KEY MANAGEMENT

10.1 Key management fundamentals

10.1.1 What is key management?

- Key management is the *secure administration of cryptographic keys*.
- Cryptographic keys are 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. Many key management systems rely on manual processes.

10.1.2 The key lifecycle

The main phases of the key lifecycle are depicted in the Figure.

Key generation concerns the creation of keys.

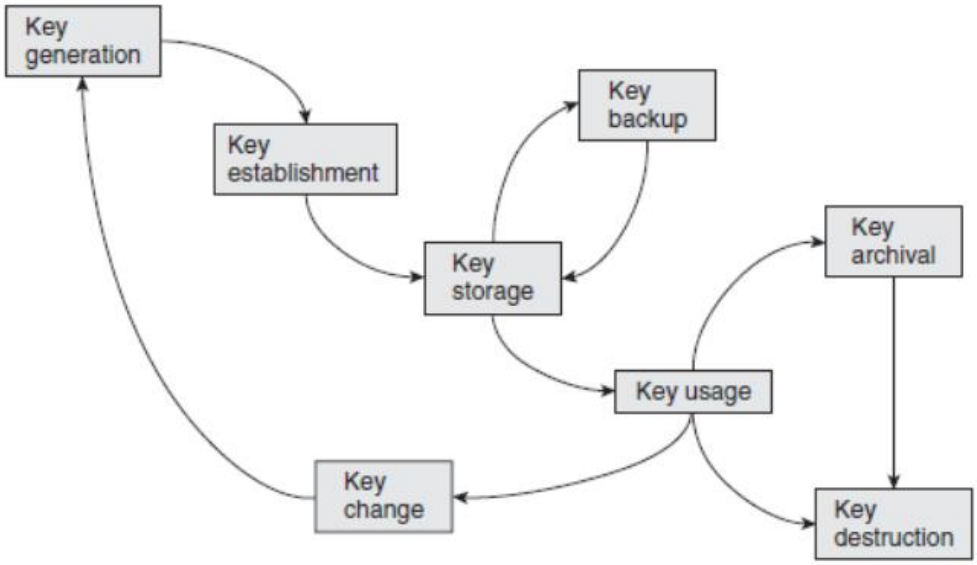
Key establishment is the process of making sure that keys reach the end points where they will be used.

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, ***key archival***.

Key usage is about how keys are used.

key change. How a key's life ends in ***key destruction***.



The key lifecycle

10.1.3 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.

Example:

- if a weak key generation mechanism is used then it might be possible to determine information about a secret key more easily than intended;
- secret keys are vulnerable 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.

This 'purpose', may include :

• information concerning which entities are associated with the key

• 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.

10.1.4 Key management systems

A 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. 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 .

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.

10.2 Key lengths and lifetimes

- ➢ Longer keys are better from a security perspective.
- ➢ A cryptographic computation normally takes more time if the key is longer. In addition, longer keys involve greater storage and distribution overheads.

10.2.1 Key lifetimes

- ➢ 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. At this point it may need to be *archived* or perhaps *destroyed.*

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**. Consider 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*.**

10.3 Key generation

Key generation processes for symmetric and public-key cryptography are fundamentally different.

10.3.1 Direct key generation

- ➢ Symmetric keys are just randomly generated numbers (normally bit strings).
- ➢ Method for generating a cryptographic key is to randomly generate a number, or more commonly a pseudorandom number.
- ➢ The choice of technique will depend on the application.

For example, use of a hardware-based non-deterministic generator might be appropriate for a master key, whereas a 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.

10.3.2 Key derivation

The term *key derivation* is the generation of cryptographic keys from other cryptographic keys or secret values.

Advantages of deriving keys from other keys:

1. **Efficiency**. Generating and establishing one key (sometimes called a *base key*), and then using it to derive many keys.

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
- ➢ It is good practice to make sure that the keys used for each of these mechanisms are different. 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:

$$K_1 = h(K||0) \text{ and } K_2 = h(K||1),$$

where h is a hash function.

2. **Longevity**:

- 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.
- 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.

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.

10.3.3 Key generation from components

For extremely 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*.

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. The generation of components should be performed as securely as possible. Let us denote the resulting components by K_A, K_B and K_C , respectively.
2. 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. 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 i.e :

$$\boldsymbol{K = K_A \oplus K_B \oplus K_C}\ .$$

- Thus Alice, Bob and Charlie are able to jointly generate a key in such a way that all three of their components are necessary for the process to complete.
- If only two of the components are present then no information about the key can be derived, even if the components are combined.

10.3.4 Public-key pair generation

Key generation for public-key cryptography is algorithm-specific.

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, 2^{1024} candidates for *e* or *d*. However, only some of these 2^{1024} numbers *can* be an *e* or a *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,
- 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.

10.4 Key establishment

Key establishment is the process of getting cryptographic keys to the locations where they will be used.

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..

The key does not need to be secret. This applies mainly to public keys.

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 termed *key predistribution*.

10.4.1 Key hierarchies

- *key hierarchy* is a technique used for managing symmetric keys.
- 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 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.

- The risk of a key being compromised, keys that are directly used to perform cryptographic computations, such as encryption of transmitted data.
- Use of a key hierarchy makes it easy to change these low-level keys without the need to replace the high-level keys, which are expensive to establish.

A SIMPLE KEY HIERARCHY

The idea of a key hierarchy is illustrated considering a simple example which is shown in the Figure. 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.

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 corresponds to the lifetime a single session, hence data keys are often referred to as *session keys*.

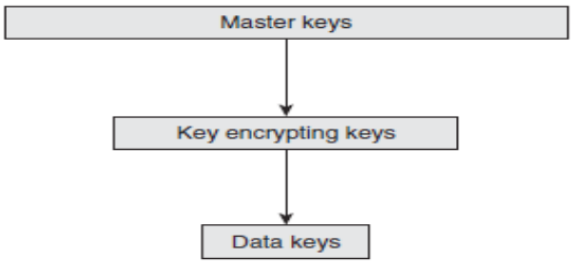


Figure 10.2. A three-level key hierarchy

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.

- 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

A key hierarchy works for a relatively simple network, but quickly becomes unmanageable for large networks.

Problem: Consider a simple two-level hierarchy consisting of only master and data keys. If there is a network of n users, then the number of possible pairs of users is 12 $n(n - 1)$. This means that, if there are 100 users then there are $12 \times 100 \times 99 = 4950$ 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.

