

MODULE 5



Cryptographic Applications

We have now completed our discussion of the cryptographic toolkit and the management of cryptographic keys. We have spent much of this discussion highlighting issues that need to be addressed when making decisions about what type of cryptography to implement and how to support it. It is notable that many of these issues were dependent on the application environment. We thus avoided making any ‘decisions’ about such issues during our discussion. Rather, we focussed on presenting the pros and cons of relevant choices.

We will now examine a number of applications of cryptography. While these may well be of independent interest, the real reason that we are studying these applications is to indicate the types of decisions that have been taken in specific application environments with respect to issues that we left ‘open’ in the previous chapters.

While these applications are all important, and hopefully interesting, their selection has primarily been to provide different application environments where different decisions have been taken. We will see during this discussion that the ‘right’ decisions are not always taken, first time around at least. We will also see that many decisions are taken based on tradeoffs. The chosen applications are:

Cryptography on the Internet. SSL/TLS is one of the most ubiquitous cryptographic protocols and provides an excellent example of the use of hybrid encryption to support open application environments.

Cryptography for wireless local area networks. The development of the cryptography used in wireless local area network standards provides a number of important lessons in practical cryptographic design.

Cryptography for mobile telecommunications. GSM and UMTS provide good examples of cryptographic design in relatively closed application environments.

Cryptography for secure payment card transactions. The banking industry has been one of the longest commercial users of cryptography and a wide variety of different techniques are used to support different types of payment transaction.

Cryptography for video broadcasting. Pay-TV provides a fascinating application with relatively straightforward cryptography requiring the support of fairly sophisticated key management.

Cryptography for identity cards. The Belgian eID card provides a good example of a technology that makes public-key cryptography widely available for use by other applications.

Cryptography for home users. Our last application concerns the potential use of cryptography for securing everyday home user applications such as file encryption, disk encryption and email security.

It is important to note that we will not attempt to provide a comprehensive introduction to these applications, since we are only interested in the role that cryptography plays in supporting them. In particular, for each application, we will be exploring:

What are the security requirements?

What are the application constraints that influence decision-making?

Which cryptographic primitives are deployed?

Which cryptographic algorithms and key lengths are supported?

How is key management conducted?

Once again we stress that the main reason that these particular applications have been chosen is illustrative. It is likely that some of the cryptographic decisions taken for these, and similar, applications will change over time.

At the end of this chapter you should be able to:

- Appreciate the influence of application constraints on making decisions about how to deploy cryptography.
- Compare a number of different application environments and their cryptographic requirements.
- Recognise the role that cryptography plays in supporting a range of applications.
- Justify the decisions taken regarding deployment of cryptography in different application environments.
- Identify appropriate key management techniques for a range of application environments.

12.1 Cryptography on the Internet

Perhaps one of the highest-profile uses of cryptography, at least to users of the Internet, is the *Secure Sockets Layer* (SSL) protocol. SSL is one of the three most important cryptographic protocols for establishing a secure network channel. The Internet is often modelled as a four-layer *Internet Protocol Suite*. While SSL operates at the Transport Layer of the Internet Protocol Suite, secure channels can also be established at the higher Application Layer using the *Secure Shell* (SSH)

protocol and at the lower Internet Layer using the *Internet Protocol Security* (IPsec) suite.

12.1.1 SSL background

SSL is a general communication security protocol for protecting data while it is being transferred between different locations. Although it has many applications, most users encounter SSL when securing a web connection between a client machine and a web server, for example, when making a purchase from an online store. SSL requires a reliable underlying transport protocol, hence its suitability for applications on the Internet running over the *Transmission Control Protocol* (TCP). Unlike many other applications of cryptography, the deployment of SSL is often made apparent to a user. When securing web sessions, an SSL connection may be indicated by:

- a dialogue box inviting the user to engage in a 'secure connection';
- the appearance of an icon, such as a padlock, on the web browser;
- the replacement of `http` by `https` in the web address displayed by the browser.

These indications provide a degree of assurance to the user that information exchanged during the session is 'secure', in contrast to traffic that is exchanged outside of an SSL-protected session.

SSL was developed by Netscape in the mid-1990s for use with their *Navigator* browser. It subsequently became the responsibility of the *Internet Engineering Task Force* (IETF), who develop standards for the Internet. In 1996, the IETF published a version known as *Transport Layer Security* (TLS). Subsequent versions of TLS have since been released. For simplicity, unless otherwise specified, we will choose to treat SSL and TLS as the same protocol and refer to this protocol as SSL.

12.1.2 SSL security requirements

SSL is designed to establish a 'secure channel' between two entities. To this end, the main security requirements are fairly standard, although SSL is designed to be highly configurable and all these 'requirements' are in fact optional:

Confidentiality. Data transferred over the secure channel should only be accessible to the entities at either end of the channel, and not by any attacker who monitors the channel.

Data origin authentication. Data transferred over the secure channel should be integrity-protected against an attacker who can conduct active attacks on the channel, including falsification of the origin of the data.

Entity authentication. In order to set up the secure channel, it should be possible to establish the identity of each communicating entity.

It is important to recognise that the secure channel enabled by SSL only operates between two specific applications, such as a client web browser and a web server. If the 'real' destination of data sent by the client web browser is a back-end database beyond the web server, then the transfer between the web server and the back-end database may require separate protection.

12.1.3 Cryptography used in SSL

SSL is designed for use in open environments, where it is unreasonable to expect communicating entities to have agreed or exchanged security-related information such as cryptographic keys. Such security-related information is often referred to as a *security association*. For example, in the case of a user wishing to make a purchase from an online store, there is no reason to suppose that the user has ever purchased anything from this store before. Indeed, the user may have only discovered the store just minutes before deciding to make a transaction. Thus an SSL session must have the capability of being established between 'strangers'.

As we know from Section 5.1.1, this is exactly the type of situation where public-key cryptography is most effective. However, given that the amount of data that we may want to exchange over the secure channel is unspecified, and thus may be large, it follows from our discussion in Section 5.5.2 that this situation lends itself naturally to hybrid encryption. Indeed, this is precisely how SSL works. In fact SSL uses a wide range of cryptographic primitives. For example:

1. public-key cryptography is used to enable symmetric key establishment;
2. digital signatures are used to sign certificates and facilitate entity authentication;
3. symmetric encryption is used to provide confidentiality;
4. MACs are used to provide data origin authentication and facilitate entity authentication.
5. hash functions are used as components of MACs and digital signatures, and for key derivation.

SSL also accommodates the fact that different entities involved in an SSL session may have different preferences for specific cryptographic algorithms and key lengths. Thus SSL supports a range of different algorithms, which include:

- many well-known block ciphers, such as AES, normally in CBC mode;
- HMAC, implemented using a choice of well-known hash functions such as SHA-256;
- digital signature algorithms such as RSA and DSA.

One of the first tasks to be performed at the start of an SSL session is for the two communicating entities to agree on which collection of these algorithms they will use to secure the session. The collection of different algorithms that they agree upon is often referred to as a *cipher suite*.

12.1.4 SSL protocols

SSL essentially consists of two cryptographic protocols:

Handshake Protocol. This protocol performs all the tasks that require agreement between the two entities before they set up the secure SSL channel. In particular, this protocol can be used to:

- agree on the cryptographic algorithms to be used to establish the secure channel;
- establish entity authentication;
- establish the keys that will be needed to secure the channel.

Record Protocol. This protocol implements the secure channel. This includes:

- formatting the data (for example, breaking it up into blocks);
- computing MACs on the data;
- encrypting the data.

SIMPLE SSL HANDSHAKE PROTOCOL DESCRIPTION

We will now describe the SSL Handshake Protocol. We will describe a ‘simple’ version of this protocol, which is only designed to provide unilateral entity authentication of the server to the client. This is the most common mode of use of the protocol, although we later indicate how mutual entity authentication can be added. Our description is simplified, since we primarily want to indicate the use of cryptography in this protocol. Note that the names that we use for the protocol messages are not strictly the same as the ‘official’ SSL message names. The message flow of the simplified SSL Handshake Protocol is indicated in Figure 12.1.

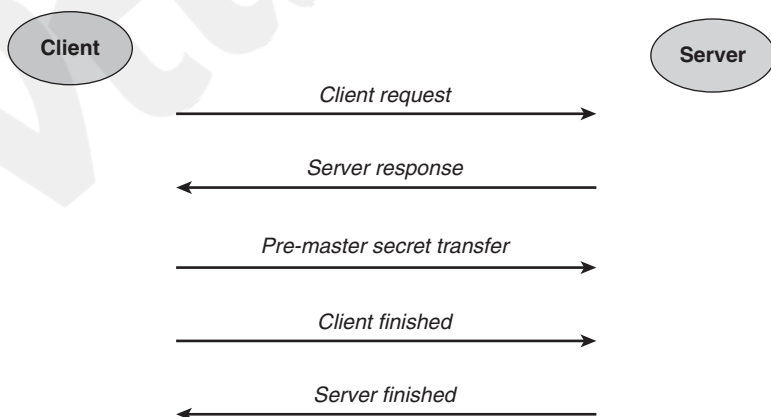


Figure 12.1. Simple SSL Handshake Protocol message flow

Client Request: This message from the client initiates the communication session and requests the establishment of an SSL-protected channel. As part of this request message, the client sends some data, including:

- a *session ID*, which acts as a unique identifier for the session;
- a pseudorandom number r_C , which will be used for the provision of freshness;
- a list of cipher suites that the client supports.

Server Response: The server responds by sending some initialisation data, including:

- the session ID;
- a pseudorandom number r_S , which is the server's freshness contribution to the protocol;
- the particular cipher suite that the server has decided to use (chosen from the list offered by the client).
- a copy of the server's public-key certificate, including details of any certificate chain required to verify this certificate.

At this point the client should check that the server's public-key certificate is valid. This will involve checking the validity of all the public-key certificates in the chain, as well as checking any relevant CRLs, as discussed in Section 11.3.3.

Pre-master Secret Transfer: The client now generates another pseudorandom number K_p , which it encrypts using the server's public key and sends to the server. Note that r_C and r_S are nonces used to provide freshness, hence they are not encrypted when they are exchanged. In contrast, K_p will be used to derive the keys that are used to secure the session, and hence must be protected when it is sent over the communication channel. This value K_p is referred to as the *pre-master secret* and is a value known only by the client and the server. (Note that it is also possible in SSL to choose to use the Diffie–Hellman protocol to establish K_p .)

The client and server are both now in a position to derive the keys that will be required to secure the SSL session. The first step in this process is to use a key derivation function to compute the *master secret* K_M . The key derivation function uses K_p as a key and takes as input, amongst other data, r_C and r_S . The client and server both then derive MAC and encryption keys from K_M (see our description of the SSL Record Protocol for details). From this point on, all messages that are exchanged are cryptographically 'protected'.

Client Finished. The client computes a MAC (the precise details of this process vary between versions, but most are based on HMAC) on the hash of all the messages sent thus far. This MAC is then encrypted and sent to the server.

Server Finished. The server checks the MAC received from the client. The server then computes a MAC on the hash of all the messages that have been sent thus far. This MAC is then encrypted and sent to the client.

Finally, the client checks the MAC received from the server.

ANALYSIS OF THE SIMPLE SSL HANDSHAKE PROTOCOL

We now confirm exactly how the simple SSL Handshake Protocol achieves its three main goals:

Agreement of cryptographic algorithms. This is achieved at the end of the second protocol message, when the server informs the client which cipher suite has been selected from the list provided by the client.

Entity authentication of the server. This relies on the following argument, assuming that the protocol run has been successful and that all checks (including certificate validity checks) have been correctly made:

1. The entity who sent the *Server Finished* message must know the master secret K_M , since the final check was correct and relied on knowledge of K_M .
2. Any entity other than the client who knows K_M must also know the pre-master secret K_P , since K_M is derived from K_P .
3. Any entity other than the client who knows K_P must know the private decryption key corresponding to the public-key certificate sent in the message *Server Response*, since this public key was used to encrypt K_P in the message *Pre-master Secret Transfer*.
4. The only entity with the ability to use the private decryption key is the genuine server, since the public-key certificate provided by the server in the message *Server Response* was checked and found to be valid.
5. The server is currently 'alive' because K_M is derived from fresh pseudorandom values (K_P and r_C) generated by the client and thus cannot be an old value.

Key establishment. SSL establishes several keys, as we will shortly discuss. These are all derived from the master secret K_M , which is a value that is established during the SSL Handshake Protocol. The master secret is derived from the pre-master secret K_P , which is a value that only the client and the server know.

Note that the *Client Finished* and *Server Finished* messages also provide retrospective data origin authentication of the entire message flow. This provides assurance that none of the messages exchanged during the SSL Handshake Protocol have been tampered with, which is particularly important since the opening messages of the protocol have no cryptographic protection.

SSL HANDSHAKE PROTOCOL WITH CLIENT AUTHENTICATION

The simple SSL Handshake Protocol does not provide mutual entity authentication, only entity authentication of the server. This is reasonable because many applications do not require client authentication at the network layer where SSL is deployed. For example, when a user purchases goods from an online store, the merchant may not care about who they are communicating with, so long as they get paid at the end of the transaction. In this scenario, client authentication is more

likely to be performed at the application layer, perhaps by using a password-based mechanism (see, for example, Section 12.4.4).

However, there are some applications of SSL, particularly in closed environments, where it may be useful to provide mutual entity authentication. In this case the simple SSL Handshake Protocol can be modified by adding an extra message from the client to the server after the message *Pre-master Secret Transfer*, as follows:

Client Authentication Data: The client sends a copy of its public-key certificate to the server. The public key in this certificate is used as a verification key. The certificate includes any details of the certificate chain required for verification. In addition, the client hashes all the protocol messages so far and digitally signs the hash using the client's signature key.

The server should now check that the client's public-key certificate (chain) is valid. The server should also verify the client's digital signature. If these checks are successful then the server has entity authentication assurance of the client by the following argument:

1. The entity who sent the *Client Authentication Data* message must know the signature key corresponding to the public key in the client's certificate, since the digital signature verified correctly.
2. The only entity who knows the signature key is the genuine client, since the public-key certificate provided by the client was checked and found to be valid.
3. The client is currently 'alive' because the digital signature was computed on a hash of some data that included the fresh pseudorandom value r_S generated by the server, and thus cannot be a replay.

SSL RECORD PROTOCOL

The SSL Record Protocol is the protocol used to instantiate the secure channel after the SSL Handshake Protocol has successfully completed. Before running the SSL Record Protocol, both the client and the server derive the cryptographic data that they will need to secure the session. This includes symmetric session keys for encryption, symmetric MAC keys and any required IVs. These are all generated using a key derivation function to compute a *key block*. This key derivation function uses K_M as a key and takes as input, amongst other data, r_C and r_S . The key block is then 'chopped up' to provide the necessary cryptographic data. In particular, the following four symmetric keys are extracted from the key block:

- K_{ECS} for symmetric encryption from the client to the server;
- K_{ESC} for symmetric encryption from the server to the client;
- K_{MCS} for MACs from the client to the server;
- K_{MSC} for MACs from the server to the client.

The SSL Record Protocol specifies the process for using these keys to protect traffic exchanged between the client and the server. For example, for data sent from the client to the server, the process is:

1. compute a MAC on the data (and various other inputs) using key K_{MCS} ;
2. append the MAC to the data and then pad, if necessary, to a multiple of the block length;
3. encrypt the resulting message using key K_{ECS} .

Upon receipt of the protected message, the server decrypts it using K_{ECS} and then verifies the recovered MAC using K_{MCS} . It is interesting to note that SSL uses a variant of the MAC-then-encrypt construction that we referred to in Section 6.3.6.

12.1.5 SSL key management

We now consider SSL key management issues by examining some of the phases of the key management lifecycle.

KEY MANAGEMENT SYSTEM

SSL essentially relies on two ‘separate’ key management systems:

Public-key management system. Since SSL is designed for use in open environments, it relies on an external key management system that governs the public-key pairs that are required by SSL users (which is *all users* if mutual entity authentication is required, or just users who are *servers* if only unilateral entity authentication is required). This key management system is beyond the scope of an SSL specification and is relied on to establish and maintain public-key certificates and information concerning their validity. If this system fails then the security provided by SSL is undermined.

Symmetric key management system. Within SSL is a self-contained symmetric key management system. SSL is used to generate symmetric sessions keys, which are designed to have limited lifetimes.

Of course these are not really independent key management systems, since there are aspects of the key management lifecycle where these two systems overlap. However, we separate them to indicate that there are certain aspects of key management that are beyond the scope of SSL itself.

KEY GENERATION

There are two types of keys deployed in SSL:

Asymmetric keys. These are generated using the public-key management system, which is not governed by the specification of SSL.

