key escrow

5.3.1 Introduction

Key escrow means that a trusted third party (such as a notary or government agency) is given access to a private or secret key owned by one of the principals taking part in a cryptographic protocol. The key is given "in escrow" to the third party, usually by just depositing a copy of the key at the third party. Under certain circumstances, say in the event of criminal investigations against one of the principals, the investigating authorities may want access to encrypted messages received by the suspect and the trusted third party assists the investigating authorities by decrypting the messages.

Some form of key escrow is also needed in backup situations. Cryptographic private keys for authentication and signing generally do not need to be backed up (if they get lost the subject is simply issued a new key-pair). Private keys for decrypting *encrypted* content, however, do need to be backed up in certain situations (such as the scenarios described in §4.8.4 and §4.10.1). If key-pairs are generated centrally, outside of tamper proof hardware, this is generally not a problem: a backup copy of the keys is made, stored in a safe place, and treated with at least the same level of security as the original. Key escrow becomes an organisational challenge in situations where keys are generated decentrally (by individual subjects) or in tamper proof hardware. PKI solutions by commercial vendors will usually generate key-pairs centrally and use temporary (symmetric) transport keys to protect private keys while being transported to the intended user or HSM and to the backup facility.

5.3.2 Key escrow controversies

Key escrow on a national scale is not uncontroversial as it requires absolute trust in the third party not to misuse (intentionally or unintentionally) the keys that were given in escrow. A particularly frightening scenario would be a government that would use its legislative powers to only allow its citizens and private industry to use encryption technologies with key escrow while holding all private keys in escrow.

93 See http://www.mozilla.org/projects/security/certs/policy/

94 See http://www.apple.com/certificateauthority/ca_program.html

95 See http://technet.microsoft.com/en-us/library/cc751157.aspx

In 1993 the US government worked on a hardware module called the Clipper chip which, according to some [104], would be a step in the direction of a government enforced key escrow scheme. The US government stopped working on the initiative in 1996 due to vulnerabilities in the used cryptographic protocol [105] discovered by Matt Blaze in 1994.

5.3.3 Key escrow or backup in practice

As mentioned on several occasions in chapter 4, it is prudent to create backups of key material when keys are used for encryption purposes. This usually concerns asymmetric private keys (symmetric keys are seldom used directly for encryption, they are either encrypted using a public key or they are derived from a password or passphrase).

When planning to deploy encryption of some sort in an organisation key backup should be an integral part of the plan. A simple but very effective way to back up key material (in case keys are generated in software) is to create a copy of the private key and store that securely. The workflow could look something like this:

1. Generate encryption key-pair
2. Generate a random secure passphrase
3. Create a copy of the private key protected with the passphrase from step 2
4. Print the passphrase and store it in a secure location
5. Save the copy of the private key on a portable medium and store it in a separate (from the passphrase) secure location
6. Issue the key to the user

In case hardware tokens like smart cards or USB tokens are used it is a little bit more complicated. In such a situation it is advisable to use a card or token management system that is responsible for personalisation of tokens for users. Most vendors of such systems will be able to offer advice on key escrow/backup.

PRODUCT SELECTION CRITERIA

6

6 PRODUCT SELECTION CRITERIA

6.1 Introduction

This chapter will address general product selection criteria that can help when choosing between different products that make use of cryptographic techniques. These criteria focus on the way in which cryptography is applied in products and the way the products themselves are designed and implemented.

6.2 Product certifications

It is common for manufacturers of security products to have the security of devices evaluated by a third party auditor. These audits are usually performed against internationally recognised standards for security evaluation. The two most important standards are FIPS 140-2 and the Common Criteria.

6.2.1 Federal Information Processing Standard (FIPS) 140-2

This standard is maintained by the National Institute of Standards and Technology (NIST), which is a United States governmental body. The FIPS 140-2 standard [79] recognises four levels to which a module can be evaluated (for a more detailed description see FIPS 140-2):

- Level 1 – the lowest level; basic security requirements are specified
- Level 2 – includes requirements for tamper evidence, user authentication
- Level 3 – includes requirements for tamper detection/resistance, data zeroisation, splitting user roles
- Level 4 – the highest level; penetration of the module has a very high probability of being detected

For most applications, a product that has at least FIPS 140-2 level 2 certification is sufficient. Note that operating a product in FIPS 140-2 level 3 or higher mode may impose restrictions on on-board key generation through the PKCS #11 API that may be incompatible with certain applications.

