Module-4 Key Management

**Introduction**

* Key management is crucial to the security of any cryptosystem.
* Without secure procedures for the handling of cryptographic keys throughout their lifecycle, the benefits of the use of strong cryptographic primitives are potentially lost.
* Indeed, it could be argued that if key management is not performed correctly then there is no point in using cryptography at all.

### What is key management?

* The scope of key management is perhaps best described as the secure administration of cryptographic keys.
* This is a deliberately broad definition, because key management involves a wide range of quite disparate processes, all of which must come together coherently if cryptographic keys are to be securel managed.

The important thing to remember is that cryptographic keys are just special pieces of data. Key management thus involves most of the diverse processes associated with information security. These include:

**Technical controls.** These can be used in various aspects of key management. For example, special hardware devices may be required for storing cryptographic keys, and special cryptographic protocols are necessary in order to establish keys.

**Process controls.** Policies, practices and procedures play a crucial role in key management. For example, business continuity processes may be required in order to cope with the potential loss of important cryptographic keys.

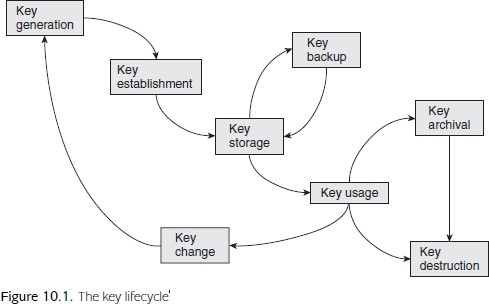
**Environmental controls.** Key management must be tailored to the environment in which it will be practiced. For example, the physical location of cryptographic keys plays a big role in determining the key management techniques that are used to administer them.

**Human factors.** Key management often involves people doing things. Every security practitioner knows that whenever this is the case, the potential for problems occurring is

high. Many key management systems rely, at their very highest level, on manual processes.

* Thus, while cryptographic keys represent an extremely small percentage of the
* data that an organisation needs to manage, much of the wider information security issues that the organisation has to deal with
* (such as physical security, access control, network security, security policy, risk management and disaster recovery) interface with key management.

# The key lifecycle



* Key generation concerns the creation of keys.
* Key establishment is the process of making sure that keys reach the end points where they will be used. **This is arguably the most difficult** phase of the key lifecycle to implement.
* Key storage deals with the safekeeping of keys. It may also be important to conduct **key backup** so that keys can be recovered in the event of loss of a key and, ultimately, key archival
* Key usage is about how keys are used. As part of this discussion we will consider key change. We will also look at how a key’s life ends in key destruction.
* Diffie–Hellman protocol simultaneously generate and establish a symmetric key.
* Also, some phases are not always relevant. For example, key backup and key archival may not be necessary in some applications.

## Fundamental key management requirements

* There are two fundamental key management requirements that apply throughout the various phases of the key lifecycle:
* **Secrecy of keys** : Throughout the key lifecycle, secret keys (in other words, symmetric keys and private keys) must remain secret from all parties except those that are authorised to know them.
* if a weak key generation mechanism is used then it might be possible t determine information about a secret key more easily than intended;
* secret keys are vulnerable( posibility of being attacked) when they are ‘moved around’, thus secure **key distribution mechanisms** must be used;
* secret keys are perhaps most vulnerable when they are **‘sitting around’,** thus key storage mechanisms must be **strong enough to resist an attacker** who has access to a device on which they reside;
* **if secret keys are not destroyed** properly then they can potentially be recovered after the **supposed time of destruction.**

### Assurance of purpose of keys.

* Throughout the key lifecycle, those **parties** relying on a key must have assurance of purpose of the key.
* In other words, **someone in possession of a key** should be confident that they can use that key for the purpose that they believe it to be for.
* This ‘purpose’, may include some, or all, of the following:
* information concerning **which entities are associated with the key** (in symmetric cryptography this is likely to be more than one entity, whereas for public-key cryptography each key is normally only **associated with one entity**);
* the cryptographic algorithm that the key is intended to be used for;
* key usage restrictions, for example, that a symmetric key can only be **used for creating and verifying a MAC,** or that a signature key can only be used for **digitally signing transactions** of less than a certain value.

Key management systems

* Any system for managing the various phases of the key lifecycle key management system may depend on:
* **Network topology**. Key management is much simpler if it is only needed to support two parties who wish to communicate securely, rather than a multinational

organisation that wishes to establish the capability for secure communication between any two employees.

* **Cryptographic mechanisms**. As we will see in this chapter and Chapter 11, some of the key management system requirements of symmetric and public-key cryptography differ.
* **Compliance restrictions**. For example, depending on the application, there may be legal requirements for key recovery mechanisms or key archival (see Section 10.5.5).
* **Legacy issues**. Large organisations whose security partly depends on that of other related organisations may find that their choice of key management system is restricted by requirements to be compatible with business partners, some of whom might be using older technology.

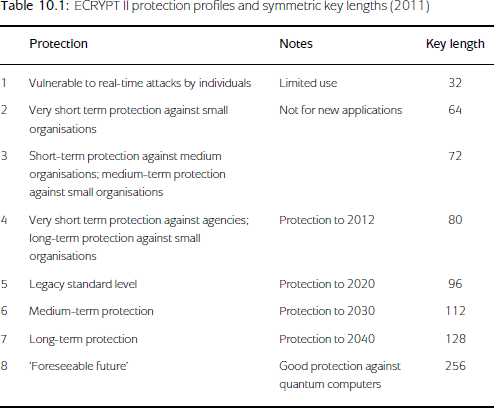
## Key lengths and lifetimes

* Longer keys are better from a security perspective. Longer symmetric keys take more time to exhaustively search for and longer public-key pairs tend to make the underlying computational problem on which a public-key cryptosystem is based harder to solve.
* So there is certainly a case for making keys as long as possible. Cryptographic computation normally takes more time if the key is longer. In addition, longer keys involve greater storage and distribution overheads.
* Hence longer keys are less efficient in several important respects. Thus key length tends to be based on an efficiency–security tradeoff.

# Key lifetimes

* The issue of key length is closely linked to the intended lifetime (also often referred as the cryptoperiod) of a cryptographic key. By this we mean that the key can only be used for a specified period of time, during which it is regarded as being live.
* Once that lifetime has been exceeded, the key is regarded as expired and should no longer be used There are many reasons why cryptographic keys have finite lifetimes. These include:
* **Mitigation against key compromise**. Having a finite lifetime prevents keys being used beyond a time within which they might reasonably be expected to be compromised, for example by an exhaustive key search or compromise of the storage medium.
* **Mitigation against key management failures**. Finite key lifetimes help to mitigate against failures in key management. For example, forcing an annual key change will guarantee that personnel who leave an organisation during the year, but for some reason retain keys, do not have access to valid keys the following year.
* **Mitigation against future attacks**. Finite key lifetimes help to mitigate against future advances in the attack environment. For this reason, keys are normally set to expire well before current knowledge suggests that they need to.
* **Enforcement of management cycles**. Finite lifetimes enforce a key change process, which might be convenient for management cycles. For example, if keys provide access to electronic resources that are paid for on an annual subscription basis, then having a one-year key lifetime allows access to keys to be directly linked to subscriptions to the service.
* **Flexibility**. Finite key lifetimes introduce an additional ‘variable’ which can be adjusted to suit application requirements. For example, a relatively short key (which is relatively inexpensive to generate, distribute and store) could be adopted under the pretext that the key lifetime is also suitably short.
* **Limitation of key exposure**. At least in theory, some information relating to a key is ‘leaked’ to an attacker every time the attacker sees a cryptographic value computed using that key.
* This is because the result of every cryptographic computation provides the attacker with information that they did not have before they saw the ciphertext. We refer to this as key exposure.
* However, this information is normally of little (often no) use to an attacker if the cryptographic algorithm is strong, hence in many applications key exposure is not a significant issue.

## Choosing a key length

* The length of a cryptographic key is commensurate with the key lifetime, which in turn is related to the length of time for which cryptographic protection is required the relationship between key length and key lifetime:
* in an ideal world the key lifetime would be chosen and then a suitable key length selected;
  + in the real world the key length may be dictated (for example, the key management system is based on the use of 128-bit AES keys) and thus the key lifetime is set to an appropriate time period.
  + decision on key length is made much simpler by the limited number of standard cryptographic algorithms, which in turn present limited options for key lengths.
  + Decisions will need to be made, particularly since some of the most popular cryptographic algorithms such as AES (see Section 4.5) and RSA (see Section 5.2) have variable key lengths.
  + Key length recommendations for symmetric cryptography tend to be algorithm in dependent, since the security of respected symmetric encryption algorithms should be benchmarked against the difficulty of an exhaustive key search
  + Key length recommendations for public-key cryptography tend to be algorithm specific, since the security of a public-key cryptosystem depends upon the perceived difficulty of the hard computational problem on which the algorithm is based (for example, factoring in the case of RSA).

There are several issues worth noting from Table 10.1:

1. Some of the key length recommendations are specifically linked to maximum recommended key lifetimes.
2. Although these recommendations are largely algorithm-independent, some further specific advice is given by ECRYPT II on the use of Triple DES, since Triple DES has a much weaker security than that suggested by its key length

**Advice on key length is not unanimous**. Ultimately these are subjective opinions, albeit hopefully informed ones. Before choosing a key length it is advisable to seek recommendations from more than one source.