Symmetric keys. These are all generated within SSL. As described in Section 12.1.4, the session keys are all derived from the master secret that is established

following the SSL Handshake Protocol. Key derivation is a suitable technique for key generation because:

- it is a lightweight key generation technique, which does not impose significant overheads;
- it allows several different session keys to be established from just one shared secret;
- as the SSL Handshake Protocol is relatively expensive to run (it requires the use of public-key cryptography), the shared master secret can be used to establish several batches of session keys, should this be desirable.

However, the generation of the master secret does rely on the client being able to randomly generate a pre-master secret. Should the client fail to do this securely using a suitable technique (see Section 8.1) then the subsequent generation of all the SSL session keys could be compromised.

The key lengths used in SSL are all negotiable and part of the cryptographic algorithm agreement process defined in the SSL Handshake Protocol.

KEY ESTABLISHMENT

The most important key establishment process in SSL is the establishment of the pre-master secret during the SSL Handshake Protocol. Probably the most common technique for conducting this is to use RSA public-key encryption during the protocol message *Pre-master Secret Transfer*. However, a variant based on Diffie–Hellman is also supported by SSL.

KEY STORAGE

Key storage is beyond the scope of SSL, but it relies on both the client and the server securely storing relevant secret keys. The most sensitive keys to store are the private keys, since they are relied upon across multiple SSL sessions. In contrast, the symmetric keys negotiated during the SSL Handshake Protocol are only used for a relatively short period of time. Nonetheless, if they are compromised then so are any sessions that they are used to protect.

KEY USAGE

One of the interesting design aspects of SSL is how it embraces the principle of key separation, discussed in Section 10.6.1. Separate encryption and MAC keys are derived from the master secret, which are then used to establish the secure channel. However, SSL takes this principle a step further by deploying separate keys for each communication direction, which provides security against reflection attacks (see Section 9.3.4). The cost of this is low because these separate keys are derived from the common master secret.

However, largely for convenience, SSL also abuses the principle of key separation. The master secret K_M is used not only as a key derivation key, but also as a MAC key. This is a good example of pragmatism in adoption of the principle of key separation. It is enforced for the vulnerable session keys that will

be used for protecting data. It is relaxed for the less-exposed master secret, which is only publicly used to create MACs during the SSL Handshake Protocol.

12.1.6 SSL security issues

SSL is a popular communication protocol and is generally regarded as cryptographically ‘strong’ if used with respected cryptographic algorithms. Most security problems experienced using recent versions of SSL have arisen from aspects that are beyond the scope of the protocol specification. These include:

Process failures. The most common ‘failure’ of SSL arises when a client does not perform the necessary checks to validate the server’s public-key certificate. A web user who is presented with a dialogue box warning them of their browser’s inability to verify a public-key certificate is quite likely to disregard it and proceed with establishing an SSL session. Indeed, it is rather hard to place too much blame on them for doing so.

A particularly common manifestation of this problem on the Internet is when a rogue web server, holding a legitimate public-key certificate in its own name, tries to pass itself off as another web server. Even if the client web browser successfully verifies the rogue web server’s certificate chain, if the client does not notice that the public-key certificate is not in the name of the expected web server then the rogue web server will succeed in establishing an SSL protected channel with the client. This is an entity authentication failure because the client has succeeded in setting up an SSL session, but it is not with the server that they think it is with. This failure is often exploited during phishing attacks.

Note that the above phishing attack is not a failure of the SSL Handshake Protocol. It is a failure in the surrounding processes that support the protocol. In this case the client has failed to conduct a protocol action (validating the server’s certificate chain) with a sufficient degree of rigour.

Implementation failures. Because it is an open protocol that can be adopted for many different applications, on different platforms, by anyone, SSL is particularly vulnerable to implementation failures. Even if the protocol specification is followed correctly, it could fail if a supporting component is weak. For example, if the client uses a weak deterministic generator to generate the pre-master secret K_p then the protocol can be compromised because the session keys become too predictable.

Key management failures. As discussed in Section 12.1.5, if either the client or the server mismanages their cryptographic keys then the protocol can be compromised. For example, if an attacker obtains the server’s private key then the attacker can recover the pre-master secret. The attacker can then compute all the resulting session keys and hence undermine any secure channel that these session keys are used to establish.

Usage failures. SSL has such a high profile that it runs the risk of being used inappropriately. Alternatively it may be appropriately deployed, but

its security properties overestimated under the misapprehension that use of SSL ‘guarantees’ security. Perhaps the classic example of the latter is our default SSL application of securing the connection between a client web browser and the web server of an online store. Such stores often claim to offer a ‘completely secure’ service because they are deploying SSL. This of course tells the client nothing about the security of their payment card details after the web server has acquired them. After all, largely due to the use of SSL, criminals seeking payment card details are more likely to hack into a back-end database, rather than engage in large-scale monitoring of network traffic.

12.1.7 SSL design issues

Having discussed the use of cryptography in SSL at some length, we close by remarking on a few of the design issues that appear to have influenced the cryptographic choices made by the designers of SSL.

Support for a range of publicly known cryptographic algorithms. Since SSL (in this case we really mean TLS) is an open standard targeted at wide-scale public use, it is fundamental that it supports not just publicly known algorithms, but a *range* of publicly known algorithms. This supports cross-platform use and has helped to foster confidence in SSL as a protocol.

Flexibility. SSL is not only flexible in terms of the components that can be used to implement it, but it is also flexible in the ways in which it can be used (for example, to provide unilateral or mutual entity authentication). This, again, is because SSL has been targeted at a wide range of application environments.

Minimal use of public-key operations. The use of hybrid encryption restricts the number of public-key operations to the minimum necessary to establish a secure channel. Although we have not discussed this in any detail, SSL is also designed so that the relatively expensive SSL Handshake Protocol may not need to be rerun if a client requires another session with the same server within a specified time period.

Unbalanced computational requirements. Recall that in the SSL Handshake Protocol it is the client who is required to generate the pre-master secret and send it encrypted to the server. This means that the client performs one public-key *encryption* operation and the server performs one public-key *decryption* operation. One reason for this is that some public-key cryptosystems, and RSA is a good example, have certain public keys that are considerably more computationally efficient to use than others (see Section 10.3.4). Thus it is reasonable to assume that encryption is a faster operation than decryption. The design of SSL thus places a slight computational ‘burden’ on the server, which is the entity expected to have the greater computational ability.

12.2 Cryptography for wireless local area networks

It is extremely important to provide security for wireless networks, since these networks are particularly vulnerable to some types of external attack. However, one of the most interesting aspects of wireless network security is the cryptographic design errors that were made when developing certain wireless network security standards. In this section we look at the example of wireless local area networks and discuss the problems that have arisen in their underlying cryptographic design.

12.2.1 WLAN background

Many users of computers are accustomed to the inherent network ‘security’ provided by the use of physical wires to communicate between different devices. Although a determined attacker can ‘tap’ a wired communication, this requires physical access to the wires themselves. Hence many attacks on wired networks tend to focus on the machines at the ends of the wires; for example, by installing malicious software on a machine that monitors traffic being sent and received on the network by that machine.

The advent of wireless communication has brought numerous benefits, perhaps the most significant being convenience. An office or home can easily establish a network without messy wiring being installed. Also, networks can be established in places where they were once awkward to install, such as railway stations, restaurants and conference venues.

However, without the security provided by physical wires, wireless networks are much more vulnerable to attack. Without built-in security, the information exchanged over them can be monitored (and potentially modified) by anyone geographically close enough to access them. For example, access to a wired home network is, by and large, restricted to someone who can enter the building and physically access either the machines or the wires. In contrast, a wireless network is potentially accessible to someone located outside the building.

The type of wireless network that is typically deployed between devices in an office or home environment is known as a *wireless local area network* (WLAN). The international standards for WLAN communications are governed by the *Institute of Electrical and Electronics Engineers* (IEEE) and are collectively referred to as IEEE 802.11. The original version of the IEEE 802.11 standard was released in 1997, but there have been many amendments since then. Some devices that are certified to be compliant with IEEE 802.11 are labelled by the trademark *Wi-Fi*, which is an indicator of interoperability.

A simple WLAN architecture is shown in Figure 12.2. A *wireless access point* is a piece of hardware that acts as a bridge between the wireless network and a wired network (for example, the wired network that delivers a connection to the Internet from a home). The access point consists of a radio, an interface with the wired network and bridging software. A *device* is any computer (for example, a desktop

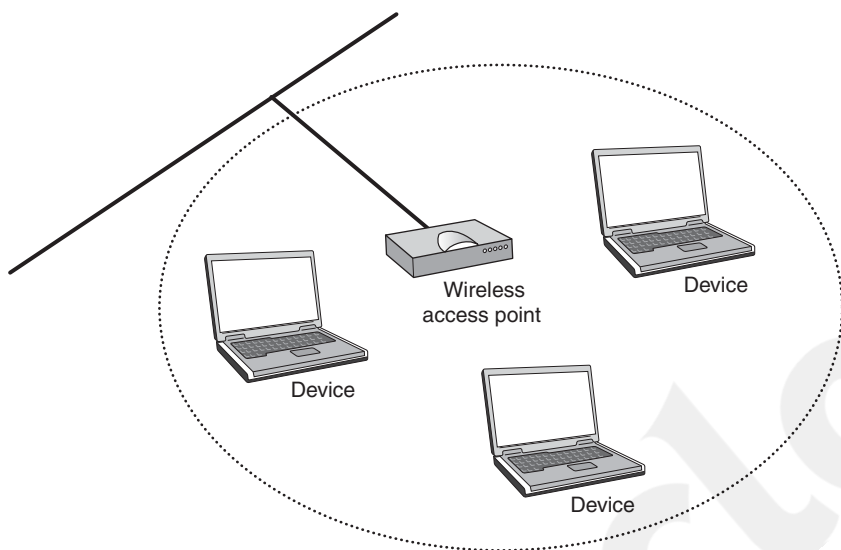


Figure 12.2. Simple WLAN architecture

PC, laptop or PDA) which has a wireless network interface card that allows it to communicate over a wireless network. A WLAN may consist of many devices all communicating with the one access point, or indeed may involve several different access points.

The original 802.11 standard defined the *Wired Equivalent Privacy* (WEP) mechanism to protect WLAN communication. WEP was designed to provide security at the data link layer, which means that it operates at a virtual networking layer that is close to being the equivalent of physical wires in a wired network. However, as we will shortly discuss, there were many serious problems with the deployment of cryptography in WEP. In 2002, an improved security mechanism known as *Wi-Fi Protected Access* (WPA) was proposed. This was intended to be a temporary solution, designed to improve security whilst being capable of running on legacy hardware. In the meantime a complete redesign of the underlying cryptographic components was underway, which was published as WPA2 in 2004 as part of the IEEE 802.11i standard. We will discuss all of these mechanisms for securing WLANs, because they not only represent an interesting and important application of cryptography, but the development process also provides several valuable cryptographic design lessons.

12.2.2 WLAN security requirements

The scope of the security requirements for a WLAN are defined by the notion that a WLAN should be *as secure as a wired network*. This idea is rather vague, since the two types of network are very different and such security equivalence cannot

be precisely established or measured. However, the notion is useful because it provides a security ‘target’ that helps to influence certain design decisions. As an example, WLAN security mechanisms are not intended to prevent denial-of-service attacks, since a wired network is not inherently protected against these; returning to our scenario of a home network, an external attacker who is located outside a building with a wired home network could potentially cut a wire that supplied communications or power to the building. In Section 12.3 we will see that a similar notion of security has been used to define the scope of security requirements for mobile telecommunications.

With this scope in mind, the security requirements for a WLAN are:

Confidentiality. Data transferred over the WLAN should be kept confidential. As we remarked earlier, tapping a wired network takes a bit of effort, so a wireless network should also offer suitable protection.

Mutual entity authentication. Communicating entities can identify one another when setting up a WLAN connection. This is motivated by the fact that a degree of inherent (very weak) ‘entity authentication’ is provided by physical wires, but there are no such guarantees once we are in a wireless environment.

Data origin authentication. The source of all data transferred over the WLAN should be assured. This is because an attacker could easily modify data transmitted during a WLAN session after the initial entity authentication has been conducted. The original WLAN security standard WEP only provides a weak level of data integrity, which is not good enough.

12.2.3 WEP

There are three cryptographic design decisions that are common to all of the WLAN security mechanisms that we discuss:

- Since WLANs may be comprised of many different types of device, from different manufacturers, it is important that the cryptography used in a WLAN is widely available. Hence it would not be wise to deploy proprietary cryptographic algorithms.
- Since these mechanisms are dedicated to WLAN security and do not require the full flexibility of the likes of SSL, it makes sense to decide which cryptographic algorithms to use in advance and then deploy them universally, rather than require an expensive equivalent of the SSL Handshake Protocol to negotiate them.
- Since speed and efficiency are important, and WLANs are usually linked to some sort of fixed infrastructure, symmetric cryptography is a natural choice.

However, the cryptographic details of each of the proposed mechanisms vary considerably. We start by looking at the original proposal of WEP, which uses:

1. The stream cipher RC4 for encryption. A stream cipher is a reasonable choice, since a wireless communication channel is prone to errors (see Section 4.2.4).

The choice of RC4 was also reasonable at the time, since it was well respected and widely deployed in other applications, including SSL (although RC4 was originally a proprietary stream cipher, the details were publicly known by the time of the design of WEP). It would not be regarded as a good choice today, since the security of RC4 is no longer regarded as sufficient for modern applications.

2. A simple *CRC checksum* for data integrity. The problem with this type of checksum is that, in a similar way to using a hash function on its own to provide data integrity (see Section 6.2.2), an attacker can modify the data and then recompute the correct CRC checksum for the modified data. Hence this checksum can only be relied upon to detect accidental changes to the data.
3. A simple challenge–response protocol to provide entity authentication.

CONFIDENTIALITY AND INTEGRITY MECHANISMS IN WEP

In many ways, the cryptographic design of WEP was very naive. We now look at the mechanisms for providing confidentiality and data integrity in more detail, in order to illustrate some of the design errors.

The first WEP design decision was to use a *shared, fixed symmetric key* in each WLAN. This same key is used by all devices, for several different purposes, when communicating using a WEP-secured WLAN. This almost eliminates any issues regarding key establishment, however, it introduces considerable risks. In particular, if one of the devices is compromised then this key may become known to an attacker, and hence the entire network will be compromised. The original version of WEP only used a 40-bit key, but later adaptations allow much longer keys.

As discussed in Section 4.2.4, one problem with deploying a stream cipher such as RC4 is the need for synchronisation, especially in a potentially noisy channel such as a wireless one. Thus WEP requires each packet of data to be encrypted separately, so that loss of a packet does not affect the rest of the data being sent. This introduces a new problem. In Section 4.2.2 we discussed the negative consequences of re-using keystream for more than one plaintext. It follows that WEP requires a mechanism for making sure that the same keystream is not reused for subsequent packets.

The solution to this problem in WEP was to introduce an initialisation vector (IV), which just like the IVs used in several of the modes of operation of a block cipher (see Section 4.6), varies each time the WEP key is used to encrypt a packet. However, RC4 does not easily allow an IV to be incorporated into the encryption process, hence the WEP IV is directly appended to the key. In this way WEP defines a ‘per-packet’ key, which consists of a 24-bit IV appended to the WEP key.

If Alice wants to set up a secure WLAN connection with Bob, based on the shared, fixed WEP key K , the encryption process for each packet of data to be sent is depicted in Figure 12.3 and is as follows.

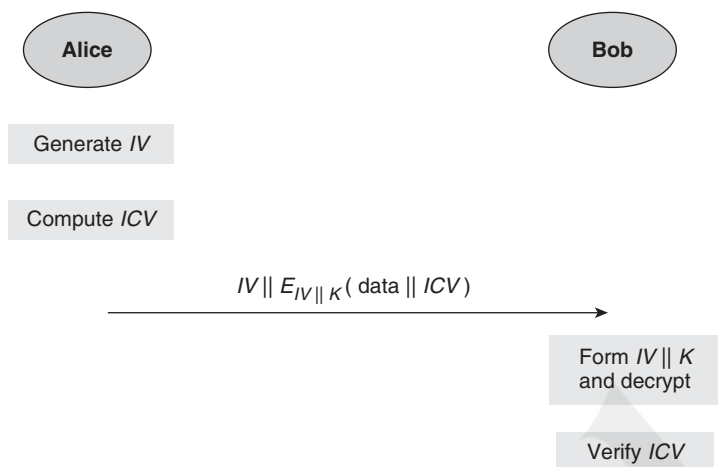


Figure 12.3. WEP encryption process

Alice:

1. generates a 24-bit pseudorandom initialisation vector IV and appends the WEP key K to IV to form the key:

$$K' = IV \parallel K;$$

2. computes a 32-bit CRC checksum ICV of the data, and appends this to the data;
3. encrypts the data plus ICV using key K' ;
4. sends IV (in the clear) and the resulting ciphertext to Bob.