6.2.2 Common Criteria Evaluation Assurance Levels (CC-EAL)

The Common Criteria[96] are an internationally recognised set of standards for evaluating security hardware and software. It is a highly regulated process with the following characteristics:

- The product or system under evaluation is referred to as the *'Target of Evaluation'* or TOE for short
- The Target of Evaluation is evaluated against a Protection Profile (PP); a profile defined by a user or user community specifically for the evaluation of certain classes of products
- The evaluation is performed on the basis of a so-called *'Security Target'* (ST) which is a detailed document describing the security functions of the *Target of Evaluation* referring tot the Protection Profile (a Security Target can be seen as a refinement of a Protection Profile)
- When a product has been evaluated the depth of and confidence in the evaluation is expressed as an *Evaluation Assurance Level* (EAL) ranging from 1 to 7 where 1 is the lowest and 7 the highest qualification.

For most applications, an evaluation level of EAL 4 or up suffices.

Please note: a Common Criteria certification only has value if the Protection Profile is strict enough. Therefore, a Common Criteria certification can only really be trusted if the Protection Profile that was used for the evaluation is checked (this is very difficult for outsiders). Within the EU, the Protection Profile for Secure Signature Creation Devices (SSCD) (see [80]) is a valuable profile for evaluation. An extensive list of US government approved protection profiles can be found in [81].

In his book [82] Ross Anderson is rather critical of the Common Criteria approach towards product evaluation. Anderson's main criticism is on the system of stakeholders writing the protection profiles and evaluating products against them. Some PPs ignore known threats other PPs are created to be purposefully lax or strict so as to influence decision makers (such as governments) to purchase products from specific vendors (see the section "Corruption, Manipulation, and Inertia" in Chapter 23 of Anderson's book). Another point of critique by Anderson is that PPs focus solely on (limitations of) technical aspects of the design, whereas decision makers have to also take procedural and organisational measures into account.

6.3 Published vs. unpublished algorithms

Section 2.3 already introduced Kerckhoffs' principle and explained why 'security by obscurity' is a bad thing to rely on. This is even more relevant for cryptographic algorithms. Every now and again, a company or individual turns up with claims like:

- 'Unbreakable new algorithm'
- 'Outperforms anything on the market today'
- '10000-bit security that no one can break'
- etc.

96 http://www.commoncriteriaportal.org/theccra.html

Invariably these same companies or individuals will claim that their algorithm is a trade secret and that they should be trusted.

Claims like these should be treated with the utmost scepticism. Past experience has shown that such 'security by obscurity' is a very bad practice and examples like the German Enigma of World War II and the recently broken Crypto-1 cipher used in MIFARE eloquently show why.

It is very rare for a completely new cryptographic algorithm to be 'discovered', so such claims are usually false. But there is another good reason to rely on published rather than unpublished (secret) algorithms: peer review.

The scientific community around cryptography is constantly investigating published algorithms. These efforts are either rewarded with broken algorithms, in which case this will lead users to move to new or other solutions or they sometimes result in proving that algorithms are secure. What this process has foremost shown, however, is that it is very rare for published algorithms to be broken completely by one discovery. Usually, a weakness is discovered that degrades the security of an algorithm by several bits of key strength. This gives users enough time to take appropriate measures (such a using larger keys or migrating to a different algorithm). A good example of this is the problems that were discovered in the MD5 hash algorithm. Once researchers had shown that this algorithm might be susceptible to attacks, it still took several years to find a feasible attack. In the mean time it had been possible to migrate to newer better algorithms from the SHA family (the same is now true for SHA-1; even though a proof exists that it is vulnerable, there is still more than enough time to migrate to SHA-2 and onward as is now happening).

A final reason to rely on published rather than unpublished algorithms is the fact that cryptographers build on each other's work and experiences. Even governments such as the United States Government who used to have a tradition of secretly designing their own algorithms (although they did publish them) like DES and SHA-1 have now come back on that policy and are instead relying on open, public competitions to come up with new algorithms such as was done for AES and is currently being done for SHA-3.

6.4 Open source vs. closed source

The distinction between open source and closed source implementations of cryptographic algorithms is much less clear-cut than the distinction between published and unpublished algorithms.

For software-only implementations of cryptographic algorithms (i.e. implementations that do not rely on custom hardware such as smart cards or HSMs), there is an advantage to these implementations being open source. The main advantage is that everyone is free to inspect the source to see whether or not algorithms have been implemented correctly, whether all current state-of-the-art security measures have been taken (certain fine-tuning of algorithms to make them less susceptible to attacks) and that anyone can have the source code audited by a third party. "Linus' Law", as formulated by Eric S. Raymond in [83] states that "given enough eyeballs, all bugs are shallow". The "open to review by anyone" feature of open source software is sometimes compared to the peer-review process used in the academic world.