**Advice on key length changes over time**. It is wise to seek the latest and most accurate information before deciding on key lengths.

# Key generation

* + This begins with key generation, which is the creation of cryptographic keys. This is a critical phase of the key lifecycle.
  + Key generation processes for symmetric and public-key cryptography are fundamentally different.
  + We will first look at ways of generating a symmetric key.

# Direct key generation

* + Symmetric keys are just randomly generated numbers (normally bit strings).
  + The most obvious method for generating a cryptographic key is to randomly generate a number, or more commonly a pseudorandom number. **Strength** of the technique should take into consideration.
  + For example, use of **a hardware-based non-deterministic generator** might be appropriate for a **master key**, whereas
  + software-based non-deterministic generator based on **mouse movements** might suffice for **generating a local key** to be used to store **personal files on a home PC.**
  + DES has some keys that are defined to be weak. In the rare event that such keys are generated by a key generation process, some guidance suggests that they should be **rejected.**

# Key derivation

* + The term key derivation is sometimes used to describe **the generation of cryptographic keys from other cryptographic keys or secret values**.
  + There are several **significant advantages** of deriving keys from other keys:
  + **Efficiency.** Key generation and establishment can be relatively expensive processes**.**
  + Generating and establishing one key (sometimes called a base key), and **then using it to derive many keys,** can be an effective technique for saving on these costs.
  + For example, many **applications require** both confidentiality and data origin authentication.
  + If separate cryptographic mechanisms are to be used to provide these two security services then they require an encryption key and a MAC key.
  + Rather than generating and establishing two symmetric keys for this purpose, a cost efficient solution is to **generate and establish one key K** and then derive two keys K1 and K2 from it.
  + For example, a very simple key derivation process might involve computing:
  + K1= h(K||0) and K2 = h(K||1).
  + **Longevity(long-lasting).** In some applications, long-term symmetric keys are preloaded onto devices before deployment.
  + Using these long-term keys directly to encrypt data exposes them to cryptanalysis .
  + However, **randomly generating a new key requires** a key establishment mechanism to be used, which may not always be possible or practical.
  + A good solution is to derive keys for use from the long-term key.
  + Key derivation must be based on a derivation function that is one-way .
  + This is important because often many different keys are derived using a single base key, hence the impact of subsequently compromising the base key could be substantial.
  + There are standards for key derivation.
  + For example, PKCS#5 defines how a key can be derived from a password or a PIN, which can be regarded as a relatively insecure type of cryptographic key, but one which is often long term (such as the PIN associated with a payment card).
  + Key derivation in this case is defined as a function f (P, S, C, L), where:
* f is a key derivation function that explains how to combine the various inputs in order to derive a key;
* P is the password or PIN;
* S is a string of (not necessarily all secret) pseudorandom bits, used to enable P to be used to derive many different keys;
* C is an iteration counter that specifies the number of ‘rounds’ to compute .
* L is the length of the derived key.

# Key generation from components

* Direct key generation and key derivation are both processes that can be performed **if one entity can be trusted** to have full control of a particular key generation process.
* important secret keys it may not be desirable to trust one entity with key generation.
* In such cases we need to distribute the key generation process **amongst a group of entities** in such a way that no members of the group individually have control over the process, but collectively they do.
* One technique for facilitating this is to **generate a key in component form.**
* We illustrate this by considering a simple scenario involving three entities: **Alice, Bob and Charlie.** Assume that we wish to generate a 128-bit key:

1. Alice, Bob and Charlie each randomly generate a component of 128 bits.

* This component is itself a sort of key, so any **direct key generation mechanism**

could be used to generate it.

* We denote the resulting components **by KA, KB and KC** , respectively.

1. Alice, Bob and Charlie securely transfer their components to a secure combiner.

* In most applications this combiner will be represented by a hardware security module.
* In many cases the ‘secure transfer’ of these components will be by manual delivery.
* The input of the components to the secure combiner is normally conducted according to **a strict protocol** that takes the form of a key ceremony.
* 3. The secure combiner derives a key K from the separate components. In this example, the best derivation function is XOR. In other words:
* K = KA ⊕ KB ⊕ KC .
* Note that the key K is only reconstructed within the **secure combiner** and not output to the entities involved in the key derivation process.
* XOR is the ‘best’ type of key derivation function since knowledge of even two of the components does not leak any information about the derived key K.

# Public-key pair generation

* Since key generation **for public-key cryptography** is algorithm-specific, we will not treat it in detail here. As for symmetric key generation:
* Public-key pair generation often requires the **random generation of numbers**.
* Relevant standards should be consulted before generating public-key pairs. However, in contrast to symmetric key generation:
* Not every number in the ‘range’ of the keyspace of a public-key cryptosystem is a valid key.
* For example, **for RSA the keys d and e** are required to have specific mathematical properties.
* If we choose an RSA modulus of 1024 bits then there are, in theory, **21024 candidates for e or d.**
* However, only some of these **21024 numbers can be an e or d,** the other choices are ruled out.
* Some keys in public-key cryptosystems are **chosen to have a specific format.**
* For example, RSA public keys are sometimes chosen to have a specific format that results in them being ‘**faster than the average case’** when they are used to compute exponentiations, thus speeding up RSA encryptions .
* There is no harm in such a deliberate choice of key since the **public key is not a secret value.**
* The generation of a key pair can be slow and complex.
* Some devices, such as smart cards, may not have the computational resources to generate key pairs.
* In such cases it may be necessary to generate key pairs off the card and import them.

## Key establishment

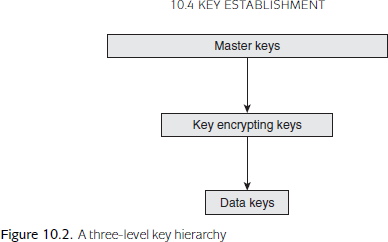
* Key establishment is the process of getting cryptographic keys to the locations where they will be used. This part of the key lifecycle tends either to be relatively straightforward, It is relatively straightforward when:
* **The key does not need to be shared.** This applies to any keys that can be locally generated and do not need to be transferred anywhere, such as symmetric keys for encrypting data on a local machine.
* Of course, if such keys are not locally generated then key establishment becomes hard again!
* **The key does not need to be secret.** This applies mainly to public keys. In this case key establishment is more of a logistical problem than a security issue
* **The key can be established in a controlled environment.** In some cryptographic applications it is possible to establish all the required keys within a controlled environment before the devices containing the keys are deployed. This is often terme key predistribution. or very hard, to manage.

# Key hierarchies

* One of the most widely used techniques for managing symmetric keys is to use
* a key hierarchy. This consists of a ranking of keys, with high-level keys being
* more ‘important’ than low-level keys. Keys at one level are used to encrypt keys
* at the level beneath.
* PHILOSOPHY BEHIND KEY HIERARCHIES
* There are two clear advantages of deploying keys in a hierarchy:
* **Secure distribution and storage.** By using keys at one level to encrypt keys at the level beneath, most keys in the system can be protected by the keys above them. This allows keys to be securely distributed and stored in encrypted
* form.

### Facilitating scalable key change.

* there are many reasons why keys may need to be changed.



### A SIMPLE KEY HIERARCHY

* The idea of a key hierarchy is best illustrated by looking at a simple example. The three levels of this hierarchy consist of:
* **Master keys.** These are the top-level keys that require careful management. They are only used to encrypt key encrypting keys. Since the key management of master keys is expensive, they will have relatively long lifetimes (perhaps several years).
* **Key encrypting keys.** These are distributed and stored in encrypted form using master keys. They are only used to encrypt data keys. Key encrypting keys will have shorter lifetimes than master keys, since they have greater exposure and are easier to change.
* **Data keys.** These are distributed and stored in encrypted form using key encrypting keys.
* These are the working keys that will be used to perform cryptographic computations.
* They have high exposure and short lifetimes.
* This may simply correspond to the lifetime a single session, hence data keys are often referred to as session keys.

### MANAGING THE TOP-LEVEL KEYS

* Top-level (master) keys need to be securely managed, or the whole key hierarchy is compromised.
* Most key management systems using key hierarchies will employ hardware security modules (HSMs) to store master keys.
* These top-level keys will never leave the **HSMs in unprotected form.**
* The generation of master keys is an **extremely critical operation.**
* Master keys are commonly generated, established and backed up in **component form.**
* If a master key needs to be shared between two different HSMs then one option is to generate the same master key from components separately on each HSM.
* An alternative is to run a key agreement protocol between the two HSMs in order to establish a shared master key.

### SCALABLE KEY HIERARCHIES

* The notion of a key hierarchy works fine for a relatively simple network, but quickly becomes unmanageable for **large networks**.
* Consider a simple **two-level hierarchy consisting** of only master and data keys.
* If we have a network **of n users**, then the number of possible pairs of users is 1/2 n(n

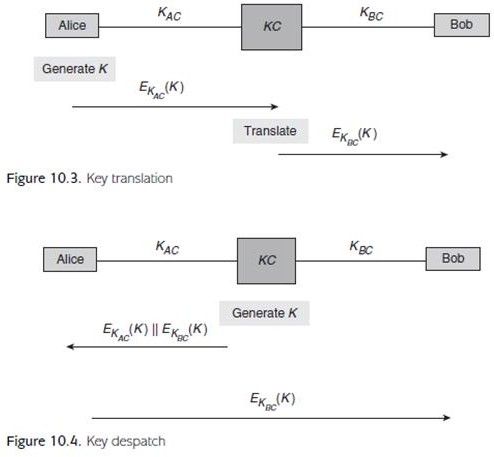
− 1).