Bob then:

1. appends the WEP key K to the received IV to form the key K' ;
2. decrypts the ciphertext using K' and extracts the checksum ICV ;
3. verifies that the checksum ICV is correct.

If the verification of ICV is successful then Bob accepts the data packet.

ENTITY AUTHENTICATION IN WEP

The WEP entity authentication technique is very simple. It is based on the same challenge-response principle that we first discussed for dynamic passwords in Section 8.5 and that was used in several of the protocols analysed in Section 9.4. If Alice (a device) wants to identify herself to Bob (a wireless access point):

1. Alice sends a request to authenticate to Bob;
2. Bob sends a nonce r_B to Alice;

3. Alice uses WEP encryption to encrypt r_B (importantly for later, note from our above explanation of the WEP encryption process that this also involves Alice generating an IV that is used to 'extend' the WEP key).
4. Alice sends the IV and the resulting ciphertext to Bob;
5. Bob decrypts the ciphertext and checks that it decrypts to r_B ; if it does, he authenticates Alice.

This simple protocol is based on Bob's assumption that only an authorised user such as Alice should know the WEP key K .

12.2.4 Attacks on WEP

WEP is perhaps the most criticised security standard ever proposed and there is very little right about it! We will briefly review some of the many concerns.

WEP KEY MANAGEMENT WEAKNESSES

We will start with key management. There are several serious problems with WEP key management:

Use of a shared fixed key. The WEP key K acts as an overall 'master key' for the WLAN and, as such, is a single point of failure. If the WEP key can be compromised (and it suffices that this compromise arises on just one of the entities forming the WLAN) and an attacker learns the WEP key then the entire WLAN security is compromised.

Exposure of the WEP key. In its role as a master key, the WEP key is unnecessarily 'exposed' through direct use as a component of an encryption key (see Section 10.4.1). It is also exposed in this way each time an authentication attempt is made.

No key separation. WEP abuses the principle of key separation (see Section 10.6.1) by using the WEP key for multiple purposes.

Key length. While WEP does allow the WEP key length to vary, the smallest RC4 key length is 40 bits, which is far too short to be secure against contemporary exhaustive key searches. Perhaps more problematically, many WEP implementations allow WEP keys to be generated from passwords which, if not long enough, reduce the effective keyspace that an attacker needs to search.

WEP ENTITY AUTHENTICATION WEAKNESSES

We now look at attacks concerning the entity authentication mechanism.

Rogue wireless access point. WEP only provides *unilateral* entity authentication from a device (Alice) to a wireless access point (Bob). This means that an attacker could set up a rogue access point and allow Alice to authenticate to it, without Alice realising that she was not dealing with the genuine access point.

Lack of session key. WEP does not establish a session key during entity authentication that is later used to protect the communication session. As a

result, WEP entity authentication is only valid for the ‘instant in time’ at which it is conducted. WEP thus suffers from the potential for a ‘hijack’ of the communication session, as discussed in Section 8.3.1.

Keystream replay attack. Another serious problem is that there is no protection against replays of the WEP authentication process. An attacker who observes Alice authenticating to Bob is able to capture a plaintext (the challenge r_B and its CRC checksum) and the resulting ciphertext (the encrypted response). Since WEP uses the stream cipher RC4, the keystream can be recovered by XORing the plaintext to the ciphertext (see Section 4.2.1). We will denote this keystream by $KS(IV \parallel K)$, since it is the keystream produced by RC4 using the encryption key $IV \parallel K$. Note that this is not yet an ‘attack’, because our standard assumptions of Section 1.5.1 dictate that good stream ciphers are designed to offer protection against an attacker who knows corresponding plaintext/ciphertext pairs, and hence can recover keystream from this knowledge. However, this relies on the same keystream not being reused in a predictable manner (see Section 4.2.2). This is where WEP fails, since the attacker can now falsely authenticate to Bob as follows (and depicted in Figure 12.4):

1. The attacker requests to authenticate to Bob;
2. Bob sends a nonce r'_B to the attacker (assuming that Bob is properly generating his nonces, it is very unlikely that $r'_B = r_B$);
3. The attacker computes the CRC checksum ICV on r'_B . The attacker then encrypts $r'_B \parallel ICV$ by XORing it with the keystream $KS(IV \parallel K)$; note:
 - the attacker does not know the WEP key K , but does know this portion of keystream;
 - in line with WEP encryption, the attacker also first sends the IV that was observed during Alice’s authentication session to Bob;
4. Bob decrypts the ciphertext, which should result in recovery of r'_B , in which case Bob accepts the attacker.

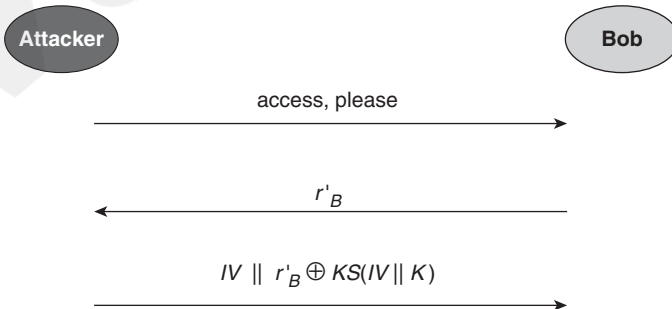


Figure 12.4. Keystream replay attack on WEP authentication

This attack works because WEP allows the attacker to ‘force’ Bob to use the same IV that Alice used in the genuine authentication session, and hence use the same encryption key $IV \parallel K$, which in turn validates the use of the previous keystream. Of course, having authenticated to the access point, the attacker cannot do much more since the attacker still does not know the WEP key K and hence cannot perform valid encryptions and decryptions. Nonetheless, the authentication process has been successfully attacked. Indeed, more generally, the use of a stream cipher in a challenge–response protocol is ill-advised (see the related activity at the end of Chapter 8).

WEP CONFIDENTIALITY AND INTEGRITY WEAKNESSES

We now identify some attacks on the confidentiality and integrity mechanisms in WEP. These include attacks that can reveal the WEP key to an attacker.

CRC manipulation attack. This attack exploits the fact that the CRC checksum used to provide data integrity is not a cryptographic primitive, but is a highly ‘linear’ function. This means that certain changes to the checksum output (ICV) can be used to deduce changes to the underlying data packet. Because of this, the encrypted ICV can be manipulated and then the receiver’s behaviour (either to accept or reject a data packet) monitored in order to deduce information about an underlying data packet. In this way it might be possible to recover an unknown data packet without knowing the WEP key.

Birthday attack on IVs. The length of the IV in WEP is only 24 bits. This means that there are 2^{24} possible different IV s. From our discussions of birthday attacks in Section 6.2.3, we know that we can expect that if around 2^{12} (which is only about 4000) data packets are sent, then two of these data packets will probably have the same IV . Even if the IV is an ascending counter then, for example, if the access point transmits data at a rate of 11 megabits per second then all the IV s are exhausted after about five hours, after which the IV will necessarily repeat. If two data packets have the same IV then they will have been encrypted using the same RC4 key. As the WEP key is fixed, an attacker can expect, over time, to find large numbers of data packets that have been encrypted using the same encryption key. This is something that we observed in Section 4.2.2 is not desirable for stream ciphers. Note that this attack is independent of the length of the WEP key. In practice this is quite a hard attack to conduct but, unfortunately for WEP, there are even more serious attacks against the encryption mechanism.

Key recovery attack. Most seriously of all, the way that RC4 keys are formed from IV s can be exploited in order to define a clever statistical attack, which by 2010 had been refined to the point where it was possible to recover WEP keys with high probability after less than ten thousand data packets had been observed. Since the WEP key is fixed, this is a fatal flaw. While this attack is not obvious, it is easy to deploy using widely available tools.

WEP DESIGN FLAWS

The designers of WEP set out with an admiral goal to provide *enough* security, but not *excessive* security, in order to keep WLANs secure but efficient. In doing so they were attempting to establish a particular efficiency–security tradeoff. It is clear that they did not succeed and that they sacrificed too much security in the name of efficiency.

However, some of the problems with the WEP cryptographic design appear to arise not from knowledgeable tradeoffs, but from fundamental misunderstandings. These include:

Poor key management. The decision to use one shared, fixed, key in a WEP-protected WLAN to provide all the WEP security services is extremely risky. We have just seen that WEP encryption keys can be recovered given enough data packets. Such an attack would be regarded as very serious for any cryptosystem, even if it changed its keys regularly. For WEP, with its fixed key, this attack is disastrous.

Failure to appreciate the effective key length. The way that the RC4 encryption keys are formed in WEP provides a false sense of security with respect to key length. Even if the WEP key is a respectable length, the RC4 encryption key only varies in 24 bits each time that an encryption operation is performed. This leads to problems, such as our *birthday attack on IVs*.

Lack of a proper cryptographic data origin authentication mechanism. The poor choice of data integrity mechanism leads to attacks that can exploit the fact that CRC checksums can be manipulated. Data origin authentication should have been provided using a suitable cryptographic primitive. Given that WEP relies on symmetric encryption, the ‘correct’ tool for this job is a MAC.

Non-standard use of a cryptographic algorithm. WEP provides an excellent example of why it is important to use cryptographic algorithms in the intended way, and not to ‘tinker’ with them. The *key recovery attack* that we mentioned is not an attack on RC4. It is an attack on the fact that RC4 was not used in a standard way in WEP. Instead, the RC4 encryption keys were formed using a technique that was invented for WEP, but not one that had been sufficiently analysed by cryptographic experts. Since it was this technique that was exploited in the powerful key recovery attack on WEP, this shows that even a small change in the way that a cryptographic algorithm is used can result in an insecure cryptographic mechanism.

Weak entity authentication mechanism. The WEP entity authentication mechanism can be exploited in several different ways, as we discussed. Use of a stream cipher in this mechanism is highly inappropriate.

Thus WEP has provided us with a wide range of valuable cryptographic design lessons, many of which have wider implications.

12.2.5 WPA and WPA2

We now look at how WPA and WPA2 overcome the many problems with WEP.

MUTUAL ENTITY AUTHENTICATION AND KEY ESTABLISHMENT

In order to avoid all the problems relating to use of a shared, fixed WEP key, a key hierarchy (see Section 10.4.1) is employed. The top key in this key hierarchy is known as the *pairwise master key PMK*, which is a key that is shared between a device and a wireless access point. There are two ways in which this key *PMK* can be established:

1. *During an AKE protocol that is run between a device and a central authentication server.* Both WPA and WPA2 support the use of a central authentication server to provide authentication in a way that is scalable and can be tailored to fit the needs of the specific application environment. A wide range of authentication techniques are supported by the *Extensible Authentication Protocol (EAP)*, which is a suite of entity authentication mechanisms that includes methods that deploy SSL (see Section 12.1) to secure a connection to an authentication server.
2. *As a pre-shared key that is programmed directly into the device and the wireless access point.* This is most suitable for small networks. The most common method for generating *PMK* is by deriving it from a password. Any users requiring access to the WLAN must be made aware of this password. A home user who purchases a wireless router may be provided with a (weak) default password from the manufacturer or service provider. It is important that this is changed on first installation to something less predictable.

Note that even if a device has successfully authenticated itself to a central authentication server and this server has passed *PMK* to the wireless access point, it is still necessary for the device to authenticate itself to the access point. Regardless of how *PMK* has been established, it forms the basis for this entity authentication process between the device and the access point. The master key *PMK* is also used to derive session keys using the following AKE protocol that runs between Alice (a device) and Bob (a wireless access point) and is shown in Figure 12.5:

1. Alice generates a nonce r_A and sends r_A to Bob.
2. Bob generates a nonce r_B . Bob then uses r_A , r_B and *PMK* to derive the following four 128-bit session keys:
 - an encryption key *EK*;
 - a MAC key *MK*;
 - a data encryption key *DEK*;
 - a data MAC key *DMK*.

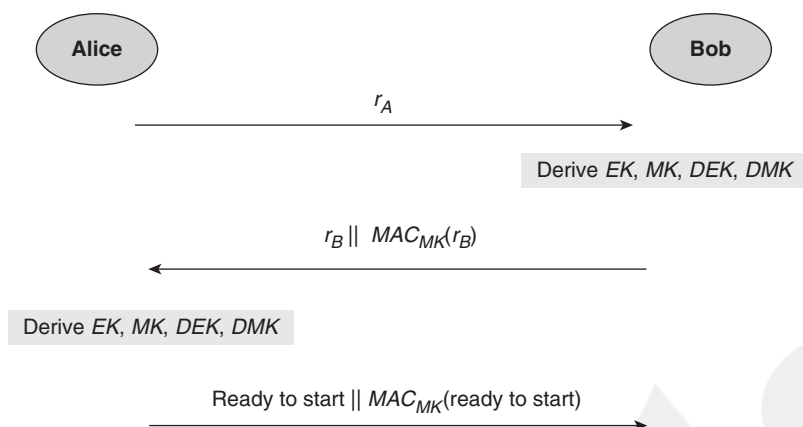


Figure 12.5. WPA authentication and key establishment protocol

- Bob then sends r_B to Alice, along with a MAC computed on r_B using MAC key MK .
- Alice uses r_A , r_B and PMK to derive the four session keys. She then checks the MAC that she has just received from Bob.
- Alice sends a message to Bob stating that she is ready to start using encryption. She computes a MAC on this message using MAC key MK .
- Bob verifies the MAC and sends an acknowledgement to Alice.

At the end of this protocol both Alice and Bob have achieved mutual entity authentication, since each has demonstrated knowledge of PMK by successfully computing MACs using MK , which is derived from PMK . (Strictly speaking, Bob has only achieved assurance that Alice is *one of* the authorised users of the WLAN, since there may be more than one user sharing PMK with Bob.) In addition, Alice and Bob have agreed on four session keys. Two of these, DEK and DMK , will be used to secure the data exchanged between Alice and Bob in the coming session (the fourth key EK plays a role in group key management, which we will not discuss).

CONFIDENTIALITY AND DATA ORIGIN AUTHENTICATION IN WPA

WPA and WPA2 differ in the way that they provide protection for the data exchanged during a communication session between a device and a wireless access point. Recall that WPA was designed as a temporary 'fix' of WEP, while WPA2 is a complete redesign.

While WPA still uses RC4, it features several design improvements:

- The RC4 encryption key is created by *mixing* DEK and an IV, rather than concatenating, as in WEP. Further, a separate encryption key is derived by such

a mixing for each packet sent. These simple changes are enough to prevent several of the attacks on WEP.

- Data origin authentication is provided using a MAC, rather than the easily manipulated CRC checksum used in WEP. The recommended MAC is a special lightweight mechanism tailored for WPA known as *Michael*.

While these are both definite improvements over WEP, they are both still unconventional cryptographic primitives, albeit ones that have been carefully designed by cryptographic experts.

CONFIDENTIALITY AND DATA ORIGIN AUTHENTICATION IN WPA2

WPA2 represents a complete redesign and uses standard cryptographic mechanisms. In particular, WPA2 adopts AES instead of RC4 as the underlying encryption algorithm.

WPA2 provides confidentiality and data origin authentication together in one mechanism by deploying AES in a protocol referred to as the *Counter Mode with CBC-MAC Protocol* (CCMP). CCMP is based on the CCM mode of operation of a block cipher that we discussed in Section 6.3.6. As indicated in Section 6.3.6, this avoids the need to provide these services using separate mechanisms. CCMP includes mechanisms for deriving fresh keys for each separate CCM ‘encryption’. Note that if CCMP is used then it is only necessary to derive one data key during the WPA2 AKE protocol, instead of separate keys *DEK* and *DMK*.

12.2.6 WLAN security issues

Perhaps the most interesting aspect of WLAN security is that some of the problems have arisen from errors in cryptographic mechanism design, which is relatively unusual. As we have repeatedly observed, it is far more common for vulnerabilities to arise elsewhere, such as during implementation and key management (see Section 3.2.4). However, it would seem that WPA2 addresses all of the previous problems and provides good cryptographic protection. To date there have been no serious attacks on the cryptography used in WPA or WPA2.

The most vulnerable aspect of WPA2 security remains the potential for the *PMK* derivation in small (home) networks to rely on a weak password or passphrase. This is a very important issue because all the subsequent session keys are derived from this pre-shared key, and the mutual entity authentication process relies on *PMK* only being known by authorised devices and the wireless access point. If this type of key derivation is being used then all the potential problems with passwords and passphrases, such as those discussed in Section 8.4.1, apply to WPA2 security. There is also the potential risk that home users use default keys that are supplied with their equipment, rather than establish their own.

12.2.7 WLAN design issues

The main cryptographic design issues concerning WLAN security are as follows:

Use of symmetric cryptography. This is a sensible decision because WLANs transfer bulk traffic between networked devices, hence speed of encryption is important. For small networks, such as a home network, key establishment is straightforward. Larger enterprise WLANs may optionally choose to use public-key mechanisms as part of the initial authentication between a device and a central authentication server, but the core WPA2 security protocol CCMP uses only symmetric cryptography.