Alternatively, install the same master key in all HSMs. Data keys for communication between Alice and Bob could then be derived 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.

Solution:

- It is common to deploy the services of a trusted third party, whom all the users trust, 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.

Consider a simple scenario.

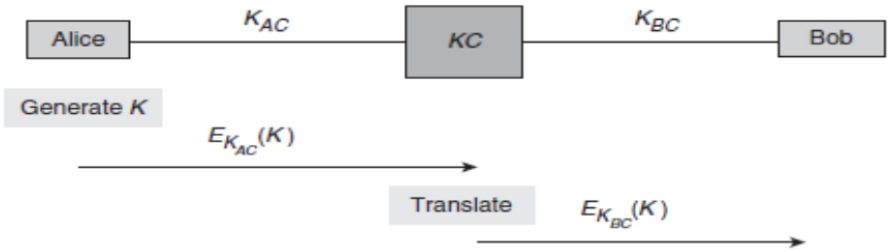
In each case let us assume that Alice wishes to establish a shared data key K with Bob, also assume that both Alice and Bob have respectively established master keys K_{AC} and K_{BC} with the KC, and that a simple two-level key hierarchy is being employed.

The two approaches are:

- ***Key translation*.**
- ***Key despatch*.**

***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*. This process is depicted in the below figure:

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



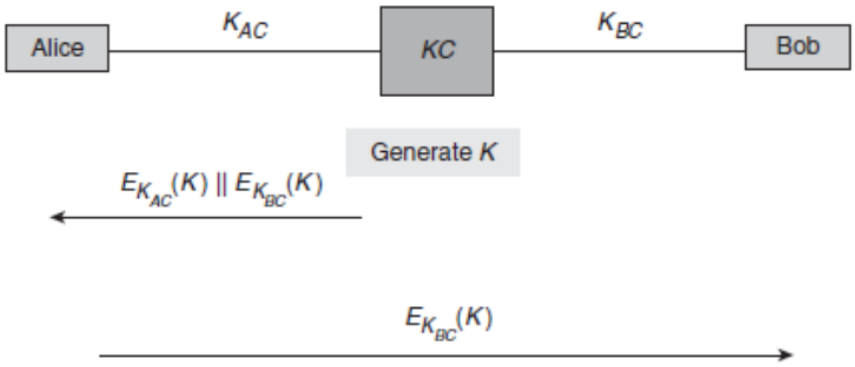
Key translation

***Key despatch*.** In this approach the KC generates the data key and produces two encrypted copies of it, one for each user. This process is depicted in the below figure and runs as follows:

1. KC generates a data key *K*, encrypts one copy of it using K_{AC} and another copy of it using K_{BC} , and then sends both encrypted copies to Alice.

2. Alice decrypts the first copy using K_{AC} and sends the other copy to Bob.

3. Bob decrypts the second copy using K_{BC} .



Key despatch

10.4.2 Unique key per transaction schemes

Unique key per transaction **(UKPT)** schemes is a different way of establishing a cryptographic key and they establish a new key each time that they are used.

MOTIVATION FOR UKPT SCHEMES

Previous key establishment mechanisms involves 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. This applies to every technique except key predistribution.

- ➢ The first requires devices that can securely store and use long-term keys, and the second introduces a communication overhead.
- ➢ One of the reasons that most of the previous schemes require above mentioned features is that the new key that is being established has been generated *independently*, in the sense that it has no relationship with any existing data.
- ➢ 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 adopts the methodology where it updates the 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.
    - They are also portable, so that they can easily be moved around, hence easily stolen.
2) Transactions should be processed speedily to avoid delays, hence efficiency is important.
3) Terminals may be managed and operated by unskilled staff, hence full automation of the key establishment process is a necessity.

EXAMPLE UKPT SCHEMES :

- Consider a UKPT scheme operating between a merchant *terminal* and a *host* (a bank or card payment server). The terminal maintains a *key register*, which is essentially the running 'key' that will be updated after every transaction.
- A generic UKPT scheme in terms of the protocol that is run between the terminal and the host during a transaction.

Note:

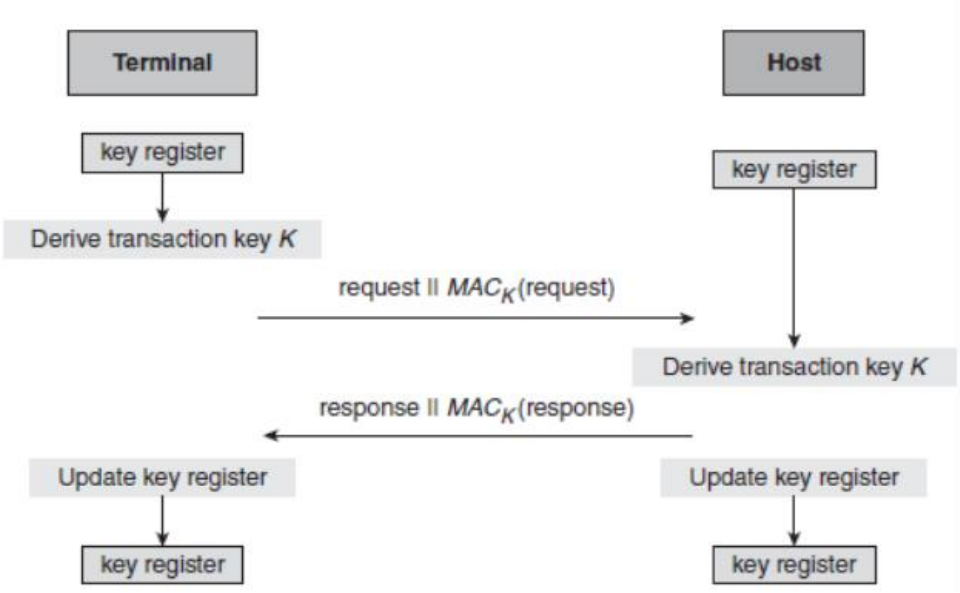
• Let us assume at the start of the protocol that the terminal and the host share an initial value that is stored in the terminal key register. This may or may not be a secret.

• a simple protocol that uses a single *transaction key* to compute MACs on the exchanged messages.

for example, an encryption key might also be needed to encrypt the PIN of the card.