* This means that, for example, if there are 100 users then there are **1/2 × 100 × 99 = 4950 possible pairs** of users.
* Hence, in the worst case, we might have to establish 4950 separate master keys amongst **the 100 HSMs in** the network, which is not practical.
* we could **install the same master key in all HSMs**.
* Data keys for communication between Alice and Bob could then be derive**d from the common master key** and Alice and Bob’s identities.
* However, compromise of Alice’s HSM would now not only compromise data keys for use between Alice and Bob, but **data keys between any pair of users in the network**. This is not normally acceptable.
* the users trust, which we will refer to as **a key centre (KC).** The idea is that each user in the network shares a key with the KC, which acts as a ‘go between’ any time any pairs of users require a shared key.
* In this way we reduce the need for 4950 master keys in a network of **100 users to just 100 master keys**, each one shared between a **specific user and the KC.**
* There are two key distribution approaches to acquiring shared keys from a KC.
* We illustrate these using a very simple scenario. In each case we assume that Alice wishes to establish a shared data key K with Bob.
* We will also assume that both Alice and Bob have respectively **established master keys KAC and KBC with the KC,** and that a simple two-level key hierarchy is being employed. The two approaches are:
* **Key translation.** In this approach the KC simply translates an encrypted key from encryption using one key to encryption using another. In this case the **KC is acting as a type of switch.**

1. Alice generates a data key K, encrypts it using KAC and sends this to KC.
2. KC decrypts the encrypted K using KAC , re-encrypts it using KBC and then sends this to Bob.
3. Bob decrypts the encrypted K using **KBC** .

* **Key despatch**. In this approach the KC generates the data key and produces **two encrypted copies of** it, one for each user.

1. **KC generates a data key K,** encrypts one copy of it using KAC and another copy of it using KBC , and then sends both encrypted copies to Alice.
2. Alice decrypts the first copy using KAC and sends the other copy to Bob.
3. Bob decrypts the second copy using KBC .



# Unique key per transaction schemes

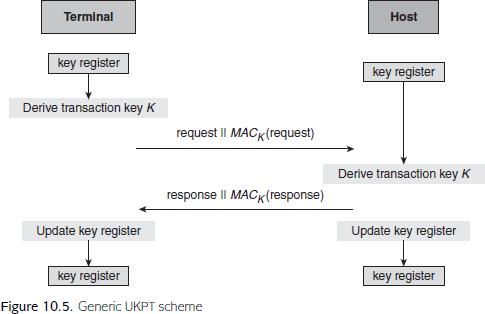
* Unique key per transaction (UKPT) schemes are so called because they **establish a new key each time** that they are used.
* MOTIVATION FOR UKPT SCHEMES
* Most of the previous key establishment mechanisms that we have discussed involve one, or both, of the following:
* Use of long-term (top-level) secret keys, for example, the use of master keys or key encrypting keys in key hierarchies.
* A special transfer of data explicitly for the purposes of key establishment.
* While these are acceptable features in many environments, they may not be desirable in others.
* The first requires devices that can securely store and use long-term keys, and the second introduces a communication overhead.
* An alternative methodology is to generate **new keys by** deriving them from information already shared by Alice and Bob
* If key derivation is used to generate new keys then the processes of
* key generation and key establishment essentially ‘merge’. This brings several advantages:

1. Alice and Bob do not **need to store a long-term key;**
2. Alice and Bob are not required to engage in any special communication solely for the purpose of key establishment;
3. Key generation and establishment can be **‘automated**

APPLICATION OF UKPT SCHEMES

* UKPT schemes adopt the methodology we have just described by updating keys using a key derivation process after each use.
* A good example of an application of UKPT schemes is retail point-of-sale terminals, which are used by merchants to verify PINs and approve payment card transactions. The advantages of a UKPT scheme all apply to this scenario:

1. Terminals have limited security controls, since they must be cheap enough to deploy widely. In addition, they are typically located in **insecure public environments such as stores and restaurants.**
2. They are also portable, so that they can easily be moved around, hence easily stolen. It is thus undesirable that **they contain important top-level keys.**
3. Transactions should be processed **speedily to avoid delays**, hence efficiency is important.
4. Terminals may be managed and operated by unskilled staff, hence full automation of the key establishment process is a necessity



* Figure 10.5 illustrates our generic UKPT scheme:

1. The terminal derives the transaction key using the contents of the key register and shared information that will be available to the host.
2. The terminal sends a request message to the host. The transaction key is used to compute a MAC on the request message.
3. The host derives the transaction key (the technique for doing this varies between schemes, as we will shortly illustrate).
4. The host validates the MAC on the request message.
5. The host sends a response message to the terminal. The transaction key is used to compute a MAC on the response message.
6. The terminal validates the MAC on the response message.
7. The terminal updates the contents of the key register.

* In order to produce a real UKPT scheme from the generic UKPT scheme of
* Figure 10.5, we need to answer three questions:

1. What is the initial value in the terminal key register?
2. How should the transaction key be derived so that the terminal and host derive the same key?
3. How should the terminal key register be updated so that the terminal and host update to the same value?

### Racal UKPT scheme. This scheme answers the three questions as follows:

1. The initial value is a secret seed, which is agreed between the terminal and the host.
2. The host maintains an identical key register to the terminal. The transaction key is derived from the key register and the card data (more precisely, the primary account number on the card), both of which are known by the terminal and the host.
3. At the end of the protocol, the **new key register value** is computed as a function of the old key register value, the card data (primary account number) and the transaction data

.Both the terminal and the host conduct the **same computation** to update their key registers.

### Derived UKPT scheme. This scheme is supported by Visa, amongst others, and

answers the three questions as follows:

1. The initial value is a unique initial key that is installed in the terminal.
2. The transaction key is derived by the terminal from the contents of the terminal key register, a transaction counter, and the terminal’s unique identifier.

The host has a special base (master) key. The host does not need to maintain a key register, but can calculate this same transaction key from the base key, the transaction counter and the terminal identifier.

1. At the end of the protocol the new terminal key register value is derived from the old key register value and the transaction counter.

The host does not need to store this value because it can compute transaction keys directly, as just described.

# Quantum key establishment

* quantum key establishment is often inappropriately described as
* ‘quantum cryptography’. The latter name suggests that it is something to do with new cryptographic algorithms that are suitable for use to protect against quantum computers .
* Quantum key establishment is in fact a technique for establishing a conventional symmetric key, which can then be used in any symmetric cryptosystem, including a one-time pad

### THE BASIC IDEA

* Quantum key establishment takes place over a quantum channel.
* This is typically instantiated by an optical fibre network or free space.
* Alice and Bob must have devices capable of sending and receiving information that is encoded as quantum states, often termed qubits, which are the quantum equivalent of bits on a conventional communication channel.
* These qubits are represented b**y photons.**
* The basic idea behind quantum key establishment is to take advantage of the fact that in a **quantum channel** such an attacker cannot ‘listen in’ without **changing the information in the channel.**
* This is a very useful property, which Alice and Bob can exploit to test whether an attacker has been listening to their communication.
* The most well known quantum key establishment protocol is the BB84 protocol.
* While the following conceptual **overview of this protocol is simplified and** omits important background information, it should provide a flavour of the basic idea.
* 1. Alice randomly generates a stream of qubits, and sends these as a stream of polarised photons to Bob.
* 2. Bob measures them using a polarisation detector, which will return either a 0 or a 1 for each photon.
* 3. Bob contacts Alice over a conventional authenticated channel (perhaps a secure email, a telephone call, or a cryptographically authenticated channel), and Alice then provides him with information that p**robably results in Bob discarding approximately 50% of the measurements** that he has just taken.
* This is because there are two different types of polarisation detector that Bob can use t measure each photon, and if he chooses the wrong one then the resulting measurement has only a 50% chance of being correct.
* Alice advises him over the authenticated channel which polarisation detector she used to encode each qubit, and Bob throws away the returns of all the wrongly measured photons.
* 4. Alice and Bob now conduct a check over the authenticated channel on the stream of bits that they think they have just agreed upon.
* They do this by randomly choosing some positions and then check to see if they both agree on the bits in these positions.
* If they find no discrepancies then they throw away the bits that were used to conduct the check, and form a key from the bits that they have not yet checked.
* There are a number of substantial limitations of quantum key establishment.
* These include:
* **Distance limitations.** Implementations of quantum key establishment are improving all the time. Nonetheless, it has still only been demonstrated to work over limited distances.
* **Data rates.** There are limits to the rate at which key material can be exchanged over the quantum channel. This is also related to the distance over which the key establishment is being conducted.
* **Cost.** Use of quantum key establishment requires expensive hardware devices and suitable quantum channels.
* **The need for conventional authentication.** Quantum key establishment requires a conventional means of authentication to be used. For example, in the BB84 protocol it is important that Alice and Bob establish an authenticated channel.