It should be noted, though, that most closed source implementations have all received some form of security auditing (see also §6.2), so these are by no means insecure.

Some would argue that closed source has the added advantage that attackers would first need to reverse engineer the binary code. Given the fact that a lot of expertise is required to perform attacks on cryptographic algorithms anyway, this is not a very strong argument.

When special hardware comes into play, such as with smart cards or HSMs, it is very rare for the (accompanying) software to be open source. For these kinds of applications one has to rely on certifications and third party security audits. It is not uncommon for high-security high-value projects to submit vendors to a detailed audit carried out by one of the many reputable security consultancy firms.

CONCLUSION AND AFTERWORD

7

7 CONCLUSION AND AFTERWORD

7.1 Conclusion

As will have become clear, cryptography is literally everywhere. It plays a quiet but important role in many day-to-day activities; when making a telephone call using a cell phone, purchasing a new computer in a web shop or getting some cash from an ATM, cryptography does its bit to improve the security.

But cryptography is not a silver bullet that solves all security woes. It needs to be used correctly in order to add to the security of a system. Indeed, there are many examples of bad applications of cryptography that have led to security disasters. Invariably, product claims like 'uses 256-bit encryption to keep your files safe' lead users to trust products. Some of these products do not live up to their claims because of mistakes in using cryptography. Unfortunately and to their detriment users mostly discover this when it's too late.

One of the most important messages of this document is not to rely on 'security by obscurity'. Any product that claims to use 'secret algorithms' that 'outperform anything on the market today both in speed as well as security' should be treated with a healthy dose of scepticism.

What will also have become clear when reading this paper is that particularly encryption (to provide privacy) is scattered across many domains and applications. There is no single solution that provides end-to-end protection all the time. One of the reasons for this the complexity of managing key material across multiple domains. This is an important area for future research and development and as such something to look out for in the future.

Finally, it is important to stay up-to-date with current developments. Cryptographic theory is a very active field of research, and many algorithms that were thought to be secure in the past have proven to be breakable and are considered weak by modern standards. The popular press often gets it wrong when discussing cryptography. Luckily, there are many good professional resources that can help in keeping pace with current developments.

7.2 Afterword

As authors we have tried to strike a balance between introducing the theory of cryptography and real-life applications. We found that many other publications focused on either one or the other and were filled with technical details about either subject. We hope this paper has helped you gain more insight into the workings and applications of cryptography but most importantly that it adds value when you need to make important decisions about products or services that rely on cryptography.

The authors would like to thank the following people for their contribution to this paper:

Reviewers
Xander Jansen (SURFnet)
Rogier Spoor (SURFnet)
René Ritzen (Utrecht University)
Frank Brokken (University of Groningen)
Bart Jacobs (Radboud University Nijmegen)
Flavio Garcia (Radboud University Nijmegen)

REFERENCES

8 REFERENCES

[1] Applied Cryptography
Bruce Schneier, John Wiley & Sons, Inc., Second Edition, November 1995

[2] Secrets and Lies – Digital Security in a Networked World
Bruce Schneier, John Wiley & Sons, Inc., January 2004

[3] Practical Cryptography
Niels Ferguson and Bruce Schneier, John Wiley & Sons, Inc., April 2003

[4] Hardening the Internet – The impact and importance of DNSSEC
Paul Brand, Rick van Rein, Roland van Rijswijk and David Yoshikawa, SURFnet bv, January 2009, http://www.surfnet.nl/Documents/DNSSSEC-web.pdf

[5] Wikipedia – Cryptography
http://en.wikipedia.org/wiki/Cryptography

[6] The Compact Oxford English Dictionary
http://www.askoxford.com/dictionaries/compact_oed/?view=uk

[7] Enigma: The Battle For the Code
Hugh Sebag Montefiore, Phoenix publishers, New Ed., October 2004

[8] Wikipedia – Caesar cipher
http://en.wikipedia.org/wiki/Caesar_cipher

[9] Wikipedia – Kerckhoffs' Principle
http://en.wikipedia.org/wiki/Kerckhoffs%27_principle

[10] Reverse-Engineering a Cryptographic RFID Tag
Karsten Nohl, David Evans, Starbug and Henryk Plötz, Proceedings of the 17th USENIX Security Symposium, July 2008
http://www.usenix.org/events/sec08/tech/full_papers/nohl/nohl.pdf