Below figure illustrates 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.
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.



Generic UKPT scheme

Two examples of real UKPT schemes are:

Racal UKPT scheme:

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 (more precisely, the two *MAC residues* of the MACs on the request and response messages, both of which can be computed by the host and the terminal but neither of which are transmitted during the protocol,

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

Derived UKPT scheme. This scheme is supported by Visa:

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.
3. 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.

10.4.3 Quantum key establishment

MOTIVATION FOR QUANTUM KEY ESTABLISHMENT

- One motivation for *quantum key establishment*, is an attempt to make the establishment of long, shared, randomly generated symmetric keys 'easy'.
- Quantum key establishment is a technique for establishing a conventional symmetric key, which can then be used in any symmetric cryptosystem, including a one-time pad.

Problem: In a conventional communication channel, one simple way of establishing a symmetric key is for Alice to generate a key and then send it to Bob. The problem with this approach is that an attacker could be listening in on the communication and thus learn

the key. Even worse, neither Alice nor Bob would be aware that this has happened.

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 by *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*. The BB84 protocol involves the following steps:

- Alice randomly generates a stream of qubits, and sends these as a stream of polarised photons to Bob.
- Bob measures them using a *polarisation detector*, which will return either a 0 or a 1 for each photon.
- 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 probably 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 to 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.
- 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.

QUANTUM KEY ESTABLISHMENT IN PRACTICE

There are a number of substantial limitations of quantum key establishment. These include:

Distance limitations. Implementations of quantum key establishment has still only been demonstrated to work over limited distances. For example, in 1988 it was shown to work over a 30 cm distance. This had improved by 2010 to around 150 km in an optical fibre network, and several demonstration networks had been built that used quantum key establishment. In contrast, there are no technical limits on the distance over which most conventional key establishment techniques can be used.

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. Although the associated costs will doubtless reduce over time, most conventional key establishment techniques do not require such special technology.

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. One way is to use symmetric cryptography. So how do they establish the key used for authentication? If a conventional key establishment technique is used then the security of the quantum key establishment relies on the security of conventional key establishment.

10.5 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.

10.5.1 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.
- 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.
- Deterministic generators require a seed, so we will require this seed to be consistently used each time we generate the key so the seed is stored inside the human brain in the form of a passphrase or strong password.
- The user generates a passphrase, which they are required to remember.
- The passphrase is then used as a seed for a deterministic generator that generates the key encrypting key on the fly.

- The key encrypting key is then used to decrypt the encrypted private key.

Drawback: is that the security of the stored key is now dependent on the security of the seed (passphrase) that is used to generate the key encrypting key.

But it is not always possible to avoid storing a key. For example:

- Suppose that a symmetric key is being used to secure communication between Alice and Bob, who are in different locations. In some applications Alice and Bob may be able to locally generate the key precisely when they require it. However, in many other applications the key will need be stored somewhere, at least for a short while (for example, if Alice and Bob are both issued with the key in advance by a mutually trusted third party).
- Many uses of cryptography require long-term access to certain keys. For example, keys used for secure data storage may themselves need to be stored for a long time in order to facilitate future access to the protected data.
- Public-key pairs are expensive to generate. Generating them precisely when they are needed is inefficient. In many cases this is impossible, since the devices on which the private keys reside (for example, a smartcard) may have no user interface. Thus private keys almost always need to be securely stored.

10.5.2 Key storage in software

- One option for storing a cryptographic key is to embed the key into software.
- One common approach is to try to 'hide' the keys somewhere in the software.
- 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**.

- **Store it in hardware**. This is the most common approach but, 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**. 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. As components are essentially keys themselves, hence not easily memorised, the most common way to store components is on hardware (such as a smart card). Thus component form is really a strengthening of a hardware- based solution, not an alternative.

10.5.2 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 known as *tamper-resistant devices*.
- Many HSMs can also perform bulk cryptographic operations, often at high speed. An HSM can either be peripheral or can be incorporated into a more general purpose device such as a point-of-sale terminal.
- HSMs are mechanisms for the secure storage of cryptographic keys, HSMs are often used to enforce other phases of the key lifecycle.
- 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 outside 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 ON AN HSM

There is at least one key, often referred to as a ***local master key*** **(LMK),** that resides inside the HSM at all times. Some HSMs may store many LMKs, each having its own specific use. Any other keys that need to be stored can either be:

1. stored inside the HSM;
2. stored outside the HSM, encrypted using an LMK.

- When a key stored outside the HSM needs to be used, it is first submitted to the HSM, where it is recovered using the LMK and then used.
- This approach places a great deal of reliance on the LMK. It is thus extremely important to back up the LMK in order to mitigate against loss of the LMK. Such loss can occur if the HSM fails, or if it is attacked, since the tamper-resistance controls are likely to delete the HSM memory.

- Indeed this applies to any keys that are only stored inside the HSM. Thus the issue of whether to store a key inside or outside the HSM involves a tradeoff between:

Efficiency – storing keys inside the HSM is more efficient in terms of processing speed since they do not need to be imported and then recovered before use.

Need for backups – since every key only stored inside the HSM needs to be securely backed up, perhaps in component form.

10.5.4 Key storage risk factors

The risks to key storage media depend not only on the devices on which keys are stored, but also on the environments within which the devices reside. This relationship is indicated in Figure 10.6, which identifies four zones based on different environmental and device controls.

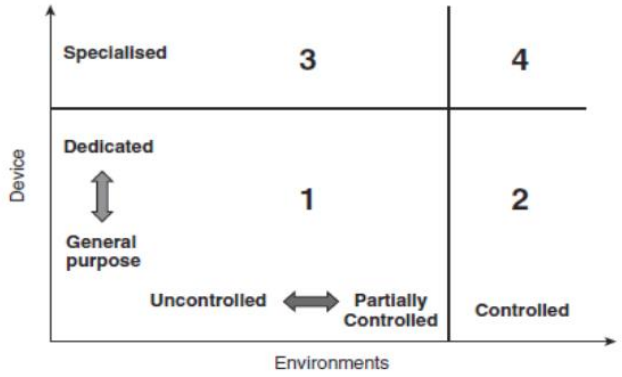


Figure 10.6. Key storage risk zones

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 installa- tions, 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. Likewise, any keys stored under the limited protection of a portable point-of-sale terminal are probably still secure from anything other than attacks from experts.