**Key storage**

* Secret keys need to be protected from exposure to parties other than the intended ‘owners’.
* It is thus very important that they are stored securely.
* **Avoiding key storage:**
* The best solution of all would be not to store cryptographic keys anywhere and just generate them on the fly whenever they are required.
* This is possible in some applications.
* Since the same key must be generated on the fly every time we need to use it, we require **a deterministic key generator to generate** the key.
* For most applications that use this technique, the **seed is stored** inside the human brain in the form of a passphrase or strong password.

# Key storage in software

### STORING KEYS IN THE CLEAR

* By far the **cheapest, and the riskiest,** approach is to store keys in the **clear in software**.
* In other words, regard keys as pieces of data that are stored on a **hard drive as unprotected data.**
* One common approach is to try to **‘hide’ the keys somewhere in the software**.
* In addition, there are **two fundamental problems** with hiding cryptographic keys in software:

1. The developer who designs the software will know **where the keys are,** so there is **at least one potential attacker** out there who knows where to look for the keys.
2. Assuming that the **hidden keys are specific to different versions** (users) of the software, an attacker who obtains **two versions of the software** could compare them.

* Any locations where differences are noted are potential locations of material relating to a key.

### STORING KEYS USING CRYPTOGRAPHY

* There are really only four options:
* **Encrypt it with yet another key.** So where do we store that key?
* **Generate it on the fly.** This is a fairly common approach and is often taken for applications where a hardware-based solution is not viable.
* **Store it in hardware.** This is probably the most common approach but, obviously, requires access to a suitable hardware device.
* The key encrypting key remains on the hardware device, which is also where all encryption and decryption using this key is performed.
* **Store it in component form.** We introduced the idea of **component form .**
* It can also be used for key storage. By using components we make the task of obtaining a key harder since, in order to **recover the key,** all of the **necessary components need to be obtained**.

# Key storage in hardware

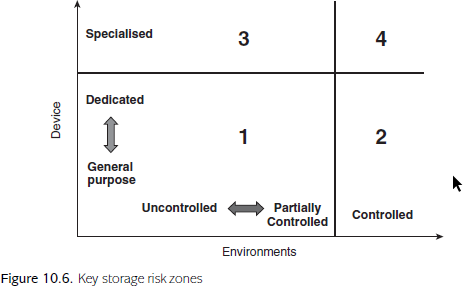
### HARDWARE SECURITY MODULES

* The securest hardware storage media for cryptographic keys are **hardware security modules (HSMs).**
* These **dedicated hardware devices** that provide key management functionality are sometimes known as **tamper-resistant devices**.
* Many HSMs can also perform bulk cryptographic operations, often at high speed.
* Keys stored on HSMs are physically **protected by the hardware**.
* If anyone attempts to penetrate an HSM, for example, to extract a key from the device, tamper-resistant circuitry is triggered and the **key** is normally deleted from the HSM’s memory
* There are various techniques that can be used to provide tamper resistance. These include:
* **Micro-switches.** A simple mechanism that **releases a switch if** an HSM is opened**.** This is not particularly effective, since a clever attacker can always drill a hole and use glue to force the switch off.
* **Electronic mesh.** A fine-gauge electronic mesh that can be attached to the inside of an HSM case. This mesh surrounds the sensitive components. If broken, it activates the tamper-detection circuitry. This mechanism is designed to protect against penetrative attacks, such as drilling.
* **Resin.** A hard substance, such as epoxy resin, that can be used to encase sensitive components. Sometimes electronic mesh is also embedded in resin. Any attempt to drill through the resin, or dissolve the resin using chemicals, will generally damage the components and trigger the tamper-detection circuitry.
* **Temperature detectors**. Sensors that are designed to detect variations in temperature outside the normal operating range.
* **Abnormal temperatures** may be an indication of an attack. For example, one type of attack involves, literally, **freezing the device memory.**
* **Light-sensitive diodes.** Sensors that can be used to **detect penetration or opening of an HSM casing.**
* **Movement or tilt detectors.** Sensors that can detect if somebody is trying to

### physically remove an HSM.

* One approach is to use mercury tilt switches, which interrupt the flow of electrical current if the physical alignment of an HSM changes.
* **Voltage or current detectors.** Sensors that **can detect variations in voltage or current outsid**e the normal operating range.
* Such anomalies may be indication of an attack.
* **Security chips**. Special secure microprocessors that can be used for cryptographic processing within an HSM.
* Even if an attacker has penetrated all the other defences of an HSM, the keys may still remain protected inside the security chip.

# Key storage risk factors

* The risks to key storage media depend not only on the devices on which key are stored, but also on the e**nvironments within which the devices reside.**
*  This relationship is indicated in Figure 10.6, which identifies four zones based on different environmental and device controls.
* The two dimensions depicted in Figure 10.6 represent:
* **Environments,** which range from
* **Uncontrolled:** public environments such as shops and restaurants, where it is not possible to implement strict access control mechanisms;
* **Partially controlled:** environments such as general offices and homes, where it is possible to implement basic access control mechanisms (for example, a physical door key);
* **Controlled:** environments such as high-security offices and military installations, where it is possible to implement strong access control mechanisms (for example, biometric swipe cards).

### Devices, which range from

* **General purpose:** general devices running conventional operating systems with their default in-built security controls (for example, a laptop);
* **Dedicated:** dedicated devices that offer some specialist security controls, such as limited tamper resistance (for example, a point-of-sale terminal or a mobile phone);
* **Specialised:** specialised devices whose main functionality is to provide security (for example, an HSM).
* The four zones identified in Figure 10.6 are mainly conceptual, but illustrate the importance of both dimensions.
* **Zone 1. This is the lowest security zone and thus offers the highest risk.** However, for many applications this may provide sufficient security.
* For example, a key stored in encrypted form on the **hard disk of a home PC may well be good enough for protection of the user’s personal files.**
* **Zone 2. The security offered by Zone 1 devices is increased substantially when** they are moved into a controlled environment. In the extreme, a key stored in the clear in software on a general PC provides excellent security if the PC is not networked and is kept in a physically secure room with an armed guard at the door! More realistically, encrypted keys stored on PCs that are located in an office with strong physical security (such as smart card access control to the rooms) and good network security controls should have better protection than those on a PC located in a public library or an internet cafe.
* **Zone 3. Specialised devices sometimes have to be located in insecure environments** because of the nature of their application.
* A good example is provided by Automated Teller Machines (ATMs), which need to be ‘customer facing’.
* Such devices are thus exposed to a range of potentially serious attacks that are made possible by their environment, such as an attacker attempting to physically remove them with the intention of extracting keys back in a laboratory.

### Zone 4. The highest-security zone is provided when a specialist device is kept in a

controlled environment.

* This is not just the most secure, but the most expensive zone within which to provide solutions.
* This level of security is nonetheless appropriate for important keys relating to high- security applications such as data processing centres, financial institutions, and certification authorities.

# Key backup, archival and recovery

* We have spent most of our discussion about cryptography assuming that the use of cryptography brings security benefits.
* However, there are situations where use of cryptography can potentially have damaging consequences.
* One such situation arises if a cryptographic key becomes ‘lost’. For example:

1. data stored in encrypted form will itself be lost if the corresponding decryption key is lost, since nobody can recover the data from the ciphertext;
2. a digital signature on a message becomes ineffective if the corresponding verification key is lost, since nobody has the ability to verify it.

## KEY BACKUP

* It can be surprisingly easy to ‘lose’ critical cryptographic keys. As we discussed in Section 10.5.3, important keys are often stored on HSMs.
* An obvious attack against a Zone 3 (see Figure 10.6) HSM would be to physically attack the HSM to the point that one of its tamper-resistant triggers is activated and the device wipes its memory.
* The attacker does not learn the keys stored on the device but, without a backup, the potential impact on the organisation relying on the HSM is high.
* Even Zone 4 HSMs are subject to risks such as a careless cleaner bumping into a device and accidentally wiping its memory.

### KEY ARCHIVAL

* Key archival is essentially a special type of backup, which is necessary in situations where cryptographic keys may still be required in the period between their expiry and their destruction.
* Such keys will no longer be ‘live’ and so cannot be used for any new cryptographic computations, but they may still be required. For example:
* There may be a legal requirement to keep data for a certain period of time. If that data is stored in encrypted form then there will be a legal requirement to keep the keys so that the data can be recovered.
* As an illustration, the London Stock Exchange requires keys to be archived for seven years.
* A document that has been digitally signed, such as a contract, may require the capability for that digital signature to be verified well beyond the period of expiry of the key that was used to sign it.

## KEY RECOVERY

* Key recovery is the key management process where a key is recovered from a backup or an archive.
* Technically this is no harder than retrieving a key from any other type of storage, so the challenges all relate to the management processes that surround key recovery.
* Clearly it should not be possible to recover a key unless the recovery is suitably authorised.
* Note that the term ‘key recovery’ is also associated with initiatives to force a ‘mandatory’ backup, also referred to as key escrow.
* The idea behind key escrow is that if **any data is encrypted then a copy of the decryption key is stored (escrowed) by a trusted third party** in such a way that, should it be necessary and the appropriate legal authority obtained, the decryption key can be obtained and used to recover the data.
* Such a situation might arise if the encrypted data is uncovered in the course of a criminal investigation

# Key usage

* Having considered the generation, establishment and storage of cryptographic keys, we now continue our study of the key lifecycle by looking at issues relating to key usage.
* The most important of these is key separation. We will also discuss the mechanics of key change, key activation and key destruction.