[11] Wikipedia – The Enigma Machine
http://en.wikipedia.org/wiki/Enigma_machine

[12] The Battle of the Atlantic
Gary Sheffield, BBC History, November 2009
http://www.bbc.co.uk/history/worldwars/wwtwo/battle_atlantic_01.shtml

[13] Wikipedia – Cryptanalysis
http://en.wikipedia.org/wiki/Cryptanalysis

[14] Wikipedia – Claude Shannon
http://en.wikipedia.org/wiki/Claude_E._Shannon

[15] Wikipedia – Entropy (in Information Theory)
http://en.wikipedia.org/wiki/Entropy_(information_theory)

[16] Wikipedia – Pseudo Random Number Generators
http://en.wikipedia.org/wiki/Pseudorandom_number_generator

[17] Data Encryption Standard (DES)
Federal Information Processing Standard 46-3 (FIPS PUB 46-3), reaffirmed October 1999
http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf

[18] Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher
NIST Special Publication SP800-67, William Barker, May 2008
http://csrc.nist.gov/publications/nistpubs/800-67/SP800-67.pdf

[19] Recommendation for Key Management (Part 1: General)
NIST Special Publication SP800-57, Elaine Barker, William Barker, William Burr, William Polk and Miles Smid, March 2007
http://csrc.nist.gov/publications/nistpubs/800-57/sp800-57-Part1-revised2_Mar08-2007.pdf

[20] Wikipedia – Feistel Network
http://en.wikipedia.org/wiki/Feistel_network

[21] Wikipedia – Block Cipher Modes of Operation
http://en.wikipedia.org/wiki/Block_cipher_modes_of_operation

[22] National Policy on the Use of the Advanced
Encryption Standard (AES) to Protect National Security Systems and National Security Information CNSS Policy No. 15, Fact Sheet No.1, The Committee on National Security Systems, June 2003
http://www.cnss.gov/Assets/pdf/cnssp_15_fs.pdf

[23] Wikipedia – International Data Encryption Algorithm
http://en.wikipedia.org/wiki/International_Data_Encryption_Algorithm

[24] Wikipedia – RC4
http://en.wikipedia.org/wiki/RC4

[25] Wikipedia – Blowfish
http://en.wikipedia.org/wiki/Blowfish_(cipher)

[26] Wikipedia – Twofish
http://en.wikipedia.org/wiki/Twofish

[27] Wikipedia – Tiny Encryption Algorithm
http://en.wikipedia.org/wiki/Tiny_Encryption_Algorithm

[28] Wikipedia – Public-key cryptography
http://en.wikipedia.org/wiki/Public-key_cryptography

[29] A Method for Obtaining Digital Signatures and Public-Key Cryptosystems
R.L Rivest, A. Shamir and L. Adleman, Massachusetts Institute of Technology, Cambridge MA, USA, September 1977
http://people.csail.mit.edu/rivest/Rsapaper.pdf

[30] Wikipedia – Modular Exponentiation
http://en.wikipedia.org/wiki/Modular_exponentiation

[31] Wikipedia – Quantum Computer
http://en.wikipedia.org/wiki/Quantum_computer

[32] Quantum Computing and Cryptography – Their impact on cryptographic practice
T. Moses, Entrust Inc., January 2009
http://www.entrust.ca/resources/download.cfm/23645/WP_QuantumCrypto_Jan09.pdf/?start

[33] Cryptographers Take On Quantum Computers
M. Heger, IEEE Spectrum, January 2009
http://spectrum.ieee.org/computing/software/cryptographers-take-on-quantum-computers

[34] The Digital Signature Standard (DSS)
National Institute of Standards and Technology, Gaithersburg, MD, June 2009
http://csrc.nist.gov/publications/fips/fips186-3/fips_186-3.pdf

[35] A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms
T. ElGamal, IEEE Transactions on Information Theory, v. IT-31, no. 4, 1985, p469-p472

[36] Cryptographic Apparatus and Method
U.S. Patent 4.200.770, M.E. Hellman, B.W. Diffie and R.C. Merkle, September 1977
http://www.google.com/patents?q=4200770

[37] The SSL Protocol Version 3.0, Internet Draft
A.O. Freier, P. Karlton and P.C. Kocher, IETF and Netscape Communications, November 18, 1996
http://www.mozilla.org/projects/security/pki/nss/ssl/draft302.txt

[38] The Transport Layer Security (TLS) Protocol Version 1.2
T. Dierks and E. Rescorla, IETF Network Working Group, August 2008
http://tools.ietf.org/html/rfc5246