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 environ- ments 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.

10.5.5 Key backup, archival and recovery

For example:

- 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;
- A digital signature on a message becomes ineffective if the corresponding verification key is lost, since nobody has the ability to verify it.
    - ➢ The first scenario illustrates the potential need for key backup of critical secret keys.

➢ The second scenario more broadly illustrates the potential need for key archival, which is the long-term storage of keys beyond the time of their expiry.

KEY BACKUP

- 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.
- Zone 4 HSMs are subject to risks such as a careless cleaner bumping into a device and accidentally wiping its memory.
- the security of a key backup process must be as strong as the security of the key itself.
- The backed-up key will need to be stored on media that is subject to at least the same level of device and environmental security controls as the key itself.
- Indeed for the highest levels of key, the use of component form might be the only appropriate method for key backup.

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.
- Managing the storage of archived keys is just as critical as for key backups.
- Once a key no longer needs to be archived, it should be destroyed.

KEY RECOVERY

- Key recovery is the key management process where a key is recovered from a backup or an archive.
- 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.
- Many suggested key escrow mechanisms employed component form storage of escrowed keys in an attempt to reassure potential users of their security.

10.6 Key usage

10.6.1 Key separation

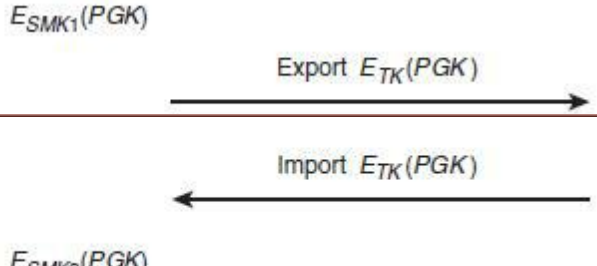
The principle of key separation is that cryptographic keys must only be used for their intended purpose.

THE NEED FOR KEY SEPARATION

- 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.
- In other applications it may be tempting to use a key that has already been established for one purpose and then use it for some other purpose.
- Disadvantages of doing this are:

Example 1. 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 it may become possible to decrypt and reveal a PIN or it may not be possible to recover any normal data encrypted with the PIN encrypting key.

- 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.
- 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 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 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 regards PGK as a MAC key.

This attack will not be possible if different variants of transport key are used for separate export and import functions.

ENFORCING KEY SEPARATION

Keys are often unstructured bit strings.There are two main techniques that can be used to enable the purpose of a key to be distinguished:

1. **Encrypting a key using a specified variant key**. This is a hardware-enforced method which involves using specific higher level keys to encrypt keys for particular purposes. This technique can be applied to keys being distributed, as well as keys being stored. This method can be used to enforce any type of key separation.
2. **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.

Examples:

1. **Employing redundancy.** 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 tagging allows the eight spare bits to define the key usage. When a key is presented in a command to an HSM, the tagging bits are checked by the HSM to ensure that the key is a valid key for the command that

it is being used for.

2. **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 shown in Figure 10.8 and has the following fields:

Figure 10.8. ANSI TR-31 key block

- Header includes information that clarifies the purpose of the key.
- Optional header includes optional data such as the expiry date of the key.
- Key is encrypted using a suitable key encrypting key.
- Authenticator is a MAC on the rest of the key block, which provides data origin authentication (data integrity) of the key block data.

iii. **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 defines the key usage.

10.6.2 Key change

THE NEED FOR KEY CHANGE

1. **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 lifetime.
2. **Unplanned key changes**. These may occur for a variety of reasons.

- A key is compromised
- A security vulnerability such as operating system vulnerability, a breakthrough in cryptanalysis, or a failure of a tamper-resistance mechanism in an HSM.
- An employee unexpectedly leaves an organization.

IMPACT OF KEY CHANGE

- Key change can be a very expensive process.
- An unplanned key change is particularly problematic.
- The minimum impact of a key change is that a new key needs to be generated and established.
- The impact can be severe, especially in the case of high-level key compromise. For example, if a master key is compromised in a financial system then the resulting costs

might include costs of investigation into the compromise, costs related to any transactions conducted using compromised keys, damage to reputation and loss of customer confidence.

- Generation and establishment of a new key.
- Withdrawing the old key (destroying or archiving it).

Planned key changes should happen automatically and require very little intervention.

More intervention may be required in the case of unplanned key changes.

There are two reasons why changing public-key pairs is normally more challenging:

1. **Knowledge of public keys**. The 'public' nature of a public key means that a key management system may have little control over which entities have knowledge of a public key. In open environments such as the Internet, a public key could be known by anyone.
2. Open application environments. Public-key cryptography tends to be used in open environments where this may be more challenging. Since private and public keys are interdependent, any requirement to change one of them requires the other also to be changed. Changing a private key is simpler than changing a symmetric key.

10.6.3 Key activation

In many applications key activation requires human interaction. 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 be up to 2048 bits long, which is a value that the human owner of the keywill not be capable of memorizing. When the user decides to digitally sign an email, the user needs to instruct the email client to activate their signature key. Several scenarios depending on how the key is stored on the computer. These include:

1. **Key stored on the computer in the clear**. the user might activate the key simply by entering an instruction, selecting the key from a list of potential keys stored on the computer. Key activation is possible for anyone with access to the computer. In this case the effective security of the keys is simply linked to the security required to access the computer itself, which requires a valid username and password.
2. **Key stored on the computer in encrypted form**. The user might activate the key in this case by providing 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.

3. **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 linked to the generation of the key. 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.
4. **Key stored off the computer**. 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.

10.6.4 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).
3. At the end of a required period of key archival.

- Since keys are sensitive data, secure mechanisms must be used. Suitable techniques are sometimes referred to as data erasure or data sanitization mechanisms.
- Simply deleting a key from a device is not sufficient if the key is to be truly destroyed.
- Operating systems may well have other (temporary) copies of the key in different locations.
- Even if the key was stored on the device in encrypted form, this may be useful to a determined attacker. Any other media storing information about keys, such as paper, should also be destroyed.

10.7 Governing key management:

- Key management is the main interface between the technology of cryptography and the users and systems that rely on it.
- Key management is a much more complex process for an organisation, due to the diversity of processes that affect key management,

10.7. 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.