# Key separation

* The principle of key separation is that cryptographic keys must only be used for their intended purpose. In this section we consider why key separation is a good idea and discuss options for enforcing it.

### THE NEED FOR KEY SEPARATION

* The problems that can arise if key separation is not enforced can be serious.
* In many applications the need for key separation may be quite obvious.
* For example, it may be the case that encryption and entity authentication are conducted by distinct processes, each with their own particular requirements regarding key lengths.
* when we look at WLAN
* security, where the process for encryption is ‘locked down’ across all applications, but the entity authentication process can be tailored to a specific application environment.
* We illustrate the potential dangers of doing this with two examples.

### Example 1. Like passwords, PINs should not be stored anywhere in the clear.

Hence PINs are often stored in encrypted form using a symmetric PIN encrypting key.

* This key should only ever be used to encrypt PINs. It should never be used to decrypt an encrypted PIN.
* In contrast, a normal symmetric data key is used both for encryption and decryption.
* If these two keys are somehow interchanged within an HSM then we have two serious problems.
* Firstly, it may become possible to decrypt and reveal a PIN.
* Secondly, it may not be possible to recover any normal data encrypted with the PIN encrypting key.

### Example 2. Suppose we have an HSM with the following two security functions:

* Function 1. This generates a four-digit PIN for a payment card by:
* 1. encrypting the card’s 16-digit account number using a PIN generation key, and
* outputting the resulting ciphertext in hex form;
* 2. scanning the hex output for the first four digits in the range 0 to 9, but ignoring any in the range A to F, which are then used to form the PIN (additional measures need to be taken in the unlikely event that there are insufficient digits generated using this process to form a PIN);
* 3. outputting the resulting PIN in encrypted form.
* Function 2. This generates a MAC on some input data by:
* 1. computing a simple CBC-MAC (using the version of CBC-MAC depicted in Figure 6.7, which is not recommended in practice) on the input data using a MAC key;
* 2.outputting the MAC in hex form.
* We illustrate the potential dangers of doing this with two examples.

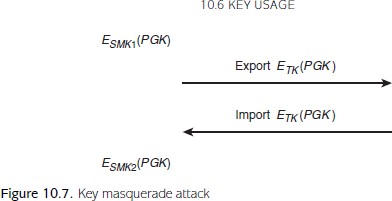
### Example 1. Like passwords, PINs should not be stored anywhere in the clear.

Hence PINs are often stored in encrypted form using a symmetric PIN encrypting key.

* This key should only ever be used to encrypt PINs. It should never be used to decrypt an encrypted PIN.
* In contrast, a normal symmetric data key is used both for encryption and decryption.
* If these two keys are somehow interchanged within an HSM then we have two serious problems.
* Firstly, it may become possible to decrypt and reveal a PIN.
* Secondly, it may not be possible to recover any normal data encrypted with the PIN encrypting key.

### Example 2. Suppose we have an HSM with the following two security functions:

* Function 1. This generates a four-digit PIN for a payment card by:
* 1. encrypting the card’s 16-digit account number using a PIN generation key, and
* outputting the resulting ciphertext in hex form;
* 2. scanning the hex output for the first four digits in the range 0 to 9, but ignoring any in the range A to F, which are then used to form the PIN (additional measures need to be taken in the unlikely event that there are insufficient digits generated using this process to form a PIN);
* 3. outputting the resulting PIN in encrypted form.
* Function 2. This generates a MAC on some input data by:
* 1. computing a simple CBC-MAC (using the version of CBC-MAC depicted in Figure 6.7, which is not recommended in practice) on the input data using a MAC key;
* 2.outputting the MAC in hex form.



* one method of enforcing key separation in an HSM is to store keys in the HSM encrypted under a master key that is specific to one usage purpose.
* In this way, access to the key is directly linked to the use of a master key that identifies the key usage purpose.
* However, many HSMs have export and import functions that allow keys to be transferred between different HSMs.
* Keys are encrypted using a transport key during export and import.
* Figure 10.7 shows how this facility could, potentially, be used to change the apparent usage purpose of a key.
* 1. A PIN generation key PGK is stored on the HSM, encrypted by a storage master key SMK1, which is the local key on the HSM that is used to store PIN generation keys.
* 2. The HSM is instructed to export PGK. It thus decrypts the encrypted PGK using SMK1, then re-encrypts PGK using the transport key TK. This is then exported.
* 3. The HSM is then instructed by the attacker to import a new MAC key. The attacker submits PGK, encrypted under TK.
* 4. The HSM decrypts the encrypted PGK using TK, then re-encrypts it using storage master key SMK2, which is the HSM key used to store MAC keys. The HSM thus now regards PGK as a MAC key.

### ENFORCING KEY SEPARATION

* In order to avoid some of the problems that we have just illustrated, mechanisms are required to enforce key separation.
* This can be regarded as part of the wider provision of assurance of purpose of keys There are two main techniques that can be used to enable the purpose of a key to be distinguished:
* **Encrypting a key using a specified variant key.** This is a hardware-enforced method that we previously mentioned, which involves using specific higherlevel keys to encrypt keys for particular purposes.
* For example, in Figure 10.7 the HSM used the key SMK1 to encrypt PIN generation keys, and key SMK2 to encrypt MAC keys.
* The HSM can interpret the usage based on the key encrypting key variant used
* **Embedding the key in a larger data block.** This involves embedding the key into a larger data object that also includes a statement on the key usage. Three examples of this are:
* **Employing redundancy.** As discussed in Section 4.4, a DES key has an effective length of 56 bits, but is usually a 64-bit value.
* Thus, there are 8 ‘spare’ bits that can be used for other purposes.
* The original DES standard recommends that the spare bits be used to provide error detection in the event that a DES key becomes corrupted.
* **Key blocks.** This is a formatted data string that allows a key to be represented along with other data relating to the key. One example is the ANSI TR-31 key block, which is depicted in Figure 10.8 and has the following fields:
* the header includes information that clarifies the purpose of the key;
* the optional header includes optional data such as the expiry date of the key;
* the key is encrypted using a suitable **key encrypting key;**
* the authenticator is a MAC on the rest of the key block, which provides data origin authentication (data integrity) of the key block data.
* **Public-key certificates.** These are types of key block used to provide assurance of purpose for public keys.
* A public-key certificate often includes a field that



# Key change

* The need for a change of key tends to arise in two different types of circumstance:

### Planned key changes.

* These will most likely occur at regular intervals.
* One reason for a planned key change might be the end of the key.
* Another reason might simply be to regularly practice key change procedures in preparation for an unplanned key change.

### Unplanned key changes.

* These may occur for a variety of reasons.
* Indeed, many of the reasons that we gave in Section 10.2.1 for having finite key lifetimes were to mitigate against unplanned events.
* An unplanned key change may thus be required if these unplanned events actually occur. For example:
* a key is compromised;
* a security vulnerability becomes apparent with the potential to lead to key compromise (such as an operating system vulnerability, a breakthrough in cryptanalysis, or a failure of a tamper-resistance mechanism in an HSM);
* an employee unexpectedly leaves an organisation.

### IMPACT OF KEY CHANGE

* Key change can be a very expensive process, depending on the importance of the key being changed.
* An unplanned key change is particularly problematic, especially in the event of a key compromise, since it raises questions about any cryptographic operations that were conducted using the affected key, such as the confidentiality of any encrypted data.
* The minimum impact of a key change is that a new key needs to be generated and established.
* However, the impact can be severe, especially in the case of high-level key compromise.

### MECHANISMS FOR CHANGING KEYS

* As mentioned above, key change requires:
* generation and establishment of a new key;
* withdrawing the old key (and potentially destroying or archiving it).
* planned key changes should happen automatically and require very little intervention.
* For example, that UKPT schemes automate planned key changes after every transaction.
* More intervention may be required in the case of unplanned key changes.

### CHANGING PUBLIC-KEY PAIRS

* This is ‘surprising’ because key change forces a new key establishment operation, which is usually a more difficult process for symmetric keys.
* There are two reasons why changing public-key pairs is normally more challenging:
* **Knowledge of public keys.** Since symmetric keys need to be carefully ‘positioned’ in a network so that entities relying on them have the right keys, a key management system tends to be fully ‘in control’ of where its symmetric keys are located.
* **Open application environments.** Symmetric cryptography tends to be employed in closed environments.
* Thus any key management system handling symmetric keys should have mechanisms and controls in place for key establishment that can be reused for key change.

# Key activation

* When assessing the security of any key management system, it is important to pay attention to the processes by **which keys are activated**, by which we mean that their use is ‘authorised’.
* We observed in Section 8.3.3 that one problem with using identity information based on a cryptographic key for entity authentication can be that the effective security is not as strong as expected.
* This problem arose because in the scenario under discussion the key was activated by a less-secure mechanism, such as a password.
* As an example, consider a signature key stored on a computer for digitally signing emails.
* If RSA is being used then this signature key might, reasonably, be up to 2048 bits long, which is clearly a value that the human owner of the key will not be capable of memorising.
* When the user decides to digitally sign an email, the user needs to instruct the email client to activate their signature key.
* Several scenarios may now apply, depending on how the key is stored (if at all) on the computer. These include:
* **Key stored on the computer in the clear.** In this case the user might activate the key simply by entering an instruction, perhaps selecting the key from a list of potential keys stored on the computer.
* Key activation is thus possible for anyone with access to the computer.
* **Key stored on the computer in encrypted form.** The user might activate the key in this case by being prompted to provide some secret identity information, such as a passphrase.
* This passphrase would then be used to generate the key that can be used to recover the signature key.
* In this case the effective security is linked to the security of the passphrase.
* **Key generated on the fly.** In this case the key is not stored on the computer, but is generated on the fly.
* The activation of the key is thus linked to the generation of the key.
* Again, one way of implementing this is to request some identity information such as a passphrase from the user.
* Thus the effective security of the key is again determined by the security of this passphrase.
* **Key stored off the computer.** Another option is that the key is stored on a peripheral device.
* The key activation takes place when the user connects the device to the computer.
* In this case the effective security is linked to the security of the peripheral device.
* This process may also require a passphrase to be used.