Use of recognised cryptographic mechanisms. This was not adhered to in WEP, where the cryptographic design was rather ad hoc. WEP thus provides a useful lesson regarding the potential folly of adopting unconventional mechanisms. In contrast, WPA2 adopts more widely accepted cryptographic mechanisms.

Flexibility, but only when appropriate. While WLANs may be deployed in quite different environments, they do not require the same cryptographic flexibility as open applications such as SSL. Thus it makes sense to ‘lock down’ the cryptographic mechanisms, where appropriate. WPA2 does this for the confidentiality and data origin authentication services. However, WPA2 allows for flexibility in choosing the initial entity authentication mechanism (between the device and a centralised authentication server), recognising that different environments may well have different approaches to identifying network users.

The potential need to cater for migration. When the flaws in WEP became apparent, it was clear that due to the difficulty of upgrading a widely deployed technology, any complete redesign of the WLAN security mechanisms could not be rolled out quickly. It was thus necessary to design a ‘fix’ that was based on the existing cryptographic mechanisms, which would provide ‘good enough’ security. The ‘fix’ is WPA, which is based on RC4. The ‘complete redesign’ is WPA2, which is based on AES.

12.3 Cryptography for mobile telecommunications

We now look at a very different application of cryptography, but one that most of us use almost every day. The aspect that makes it so different from the previous two applications that we have studied is the nature of the operating environment. Mobile telecommunication services are provided by companies who have agreed on certain operational standards in order to make their services compatible. Collectively they thus represent a sort of ‘closed’ environment, albeit one distributed across a substantial number of different organisations. We will see that this has influenced some of the cryptographic design decisions that have been taken.

12.3.1 GSM and UMTS background

In Section 12.2.1 we observed that there is an inherent level of physical security provided by a wired computer network. The same can be said for a wired telecommunications network. Thus, in a very similar way to the development of WLANs, the advent of mobile telecommunications brought with it a range of new threats that did not exist for traditional wired telecommunications networks.

These problems were not recognised by the designers of the first mobile phone systems. These used analogue signals and did not have suitable protection. Mobile handsets sent their serial numbers in the clear, leaving them highly susceptible to cloning. A cloned phone could then be used by an attacker at the expense of the genuine user. Eavesdropping on calls was also straightforward.

This situation was clearly unacceptable for everyone involved. It raised privacy concerns for mobile phone customers, as well as the considerable inconvenience of dealing with the aftermath of a phone cloning incident. More significantly, mobile telecommunications operators faced loss of revenue and reputation through incidents of fraud.

The shift from analogue to digital communications brought with it the opportunity to use cryptographic techniques to provide security. In doing so, the development of the *Global System for Mobile Communication* (GSM) standard by the *European Telecommunications Standards Institute* (ETSI) brought security to mobile telecommunications. We will look in some detail at the cryptographic aspects of GSM security. Third generation, or 3G, mobile phones are characterised by higher data transmission rates and a much richer range of services. We will briefly discuss the enhanced security of GSM's successor for 3G phones, the *Universal Mobile Telecommunications System* (UMTS).

The basic architecture of a mobile telecommunications network is shown in Figure 12.6. The network is divided into a large number of geographic *cells*, each of which is controlled by a *base station*. A mobile phone first connects with its nearest base station, which directs communications either to the home network of the mobile phone user or to other networks in order to transfer call data.

12.3.2 GSM security requirements

One of the main drivers behind GSM's security mechanisms was revenue protection. Mobile telecommunications is big business and mobile operators pay substantial sums for the frequency ranges that they use. It is very important to mobile operators that they charge the right customers for the services that these customers have genuinely used. However, because mobile telecommunications is a business, the security provided by GSM must be cost-effective and limited to that which is strictly necessary.

The overarching design guideline for GSM was that the resulting system should be *as secure as the Public Switched Telephone Network* (PSTN). This is very

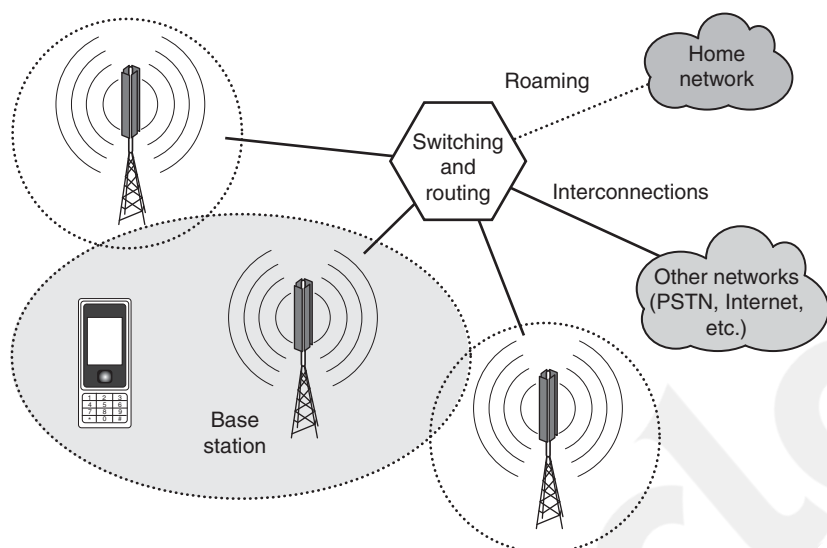


Figure 12.6. Basic architecture of mobile telecommunications network

much in the same spirit as the guidelines for developing WLAN security that we discussed in Section 12.2.2. It is widely rumoured that GSM was not designed to go further and provide *end-to-end security* (meaning security throughout the entire path from source to destination) because governments were keen to maintain the degree of intercept access that they have to the PSTN. This led to the following specific security requirements:

Entity authentication of the user. Mobile operators need to have strong assurance of the identity of users connecting with their services in order to reduce fraud. This issue is much simpler to deal with in traditional telephone networks, since a user needs to have physical access to the end of a telephone wire in order to use the services.

Confidentiality on the radio path. In simple terms, a mobile connection passes ‘over the air’ (the *radio path*) between the handset and a base station, after which it is passed through a switching centre and enters the traditional PSTN (see Figure 12.6). Thus in order to provide ‘PSTN-equivalent security’, the main link for which GSM needs to provide additional security is the radio path. Since this path is easily intercepted by anyone with a suitable receiver it is necessary to provide confidentiality on this radio path.

Anonymity on the radio path. GSM provides a degree of anonymity (confidentiality of the identity of users) on the radio path in order to prevent an attacker from linking the source of several intercepted calls. This is handled by using

temporary user identities for each call, rather than permanent ones. We will not discuss this requirement further.

Notably, GSM did not specify the need for entity authentication of the mobile operator to the mobile phone, since at the time of development of GSM this was not regarded as a serious threat. This was because it was perceived to be extremely expensive for an attacker to masquerade as a mobile operator.

The meeting of these security requirements was subject to certain constraints:

- The security mechanisms should not be excessively strong, in order to avoid export control issues (which were more relevant in the period of GSM development, the 1990s, than they are today).
- The security mechanisms should not add significant overheads to the operation of the system, including call setup.

12.3.3 Cryptography used in GSM

The main cryptographic design decisions for GSM were:

A fully symmetric cryptographic architecture. While it is obvious that the need for fast real-time encryption of the radio link requires the use of symmetric cryptography, it might still be beneficial to deploy public-key cryptography to enable key establishment. However, GSM is an entirely closed system. All key material can be loaded onto the necessary equipment prior to it being issued to users, so there is no need to use public-key cryptography for this purpose.

Stream ciphers for data encryption. The requirement for fast real-time encryption over a potentially noisy communication channel means that, as we discussed in Section 4.2.4, a stream cipher is the most appropriate primitive.

Fixing the encryption algorithms. It is necessary that the mobile operators agree on which encryption algorithms to use, so that the devices on which they operate can be made compatible with one another. However, other cryptographic algorithms, such as those used in GSM authentication, do not have to be fixed. In the case of authentication, an individual mobile operator is free to choose the cryptographic algorithm that it deploys to authenticate its own users (since users of another mobile operator are not directly impacted by this decision).

Proprietary cryptographic algorithms. The designers of GSM chose to develop some proprietary cryptographic algorithms, rather than use open standards. We have discussed the pros and cons of this choice in Section 1.5.3. While the use of proprietary algorithms is not wise in many application environments, in the case of GSM there were three factors that favoured at least considering this option:

- GSM is a closed system, hence deploying proprietary algorithms is feasible.
- ETSI have a degree of cryptographic expertise, and maintain links with the open research community.

- The need for fast real-time encryption means that an algorithm designed explicitly to run on the hardware of a mobile phone will probably perform better than an 'off-the-shelf' algorithm.

The fundamental component involved in GSM security is the *Subscriber Identification Module (SIM) card*, which is a smart card (see Section 8.3.3) that is inserted into the mobile phone of the user. This SIM card contains all the information that distinguishes one user account from another. As a result, a user can potentially change phone equipment simply by removing the SIM and inserting it into a new phone. The SIM contains two particularly important pieces of information:

1. the *International Mobile Subscriber Identity (IMSI)*, which is a unique number that maps a user to a particular phone number;
2. a unique 128-bit cryptographic key K_i , which is randomly generated by the mobile operator.

These two pieces of data are inserted onto the SIM card by the mobile operator before the SIM card is issued to the user. The key K_i forms the basis for all the cryptographic services relating to the user. The SIM card also contains implementations of some of the cryptographic algorithms required to deliver these services.

GSM AUTHENTICATION

Entity authentication of the user in GSM is provided using a challenge–response protocol, in a similar way to the dynamic password schemes that we discussed in Section 8.5. This is implemented as part of an AKE protocol, which also generates a key K_c for subsequent data encryption. GSM does not dictate which cryptographic algorithms should be used as part of this AKE protocol, but it does suggest one candidate algorithm and defines the way in which algorithms should be used.

As indicated in Figure 12.7, an algorithm A3 is used in the challenge–response protocol and an algorithm A8 is used to generate the encryption key K_c . Both of these algorithms can be individually selected by the mobile operator and are implemented on the SIM and in the operator's network. Both A3 and A8 can be loosely considered as types of key derivation function, since their main purpose is to use K_i to generate pseudorandom values.

In the following we use the notation $A_{3K}(data)$ to denote the result of computing algorithm A3 on the input *data* using key K (the notation $A_{8K}(data)$ should be similarly interpreted). If Alice (a mobile) is able to directly authenticate to Bob (the authentication centre of a mobile operator) then the GSM AKE protocol is as follows:

1. Alice sends an authentication request to Bob.
2. Bob generates a 128-bit randomly generated challenge number *RAND* and sends it to Alice.

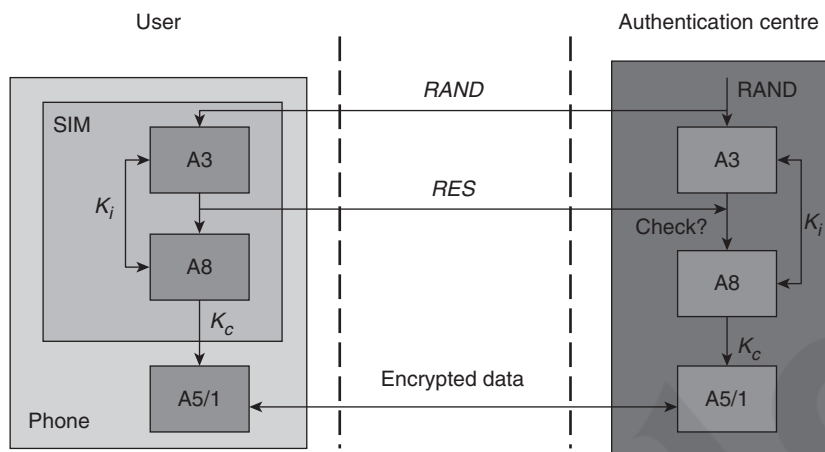


Figure 12.7. GSM authentication and encryption

- Alice's SIM card uses K_i and $RAND$ to compute a response RES using algorithm A3:

$$RES = A3_{K_i}(RAND).$$

The response RES is sent back to Bob.

- Bob, who maintains a database of all the user keys, selects the appropriate key K_i for Alice and then computes the expected response in the same way. If the result matches the received RES then Alice is authenticated.
- Alice and Bob both use K_i and $RAND$ to compute an encryption key K_c using algorithm A8:

$$K_c = A8_{K_i}(RAND).$$

This simple protocol relies on the belief that only the mobile user and the mobile operator authentication centre can possibly know the key K_i that has been installed on the user's SIM card.

GSM ENCRYPTION

While authentication is a service that is 'private' to a mobile user and their mobile operator, encryption must be provided using a mechanism that is common to all mobile operators, in order to facilitate cross-network calls. Thus the encryption algorithm A5/1 is fixed by the GSM standard (in fact GSM offers three different versions of A5, but A5/1 is the most commonly deployed). As indicated in Figure 12.7, it is implemented on the mobile phone itself, not the SIM card, since the phone has more computation power than the SIM.

The A5/1 algorithm is a stream cipher with a 64-bit key. It was designed to be implemented very efficiently in the hardware of a mobile phone. In GSM, A5/1 is

used to encrypt all radio path communication (both signalling information and the message data) using the key K_c . Potentially, this key may be freshly generated each time a user makes a mobile call.

Encryption is also used to protect the transfer of temporary identification numbers, which are used instead of the IMSI to provide user anonymity.

FACILITATING GSM ROAMING

While we previously argued that authentication is a ‘private’ service between a user and their mobile operator, there is one situation where this is not strictly true. This happens when a mobile user is traveling outside the area serviced by their mobile operator, for example, overseas (this is referred to as *roaming*).

Although different mobile operators are in some sense part of a wider ‘closed’ GSM network, they are still individual businesses with their own private user relationships. It would thus be unacceptable for one operator to share its security critical data (particularly key K_i) with another for the purpose of facilitating roaming. On the other hand, it is equally unacceptable from a practical perspective for every authentication request from a roaming user to be referred back to the user’s mobile operator, since this might result in extensive delays.

GSM has a clever solution to this problem, through the use of *authentication triplets*. When a roaming mobile user Alice first connects with Charlie, a local mobile operator with whom she has no direct business relationship, the following procedure is followed:

1. Charlie contacts Bob (Alice’s mobile operator) and requests a batch of GSM authentication triplets.
2. Bob generates a fresh batch of randomly generated challenge numbers $RAND(1), RAND(2), \dots, RAND(n)$ and computes the matching values for RES and K_c using Alice’s key K_i . These form the batch of triplets:

$$TRIP(1) = (RAND(1), RES(1), K_c(1))$$

$$TRIP(2) = (RAND(2), RES(2), K_c(2))$$

$$\vdots$$

$$TRIP(n) = (RAND(n), RES(n), K_c(n)),$$

where $RES(j) = A3_{K_i}(RAND(j))$ and $K_c(j) = A8_{K_i}(RAND(j))$. Bob sends this batch of triplets to Charlie.

3. Charlie sends the challenge $RAND(1)$ to Alice.
4. Alice computes the response $RES(1)$ using $RAND(1)$ and key K_i and sends $RES(1)$ to Charlie.
5. Charlie checks that the received $RES(1)$ matches the value in the first triplet that he received from Bob. If it does then Charlie authenticates Alice. Note that Charlie has done this without needing to know the key K_i . Alice and Charlie can now safely assume that they share the encryption key $K_c(1)$.

6. The next time Alice contacts Charlie to request a new authentication, Charlie uses the second triplet received from Bob and sends the challenge $RAND(2)$. Thus although Bob has to be involved in the first authentication attempt, there is no need to contact Bob again until the current batch of triplets have all been used up.

SECURITY OF GSM ALGORITHMS

Despite the justification that we made for the use of proprietary algorithms in GSM, the general concerns about the use of proprietary algorithms that we outlined in Section 1.5.3 have come to the fore since the initial development of GSM.

A popularly implemented early instantiation of the algorithms A3 and A8 was a proprietary algorithm called COMP128. The details of this algorithm were leaked in 1997 and weaknesses in COMP128 were subsequently found. New versions of this algorithm have since been introduced.

The initial design of A5/1 was also secret but the algorithm was subsequently reverse-engineered and some powerful attacks against it have now been demonstrated. We will see shortly that a different approach has been taken over UMTS algorithm selection.

Nonetheless, GSM has proved to be a successful security standard. GSM effectively solved the problem of cloning mobiles to gain unauthorised access to mobile telecommunications networks. GSM addressed the problem of eavesdropping on the radio path. It is interesting to note that GSM was also one of the first applications to demonstrate the advantages of basing security of consumer devices on smart cards.