Important outcome of key management governance is:

- **By design**: in other words, that the entire key management lifecycle has been planned from the outset, and not made up in response to events as they occur.
- **Coherent**: the various phases of the key lifecycle are considered as linked component parts of a larger unified process and designed with 'big picture' in mind.
- **Integrated**: the phases of the key management lifecycle are integrated with the wider requirements and priorities of the organisation.

10.7.2 Example procedure: key generation ceremony

- **Key ceremony**: It is an important type of key management procedure that might be required by a large organization and used to implement key generation from components.
-

In key generation ceremony the participants are:

1. **Operation manager:** responsible for the physical aspects, including the venue, hardware, software and any media on which components are stored or transported.
2. **Key manager:** responsible for making sure that the key ceremony is performed in accordance with the relevant key management policies, practices and procedures.
3. **Key custodians**: the parties physically in possession of the key components, responsible for handling them appropriately and following the key ceremony as instructed.
4. **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.

Key ceremony involves several phases:

1. **Initialization**. 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.
2. **Component retrieval**. The components required for the key ceremony are transported to the key ceremony location.
3. **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. Throughout the key ceremony, the witnesses record the events and any deviations from the defined procedure are noted. At the end, an official record is presented to the key manager.
4. **Validation**. The official record can be scrutinized to validate that the correct procedure was followed.

Public-Key Management

11.1 Certification of public keys

The main challenge for the management of public keys is providing assurance of purpose of public keys. The most popular mechanism for providing this assurance of purpose, the ***public-key certificate*.**

11.1.1 Motivation for public-key certificates

A SCENARIO

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. This requires Bob to have access to Alice's verification key. Suppose that Bob is presented with a key 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? And many more questions:

- *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

The above scenario requires:

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).

PROVIDING A POINT OF TRUST

The problem in designing any public-key management system is that we need to find a source for the provision of the 'strong association' between a publickey value and its related data. In public-key management systems this is provided by introducing points of trust in the form of trusted third parties who 'vouch' for this association.

USING A TRUSTED DIRECTORY

An 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.

This approach has 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.

11.1.2 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.

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. This provides the 'strong association'.

Example:

Table 11.1: Fields of an X.509 Version 3 public-key certificate

Field	Description
Version	Specifies the X.509 version being used (in this case V3)
Serial Number	Unique identifier for the certificate
Signature	Digital signature algorithm used to sign the certificate
Issuer	Name of the creator of the certificate
Validity	Dates and times between which the certificate is valid
Subject	Name of the owner of the certificate
Public-Key Info.	Public-key value; Identifier of public-key algorithm
Issuer ID	Optional identifier for certificate creator
Subject ID	Optional identifier for certificate owner
Extensions	A range of optional fields that include:
	Key identifier (in case owner owns several public keys);
	Key usage (specifies usage restrictions);
	Location of revocation information;
	Identifier of policy relating to certificate;
	Alternative names for owner.

INTERPRETING A PUBLIC-KEY CERTIFICATE

- 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:

1. **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.
2. **Certificate revocation**: The CA is responsible for revoking the certificate in the event that it becomes invalid.
3. **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 fulfill this role, the CA will need to actively maintain its profile as a trusted organization. It may also need to enter into relationships with other organizations in order to facilitate wider recognition of this trust.

RELYING ON A PUBLIC-KEY CERTIFICATE

In order to obtain assurance of purpose of the public key follow three steps:

1. Trust the CA: The relying party needs to be able to trust the CA to have performed its job correctly when creating the certificate.

2. Verify the signature on the certificate: The relying party needs to have access to the verification key of the CA in order to verify the CA's digital signature on the public-key certificate. If the relying party does not verify this signature then they have no guarantee that the contents of the public-key certificate are correct.

3. **Check the fields**: The relying party needs to check all the fields in the public-key certificate. In particular, they must check the name of the owner and that the public-key certificate is valid. If the relying party does not check these fields then they have no guaranteed that the public key in the certificate is valid for the application for which they intend to use it.

DIGITAL CERTIFICATES

Public-key certificates represent a special class of digital certificates. An example of another type of digital certificate is an attribute certificate, which can be used to provide a strong association between a specific attribute and an identity, such as:

• The identity specified is a member of the access control group administrators.

• The identity specified is over the age of 18.

Attribute certificates might contain several fields that are similar to a public-key certificate (for example, owner name, creator name, validity period) but would not contain a public-key value.

11.2 The certificate lifecycle

11.2.1 Differences in the certificate lifecycle

1. Key generation.

The generation of a public-key pair is an algorithm-specific, and often technically complex, operation. 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.

2. **Key establishment**.

Private Key establishment is potentially easier than symmetric key establishment since the private key only needs to be established by one entity. This entity could even generate the private key themselves. 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: The owner of the public-key certificate provides the certificate whenever it is required by a relying party.

Pulling a public-key certificate: The relying parties must retrieve public-key certificates from some sort of repository when they first need them. One potential advantage of pulling public-key certificates is that they could be pulled from a trusted database that only contains valid public-key certificates.

3. Key storage, backup, archival:

They are all less-sensitive operations when applied to public key certificates.

4.Key usage:

Many public-key certificate formats, such as the X.509 Version 3 certificate format include fields for specifying key usage.

5.Key destruction:

Destruction of public-key certificates is less sensitive, and may not even be required.

11.2.2 Certificate creation

LOCATION OF KEY PAIR AND CERTIFICATE CREATION

- Key pair generation and Certificate creation two separate processes here:
- 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).

The choice of location for this operation results in different certificate creation scenarios:

➢ **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.

Advantages:

- 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.
- The key pair generation process does not require the owner to do anything.

Disadvantages:

- The owner needs to trust the third party to securely distribute the private key to the owner;
- The owner needs to trust the third party to destroy the private key after it has been distributed to 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.

Advantages:

- **T**he 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.

Disadvantages:

- 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. Since a public-key certificate generated by a CA provides 'independent' assurance of purpose of a public key, whereas self-certification requires relying parties to trust in the assurance of purpose provided by

the owner of the public key. However, if relying parties trust the owner then this scenario may be justifiable.

Examples:

- The owner is a CA; it is not uncommon for CAs to self-certify their own public-keys.
- 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 publickey 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.
- These credentials not only provide a means of authenticating the owner, but also provide information that will be included in some of the fields of the public-key certificate.
- Registration process that varies greatly between different applications.