# Key destruction

* When a key is no longer required for any purpose then it must **be destroyed in a secure manner**. The point at which key destruction is required may either be:

1. when the key expires (the natural end of the key’s lifetime);
2. when the key is withdrawn (before its expiry, in the event of unplanned events such as those discussed in Section 10.6.2);
3. at the end of a required period of key archival.

* Since keys are a special type of data, the mechanisms available for destroying keys are precisely those for **destroying general data.**
* Since keys are sensitive data, **secure mechanisms** must be used.
* Suitable techniques are sometimes referred to as data erasure or data sanitisation mechanisms.

# Governing key management

* key management is the main interface between the technology of cryptography and the users and systems that rely on it.
* To this extent, key management is a small, but important, part of the wider management of the security of an information system.

# Key management policies, practices and procedures

* Within an organisation, the most common way to govern key management is through the specification of:
* **Key management policies**. These define the overall requirements and strategy for providing key management. For example, a policy might be that all cryptographic keys are stored only in hardware.
* **Key management practices.** These define the tactics that will be used in order to achieve the key management policy goals. For example, that all devices using cryptography will have an in-built HSM.
* **Key management procedures**. These document the step-by-step tasks necessary in order to implement the key management practices.
* For example, the specification of a key establishment protocol that will be used between two devices.

## Example procedure: key generation ceremony

* We illustrate the **potential complexities of key management governance** by giving an example of an important type of key management procedure that might be required by a large organisation. This is that of a key ceremony.
* Note that the key in question could be a top-level (master) symmetric key or top-level (root) private key, which needs to be installed into an HSM. The key might be:
* a new key being freshly generated;
* an existing key being re-established (from backed-up stored components).
* The participants are:
* **Operation manager:** responsible for the physical aspects, including the venue, hardware, software and any media on which components are stored or transported;
* **Key manager**: responsible for **making sure that the key ceremony is performed in accordance** with the relevant key management policies, practices and procedures;
* **Key custodians: the parties** physically in possession of the key components, responsible for handling them appropriately and following the key ceremony as instructed;
* **Witnesses:** responsible for observing the key ceremony and **providing independent assurance** that all other parties perform their roles in line with **the appropriate policies**, practices and procedures (this might involve recording the key ceremony).
* The key ceremony itself involves several phases:
* **Initialisation.** The operation manager installs and configures the required hardware and software, including the HSM, within a controlled environment.
* This process **might need to be recorded by witnesses**.
* **Component retrieval.** The components required for the key ceremony, held by the

**relevant key custodians**, are transported to the **key ceremony location.**

* These key custodians may be from different organisations (departments) and **may not be aware of each others’ identities**.
* **Key generation/establishment.** The key is installed onto the HSM under the guidance of **the key manager.**
* This process will involve the various key custodians taking part in the key ceremony, but not necessarily simultaneously.
* **Validation.** If necessary, **following the completion of the key ceremony**, the official record can be scrutinised to validate that the correct procedure was followed (perhaps as part of an audit).

**Chapter 11 Public-Key Management**

## Certification of public keys

* Suppose that Bob receives a digitally signed message that claims to have been signed by Alice and that Bob wants to verify the digital signature.
* As we know this requires Bob to have access to Alice’s verification key.
* Suppose that Bob is presented with a key (we do not concern ourselves with how this is done) that is alleged to be Alice’s verification key.
* Bob uses this key to ‘verify’ the digital signature and it appears to be correct.
* What guarantees does Bob have that this is a valid digital signature by Alice on the message?
* Here are some questions that Bob would be strongly advised to consider, especially if the digital signature is on an important message:
* Does the verification key actually belong to Alice?
* Could Alice deny that this is her verification key?
* Is the verification key valid?
* Is the verification key being used appropriately?

### PROVIDING ASSURANCE OF PURPOSE

* 1. provide a ‘strong association’ between a public key and the owner of that key
* (the entity whose identity is linked to the public key);
* 2. provide a ‘strong association’ between a public key and any other relevant data
* (such as expiry dates and usage restrictions).

### USING A TRUSTED DIRECTORY

* Perhaps the crudest approach to providing assurance of purpose for public keys is to use a trusted ‘directory’, which lists all public keys next to their related data (including the name of the owner).
* Anyone requiring assurance of purpose of a public key, simply looks it up in the trusted directory
* While this approach may suffice for some applications of public-key cryptography, there are several significant problems:
* **Universality.** The directory has to be trusted by all users of the public-key management system.
* **Availability.** The directory has to be online and available at all times to users of the public-key management system.
* **Accuracy.** The directory needs to be maintained accurately and **protected from unauthorised modification.**

## Public-key certificates

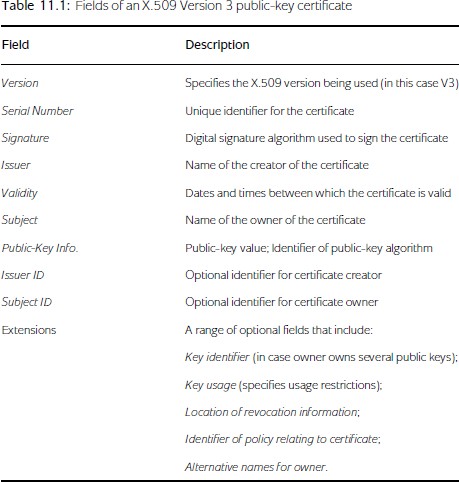
* A public-key certificate is data that binds a public key to data relating to the assurance of purpose of that public key.
* It can be thought of as a trusted directory entry in a sort of distributed database.

### CONTENTS OF A PUBLIC-KEY CERTIFICATE

* A public-key certificate contains four essential pieces of information:
* **Name of owner.** The name of the owner of the public key. This owner could be a person, a device, or even a role within an organisation. The format of this name will

depend upon the application, but it should be a unique identity that identifies the owner within the environment in which the public key will be employed.

* **Public-key value. The public key itself**. This is often accompanied by an identifier of the cryptographic algorithm with which the public key is intended for use.
* **Validity time period.** This identifies the date and time from which the public key is valid and, more importantly, the date and time of its expiry.
* **Signature.** The creator of the public-key certificate digitally signs all the data that forms the **public-key certificate,** including the **name of owner, public-key value and validity time period.** This digital signature not only binds all this data together, but is also the **guarantee that the creator of the certificate believes that all the data is correct.**



### INTERPRETING A PUBLIC-KEY CERTIFICATE

* public-key certificate binds the assurance-of purpose data relating to a public key to the public-key value, but does nothing more than this. In particular:
* A public-key certificate cannot be used to encrypt messages or verify digital Signatures
* A public-key certificate is not a proof of identity.

### PUBLIC-KEY CERTIFICATE CREATORS

* A creator of a public-key certificate is referred to as a **certificate authority (CA).**

The certificate authority normally plays three important roles:

* **Certificate creation.** The CA takes responsibility for ensuring that the information in a public-key certificate is correct before creating and signing the public-key certificate, and then issuing it to the owner.
* **Certificate revocation.** The CA is responsible for revoking the certificate in the event that it becomes invalid.
* **Certificate trust anchor.** The CA acts as the point of trust for any party relying on the correctness of the information contained in the public-key certificate. To fulfil this role, the CA will need to actively maintain its **profile as a trusted organisation.**

# The certificate lifecycle

* Differences in the certificate lifecycle
* We now recall the main phases of the key lifecycle from Figure 10.1 and comment on where differences lie:
* **Key generation.** This is one of the phases which differs significantly. The generation of a public-key pair is an algorithm-specific, and often technically complex, operation.
* Having done this, creation of a public-key certificate is even harder from a process perspective since it involves determining the validity of information relating to the public key.
* **Key establishment.** Private key establishment is potentially easier than symmetric key establishment since the private key only needs to be established by one entity.
* If another entity generates the private key then private key establishment may involve the private key being distributed to the owner using a secure channel of some sort, such as physical **distribution of a smart card** on which the private key is installed.
* Public-key certificate establishment is not a sensitive operation, since the public-key certificate does not need to be kept secret. Most techniques can either be described as:
* **Pushing a public-key certificate**, meaning that the owner of the public-key certificate provides the certificate whenever it is required by a relying party.
* **Pulling a public-key certificate**, meaning that relying parties must retrieve public- key certificates from some sort of repository when they first need them.
* **Key storage, backup, archival**. They are all **less-sensitive operations** when applied to public-key certificates.
* **Key usage.** The principle of key separation, applies equally to public-key pairs. Many public-key certificate formats, such as the X.509 Version 3 certificate format depicted in Table 11.1, include fields for specifying key usage.
* **Key change.** This is the other phase of the key lifecycle that differs significantly for public-key pairs. We identified why this is the case in Section 10.6.2 and we will discuss potential techniques for facilitating key change in Section 11.2.3.
* **Key destruction**. Destruction of private keys is covered by Section 10.6.4.
* Destruction of public-key certificates is less sensitive, and may not even be required.