12.3.4 UMTS

The main reason for developing a new standard for mobile telecommunications was not so much GSM security concerns, but rather to provide additional features and functionality, such as the ability to access internet services. However, the opportunity was taken during the development of UMTS to build on the successful aspects of GSM security, and further strengthen it where appropriate. The main cryptographic improvements over GSM are as follows:

Mutual entity authentication. GSM offers entity authentication only of the mobile user. Since the development of GSM, so-called *false base station attacks* have become much more feasible due to reductions in the costs of suitable equipment. In one example of such an attack, a mobile user connects to the false base station, which immediately suggests to the user that encryption is turned off. By additionally requiring the user to authenticate to the mobile base station, such attacks are prevented.

Prevention of triplet reuse. A GSM triplet can, in theory, be reused many times for the particular mobile that it was generated for. In UMTS this is prevented

by upgrading authentication triplets to *quintets*, which additionally include a sequence number that prevents successful replay and a MAC key.

Use of publicly known algorithms. UMTS adopts cryptographic algorithms based on well-established and well-studied techniques. While it does not quite use ‘off-the-shelf’ algorithms, due to the desire to tailor algorithms to the underlying hardware, the algorithms deployed are very closely based on standard algorithms and the modifications have been publicly evaluated.

Longer key lengths. Following the relaxation of export restrictions that were in place at the time of GSM development, the key lengths of the underlying cryptographic algorithms were increased to 128 bits.

Integrity of signalling data. UMTS provides additional integrity protection to the critical signalling data. This is provided using a MAC, whose key is established during the UMTS authentication (AKE) protocol.

UMTS SECURITY PROTOCOLS

We will omit the details of the UMTS security protocols since they are, in essence, just slightly more complex versions of the original GSM protocols. Entity authentication of the mobile user is conducted via a similar challenge–response mechanism to GSM, at the end of which encryption and MAC keys are established.

Entity authentication of the base station is added to UMTS through the use of a MAC. The freshness mechanism used as part of this authentication is a sequence number (see Section 8.2.2), which is maintained by the mobile user and the base station. This is preferable to also using a challenge–response protocol in the opposite direction since, as we discussed in Section 8.2.3, this would introduce one extra message exchange, as well as require the mobile user to randomly generate a challenge number. It would also be very inconvenient when roaming, since the local mobile operator would have to contact a user’s home mobile operator during each authentication attempt.

Roaming works on exactly the same principle as for GSM, except that the additional fields of the authentication quintet provide protection against replays.

UMTS CRYPTOGRAPHIC ALGORITHMS

Just as for GSM, mobile operators are free to use their own cryptographic algorithms as part of the UMTS AKE protocol. However, UMTS recommends the use of a set of algorithms called MILENAGE, which is based on AES and implements all the functionality required for UMTS authentication.

Once again the encryption algorithm must be fixed across all mobile operators. The selected UMTS algorithm is KASUMI, which is a 128-bit block cipher based on a well-studied design known as MISTY. Since what we really want is a stream cipher, KASUMI is deployed in a mode of operation that deploys a block cipher as a stream cipher keystream generator (similar to some of those discussed in Section 4.6). UMTS also specifies ‘backup’ stream ciphers in the (unexpected) event that a serious vulnerability is found in KASUMI.

There are currently no known practical attacks against these cryptographic algorithms.

12.3.5 GSM and UMTS key management

Key management in GSM and UMTS is fairly straightforward.

KEY MANAGEMENT SYSTEM

GSM and UMTS have an entirely symmetric key management system, facilitated by the fact that a mobile operator is completely in control of all keying material relating to their users. We can think of the underlying key management system as a very simple key hierarchy (see Section 10.4.1) with the user keys K_i acting as individual user ‘master keys’ and the encryption keys K_c acting as data (session) keys.

KEY GENERATION

The user keys K_i are randomly generated, normally by the SIM manufacturer (on behalf of the mobile operator) using a technique of their choice. The encryption keys K_c are derived from the user keys K_i , using the mobile operator’s chosen cryptographic algorithm.

KEY ESTABLISHMENT

The establishment of user key K_i is under the control of the SIM manufacturer (on behalf of the mobile operator) who installs K_i on the user’s SIM card before it is issued to the user. The significant key management advantage that is being exploited here is that a mobile service has no utility until a customer obtains a physical object from the mobile operator (in this case a SIM card), hence key establishment can be tied to this process. The keys K_c are established during the AKE protocol used for entity authentication. It is clearly very important that the SIM manufacturer transfers all the keys K_i to the mobile operator using highly secure means, perhaps in the form of an encrypted database.

KEY STORAGE

The critical user keys K_i are stored in the hardware of the user’s SIM card, which offers a reasonable degree of tamper-resistance. Only the encryption key K_c , and in UMTS a MAC key derived from K_i , leave the SIM card. These are session keys that are discarded after use.

KEY USAGE

Both GSM and UMTS enforce a degree of key separation by making sure that the long-term user key K_i is only ever indirectly ‘exposed’ to an attacker through its use to compute the short responses to the mobile operator’s challenges. The key K_c that is used for bulk data encryption, and is thus most ‘exposed’ to an attacker, is a derived key that is not used more than once. In UMTS, separate keys

for encryption and MACs are derived from K_i . The use of a SIM also makes key change relatively straightforward.

12.3.6 GSM and UMTS security issues

GSM broke new ground for the mass use of cryptography. It provided, and to an extent still provides, good security for a rapidly expanding mobile phone network. GSM was, by and large, well designed and the basic security architecture of GSM is preserved in UMTS, which tightens up on the security offered by GSM.

It is worth remembering, however, that GSM and UMTS are deliberately not designed to provide end-to-end security. The design goal of being ‘as secure as the PSTN’ means that, just like a conventional telephone call, a mobile telephone call may still be intercepted after it has been switched into the conventional PSTN infrastructure.

12.3.7 GSM and UMTS design issues

The main design issues emerging from our study of GSM and UMTS are the following:

Use of symmetric cryptography. The closed nature of the application environment lends itself to adoption of a fully symmetric solution. The properties of stream ciphers are highly suited to mobile telecommunications.

Adaptation to evolving constraints. GSM was designed under several constraints, including cryptographic export restrictions and the apparent lack of a need for mobile operator authentication. As the environment determining these constraints evolved, the redesigned security mechanisms of UMTS took these into account.

Shift from proprietary to publicly known algorithms. Mobile telecommunications provide a plausible environment for the adoption of proprietary cryptographic algorithms. However, subsequent weaknesses in some of the original GSM algorithms may well have influenced the use of publicly known algorithms in UMTS.

Flexibility, but only when appropriate. Just as we saw for WLAN security in Section 12.2, GSM and UMTS only prescribe particular cryptographic algorithms when this is essential, leaving a degree of flexibility to mobile operators. That said, in UMTS mobile operators are strongly encouraged to follow central recommendations.

12.4 Cryptography for secure payment card transactions

Financial sector organisations are the most established commercial users of cryptography. They oversee global networks that use cryptographic services to provide security for financial transactions. We will demonstrate some of the ways

in which cryptography is used in the financial sector by considering some of the services offered by international payment card organisations.

12.4.1 Background to payment card services

A *payment card organisation* (PCO), such as Visa and MasterCard, essentially operates as a ‘club’ of member banks who cooperate in order to facilitate transactions. Figure 12.8 indicates the key players in this cooperative organisation.

Issuing banks issue payment cards to customers. *Acquiring banks* have relationships with merchants of goods. PCOs run networks that connects these banks and facilitate payments from issuing bank customers to acquiring bank merchants. The two main uses of a payment card network are to:

1. authorise payments;
2. arrange clearing and settlement of payments.

PCOs oversee the use of both credit and debit cards. The main difference between the two is the process by which the issuing bank decides to bill the customer. From a cryptographic perspective we will not distinguish between these two types of payment card.

In this section we will look at a number of different uses of cryptography with respect to payment cards. We begin by looking at the cryptographic security relating to magnetic stripe payment cards. We then consider the upgraded security

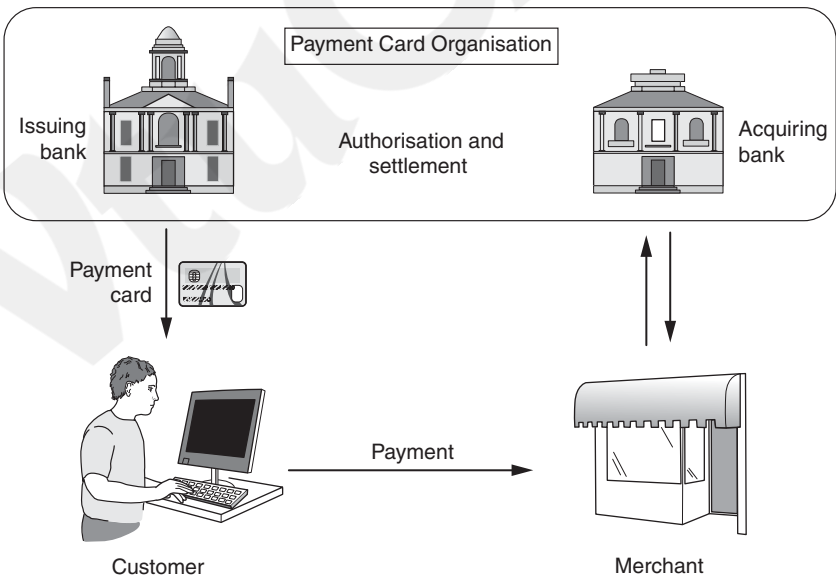


Figure 12.8. Payment card organisation infrastructure

offered by so-called *chip-and-PIN* cards, more correctly referred to as *Europay MasterCard Visa* (EMV) cards, named after the three major global payment card schemes operating in Europe who worked together to establish a common interoperable standard for payment card security. We then discuss the use of payment cards in online transactions. Finally we consider the use of payment cards as tokens for authenticating to other services.

12.4.2 Magnetic stripe cards

Most payment cards have magnetic stripes. Even payment cards with chips often retain the magnetic stripe and may resort to using it when they are deployed in environments that do not support EMV. The following description of cryptography used by magnetic stripe cards is based on the practices of Visa and MasterCard.

PIN PROTECTION

Our first example of cryptography being used by payment cards concerns online authentication of a user who inserts their magnetic stripe payment card into an ATM. Before releasing any funds, the ATM needs to know whether the user is genuine and whether they are entitled to make the requested withdrawal.

The process begins when the user is asked to enter their PIN into the ATM. The ATM clearly cannot verify this PIN on its own, so it needs to refer the PIN to the user's issuing bank. Since PINs are sensitive, this information should be encrypted. It is impractical for every ATM to share an encryption key with every issuing bank, so a process of key translation (see Section 10.4.1) is used:

1. The ATM encrypts the PIN and the authentication request message using a key shared by the ATM and the acquiring bank responsible for that ATM (each ATM should have a unique key of this type).
2. The acquiring bank decrypts the ciphertext and then re-encrypts it under a key known as the *acquirer working key*, which is a key shared by the acquiring bank and the PCO.
3. The PCO decrypts the ciphertext and re-encrypts it using an *issuer working key*, which is a key that the PCO shares with the issuing bank.
4. The issuing bank decrypts the ciphertext and makes the necessary checks of the PIN and the authentication request message. The response is then relayed back to the ATM.

Symmetric cryptography is used for this application mainly for legacy reasons, since this type of application predates the invention of public-key cryptography. However, symmetric cryptography is also feasible to use in this situation because the underlying infrastructure is 'closed' and thus symmetric keys can be managed. The symmetric algorithm employed is 2TDES (see Section 4.4.4). Again this is a legacy choice, since the original specifications used single DES.

Note that it would be dangerous simply to encrypt the PIN directly during this process, since the limited number of PINs will result in a limited number of possible ciphertexts representing encrypted PINs. If the same key were to be used to encrypt several PINs then an attacker could conduct a dictionary attack that matched a ciphertext representing an unknown PIN against a ‘dictionary’ of ciphertexts corresponding to known PINs. This threat is prevented by two important mechanisms:

Use of a PIN block. The PIN is never encrypted directly. Instead a *PIN block* is formed, one example of which consists of a 64-bit string containing the PIN being XORed to a 64-bit string containing the *Personal Account Number* (PAN) corresponding to the card. This means that two cards with the same PIN will not be encrypted to the same ciphertext under the same encryption key.

Session key encryption. Further security is provided by ensuring that ATMs use session keys, which are generated for a single PIN encryption event and then destroyed.

CARD VERIFICATION VALUES

One major problem with magnetic stripe cards is that they are relatively easy to clone. Early payment cards only included routine information such as the PAN and expiry date on the magnetic stripe. Since this information is easily obtained by a potential attacker (most of it is even displayed on the card itself, or can be obtained from receipts), it was very easy for an attacker to forge such a card.

The problem was alleviated by the inclusion of a cryptographic value known as the *Card Verification Value* (CVV) on the magnetic stripe (we adopt Visa terminology here, while MasterCard uses the term *Card Validation Code* for the same concept). The CVV consists of three digits that are extracted from a hex ciphertext, which is computed by encrypting the routine card information using a key known only to the issuer. The CVV is not displayed on the card and can only be created and verified by the card issuer.

Of course the CVV can be obtained by an attacker who has read off all the information contained on the magnetic stripe, for example, a rogue merchant. Payment cards thus include a second CVV value, CVV2, which is a cryptographic value computed in a similar (but slightly different) way to the CVV. The CVV2 is displayed on the reverse of the payment card, but is not included in the magnetic stripe. The CVV2 is primarily used as a simple check of the physical presence of a card, particularly in transactions made over the telephone or online (see Section 12.4.4).

PIN VERIFICATION VALUE

In order to improve availability, PCOs also provide a service which allows PINs to be verified when the card issuer is unable to process PIN verification requests. This is conducted using a *PIN Verification Value* (PVV), which is computed in a similar way to the CVVs, except that the PIN itself forms part of the plaintext

that is encrypted in order to generate the PVV. The issuing bank needs to share the key that it uses to compute this PVV with the PCO.

The PVV is four digits long, so that its security is 'equivalent' to that of the PIN itself. Like the CVV, the PVV is normally stored on the magnetic stripe but not displayed on the card. During a PIN verification request, the PCO recomputes the PVV using the PIN that has been offered by the customer and checks whether this value matches the PVV on the magnetic stripe. If it does then the PIN verification is accepted.

Although CVVs and PVVs are short values, and hence theoretically could be exhaustively searched for by an attacker, PCOs use procedural controls to stop any apparent attack of this type. A typical control rejects the card after a maximum of three or four attempts. In this way a relatively weak cryptographic mechanism is strengthened by an appropriate management control.

PAYMENT CARD AUTHORISATION

When a payment card is inserted into a terminal, the main goal of the terminal is normally to determine the validity of the card and decide whether the transaction that is being requested is likely to go through. Prior to magnetic stripe cards, this process required a merchant to make a telephone call to the issuer. The ability for a terminal to extract data from the magnetic stripe and automatically contact the issuer in order to authorise a transaction certainly makes this process easier. However, it is important to note that with magnetic stripe cards this process still requires direct (online) communication with the card issuer. This requirement has restricted the adoption of payment cards of this type in countries with poor communication infrastructures.

12.4.3 EMV cards

EMV cards were introduced for two main reasons. The first reason was in order to improve the security of payment card transactions. The other reason was to lower telecommunication costs by introducing a secure means of authorising a transaction offline, hence reducing the number of times that a merchant might have to contact a card issuer.

The introduction of EMV cards has greatly increased the use of cryptography to protect payment card services since the chip on the card is capable of storing cryptographic keys. As we will see in Sections 12.4.4 and 12.4.5, they have also increased the diversity of secure services for which a card can be used.

PIN VERIFICATION

PIN verification becomes much more straightforward for EMV than for magnetic stripe cards, since the PIN can be stored on the chip itself. This allows a terminal to easily verify the PIN without having to contact the card issuer, or use a service based on a PVV.

OFFLINE DATA AUTHENTICATION

In order to authorise an EMV card transaction, a terminal must first decide whether to do an offline check, or whether to conduct a stronger online check that involves communicating with the card issuer. The decision as to which check to conduct depends on the transaction amount and the number of transactions conducted since the last online check.

Offline data authentication does not involve the card issuer. In its most basic form, it provides a means of gaining assurance that the information stored on an EMV card has not been changed since the payment card was created by the card issuer. In other words, it provides data origin authentication of the fundamental card data. The stronger mechanisms also provide entity authentication of the card. Offline data authentication of a payment card can be conducted directly by a terminal that the card has been inserted into.

It is impractical to provide this offline service using symmetric cryptography, since each terminal would need to share a symmetric key with every possible issuer. The use of key translation, as discussed in Section 12.4.2 for magnetic stripe PIN verification, requires the issuer to be online. Thus public-key cryptography, in the form of a digital signature scheme, is used to provide offline data authentication. For space efficiency reasons, EMV cards use a type of RSA digital signature scheme with message recovery (see Section 7.3.5) to provide this assurance.