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 the most complex part of the certificate creation process.
    - Centralised checking of credentials represents a major bottleneck in the process, particularly for large organisations.
    - Distributing the registration activities across a number of local RAs, which perform the checking and then report the results centrally.

What credentials should be presented to the RA during registration?

Some examples of credentials:

- A very low level of public-key certificate might simply require a valid email address to be presented at registration. The registration process might include checking that the applicant can receive email at that address.
- Registration for public-key certificates for use in a closed environment, such as an organisation's internal business environment, might involve presentation of an employee number and a valid internal email address.
- Commercial public-key certificates for businesses trading over the Internet might require a check of the validity of a domain name and the confirmation that the applicant business is legally registered as a limited company.
- Public-key certificates for incorporation into a national identity card scheme require a registration process that unambiguously identifies a citizen. Credentials might include birth certificates, passports, domestic utility statements, etc.

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.

Problem:

- The attacker now has a public-key certificate issued in their name for a verification key for which they do not know the corresponding signature key.
- A problem arises if Alice now digitally signs a message with her signature key, since the attacker will be able to persuade relying parties that this is actually the attacker's digital signature on the message.
- This is because the attacker's name is on a public-key certificate containing a verification key that successfully verifies the digital signature on the message.
- This attack can be prevented if the CA conducts a simple check that the publickey certificate applicant knows the corresponding private key.

- This type of check is 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:

- 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.
- 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. This raises the question of how assurance of purpose of the CA's verification key will be provided.

Solution: is to create a public-key certificate for the CA's public key.

But who will sign the public-key certificate of the CA?

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 CAs then the CA may choose to have their public key certified by another CA.

Self-certification. A top-level CA has no choice other than self certification.

As an example, CAs who certify public keys that are used in web-based commercial applications need to have their public-key certificates incorporated into leading web browsers, or have them certified by a higher-level CA who has done this.

11.2.3 Key pair change

REVOCATION OF PUBLIC-KEY CERTIFICATES

Withdrawing an existing public key is very difficult. This process is referred to as *revoking* the public key. Revoking a public key essentially means revoking the public-key certificate.

REVOCATION TECHNIQUES

Three ways of Revocation of public-key certificates are:

Blacklisting.

- This involves maintaining a database that contains serial numbers of public-key certificates that have been revoked.

- This type of database is referred to as a *certificate 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.
- The CRLs need to be digitally signed by the CA and made available to relying parties.

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.

An example is the *Online Certificate Status Protocol* (OCSP).

Rapid expiration. This removes the need for revocation by allocating very short lifetimes to public-key certificates. This, of course, comes at the cost of requiring certificates to be reissued on a regular basis.

Blacklisting is a common technique when real-time revocation information is not required. There are many different ways of implementing the blacklisting concept, often involving networks of distributed CRLs rather than one central CRL.

The main problem with blacklisting is one of synchronisation. i.e:

- Reporting delays between the time that a public-key certificate should be revoked (for example, the time of a private key compromise) and the CA being informed;
- CRL issuing delays between the time that the CA is informed of the revocation of a public-key certificate and the time that the next version of the CRL is signed and made publicly available.

Thus a relying party could rely on a public-key certificate in the gap period between the time the public-key certificate should have been revoked and the publication time of the updated CRL. This issue must be 'managed' through suitable processes and procedures.

For example:

• The CA should inform all relying parties of the update frequency of CRLs.

• The CA should clarify who is responsible for any damage incurred from misuse of a public key in such a gap period.

Address these issues by:

– the CA accepting limited liability during gap periods;

– relying parties accepting full liability if they fail to check the latest CRL before relying on a public-key certificate.

11.3 Public-key management models

11.3.1 Choosing a CA

Choosing an organization to play the role of a CA in an open environment is less straightforward. Currently, most CAs serving open environments are commercial organizations who have made it their business to be 'trusted' to play the role of a CA. While CAs serving open environments can be regulated to an extent by commercial pressure 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 best practices.

11.3.2 Public-key certificate management models

1. CA-FREE CERTIFICATION MODEL

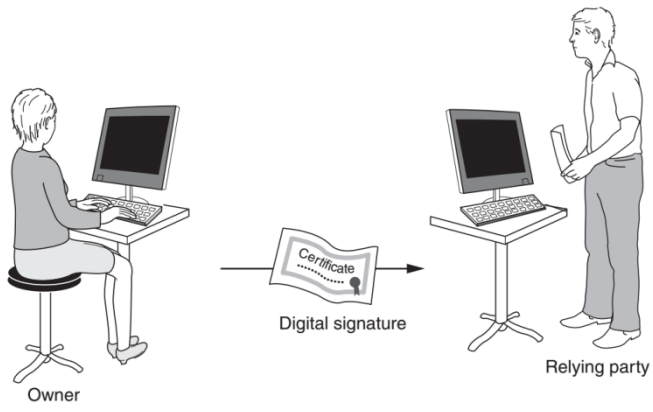


Figure 11.1. CA-free certification model

- 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.
- For example, the owner could include their public key in an email signature or write it on a business card. The relying party then has to make an independent decision as to

whether they trust the owner or not. The relying party thus carries all the risk in this model.

2. REPUTATION-BASED CERTIFICATION MODEL

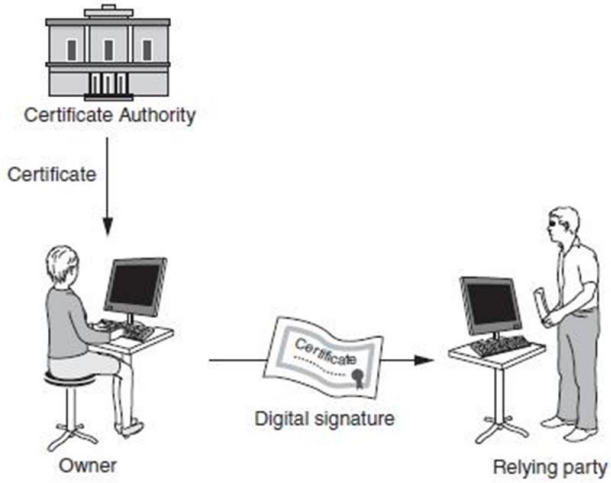


Figure 11.2. Reputation-based certification model

➢ 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.