# Certificate creation

* LOCATION OF KEY PAIR AND CERTIFICATE CREATION It is important to be aware of the fact that we are dealing with two separate processes here:
* generation of the public-key pair itself;
* creation of the public-key certificate.
  + Key pair generation can be performed either by the owner of the public-key pair or a trusted third party (who may or may not be the CA).
  + **Trusted third party generation.** In this scenario, a trusted third party (which could be the CA) generates the public-key pair.
  + If this trusted third party is not the CA then they must contact the CA to arrange for certificate creation. The

advantages of this approach are that

* the trusted third party may be better placed than the owner to conduct the relatively complex operations involved in generation of the public-key pair (see Section 10.3.4);
* the key pair generation process **does not require the owner to do anything.**
  + **The possible disadvantages are that:**
* the owner needs to trust the third party to securely distribute the private key to the owner; the only exception to this is if the private key is managed on behalf of the owner by the trusted third party, in which case processes must exist for securely managing ‘access’ to the private key when the owner needs to use it.
* the owner needs to trust the third party to destroy the private key after it has been distributed to the owner; an exception to this would be if the third party provides a backup and recovery service for the owner
  + **Combined generation.** In this scenario, the owner of the key pair generates the public-key pair. The owner then submits the public key to a CA for generation of the public-key certificate.
  + The main advantages of this approach are that:
  + • the owner is in full control of the key pair generation process;
  + • the private key can be locally generated and stored, without any need for it to be distributed.
  + The possible disadvantages of this approach are that:
  + • the owner is required to have the ability to generate key pairs;
  + • the owner may need to demonstrate to the CA that the owner knows the private key that corresponds to the public key submitted to the CA for certification.
  + **Self-certification.** In this scenario, the owner of the key pair generates the key pair and certifies the public key themselves. This approach is certainly
  + simple.Examples of situations
  + where this might be the case are:
* the owner is a CA; it is not uncommon for CAs to self-certify their own public-keys, which is an issue that we will discuss in a moment;
* all relying parties have an established relationship with the owner and hence trust the owner’s certification; for example, a small organisation using a self-certified public key to encrypt content on an internal website.

**REGISTRATION OF PUBLIC KEYS**

* + If either trusted third-party generation or combined generation of a public key pair is undertaken then the owner of the public-key pair must engage in **a registration process with the CA** before a public-key certificate can be issued.
  + This is when the **owner presents their credentials** to the CA for checking.
  + In many application environments a separate entity known as a **Registration Authority (RA)** performs this operation. The roles of RA and CA can be separated for several reasons:
* **Registration** involves a distinct set of procedures that generally require an **amount of human intervention,** whereas certificate creation and issuance can be automated.
* Checking the credentials of a public-key certificate applicant is often the most **complex part** of the certificate creation process.

### PROOF OF POSSESSION

* + If a public key and its certificate are created using **combined generation** then, strictly speaking, **it is possible for an attacker to attempt to register a public key** for which they do not know the corresponding private key.
  + Such an ‘attack’ on a verification key for a digital signature scheme might work as follows:

1. The attacker obtains a copy of Alice’s verification key. This is a public piece of information, so the attacker can easily obtain this.
2. The attacker presents Alice’s verification key to an RA, along with the attacker’s legitimate credentials.
3. The RA verifies the credentials and instructs the associated CA to issue a publickey certificate in the name of the attacker for the presented verification key.
4. The CA issues the public-key certificate for the verification key to the attacker.
   * This attack can be prevented if the CA conducts a simple check that the public key certificate applicant knows the corresponding **private key.**
   * This type of check is often referred to as **proof of possession** (of the corresponding private key).
   * If the public key is an encryption key then one possible proof of possession is as follows:
5. The RA encrypts a test message using the public key and sends it to the certificate applicant, along with a request for the applicant to decrypt the resulting ciphertext.
6. If the applicant is genuine, they decrypt the ciphertext using the private key and return the plaintext test message to the RA.

An applicant who does not know the corresponding private key will not be able to perform the decryption to obtain the test message.

### GENERATING CA PUBLIC-KEY PAIRS

* + Public-key certificates involve a CA digitally signing the owner’s public key together with related data. This in turn requires the CA to possess a public-key pair.
  + The two most common methods of certifying the CA’s verification key are:
  + **Use a higher-level CA. If** the CA is part of a chain of Ca’s then the CA may choose to have their public key certified by another CA. Of course, this does not address the question of who certifies the public key of the higher-level CA.
  + **Self-certification.** A top-level CA probably has no choice other than self- certification**.** It may suffice that this process involves making the public key available in high-profile media, such as daily newspapers

# Key pair change

* + **REVOCATION(officially cancelling)** OF PUBLIC-KEY CERTIFICATES
  + Revoking a public key essentially means revoking the public-key certificate.
  + With this in mind, it is worth observing that there may be situations where a public- key certificate needs to be revoked and then a **new public-key certificate created** for the same public-key value.

### REVOCATION TECHNIQUES

* + Revocation of public-key certificates can only realistically be approached in one
  + of three ways:
  + **Blacklisting.** This involves maintaining a database that contains serial numbers of public-key certificates that **have been revoked.**
  + This type of database is often referred to as a c**ertificate revocation list (or CRL)**.
  + These CRLs need to be maintained carefully, normally by the **CA who is responsible for issuing the certificates**, with clear indications of how often they are updated.
  + **Whitelisting.** This involves maintaining a database that contains serial numbers of public-key certificates that are valid.
  + This database can then be queried by a relying party to find out if a **public-key certificate is valid.**
  + **Rapid expiration.** This removes the need for revocation by allocating very **short**

### lifetimes to public-key certificates.

**Public-key management models**

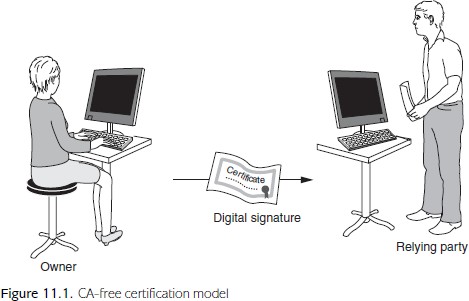
* + Choosing an organisation to play the role of a CA in an open environment is less straightforward the importance of their role may demand tighter regulation of their practices.
  + Options for this include:
  + **Licensing.** This approach requires CAs to obtain a government license before they can operate.
  + Government, thus, ultimately provides the assurance that a CA conforms to minimum standards.
  + **Self-regulation.** This approach requires CAs to form an **industry group** and set **their own minimum operational standards** through the establishment of bestpractices.

# Public-key certificate management models

1. CA-FREE CERTIFICATION MODEL
   * The CA-free certification model is depicted in Figure 11.1 and applies when there is
   * no CA involved. In the CA-free certification model**, the owner generates a key pair**

and then either **self-certifies the public key** or does not use a public-key certificate.

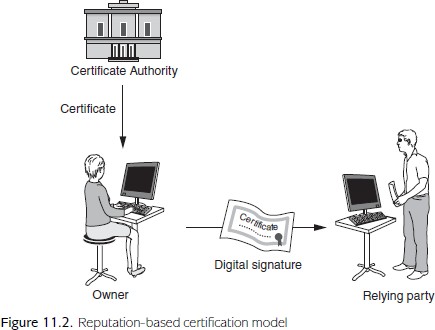
* + Any relying party obtains the (self-certified) public key directly from the owner.

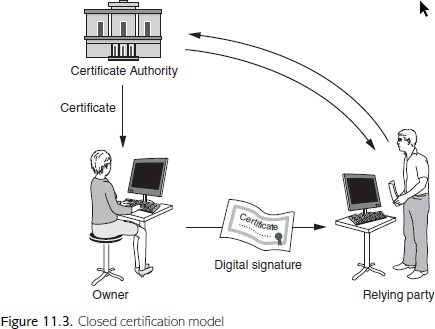


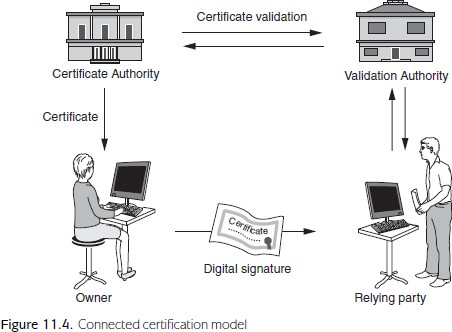
1. REPUTATION-BASED CERTIFICATION MODEL

* The reputation-based certification model is depicted in Figure 11.2 and applies when the owner has obtained a public-key certificate from a CA, but the relying party has **no relationship with this CA.**
* for example, it is a well-known organisation or **trust service provider**, then the

**relying party** might be **willing to accep**t the information in the public-key certificate.



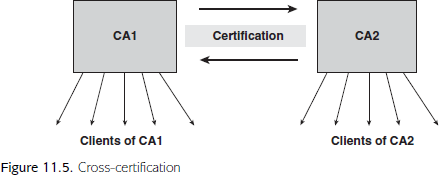
1. CLOSED CERTIFICATION MODEL
   * The closed certification model is depicted in Figure 11.3 and applies when the relying party **has a relationship with the owner’s CA**.
   *  The closed certification model is the most ‘natural’ certification model, but is only really applicable to closed environments where a single CA oversees the management of all public-key certificates.
2. CONNECTED CERTIFICATION MODEL
   * The connected certification model is depicted in Figure 11.4 and applies when the relying party has a relationship with a trusted third party, which in turn has a relationship with the owner’s CA.
   * The trusted third party that the relying party has a relationship with could be another CA.
   * In Figure 11.4 we describe it as a **validation authority** because its **role is to assist the relying party to validate the information** in the owner’s public-key certificate.
   * Strictly speaking, this validation authority may not necessarily be a CA.