[39] Crying Wolf: An Empirical Study of SSL Warning Effectiveness
J. Sunshine, S. Egelman, H. Almuhimedi, N. Atri and L. Faith Cranor, Proceedings of the USENIX Security Conference, 2009
http://www.usenix.org/events/sec09/tech/full_papers/sunshine.pdf

[40] Security Architecture for the Internet Protocol
S.Kent and K. Seo, IETF Network Working Group, December 2005
http://tools.ietf.org/html/rfc4301

[41] Internet Key-Exchange (IKEv2) Protocol
C. Kaufman (editor), IETF Network Working Group, December 2005
http://tools.ietf.org/html/rfc4306

[42] An Illustrated Guide to IPsec
S.J. Friedl, Unixwiz.net, August 2005
http://unixwiz.net/techtips/iguide-ipsec.html

[43] IP Authentication Header
S. Kent, IETF Network Working Group, December 2005
http://tools.ietf.org/html/rfc4302

[44] IEEE 802.11 – Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications
IEEE Computer Society, IEEE Std. 802.11™-2007, IEEE, New York, USA, June 2007
http://standards.ieee.org/getieee802/download/802.11-2007.pdf

[45] GSM – SRSLY?
K. Nohl and C. Paget, Presentation at the 26th Chaos Computer Club Congress, December 2009
http://events.ccc.de/congress/2009/Fahrplan/attachments/1519_26C3.Karsten.Nohl.GSM.pdf

[46] Cellphone Encryption Code Is Divulged
K. J. O'Brien, New York Times, December 2009
http://www.nytimes.com/2009/12/29/technology/29hack.html?_r=1

[47] Instant Ciphertext-Only Cryptanalysis of GSM Encrypted Communication
E. Barkan, E. Biham and N. Keller, Technion – Israel Institute of Technology, Journal of Cryptology 21 (3): p392-429, 2008
http://cryptome.org/gsm-crack-bbk.pdf

[48] A Related-Key Rectangle Attack on the Full KASUMI
E. Biham, O. Dunkelman and N. Keller
http://www.ma.huji.ac.il/~nkeller/kasumi.ps

[49] A Second GSM Cipher Falls
D. Fisher, Threatpost by Kaspersky Labs, January 2010
http://threatpost.com/en_us/blogs/second-gsm-cipher-falls-011110

[50] Computerworld: 'Don't use WEP for Wi-Fi security, researchers say'
P. Sayer, ComputerWorld, IDG News Service, April 2007
http://www.computerworld.com/s/article/9015559/Don_t_use_WEP_for_Wi_Fi_security_researchers_say

[51] Cracking the Bluetooth PIN
Y. Shaked and A. Wool, Tel Aviv School of Electrical Engineering Systems, 2005
http://www.eng.tau.ac.il/~yash/shaked-wool-mobisys05/

[52] Guide to Bluetooth Security
NIST Special Publication SP800-121, K. Scarfone and J. Padgette, September 2008
http://csrc.nist.gov/publications/nistpubs/800-121/SP800-121.pdf

[53] Wikipedia – Comparison of disk encryption software
http://en.wikipedia.org/wiki/Comparison_of_disk_encryption_software

[54] Review: 7 secure USB drives
B. O'Brien, R. Ericson and L. Mearian, Computerworld, March 2008
http://www.computerworld.com/s/article/9062527/Review_7_secure_USB_drives?taxonomyId=18&pageNumber=9

[55] Secure USB flash drives
ENISA, Heraklion, Greece, June 2008
http://www.enisa.europa.eu/act/ar/deliverables/2008/secure-usb-flash-drives-en

[56] Directive 1999/93/EC of the European Parliament and of the Council on a Community framework for electronic signatures
Official Journal of the European Communities, January 2000
http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=OJ:L:2000:013:0012:0020:EN:PDF

[57] Quantum Cryptography: Public Key Distribution and Coin Tossing
Charles Bennett and Gilles Brassard, Proceedings of the International Conference on Computers, Systems and Signal Processing, Bangalore, India, December 1984
http://www.research.ibm.com/people/b/bennetc/bennettc198469790513.pdf

[58] Quantum Cryptography Protocols Robust against Photon Number Splitting Attacks for Weak Laser Pulse Implementations
Valerio Scarani, Antonio Acín, Grégoire Ribordy and Nicolas Gisin, Physical Review Letters, Vol. 92, no. 5, February 2004
http://www.gapoptique.unige.ch/Publications/PDF/PRL57901.pdf