➢ Even if the relying party obtains the verification key of the CA, which enables them to verify the public-key certificate, because they do not have any relationship with the CA itself they do not by default gain assurance of purpose from verification of the certificate.

➢ They are left to choose whether to trust that the CA has done its job correctly and hence that the information in the public-key certificate is correct. In the worst case, they may not trust the CA at all, in which case they have no reason to trust any information affirmed by the CA.

➢ The only assurance that they might gain is through the reputation of the CA that signed the public-key certificate. If the relying party has some trust in the reputation of the CA, for example, it is a well-known organization or trust service provider, and then the relying party might be willing to accept the information in the public-key certificate.

3.CLOSED CERTIFICATION MODEL

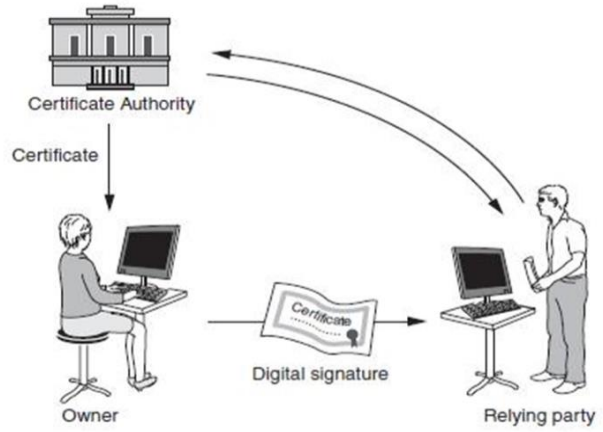


Figure 11.3. Closed certification model

- 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.
- In the closed certification model, the more onerous issues concerning public-key management, such as those relating to liability and revocation, are more straightforward to solve than for the other models.
  - This is because public key certificate owners and relying parties are all governed by the same certificate management policies and practices.

4.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.
- The role of validation authority is to assist the relying party to validate the information in the owner's public-key certificate.
  - This validation authority may not necessarily be a CA. the CA and validation authority could both be members of a federation of organizations who have agreed to cooperate in the validation of public-

key certificates and have signed up to common certificate management policies and practices.

✓ The connected certification model is a pragmatic 'stretching' of the closed certification model, in order to allow public-key certificates to be managed in environments that are either:

Open: Owners and relying parties are not governed by any single management entity. **Distributed:** In the case of a closed environment that is distributed. For example a large organization with different branches or regional offices.

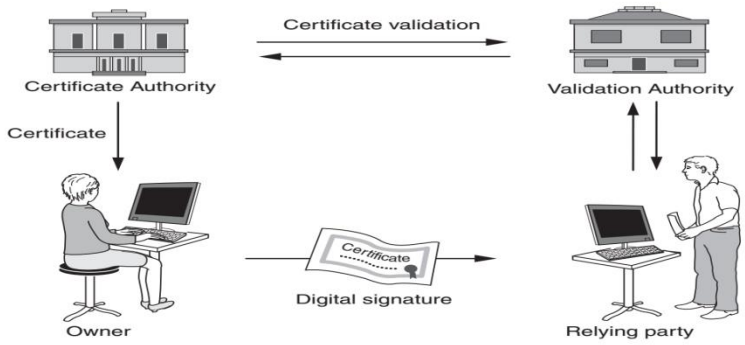


Figure 11.4. Connected certification model

11.3.3 Joining CA domains

There are three approaches:

1. **CROSS-CERTIFICATION**

Figure 11.5. Cross-certification

The first technique for joining two CA domains is to use cross-certification; each CA certifies the other CA's public key. This idea is depicted in Figure 11.5. Cross-certification implements a transitive trust relationship. By cross-certifying, relying party Bob of CA2, who wishes to trust a public-key certificate issued to Alice by CA1, can do so by means of the following argument:

1. Bob trust CA2 (because bob 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. Bob trust the information in Alice's public-key certificate.

2.CERTIFICATION HIERARCHIES

The second technique is 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. The simple certification hierarchy in Figure 11.6 uses a higher-level CA (termed the root CA) to issue public-key certificates for both CA1 and CA2. Relying party Bob of CA2, who wishes to trust a public-key certificate issued to Alice by CA1, can do so by means of the following argument:

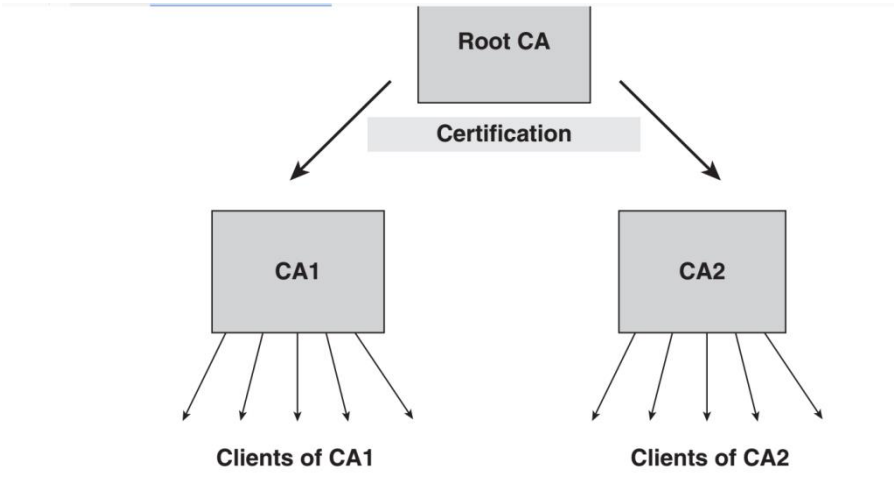


Figure 11.6. Certification hierarchy

1. Bob trust CA2 (because Bob 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. Bob trust the information in Alice's public-key certificate.