# Joining CA domains

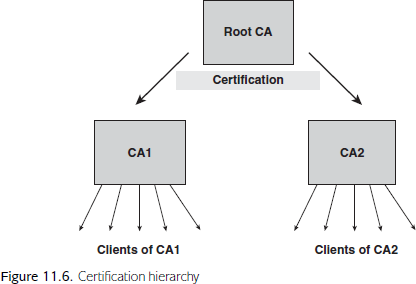
* + consider the nature of the relationship between CA1 and CA2.
  + In particular, we will look at techniques for ‘joining’ their respective CA domains and allowing certificates issued by CA1 t be ‘trusted’ by relying parties who have trust relationships with CA2.
  + CROSS-CERTIFICATION
  + The first technique for joining two CA domains is to use cross-certification, whereby each CA certifies the other CA’s public key.

1. I (Bob) trust CA2 (because I have a business relationship with CA2);
2. CA2 trusts CA1 (because they have agreed to cross-certify one another);
3. CA1 has vouched for the information in Alice’s public-key certificate (because CA1 generated and signed it);
4. Therefore, I (Bob) trust the information in Alice’s public-key certificate.

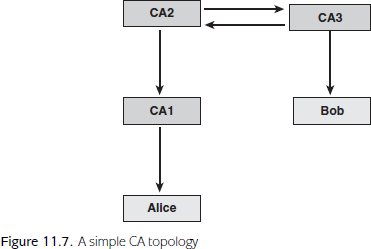


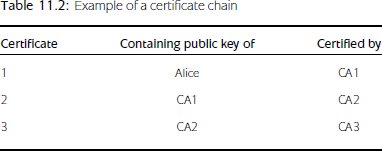
* + CERTIFICATION HIERARCHIES
  + The second technique, is to use a certification hierarchy consisting of different levels of CA.
  + A higher-level CA which both CA1 and CA2 trust, can then be used to ‘transfer’ trust from one CA domain to the other can do so by means of the following argument:

1. I (Bob) trust CA2 (because I have a business relationship with CA2);
2. CA2 trusts root CA (because CA2 has a business relationship with root CA);
3. Root CA has vouched for the information in CA1’s public-key certificate (because root CA generated and signed it);
4. CA1 has vouched for the information in Alice’s public-key certificate (because CA1 generated and signed it).
5. Therefore, I (Bob) trust the information in Alice’s public-key certificate.



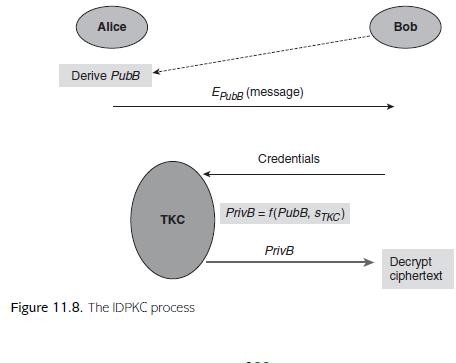
* + CERTIFICATE CHAINS
  + The joining of CA domains makes verification of public certificates a potentially
  + complex process. In particular, it results in the creation of certificate chains, which
  + consist of a series of public-key certificates that must all be verified in order to
  + have trust in the end public-key certificate.





### Alternative approaches

* + Webs of trust
  + that public keys could be made available directly by owners to relying parties without the use of a CA.
  + The problem with this approach is that the relying party is left with no trust anchor other than the owner themselves.
  + A stronger assurance can be provided if a web of trust is implemented.
  + Suppose that Alice wishes to directly provide relying parties with her public key.
  + THE IDEA BEHIND IDPKC
  + One way in which this binding could be built in is if the public-key value can be
  + uniquely derived from the identity. And one way in which this can be done is to
  + make the public-key value and the identity the same value. This is the motivation
  + behind identity-based public-key cryptography (IDPKC).
  + A significant difference between IDPKC and certificat
  + A MODEL FOR IDPKC ENCRYPTION
  + Figure 11.8 shows the process behind using IDPKC to encrypt a message from
  + Alice to Bob. The model consists of the following stages:



### Encryption. Alice derives Bob’s public key PubB from Bob’s identity using the

* + publicly known rules. Alice then encrypts her message using PubB and sends
  + the resulting ciphertext to Bob.

### Identification. Bob identifies himself to the TKC by presenting appropriate

* + credentials and requests the private key PrivB that corresponds to PubB.

### Private key derivation. If the TKC accepts Bob’s credentials then the TKC

* + derives PrivB from PubB and a system secret value sTKC, known only by
  + the TKC.

### Private-key distribution. The TKC sends PrivB to Bob

* + **Decryption. Bob decrypts the ciphertext using PrivB.**

**Decryption. Bob decrypts the ciphertext using PrivB.**

* + **Decryption.** Bob decrypts the ciphertext using PrivB.
  + **Encryption. Alice** derives Bob’s public key PubB from Bob’s identity using the publicly known rules. Alice then encrypts her message using PubB and sends the resulting ciphertext to Bob.

### Identification. Bob identifies himself to the TKC by presenting appropriate

credentials and requests the private key PrivB that corresponds to PubB.

### Private key derivation. If the TKC accepts Bob’s credentials then the TKC

derives PrivB from PubB and a system secret value sTKC, known only by the TKC.

### Private-key distribution. The TKC sends PrivB to Bob using a secure channel.

* + **Decryption. Bob decrypts the ciphertext using PrivB.**

**IDPKC ENCRYPTION ALGORITHMS**

* + - The most important issue regarding algorithms for IDPKC is that *conventional public-key cryptosystems cannot be used for IDPKC*. There are two principal reasons for this:

1. In conventional public-key algorithms, such as RSA, it is not possible for *an* value to be a public key. Rather, a public key is a value that satisfies certain specific mathematical properties. Given an arbitrary numerical identity of a public-key owner, it is unlikely that this corresponds to a valid public key (it *might*, but this would be lucky, rather than expected).
2. Conventional public-key algorithms do not feature a system secret *sTKC* that can be used to ‘unlock’ each private key from the corresponding public key.

### PRACTICAL ISSUES WITH IDPKC

While IDPKC directly solves some of the problems associated with certificate based public-key cryptography, it results in some new issues. These include:

**The need for an online, centrally trusted TKC**. There is no getting around this requirement, which immediately restricts IDPKC to applications where the existence of such a trusted entity is acceptable and practical. Note in particular that:

* the TKC should be *online*, since it could be called upon at any time to establish private keys;
* only the TKC can derive the private keys in the system, hence it provides a source of key escrow, which can either be seen as desirable or undesirable (see Section 10.5.5);
  + the TKC requires secure channels with all private key owners, with the similar advantages and disadvantages as discussed for trusted third-party generation of key pairs in Section 11.2.2, except that the TKC cannot ‘destroy’ the private
  + keys as it always has the ability to generate them. Nonetheless, there are many potential applications where the existence of such a TKC is reasonable, especially in closed environments.

**Revocation issues**. One of the problems with tying a public-key value to an identity is the impact of revocation of a public key, should this be required. If the public key has to change, we are essentially requiring the owner’s ‘identity’ to change, which clearly might not be practical. An obvious solution to this is to introduce a temporal element into the owner’s ‘identity’, perhaps featuring a time period for which the public key is valid.

**Multiple applications**. Another issue is that it is no longer immediately obvious how to separate different applications. In conventional public-ke cryptography, one owner can possess different public keys for different applications.

### MORE GENERAL NOTIONS OF IDPKC

One of the most promising extensions of the IDPKC idea is to associate public keys in an IDPKC system with *decryption policies*. The idea involves only a slight modification of the process described in Figure 11.8:

**Encryption**. Alice derives a public key *PubPolicy* based on a specific decryption policy using publicly known rules. For example, this policy could be *Qualified radiographer working in a UK hospital*. Alice then encrypts her message (say, a health record) using *PubPolicy* and (continuing our example) stores it on a medical database, along with an explanation of the decryption policy.

**Identification**. Qualified UK radiographer Bob, who wishes to access the health record, identifies himself to the TKC by presenting appropriate medical credentials and requests the private key *PrivPolicy* that corresponds to *PubPolicy*.

**Private-key derivation**. If theTKCaccepts Bob’s credentials then theTKCderives *PrivPolicy*

from *PubPolicy* and a system secret value *sTKC*, known only by the TKC.

**Private-key distribution**. The TKC sends *PrivPolicy* to Bob using a secure channel.

**Decryption**. Bob decrypts the cipher text using *PrivPolicy*.