EMV provides three offline data authentication mechanisms:

Static Data Authentication (SDA) is the simplest technique. All that is checked is the digital signature on the card data that is stored on the card. Verification of this digital signature requires access to the issuer's verification key. Clearly it is not reasonable to expect every terminal to have direct access to every issuer's verification key. Thus EMV employs a simple certificate hierarchy (see Section 11.3.3). In this case the card stores a public-key certificate containing the verification key of the issuer. This certificate is signed by the PCO, and the PCO's verification key is installed in every terminal supporting EMV.

Dynamic Data Authentication (DDA) goes one step further and provides this assurance in a dynamic way that differs for each transaction, hence providing another layer of security against card counterfeiting. During DDA, a challenge-response protocol is run that provides entity authentication of the card. In this case each card has its own RSA key pair and includes a public-key certificate for the card's verification key, signed by the issuer, as well as the issuer's public-key certificate, signed by the PCO. We thus have a three-level public-key certificate chain. The card computes a digital signature on the card data as well as some information unique to the current authentication session. The terminal uses the certificates offered by the card to verify this digital signature.

Combined Data Authentication (CDA) is similar to DDA, except that the card also signs the transaction data, thus providing assurance that the card and terminal have the same view of the transaction. This protects against man-in-the-middle

attacks that seek to modify the transaction data communicated between card and terminal. In contrast, DDA can take place before the transaction details have been established.

ONLINE AUTHENTICATION

Online authentication is the stronger check, which requires communication with the card issuer. As with DDA, the objective of online card authentication is for a terminal to gain entity authentication assurance of a payment card that is involved in a transaction. This is provided by means of a simple challenge–response protocol, based on a symmetric key that is shared by the card issuer and the payment card, which stores it on the chip.

The only complication is that the terminal does not share this key, hence the issuer must be contacted online in order to verify the response. More specifically:

1. The terminal generates transaction data (which includes the payment card details) and a randomly generated challenge, which it then sends to the card.
2. The card computes a MAC on this data with the key that it shares with the issuer. This MAC is called the *authorisation request cryptogram*, and is passed on to the issuer.
3. The issuer computes its own version of the authorisation request cryptogram and compares it with the value received from the card. The issuer is also able to conduct a check that there are sufficient funds in the account to proceed with the transaction.

There are also situations where a payment card may require entity authentication of its issuer (for example, if it is being instructed to perform some internal management procedures, such as resetting counters). This can be built into the card authentication procedure as follows:

1. The card issuer treats the authorisation request cryptogram as a randomly generated challenge and computes a MAC on it using the key that it shares with the card. This response is sent to the card.
2. The card uses the key it shares with the card issuer to check this response. If it matches then it successfully authenticates the card issuer.

TRANSACTION CERTIFICATES

At the end of each transaction a *transaction certificate* (TC) is generated. This is a MAC computed on the details and outcome of the transaction and is passed back to the card issuer. The TC is computed using the key shared by the card and the card issuer. The TC is normally only required as evidence in the event of a subsequent dispute about certain aspects of the transaction.

SECURITY OF MANAGEMENT FUNCTIONS

A number of important management functions concerning security features of the payment card can be remotely managed by sending instructions to the card.

These include PIN changes, PIN unblocking instructions and changes to card data items (such as credit limits). These instructions are sent by the card issuer to the card (via a terminal). They are authorised by computing and verifying a MAC on the instruction, which is generated using a symmetric key that is shared by the card issuer and the card. Since this is a very different use of symmetric cryptography, in line with the principle of key separation (see Section 10.6.1) this key is different from the one used in online authentication.

12.4.4 Using EMV cards online

The previous security features of EMV cards all relate to applications where a card comes into contact with a terminal owned by a merchant. In this sense the card is physically ‘present’ and security features of the card can be directly employed.

However, an increasing number of transactions are conducted when the card is remote from the merchant, most commonly when a customer makes an online transaction. These are referred to as *card-not-present* (CNP) transactions. The potential for fraud in such transactions is high, since the most common information used to authenticate CNP transactions is simple card data (PAN, expiry date, CCV2), which is relatively easily acquired by a determined attacker. From the card holder perspective, the counter to this fraud threat has been the ability to challenge fraudulent transactions. However, this brings significant costs to the merchants, as well as being an inconvenience to the PCOs and cardholders when new cards have to be reissued to customers who have been fraud victims.

Secure Electronic Transactions (SET) was a standard that proposed a heavy architecture and set of procedures for securing CNP transactions. It relied on an overarching public-key management system and required all merchants to acquire special supporting equipment. Its complexity prevented it from being successful and so Visa and Mastercard developed a more lightweight approach known as *3DSecure*. The two main goals of 3DSecure are:

1. The card issuer is able to authenticate its payment card holders during a CNP transaction.
2. A merchant gains assurance that it will not later be financially punished because of a fraudulent transaction.

3DSecure is much more flexible than SET because it allows a card issuer to decide by what means it will authenticate its card holders during a CNP transaction. The overall benefits to all parties are that the increased transaction security allows the PCOs to charge merchants less money for using their services.

At its heart, 3DSecure relies on the following process:

1. A merchant that is 3DSecure-enabled puts in a request for authorisation of the card.

2. The card issuer contacts the card holder and requests authentication information. While EMV-CAP (see Section 12.4.5) provides a natural way to enable this, a common instantiation is for the card issuer and card holder to have pre-agreed a password, which the card holder must enter into a form presented to them in an embedded frame on their browser.
3. If authentication is successful, the card issuer computes a MAC on the critical transaction data, using a symmetric key known only to them. This MAC is known as a *Cardholder Authentication Verification Value* (CAVV) and acts as sort of 'signature', vouching for the authentication of the card holder and the transaction data. The CAVV will be used to resolve any subsequent disputes about the transaction.

3D Secure appears to be a popular technique and is widely adopted.

12.4.5 Using EMV cards for authentication

The increase in uptake of remote banking services, both online and over the telephone, presents banks with the challenge of coming up with strong entity authentication mechanisms in order to reduce the risk of fraud. A wide range of entity authentication mechanisms are used for access to remote banking services, including dynamic password schemes, as discussed in Section 8.5. Such solutions require bank customers to possess a device with cryptographic capability.

Since EMV cards have cryptographic capability, and EMV-supporting bank customers have such a card by default, it is natural to consider using the EMV card as part of an entity authentication mechanism. This is precisely the thinking behind the *Chip Authentication Program* (CAP), which specifies a range of entity authentication options (EMV-CAP explicitly refers to MasterCard technology, while Visa have a similar scheme known as *Dynamic Passcode Authentication*). These are supported by a *CAP reader*, which is a handheld device with a display and keypad. This is much the same as the token that we described in our example dynamic password scheme in Section 8.5.2, except that the CAP reader also has a slot into which an EMV card can be inserted. The customer authenticates directly to the CAP reader by means of a PIN. The CAP reader can then support several different entity authentication mechanisms:

Identify. This option displays a number on the CAP reader that is computed from a symmetric key on the EMV card and an EMV customer transaction counter, which is also stored and updated on the card. This mechanism is a type of sequence-number-based dynamic password scheme. The cryptographic computation essentially involves computing a CBC-MAC (see Section 6.3.3) on the input.

Response. This option works in almost the same way as our example dynamic password scheme in Section 8.5.2. In this case the bank provides the customer with a randomly generated challenge. The customer types the challenge into

the CAP reader, which computes a response using the symmetric key on the EMV card (again, based on CBC-MAC). Finally, the customer provides the bank with the displayed response.

Sign. This is stronger version of the *response* mechanism, which involves the CBC-MAC being computed on basic transaction data (amount and recipient account) as well as the challenge value. This can be used to provide a type of ‘digital signature’ on the transaction. This is an example of the ‘asymmetric trust relationship’ use of MACs to provide non-repudiation that we discussed in Section 7.2.2.

The use of CAP has been gaining in popularity amongst European banks, where its provision of relatively strong two-factor authentication based on EMV cards does appear to have reduced certain types of fraud, such as those based on phishing attacks.

12.4.6 Payment card key management

We now briefly review some of the main key management issues relating to the cryptography employed by payment cards.

KEY MANAGEMENT SYSTEM

While the cryptography used by magnetic stripe cards is entirely symmetric, EMV uses a hybrid of symmetric and public-key cryptography. While PCOs allow issuing and acquiring banks to manage the keys of their own customers, the PCOs provide overarching key management services that link up these banks and facilitate secure transactions.

The model depicted in Figure 12.8 that underlies payment card transactions is essentially the same as the connected certification model that we presented in Figure 11.4. We argued in Section 11.3.2 that this model is suitable for public-key certificate management across a large distributed organisation. It is thus a good model to adopt given the distributed nature of a PCO’s network of banks.

KEY GENERATION

A PCO generates its own master public-key pair. PCOs maintain master RSA key pairs of different lengths in order to cope with potential improvements in factorisation techniques.

Individual banks are responsible for the generation of all keys that are placed on their own cards. Banks are also responsible for generating their own RSA key pairs, which they submit to the PCOs for certification. The symmetric keys are all 2TDES keys. The keys stored on a customer’s card are typically derived from the user’s PAN and a master derivation key. The keys stored on the card are never used directly (in contrast to our slightly oversimplified description). Instead, session keys are derived from these long-term keys and a transaction counter, which is

also maintained on the card and is communicated to relying parties during a transaction.

KEY ESTABLISHMENT

The advantage of a closed system of this type is that the keys stored on a card can be pre-installed during the manufacturing (or personalisation) process. This is slightly more complex for RSA key pairs, since they cannot be mass generated as efficiently as symmetric keys. The session keys used in individual transactions are established on the fly during the transaction, as just discussed.

A PCO's verification key is installed into terminals during their manufacture. PCOs also oversee an important symmetric key hierarchy. At the top level are *zone control master keys*, which are manually established using component form (see Section 10.3.3). These are used to establish the acquirer working keys and issuer working keys.

KEY STORAGE

All the long-term secret or private keys used in EMV payment card systems are protected in tamper-resistant hardware, either in the form of an issuer's hardware security module or the chip on the payment card.

KEY USAGE

In general, key separation is enforced in EMV. The two main security functions that involve encryption using keys stored on a card are conducted using separate symmetric keys.

12.4.7 Payment card security issues

A PCO's overall security goal is to keep fraud using its cards down to a manageable level. PCOs such as Visa and MasterCard thus maintain a risk management division whose function is to assess whether the current security controls are good enough. The evolution from magnetic stripes to EMV cards also reflects an adaptation to cope with new perceived threats.

3DSecure is a further response to the growing problems of online fraud. Of course, 3DSecure is only as secure as the authentication mechanism deployed by the issuing bank to authenticate its customers. Nonetheless, it seems to offer a reasonable compromise between security and usability that has helped to reduce the amount of CNP fraud.

12.4.8 Payment card cryptographic design issues

The main cryptographic design issues concerning payment card cryptographic security mechanisms are:

Use of well-respected cryptographic algorithms. Payment cards use 2TDES and RSA, which are well-established algorithms.

Targeted use of public-key cryptography. Payment cards use public-key cryptography precisely when it delivers substantial benefits, namely in simplified key management for the support of offline data authentication.

Balance of control and flexibility. PCOs strictly control the part of the key management infrastructure that they need to, but otherwise devolve control to participating banks. This provides scalable key management and allows banks to develop their own relationships with their customers.

Efficient use of related data. Payment cards use data in a number of imaginative ways. For example, PANs are used to derive keys, and items of transaction data are used as challenges in authentication protocols. This is both efficient and clever.

12.5 Cryptography for video broadcasting

The next application that we will examine is fundamentally different from the previous ones. This is the use of cryptography to protect digital video content that is being broadcast, sometimes referred to as *pay-TV*. What is most fascinating about this application is that the security service requirements are fairly straightforward but the key management is very sophisticated. This is because the environment poses some unusual operational restrictions that require special key management techniques.

12.5.1 Video broadcasting background

Commercial television broadcasters have traditionally financed the provision of their services either through government subsidy or advertising revenue. This is primarily because most analogue broadcast content can be received by anyone with access to a suitable device, such as a television set. This makes alternative business models, such as those based on annual subscription, hard to enforce. For example, enforcement of the annual television license in the UK involves locating devices and attempting to retrospectively collect revenue.

An alternative option is to 'encrypt' analogue content using special techniques that are developed for particular broadcast technologies. This process is often referred to as *scrambling*. This requires a consumer of content to acquire dedicated hardware in order to use decryption to recover the content. This requirement thus presents an opportunity for revenue collection.

Digital video broadcast networks process digital content, thus making it possible to use the full range of modern cryptographic mechanisms to protect content. This, in turn, enables a wide variety of different business models. Most of these require consumers to obtain specific hardware (or occasionally software) in order to recover content. Common models include *full subscription services*

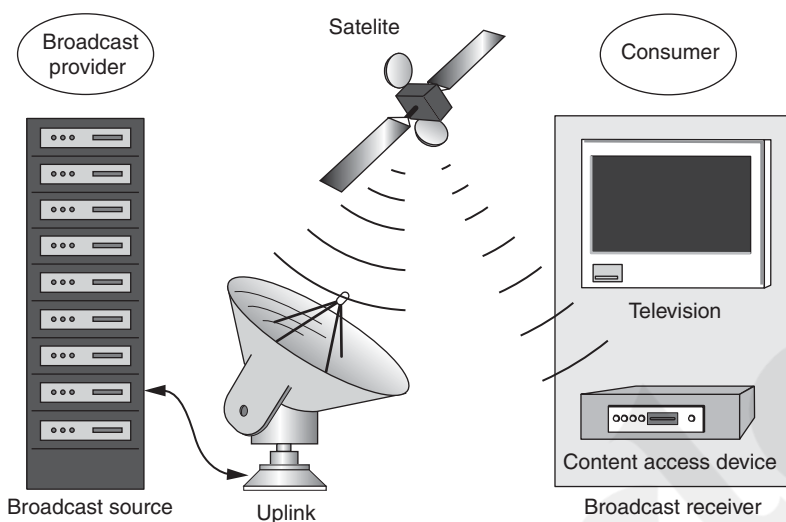


Figure 12.9. Digital video broadcast network

that allow consumers to access all broadcast content for a specific period of time, *package subscription services* that allow consumers to access 'bundles' of predefined broadcast content, and *pay-per-view services* that allow the purchase of specific broadcast content (for example, a live broadcast of a sports event). The compression of digital video broadcasts also allows more content to be broadcast than that of analogue over a similar bandwidth. It thus creates the opportunity for a much more diverse provision environment.

Figure 12.9 shows a simple example of a possible infrastructure for a digital video broadcast network. The *broadcast source* transmits the broadcast content, and is under the control of the *broadcast provider*. The broadcast content is transmitted to the *consumer* of the content, who requires access to a suitable *broadcast receiver* in order to receive the signal. In the example in Figure 12.9, the communication channel is over the air via a satellite link, hence the broadcast receiver takes the form of a satellite dish. However, a digital video broadcast could just as well be transmitted by other media, such as a fibre optic cable, in which case the broadcast receiver is any hardware device capable of receiving the content. As well as receiving the data transmitted by the broadcast source, the consumer requires a *content access device*, which has the capability of decrypting to recover the broadcast content. While this can be implemented in software, most content access devices are hardware devices that contain a smart card. The critical data that is required to control access to the broadcast content, such as cryptographic keys, will normally be stored on the smart card, thus allowing the potential for a content access device to be used to obtain content from different

broadcast providers. In such cases we will choose to regard the ‘content access device’ as the hardware *and* the smart card, unless otherwise specified. Note that this generic network infrastructure is independent of the business model being used to sell the broadcast content.

In the subsequent discussion we will consider a generic broadcast video application, rather than a specific provider’s system. We will assume, however, that the broadcast provider is using the *Common Scrambling Algorithm* (CSA), which is a standard proprietary encryption algorithm around which many providers base their security (see Section 12.5.3).

12.5.2 Video broadcasting security requirements

In order to appreciate the security requirements for digital video broadcasting, it is first necessary to appreciate two important constraints on the broadcast network environment:

One-way channel. The broadcast communication channel only operates in one direction: from broadcast source to broadcast receiver. There is no means by which a consumer can send information back to the broadcast source on this communication channel.

Uncontrolled access. Just as for analogue broadcasts, digital video broadcast content can be received by anyone with the right broadcast receiver technology (a satellite dish in our example in Figure 12.9).

The security requirement for digital video broadcast is thus, simply:

Confidentiality of the broadcast content. In order to control the revenue stream the broadcast provider must make the broadcast content essentially ‘worthless’ to anyone who has not purchased the necessary content access device. In other words, confidentiality is required on the broadcast channel, with only authorised consumers having access to the necessary decryption keys. It is important to note that this requirement for confidentiality does not arise due to the sensitivity of the broadcast content. On the contrary, the broadcast provider *wants* people to view this content, so long as they have paid to do so. This requirement is sometimes referred to as *conditional access*.