3.CERTIFICATE CHAINS

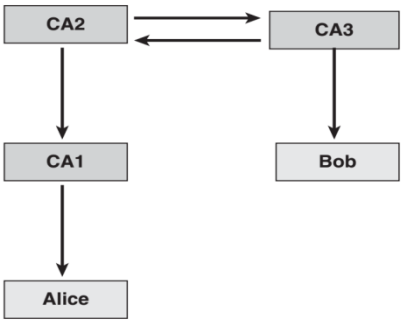


Figure 11.7. A simple CA topology

The joining of CA domains makes verification of public certificates a potentially complex process. 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. Example: consider the apparently simple CA topology in Figure 11.7. In this topology Alice has a relationship with CA1, which is a low-level CA, whose root CA is CA2. Bob has a relationship with CA3, which has cross-certified with CA2. Now suppose that Bob wishes to verify Alice's public-key certificate. He will need to verify a certificate chain that consists of the three public- key certificates shown in Table 11.2. To properly verify the above certificate chain, for each of these public key certificates Bob should:

1. Verify the signature on the public-key certificate.
2. Check all the fields in the public-key certificate.
3. Check whether the public-key certificate has been revoked.

Table 11.2: Example of a certificate chain

Certificate	Containing public key of	Certified by
1	Alice	CA1
2	CA1	CA2
3	CA2	CA3

11.4 Alternative approaches

11.4.1 Webs of trust

✓ 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 of a web of trust involves other public-key certificate owner's acting as 'lightweight CAs' by digitally signing Alice's public key.

✓ Alice gradually develops a key ring, which consists of her public key plus a series of digital signatures by other owners attesting to the fact that the public-key value is indeed Alice's.

✓ These other owners are not acting as formal CAs, and the relying party may have no more of a relationship with any of these other owners than with Alice herself.

✓ Alice builds up her key ring there are two potentially positive impacts for relying parties:

1 A relying party sees that a number of other owner's have been willing to sign Alice's public key. This is at least some evidence that the public key may indeed belong to Alice.

2 There is an increasing chance (as the key ring size increases) that one of the other owners is someone that the relying party knows and trusts. If this is the case then the relying party might use a transitive trust argument to gain some assurance about Alice's public key.

Webs of trust have limitations.

- They represent a lightweight and scalable means of providing some assurance of purpose of public keys in open environments.
- Webs of trust make a real impact is unclear since, for the types of open applications in which they make most sense, relying parties are often likely to choose to simply trust the owner.

11.4.2 Identity-based public-key cryptography

THE IDEA BEHIND IDPKC

✓ A significant difference between IDPKC and certificate-based approaches to management of conventional public-key cryptography is that IDPKC requires a trusted third party to be involved in private-key generation.

✓ This trusted third party is referred as a trusted key centre (TKC), since its main role is the generation and distribution of private keys.

- A public-key owner's 'identity' is their public key. There is a publicly known rule that converts the owner's 'identity' into a string of bits, and then some publicly known rule that

converts that string of bits into a public key.

• The public-key owner's private key can be calculated from their public key only by the TKC, who has some additional secret information.

A MODEL FOR IDPKC ENCRYPTION

•

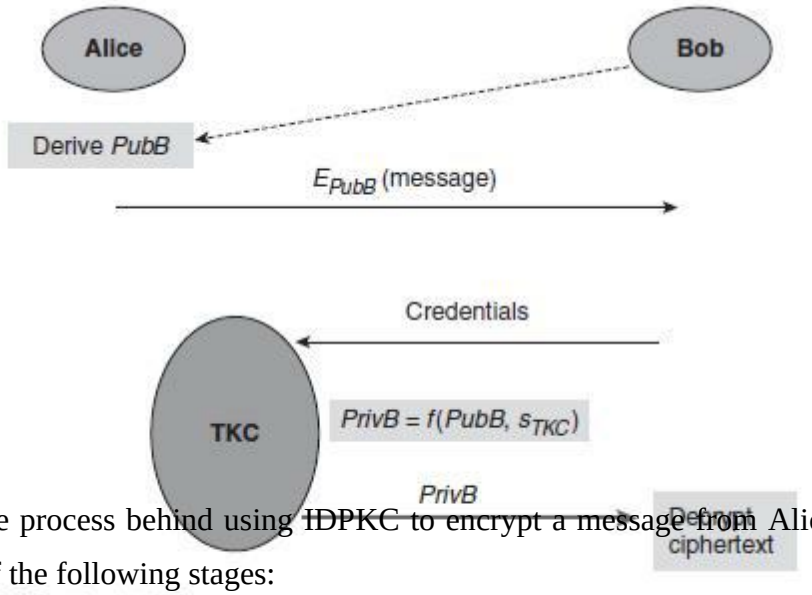


Figure 11.8. The IDPKC process

Figure 11.8 shows the process behind using IDPKC to encrypt a message from Alice to Bob. The model consists of the following stages:

1. **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.
2. **Identification**. Bob identifies himself to the TKC by presenting appropriate credentials and requests the private key PrivB that corresponds to PubB.
3. **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.
4. **Private-key distribution**. The TKC sends PrivB to Bob using a secure channel.
5. **Decryption**. Bob decrypts the ciphertext using PrivB.

One of the most interesting aspects of this IDPKC encryption model is that encryption can occur before private-key derivation and distribution. It is possible to send an encrypted message to someone who has not yet established a private decryption key. Once a user has obtained their private key, there is no need for them to repeat the middle three stages until either PubB or the system secret STKC change.

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 any value to be

a public key. Rather, a public key is a value that satisfies certain specific mathematical properties.

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

1. The need for an online, centrally trusted TKC.
    - 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.
    - The TKC requires secure channels with all private key owners.

2. **Revocation issues.**

- One of the problems with tying a public-key value to an identity is the impact of revocation of a public key.

- 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.

- For example, Bob's public key might become PubB 3rdApril for any encrypted messages intended for Bob on 3rd April, but change to PubB 4thApril the next day. The cost of this is a requirement for Bob to obtain a new private key for each new day.

3. Multiple Applications

- In conventional public-key cryptography, one owner can possess different public keys for different applications.

- By tying an identity to a public-key value, this is not immediately possible. One solution is to introduce variety into the encryption key using PubB Bank as Bob's public key for his online banking application.

- IDPKC offers a potential advantage in this area since it is possible for the same public key PubB to be used across multiple applications.

MORE GENERAL NOTIONS OF IDPKC

- The idea behind IDPKC is both compelling and intriguing, since it represents a quite different approach to implementing public-key cryptography.

- They could take the form of almost any string of data. 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 using PubPolicy and 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 the TKC accepts Bob's credentials then the TKC derives 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 ciphertext using PrivPolicy.