[59] Post-Quantum Cryptography (presentation)
Tanja Lange, University of Eindhoven, December 2008
http://hyperelliptic.org/tanja/vortraege/indo-PQ.pdf

[60] Quantum Hacking: Experimental Demonstration of Time-Shift Attack Against Practical Quantum Key Distribution Systems
Yi Zhao, Chi-Hang Fred Fung, Bing Qi, Christine Chen and Hoi-Kwong Lo, University of Toronto, Physical Review A #78, October 2008
http://arxiv.org/pdf/0704.3253v2

[61] Quantum Cryptography
Stephen Northcutt, SANS Institute, January 2008
http://www.sans.edu/resources/securitylab/quantum_crypto.php

[62] An Illustrated Guide to the Kaminsky DNS Vulnerability
Steve Friedl, August 2008
http://www.unixwiz.net/techtips/iguide-kaminsky-dns-vuln.html

[63] Hardening the Internet: The Impact and Importance of DNSSEC
Roland van Rijswijk, Paul Brand, Rick van Rein and David Yoshikawa, January 2009
http://www.surfnet.nl/Documents/DNSSEC-web.pdf

[64] The MD4 Message-Digest Algorithm
Ronald Rivest, IETF Network Working Group, April 1992
http://tools.ietf.org/html/rfc1320

[65] The MD5 Message-Digest Algorithm
Ronald Rivest, IETF Network Working Group, April 1992
http://tools.ietf.org/html/rfc1321

[66] Secure Hash Standard – FIPS 180-3
National Institute of Standards and Technology, Gaithersburg, MD, October 2008
http://csrc.nist.gov/publications/fips/fips180-3/fips180-3_final.pdf

[67] Cryptography Engineering
Niels Ferguson, Bruce Schneier, Tadayoshi Kohno, Wiley, March 2010

[68] A Practical Attack on MIFARE Classic
Gerhard de Koning Gans, Jaap-Henk Hoepman and Flavio D. Garcia, Nijmegen, The Netherlands, 2008
http://www.sos.cs.ru.nl/applications/rfid/2008-cardis.pdf

[69] Dismantling MIFARE Classic
Flavio D. Garcia, Gerhard de Koning Gans, Ruben Muijrers, Peter van Rossum, Roel Verdult, Ronny Wichers Schreur and Bart Jacobs, in Proceedings of the 13th Symposium on Research in Computer Security (ESORICS) 2008, LNCS, Springer
http://www.cs.ru.nl/~flaviog/publications/Dismantling.Mifare.pdf

[70] Project Details on: ISO/IEC 14443, Proximity cards (PICCs)
ISO Working Group 8 (WG8)
http://wg8.de/sd1.html#14443

[71] The Crypto1 Cipher
CryptoLib website
http://cryptolib.com/ciphers/crypto1/crypto1.png

[72] Algebraic Attacks on the Crypto-1 Stream Cipher in MiFare Classic and Oyster Cards
Nicolas T. Courtois, Karsten Nohl and Sean O'Neil, Cryptology Report 2008/166, IACR, April 2008
http://eprint.iacr.org/2008/166.pdf

[73] Legic Prime: Obscurity in Depth
Henryk Plötz and Karsten Nohl, slides for the 26th Chaos Communication Congress, December 2009
http://events.ccc.de/congress/2009/Fahrplan/attachments/1506_legic-slides.pdf

[74] Nur die halbe Wahrheit (German: Only half of the truth)
Michael Gückel, Sicherheit.info, January 2010
http://sicherheit.info/SI/cms.nsf/si.ArticlesByDocID/2103757?Open

[75] Frequently Asked Questions and Answers for iCLASS
HID Corporation
http://www.hidglobal.com/page.php?page_id=24

[76] HSM Buyer's Guide
Roland van Rijswijk, OpenDNSSEC project, March 2009
http://www.opendnssec.org/documentation/hsm-buyers-guide/

[77] Trusted Platform Module
International Standards Organisation ISO 11889
http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=50970

[78] Direct Anonymous Attestation
Ernie Brickell, Jan Camenisch and Liqun Chen, Cryptology Report 2004/205, August 2004
http://eprint.iacr.org/2004/205

[79] Security Requirements for Cryptographic Modules
Information Technology Laboratory, NIST, Gaithersburg, MD, May 2001
http://www.nist.gov/itl/upload/fips1402.pdf

[80] CWA 14169:2004 - Secure Signature Creation Devices 'EAL 4+'
European Committee for Standardization (CEN) Workshop Agreement, Brussels, 2004
ftp://ftp.cenorm.be/PUBLIC/CWAs/e-Europe/eSign/cwa14169-00-2004-Mar.pdf