It is worth briefly considering why this is the only security service requirement.

Entity authentication. Most of our previous applications required some level of entity authentication, which would be one way of controlling which consumers get access to broadcast video content. However, this requires the consumer to be able to communicate with the broadcast source, which in this case is not possible. Entity authentication of the broadcast source is possible, but unnecessary, since the threat of an attacker posing as a broadcast source and sending false video broadcasts is not particularly relevant to most commercial broadcast environments.

Data integrity. There is no doubt that data integrity is important, since a video broadcast channel is potentially prone to errors in the transmission channel. However, the threat against data integrity is more likely to be accidental errors rather than deliberate ones introduced by a malicious attacker. Hence the solutions lie in the area of error-correcting codes (see Section 1.4.4) and not cryptographic mechanisms.

There are several other operational constraints that influence the design of the key management for digital video broadcasting. We will discuss these in Section 12.5.4.

12.5.3 Cryptography used in video broadcasting

Since confidentiality is required, we need to determine which encryption algorithm to use. The cryptographic design decisions behind this are almost identical to those for GSM encryption (see Section 12.3.3), namely:

A fully symmetric cryptographic architecture. Video broadcast networks are closed systems.

Stream ciphers for data encryption. Video broadcasts involve streaming data in real time over potentially noisy communication channels.

Fixing the encryption algorithm. Agreeing on use of a fixed encryption algorithm allows this algorithm to be implemented in all broadcast receivers, aiding interoperability.

Proprietary encryption algorithm. Choosing to design a proprietary encryption algorithm was justifiable for the same reasons as for GSM. In this case the expertise lay with members of the *Digital Video Group* (DVB), which is a consortium of broadcasters, manufacturers, network operators, software developers, and regulatory bodies with interests in digital video broadcasting. As for GSM, one of the influences behind the design was to make decryption as efficient as possible, since content access devices are less powerful than broadcast sources.

The proprietary encryption algorithm that was designed was CSA. While CSA was standardised by ETSI (see Section 12.3), it was only available for scrutiny under a non-disclosure agreement. However, in 2002, the CSA was implemented in a software application and subsequently reverse-engineered.

The CSA is essentially a double stream cipher encryption. The first encryption is based on a proprietary block cipher deployed in CBC mode, which means that it operates as a stream cipher (see Section 4.6.3). The second layer of encryption uses a dedicated stream cipher to encrypt the ciphertext produced during the first encryption (this is a slight simplification). The key length is 64 (only 48 of the bits are actually used for encryption) and the same encryption key is used for both encryption processes. It is not clear why this ‘double encryption’ layered design

was adopted, but the natural reason would be as a form of insurance against one of the layers of encryption being broken.

12.5.4 Key management for video broadcasting

The primary key management task for digital video broadcasting is simple to state: the keys required to recover broadcast content should be available only to those consumers who are authorised to view the broadcast content. However, there are several complications that combine to make this a challenging task:

The number of potential consumers. A digital video broadcast network is likely to have a large number of consumers (in some cases this could be several million), hence the key management system design must be sufficiently scalable that it works in practice.

Dynamic groups of authorised consumers. The groups of consumers who are authorised to view digital broadcast content is extremely dynamic. Pay-per-view services provide the extreme example of this, where the group of authorised consumers is likely to be different for every content broadcast.

Constant service provision. In many applications a broadcast source will be constantly streaming digital video content that needs to be protected. There are no break periods in which key management operations could be conducted. Most key management must therefore be conducted on the fly.

Precision of synchronisation. As we know from Section 4.2.4, stream ciphers require the keys at each end of the communication channel to be synchronised. In digital video broadcasting this synchronisation has to happen between the broadcast source and *all* (and as we have just pointed out, this could be ‘millions of’) authorised consumers. This synchronisation must be close to being perfect, otherwise some consumers may incur a temporary loss of service.

Instant access. Consumers normally want instant access to broadcast content and will not tolerate delays imposed by key management tasks. A good example of the extreme nature of this problem arises in the case of subscription services, where consumers often choose to select a series of different broadcast channels, each for a very short period of time, in order to make a selection (often referred to as ‘channel surfing’). Since these different channels need to be encrypted using different encryption keys, the content access device needs to have instant access to all the relevant decryption keys.

We will now look at how digital video broadcasting systems typically address these challenges.

VIDEO BROADCAST KEY MANAGEMENT SYSTEM DESIGN

As we indicated in Section 12.5.2, all video broadcast content must be encrypted during transmission. In Section 12.5.3 we identified that this must be using a symmetric key, which we will refer to as the *content encryption key* (CEK).

Since the broadcast source only transmits one version of an item of broadcast content, the content encryption key used to encrypt a specific item of content must be the same for all consumers. Since consumers have different access rights to digital content, the CEK for two different items of broadcast content must be different.

The challenge is thus to make sure that only consumers who are authorised to access content can obtain the appropriate CEK. In order to facilitate this, the operational constraints that we outlined impose the following key management design decisions:

Encrypted CEK is transmitted in the broadcast signal. The most important reason for this is the need for instant access. The CEK is transmitted along with the content itself and is made ‘instantly available’ by being continuously repeated, perhaps every 100 milliseconds or so. Clearly the CEK cannot be transmitted in the clear, otherwise anyone receiving the broadcast signal could obtain it and hence recover the content. Thus the CEK is transmitted in encrypted form. We will refer to the key used to encrypt the CEK as the *key encrypting key* (KEK).

CEK is frequently changed. Once someone has access to the CEK, they can use it to recover all broadcast content that is encrypted using it. Thus it is important to frequently change the CEK, for the reasons discussed in Section 10.6.2. In most video broadcast systems the CEK typically changes every 30 seconds, but this can happen as often as every five seconds.

CEK is transmitted in advance. In order to aid synchronisation and instant access, the CEK is issued in advance of the transmission of any content broadcast using it. Clearly this cannot be too far in advance because of the dynamic nature of the authorised consumer base. The compromise is to constantly transmit two (encrypted) CEKs, which consist of:

1. the current CEK that is being used to encrypt the current broadcast content;
2. the ‘next’ CEK that will be used to encrypt the next broadcast content.

Hence the content access device has time to recover the next CEK and have it instantly available as soon as the CEK is changed.

Use of symmetric key hierarchies. We have already seen that video broadcast schemes use KEKs to encrypt the CEKs. This of course just ‘transfers’ the access problem to making sure that only authorised consumers have access to the required KEKs. In order to manage this problem in a scalable way, video broadcast systems use symmetric key hierarchies (see Section 10.4.1), the details of which we will discuss shortly.

VIDEO BROADCAST KEY ESTABLISHMENT

We now discuss how a video broadcast scheme establishes the KEKs that are necessary for authorised consumers to obtain the CEKs that they are entitled to.

As previously mentioned, video broadcast schemes use symmetric key hierarchies. At the 'top' of each these hierarchies are keys that are shared only by the broadcast provider and a particular consumer, which we refer to as *consumer keys* (CKs). In a simple system, with relatively few consumers, these CKs could be used to encrypt the KEKs. However, there are two reasons why this is not very practical:

1. Most video broadcast systems have so many consumers that sending an encrypted KEK in this way would require too much bandwidth, since a unique ciphertext would have to be sent for each consumer.
2. Each KEK itself must be frequently changed, for similar reasons to the CKs. This might happen, say, on a daily basis. Thus the bandwidth problems are further exacerbated by the need to frequently update the KEKs.

The compromise is to deploy *zone keys* (ZKs), which are keys shared by groups of consumers. Zone keys have longer lifetimes than KEKs, but shorter lifetimes than CKs. A relevant ZK is initially sent to a consumer encrypted using their CK. The consumer then uses the ZK to recover KEKs, which are used to recover CEKs. When a ZK needs to be changed, the new ZK does need to be sent to every consumer who requires it, but this event occurs much less frequently than for KEKs (which in turn occurs much less frequently than for CEKs).

The consumer keys, which sit at the top of these key hierarchies, are stored on the smart cards of the content access devices. They are thus established prior to the issuing of the smart cards to the consumers. A simple example set of key hierarchies is shown in Figure 12.10. In this example there are five consumers, divided into two zones. In practice, multiple layers of zone keys can be deployed in order to enhance scalability.

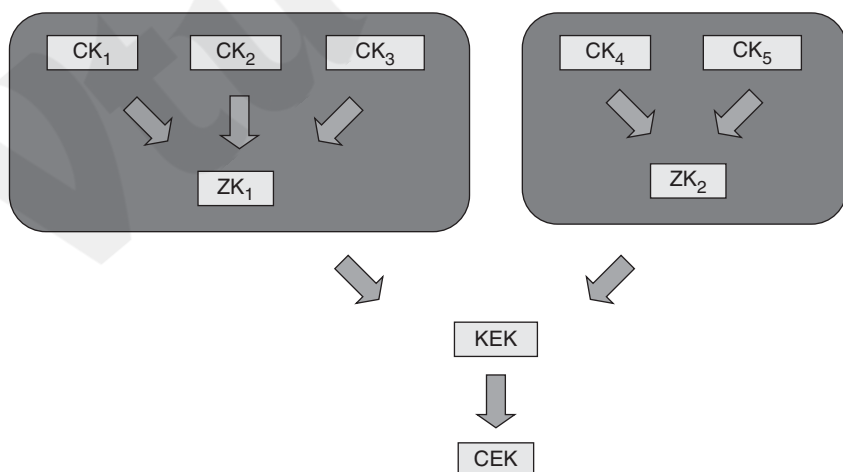


Figure 12.10. Digital video broadcast scheme key hierarchy

Video broadcast schemes provide a good example of the benefits of using a symmetric key hierarchy in order to provide scalable key management. It is worth noting that, unlike the encryption of the content, there are no standards mandating the encryption algorithms used to distribute keys in the hierarchy. Thus different content providers are free to choose their own methods for doing this.

VIDEO BROADCAST ACCESS CONTROL

Encryption of the broadcast content prevents anyone who does not have a content access device with a smart card containing a consumer key from recovering broadcast content. However, it should be apparent that a consumer who does have a valid CK will be able to recover content. This poses a potential problem when a consumer's contract with a broadcast provider ends since, in theory, the consumer will still be able to access broadcast content until the next update of the relevant ZK.

In practice, this problem is addressed by the enforcement of access control in the content access device (see Section 1.4.6). Each consumer is issued in advance with their *content access rights*, which identifies which content the consumer has permission to access. These rights are distributed to a consumer in a special management message, encrypted using the consumer's CK. These content access rights can be updated at any stage using a similar process. Before a content access device attempts to recover any content, it first checks the consumer's content access rights to find out whether the consumer is entitled to access the content. If they are then the content is recovered, otherwise the content access device refuses to proceed.

Thus video broadcast schemes have a two-tiered approach to protecting content. At one level there is access control, enforced by the content access device. The other level is cryptographic control, enforced by using the symmetric key hierarchy. A 'normal' consumer should be prevented by the hardware controls from determining their CK and thus cannot alter the content access rights that they have been issued with. However, even a consumer who is able to do this will ultimately be 'shut out' of their ability to access content through the key establishment controls that we previously described.

VIDEO BROADCAST KEY STORAGE

In hardware-based content access devices all the relevant keys, including the important consumer key, are stored on the smart card. However, there remains one potential point of vulnerability. Since content access devices are designed to be interoperable between different broadcast providers, the interface between the smart card and the rest of the content access device is standardised and hence well understood. This potentially allows an attacker to attempt to obtain a CK when it is transferred between the smart card and the rest of the content access device. For this reason, a shared symmetric key between the smart card and the rest of the content access device is often established, in order to secure this interface.

12.5.5 Video broadcast security issues

Digital video broadcast schemes appear to have the capability of providing strong protection for broadcast content. This is extremely important for the broadcast providers, since they rely on encryption to protect their source of revenue. The protection of the content ultimately relies on the security of CSA, which despite some concerns remains fairly well regarded. Although the key length is short by modern standards, the keys are frequently changed. The security of these keys also relies on the encryption algorithms and key management techniques deployed by a specific broadcast provider to protect all the keys in the key hierarchy.

12.5.6 Video broadcast design issues

Video broadcast networks have provided us with a very interesting application of cryptography, with the following design issues:

Use of symmetric cryptography. The closed nature of a video broadcast scheme facilitates the use of a fully symmetric cryptosystem.

Use of a symmetric key hierarchy. Video broadcast schemes provide a good example of the benefits of deploying a key hierarchy to support symmetric key management.

The influence of operational constraints. While the security requirements for video broadcast networks are fairly straightforward, the operational constraints require some innovative key management controls. Of particular interest are the techniques used in order to provide instant and synchronised access to broadcast content across a potentially large consumer base.

Partially standardised infrastructure. Video broadcast schemes follow some common standards, for example, for content encryption, while leaving other aspects such as higher-level key establishment open to custom design by individual broadcast providers. While this provides the opportunity for a diverse market of interoperable schemes, it also presents a potential source of vulnerability in specific systems.

Note that the security of video broadcast networks is just one example of a larger set of applications relating to the wider problem of *digital rights management*, which concerns technologies for limiting the access of users to digital content.

12.6 Cryptography for identity cards

The applications that we have looked at so far have all had fairly specific goals, and hence well-defined specific security requirements. Our next use of cryptography is quite different in this regard. National (citizen) identity cards are normally

intended to be general purpose ‘tokens’ that can be used by a range of applications that require information relating to the identity of a citizen. They are thus tools that can be deployed in applications, rather than being applications in their own right. Since identity cards can be deployed with many different functionalities (and indeed many such schemes do not deploy cryptography at all), we will focus our discussion on one specific scheme, the Belgian *eID* card scheme, which was one of the first such schemes to provide cryptographic capability on each identity card. This is an example of cryptography being made widely available for use by other applications, rather than cryptography being deployed to provide specific support to a particular application.

12.6.1 eID background

Within a specific context, such as a workplace, most people accept cards that contain and/or display data relating to the identity of the holder. However, the attitude towards *national* identity card schemes is surprisingly diverse and, to an extent, cultural. In some countries, such as the UK, there is a great deal of hostility to such schemes. This is largely due to concerns over privacy issues, costs of deployment, data management and doubts about the utility of such a scheme. In many other countries, such as Belgium, national identity card schemes have been rolled out and are integrated into daily life.

The main application of national identity cards is to present independently issued evidence of the identity of the card holder. Such cards typically display a photograph of the card holder and some personal details, which may include a handwritten signature. However, the progress in smart card technology and the development of cryptographic applications has presented the opportunity for national identity cards to provide additional functionality and thus, perhaps, become more useful.



Figure 12.11. eID card

The eID card scheme was motivated by the establishment of the 1999 European Directive on Electronic Signatures, which created a framework that enabled electronic signatures (see Section 7.1.2) to become legally binding. The first eID cards were issued to Belgian citizens in 2003 and from 2005 all newly issued identity cards were eID cards.

The eID card has four core functions:

Visual identification. This allows the card holder to be visually identified by displaying a photograph on the card alongside a handwritten signature and basic information such as date of birth (see Figure 12.11). This functionality is also provided by previous Belgian identity cards.

Digital data presentation. This allows the data on the eID card to be presented in electronic form to a verifying party. The card data has a specific format and includes:

- a digital photograph of the card holder;
- an *identity file* which consists of:
 - personal data such as name, national identity number, date of birth, and special status (for example, whether the card holder has a disability);
 - a hash of the digital photograph of the card holder;
 - card-specific data such as chip number, card number and validity period;
- an *address file* which consists of the card holder's registered address.

Applications of digital data presentation include access control to facilities such as libraries, hotel rooms and sports halls.

Digital card holder authentication. This allows a card holder to use the eID card to 'prove' their identity in real time to a verifying party. In other words, it facilitates entity authentication of the card holder. The many listed applications of digital card holder authentication include remote access to various internet services, including official document requests (for example, birth certificates), access to an online tax declaration application, and access to patient record information.

Digital signature creation. This allows the card holder to use the eID card to digitally sign some data. Applications of digital signature creation include signing of electronic contracts and social security declarations. Digital signatures created using an eID card are legally recognised.

12.6.2 eID security requirements

The three digital functions of the eID card motivate the following three security requirements:

Data origin authentication of the card data. In order to provide digital data presentation, assurance that the card data has not been changed since the card was issued must be provided.

Ability to provide a data origin authentication service. In order to support digital card holder authentication, it is necessary for an eID card to be used as part of an entity authentication service. The eID card's role in this is to provide a data origin authentication service, which can then be used to support an entity authentication protocol between the card holder and a verifying party.

Ability to provide a non-repudiation service. In order to support digital signature creation, an eID card must be able to provide non-repudiation.

12.6.3 Cryptography used in eID cards

The cryptography in the eID card is relatively straightforward. The following design issues are important in determining the eID card's cryptographic capability:

Use of public-key cryptography. The open nature of the potential application space for eID cards dictates that public-key cryptography must be supported. It is impractical for an eID card to contain pre-loaded symmetric keys that will be 'meaningful' to all unknown future applications.

A digital signature scheme suffices. All three of the security requirements for eID cards can be met by using a digital signature scheme. The first requirement does not even require the digital signature scheme to be implemented on the eID card, however, the second and third requirements do need this. Note that the eID card is not required to have the capability to encrypt or decrypt data.

Use of a publicly known digital signature scheme. In order to encourage use of the eID card and aid interoperability, it is imperative that the digital signature scheme that is deployed is widely respected and supported.

The eID card scheme addresses these design issues by using RSA digital signatures with appendix (see Section 7.3.4). The initial eID cards used 1024-bit RSA, however, there are plans to support 2048-bit RSA in future cards. A number of hash functions are currently supported, including MD5 and SHA-1.

The eID card scheme thus presents an interesting case study of the use of public-key cryptography. We will now briefly look at how the three core digital functions are provided using an eID card and then take a look at how the public-key management is supported.

12.6.4 Provision of the eID card core functions

The eID card scheme is governed by an entity called the *National Register* (NR). The NR can be considered as a trusted third party that facilitates the scheme. As we will see in Section 12.6.5, the NR is responsible for issuing eID cards and hence also takes 'ownership' of the personal data contained on them.

Each eID card contains two signature key pairs and one additional signature key:

Authentication key pair. This key pair is used to support digital card holder authentication.

Non-repudiation key pair. This key pair is used to support digital signature creation.

Card signature key. This signature key can be used to authenticate the card, rather than the card holder. Only the NR knows the verification key that corresponds to a particular eID card. This signature key is only used for administrative operations between the card and the NR.

Note that the eID card enforces the principle of key separation (see Section 10.6.1) by having separate signature key pairs for the two different security services. As well as being an example of good key management practice, there are legal reasons for this separation since the non-repudiation verification key requires a higher level of certificate in order to facilitate legal recognition of digital signatures verified using it (see Section 12.6.5).

We now consider how the first two key pairs are used to support the three core digital eID card functions.

DIGITAL DATA PRESENTATION

This involves a verifying party reading the card data and then gaining assurance that the data on the card is correct. To gain this assurance, the verifying party needs to verify two digital signatures that are created by the NR and stored on the eID card:

Signed identity file. This is a digital signature generated by the NR on the identity file.

Signed identity and address file. This is a digital signature generated by the NR on a concatenation of the signed identity file and the address file. In other words, this takes the form:

$$\text{sig}_{NR}(\text{sig}_{NR}(\text{identity file}) || \text{address file}).$$

A verifying party can then verify the card data by first using the verification key of the NR to verify the signed identity file. If this check is fine then they can proceed to verify the signed identity and address file.

The reason that the NR does not simply sign all the card data is that address changes are much more frequent than changes to the content of the identity file. Thus the NR can update an address on the card without having to reissue a new eID card. Hence a major administrative operation is saved at the expense of a slightly more complex verification process of the card data.

DIGITAL CARD HOLDER AUTHENTICATION

Each eID card holder can activate the signature keys on the eID card through the use of a PIN. The card holder also requires access to an *eID card reader*, which

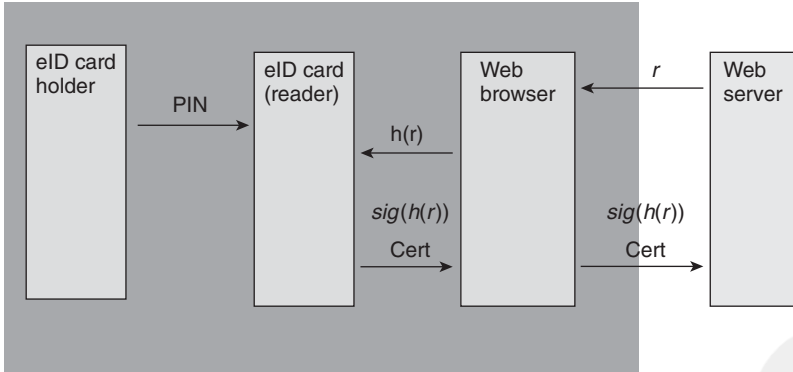


Figure 12.12. eID card holder authentication

may include a PIN pad. This provides an interface between the eID card and the card holder's computer. A typical card holder authentication process is illustrated in Figure 12.12. In this example, a visited web server is requesting authentication of the card holder:

1. The web server randomly generates a challenge r . This is sent to the card holder's browser, which displays a request to login.
2. The card holder enters their PIN into the eID card reader which, if correct, authorises the eID card to proceed with the authentication.
3. The card holder's browser computes a hash $h(r)$ of the challenge r , using a suitable hash function (see Section 6.2) and sends this to the eID card via the card reader.
4. The eID card digitally signs $h(r)$ using the authentication signature key and sends this to the web server via the card holder's browser, along with the card holder's authentication verification key certificate.
5. The web server verifies the received certificate and, if this is successful, verifies the signature and checks that it corresponds to the challenge r . If everything is in order, the card holder is successfully authenticated.

This process is a straightforward application of challenge–response to provide entity authentication (see Section 8.5). Note that the overall security of the authentication process relies on the security of the card holder's PIN. An attacker with access to both the eID card and the PIN can falsely authenticate to the web server.

DIGITAL SIGNATURE CREATION

The digital signature creation process is as described in Section 7.3.4, except that the card holder enters their PIN before the digital signature is created. The digital signature is generated using the non-repudiation signature key.

The non-repudiation verification key certificate is sent to the verifier along with the digital signature. The verifier should then perform all the standard verification checks, which may include checking the appropriate CRLs (see Section 12.6.5), before verifying the digital signature using the procedure outlined in Section 7.3.4.

12.6.5 eID key management

The eID card scheme provides an interesting example of a key management system supporting public-key cryptography. We will look at how the eID card scheme key management is supported, with a particular focus on the two phases in the certificate lifecycle identified in Section 11.2 as being particularly challenging, namely certificate issuing and certificate revocation.

eID CERTIFICATES

The eID card scheme key management is based on the closed certification model that we described in Section 11.3.2. It uses a certification hierarchy, as described in Section 11.3.3, in order to provide a scalable approach to certificate issuing. This certification hierarchy is indicated in Figure 12.13. The main CAs involved are:

Belgium Root CA. This CA is the root CA that oversees all the eID scheme certification. It possesses a 2048-bit RSA verification key certificate that is both self-signed and signed by a commercial CA.

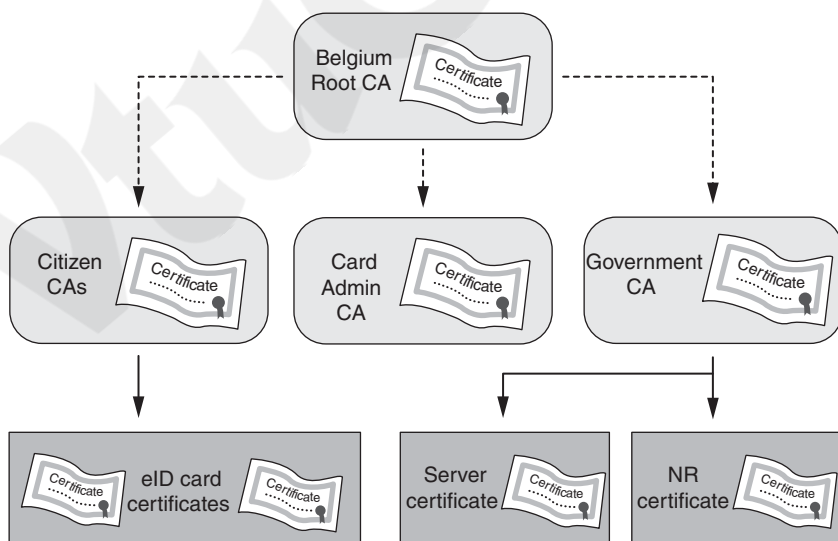


Figure 12.13. eID certification hierarchy

Citizen CAs. These CAs issue certificates to card holders and are responsible for signing the eID card authentication and non-repudiation verification key certificates. Citizen CAs have a 2048-bit RSA verification key signed by the Belgium Root CA.

Card Admin CA. This CA issues certificates to organisations carrying out administrative operation of the eID card scheme, such as those managing address changes and key pair generation. The Card Admin CA has a 2048-bit RSA verification key signed by the Belgium Root CA.

Government CA. This CA issues certificates to government organisations and web servers, including the NR. The Government CA has a 2048-bit RSA verification key signed by the Belgium Root CA.

Each eID card stores five certificates:

1. the Belgium Root CA certificate;
2. the Citizen CA certificate for the Citizen CA that issued the eID card's certificates;
3. the eID card authentication verification key certificate;
4. the eID card non-repudiation verification key certificate;
5. the NR certificate.

All eID card scheme certificates are X.509 Version 3 certificates (see Section 11.1.2). The card holder non-repudiation verification key certificate must, in addition, be a *qualified certificate*, which means that it satisfies further conditions, including that the precise identity of the certificate holder has been established. A certificate is required under European law to be qualified if any digital signatures produced using the corresponding signature key are to be legally binding.

eID CARD ISSUING PROCESS

The process of issuing an eID card is quite complex and involves several different organisations. It serves as a good illustration of the intricacies of generating public-key certificates, which we discussed in general terms in Section 11.2.2. The process is indicated in Figure 12.14 and consists of the following steps:

1. Either after requesting, or being invited to apply for, an eID card, the eID applicant attends a local government office. This office essentially acts as the RA (see Section 11.2.2). The applicant presents a photograph to the RA, which then verifies the personal details of the applicant and formally signs an *eID card request*.
2. The eID card request is sent from the local government office to the *card personaliser* (CP), and the NR is notified. The CP checks the eID card request. For simplicity we will assume the existence of a single CP, who is responsible for creating the physical aspects of the card and for inputting the relevant data onto the chip on the card.
3. The CP creates a new eID card and generates the required key pairs on the card itself. The CP then sends a request for certificates to the relevant Citizen CA via the NR, who issues a certificate serial number for each certificate.

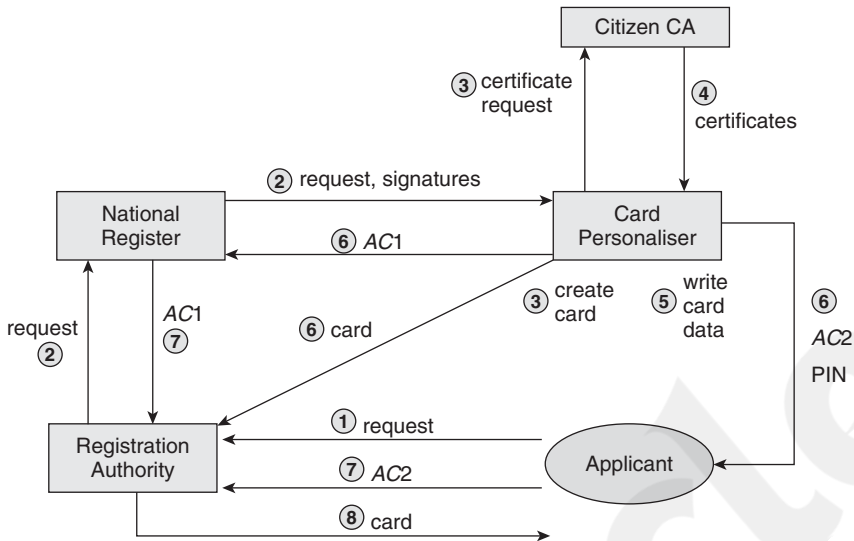


Figure 12.14. eID card issuing process

4. The Citizen CA generates certificates and sends them to the CP, who stores them on the card. The CA then immediately suspends these certificates.
5. The CP writes all the remaining card data onto the card and then deactivates the card.
6. The CP sends:
 - the first part of an activation code AC1 to the NR;
 - the second part of the activation code AC2 and a PIN to the applicant;
 - the inactive eID card to the RA.
7. The applicant revisits the RA and presents AC2. This is then combined with AC1, which the RA requests from the database of the NR.
8. The CA activates the suspended card certificates and the active eID card is issued to the applicant.

eID CERTIFICATE REVOCATION

In Section 11.2.3 we discussed the challenges of public-key certificate revocation. As well as the reasons identified in Section 10.6.2 for revoking a public-key certificate, there are two special situations in which an eID card certificate has the status of being revoked:

1. the eID card non-repudiation verification key certificate is revoked for juveniles under the age of 18;
2. the eID card authentication verification key certificate is revoked for children under the age of 6.

The main technique used to manage certificate revocation in the eID card scheme is CRLs (see Section 11.2.3). A significant problem of the eID card scheme is that the potential size of CRLs is considerable. The eID card scheme Citizen CAs issue new *base CRLs* every three hours. During the period between updates of the base CRL, much smaller *delta CRLs* are issued, which identify changes to the last base CRL. In this way anyone who wishes to maintain their own local copy of the complete CRLs for the eID card scheme does not have to regularly download the full database. All CRLs are digitally signed by the issuing Citizen CA using 2048-bit RSA.

In general, applications using the eID card scheme are free to decide how to manage the certificate revocation information. Options include trusting the revocation status management to a third-party provider who operates an OCSP querying service (see Section 11.2.3), which itself will rely on the information provided in the Citizen CA CRLs.

eID SIGNATURE VALIDITY

Given the importance of some of the applications of eID cards, particularly with respect to digital signature creation, it is worth briefly commenting on the potential validity of digital signatures during two specific periods of time:

Digital signatures created after an incident but before revocation. As discussed in Section 11.2.3, a potential problem arises if a relying party verifies an eID card signature in the period between occurrence of a security incident (of a type that invalidates the eID card non-repudiation verification key certificate) and the revocation of that certificate. If the time of the incident can be precisely verified then, technically speaking, a digital signature created during this period is unlikely to be valid. Applications need to be aware of this potential problem and have procedures for coping with it. The Citizen CAs assist this process by frequently issuing base and delta CRLs.

Validity of digital signatures after expiry or revocation of the eID card (non-repudiation verification key certificate). So long as a digital signature is verified before expiry or revocation of the eID card (or its non-repudiation verification key certificate) then it should still be regarded as valid (and, indeed, may be legally binding) after the expiry or revocation date. One method for making this more explicit is for the signer who signs some data to obtain a digital signature from a trusted third party that attests to the validity of that signature at a specific point in time. Namely, the signer Alice presents her digital signature $sig_A(\text{data})$ to the TTP, who verifies this signature at time t and then generates the digital signature:

$$sig_{TTP}(sig_A(\text{data}) || t).$$

The TTP thus acts as an archiving service. After the expiry or revocation of her eID card, Alice can still present the archived signature as evidence of

its validity. Note that any future relying party does not need to verify Alice's original signature, but does have to trust the TTP. This process assumes that:

- The TTP's verification key has a longer lifetime than Alice's. On expiry of the TTP's verification key, Alice can always ask the TTP to resign the archived signature with its new signature key.
- No flaws are subsequently found in any of the processes or algorithms used to generate or validate Alice's digital signature.

12.6.6 Security issues

The eID card scheme represents a relatively straightforward use of cryptography. Its primary function is to issue citizens with smart cards that have digital signature capability. These applications can then engage with this cryptographic functionality in order to meet their own security requirements. The main 'security issues' are thus likely to arise from the specific ways in which these applications interact with eID cards. Since the eID card scheme supports digital signatures, it is also important to be aware of the many security issues that we discussed regarding the use of digital signatures in Section 7.4.

The main security issues for the eID cards themselves arise from the key management. Card issuing is a fairly complex and controlled process, since the implications of eID cards being either fraudulently issued, or issued with incorrect data, are potentially very serious. Certificate revocation is managed in a scalable way and it is up to individual applications to make sure that they obtain the latest revocation data in order to verify data signed using an eID card.

12.6.7 Design issues

The main design issues concerning the eID card scheme are as follows:

Use of public-key cryptography. While eID cards are issued within a closed environment, they are intended for use in open environments. Thus the use of public-key cryptography is appropriate.

Use of publicly known algorithms. To increase confidence and support interoperability, the eID card scheme uses the well-respected RSA digital signature scheme.

Use of certification hierarchies. The eID card scheme's national reach lends itself very naturally to a certification hierarchy, with central CAs supporting regional registration authorities.

Specific data handling. The eID card design demonstrates that in real applications different data items may require different management. This is reflected in the way that card data is digitally signed, which recognises that address data normally changes much more frequently than other types of personal data.

Flexibility. The eID card scheme is primarily an enabler for cryptographic applications. It therefore leaves specific applications a degree of flexibility on how they manage security of applications interacting with eID cards. In particular, applications must manage their own certificate revocation processing.