[81] Archived U.S. Government Approved Protection Profiles
http://www.niap-ccevs.org/pp/archived/ (Last retrieved August 2010)

[82] Security Engineering: A Guide to Building Dependable Distributed Systems
Ross Anderson, John Wiley & Sons, 2001

[83] The Cathedral and The Bazaar
Eric S. Raymond, February 2010
http://www.catb.org/~esr/writings/cathedral-bazaar/

[84] Predicting the winner of the 2008 US Presidential Elections using a Sony Playstation 3
Marc Stevens, Arjen K. Lenstra and Benne de Weger, November 2007
http://www.win.tue.nl/hashclash/Nostradamus/

[85] Finding Collisions in the Full SHA-1
Xiaoyun Wang, Yiqun Lisa Yin and Hongbo Yu, in Proceedings of the 25th Annual Cryptology Conference (Crypto 2005), Santa Barbara, CA, August 2005
http://people.csail.mit.edu/yiqun/SHA1AttackProceedingVersion.pdf

[86] RIPEMD-160: A Strengthened Version of RIPEMD
Hans Dobbertin, Antoon Bosselaers and Bart Preneel, Cryptobytes, Vol. 3, No. 2, 1997, pp. 9-14
ftp://ftp.rsasecurity.com/pub/cryptobytes/crypto3n2.pdf

[87] The Case for Elliptic Curve Cryptography
NSA, January 2009
http://www.nsa.gov/business/programs/elliptic_curve.shtml

[88] Secure/Multipurpose Internet Mail Extensions
(S/MIME) Version 3.1 Message Specification
B. Ramsdell, IETF Network Working Group, July 2004
http://tools.ietf.org/html/rfc3851

[89] Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies
N. Freed and N. Borenstein, IETF Network Working Group, November 1996
http://tools.ietf.org/html/rfc2045

[90] Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types
N. Freed and N. Borenstein, IETF Network Working Group, November 1996
http://tools.ietf.org/html/rfc2046

[91] MIME (Multipurpose Internet Mail Extensions) Part Three: Message Header Extensions for Non-ASCII Text
K. Moore, IETF Network Working Group, November 1996
http://tools.ietf.org/html/rfc2047

[92] Media Type Specifications and Registration Procedures
N. Freed and J. Klensin, IETF Network Working Group, December 2005
http://tools.ietf.org/html/rfc4288

[93] Multipurpose Internet Mail Extensions (MIME) Part Four: Registration Procedures
N. Freed and J. Klensin, IETF Network Working Group, December 2005
http://tools.ietf.org/html/rfc4289

[94] Multipurpose Internet Mail Extensions (MIME) Part Five: Conformance Criteria and Examples
N. Freed and N. Borenstein, IETF Network Working Group, November 1996
http://tools.ietf.org/html/rfc2049

[95] PKCS #7: Cryptographic Message Syntax Standard
RSA Laboratories, November 1993
ftp://ftp.rsasecurity.com/pub/pkcs/ps/pkcs-7.ps

[96] OpenPGP Message Format
J. Callas, L. Donnerhacke, H. Finney, D. Shaw and R. Thayer, IETF Network Working Group, November 2007
http://tools.ietf.org/html/rfc4880

[97] Document management – Portable document format – Part 1: PDF 1.7
Adobe Systems Incorporated, July 2008
http://www.adobe.com/devnet/acrobat/pdfs/PDF32000_2008.pdf

[98] A Primer on document security: Technical Whitepaper
Adobe Systems Incorporated, 2007
http://www.adobe.com/security/pdfs/acrobat_livecycle_security_wp.pdf

[99] 2007 Office System Document: Digital Signing of Microsoft 2007 Office System Documents
Microsoft Corporation, 2007
http://www.microsoft.com/downloads/details.aspx?FamilyID=79d06e72-4b45-4669-9eac-0eca5821e8ff

[100] XMLDSig: XML Signature Syntax and Processing – (Second Edition)
Mark Bartel, John Boyer, Barb Fox, Brian LaMacchia and Ed Simon, The World Wide Web Consortium, June 2008
http://www.w3.org/TR/xmldsig-core/

[101] .ZIP File Format Specification version 6.3.2
PKWARE Inc., Milwaukee, WI, September 2007
http://www.pkware.com/documents/casestudies/APPNOTE.TXT

[102] The Zurich Trusted Information Channel: An Efficient Defence against Man-in-the-Middle and Malicious Software Attacks
Thomas Weigold, Thorsten Kramp, Reto Hermann, Frank Höring, Peter Buhler and Michael Baentsch, in proceedings of Trust 2008, LNCS 4968, pp. 75–91, 2008
http://www.zurich.ibm.com/pdf/csc/ZTIC-Trust-2008-final.pdf

[103] Internet X.509 Public Key Infrastructure Certificate Policy and Certification Practices Framework
S. Chokhani, W. Ford, R. Sabett, C. Merrill and S. Wu, IETF Network Working Group, November 2003
http://tools.ietf.org/html/rfc3647

[104] Wikipedia – Key Escrow
http://en.wikipedia.org/wiki/Key_escrow

[105] Protocol Failure in the Escrowed Encryption Standard
Matt Blaze, in Proceedings of the Second ACM Conference on Computer Communications Security, Fairfax, VA, November 1994
http://www.crypto.com/papers/eesproto.pdf

[106] 2007 Microsoft Office System Document Encryption
Microsoft Corporation, June 2007
http://www.microsoft.com/downloads/details.aspx?FamilyID=0444ea0e-3f62-4da0-8551-52349b70272e&displaylang=en

[107] Authentication Gap in TLS Renegotiation
Marsh Ray, November 2009
http://extendedsubset.com/?p=8

[108] Hackers blind quantum cryptographers
Zeeya Merali, Nature Online, August 2010
http://www.nature.com/news/2010/100829/full/news.2010.436.html

[109] The Secure Shell (SSH) Protocol Assigned Numbers
S. Lehtinen and C. Lonvick, IETF Network Working Group, January 2006
http://tools.ietf.org/html/rfc4250

[110] The Secure Shell (SSH) Protocol Architecture
T. Ylönen and C. Lonvick, IETF Network Working Group, January 2006
http://tools.ietf.org/html/rfc4251

[111] The Secure Shell (SSH) Authentication Protocol
T. Ylönen and C. Lonvick, IETF Network Working Group, January 2006
http://tools.ietf.org/html/rfc4252

[112] The Secure Shell (SSH) Transport Layer Protocol
T. Ylönen and C. Lonvick, IETF Network Working Group, January 2006
http://tools.ietf.org/html/rfc4253

[113] The Secure Shell (SSH) Connection Protocol
T. Ylönen and C. Lonvick, IETF Network Working Group, January 2006
http://tools.ietf.org/html/rfc4254

[114] Using DNS to Securely Publish Secure Shell (SSH) Key Fingerprints
J. Schlyter and W. Griffin, IETF Network Working Group, January 2006
http://tools.ietf.org/html/rfc4255

[115] Communication Theory of Secrecy Systems
Claude E. Shannon, ca. 1948, as re-published by Jiejun Kong
http://netlab.cs.ucla.edu/wiki/files/shannon1949.pdf

[116] On the Security and Composability of the One Time Pad
Dominik Raub, Rainer Steinwandt and Jörn Müller-Quade, in SOFSEM 2005: Theory and Practice of Computer Science, Lecture Notes in Computer Science, 2005, Vol. 3381/2005, pp. 288-297
http://www.springerlink.com/content/5qdfg38q1y902ffw/

[117] Timing Analysis of Keystrokes and SSH Timing Attacks
Dawn Xiadong Song, David Wagner and Xuqing Tian, in proceedings of the 10th USENIX Security Symposium, 2001
http://www.cs.berkeley.edu/~dawnsong/papers/ssh-timing.pdf

[118] Wikipedia – RADIUS
http://en.wikipedia.org/wiki/RADIUS

[119] How to use digital signatures
OpenOffice.org WIKI
http://wiki.services.openoffice.org/wiki/How_to_use_digital_Signatures

[120] Wirelessly Pickpocketing a Mifare Classic Card
Flavio D. Garcia, Peter van Rossum, Roel Verdult and Ronny Wichers Schreur, in proceedings of the 30th IEEE Symposium on Security and Privacy (S&P 2009), pp. 3-15, 2009
http://www.cs.ru.nl/~flaviog/publications/Pickpocketing.Mifare.pdf

COLOPHON

Authors
Roland van Rijswijk (SURFnet)
Martijn Oostdijk (Novay)

Editor
Roland van Rijswijk

Layout and graphic design
Vrije Stijl

This project is realised with the support of SURF,
the collaboration organisation for ICT in
higher education and research.

SURFnet
Postbus 19035
3501 DA Utrecht
The Netherlands

T +31 302 305 305
F +31 302 305 329
E admin@surfnet.nl
I www.surfnet.nl

September 2010