Chapter 13

Key Management Techniques

Contents in Brief

13.1 Introduction

This chapter considers key management techniques for controlling the distribution, use, and update of cryptographic keys. Whereas Chapter 12 focuses on details of specific key establishment protocols which provide shared secret keys, here the focus is on communications models for key establishment and use, classification and control of keys based on their intended use, techniques for the distribution of public keys, architectures supporting automated key updates in distributed systems, and the roles of trusted third parties. Systems providing cryptographic services require techniques for initialization and key distribution as well as protocols to support on-line update of keying material, key backup/recovery, revocation, and for managing certificates in certificate-based systems. This chapter examines techniques related to these issues.

Chapter outline

The remainder of this chapter is organized as follows. §13.2 provides context including background definitions, classification of cryptographic keys, simple models for key establishment, and a discussion of third party roles. §13.3 considers techniques for distributing confidential keys, including key layering, key translation centers, and symmetric-key certificates. §13.4 summarizes techniques for distributing and authenticating public keys including authentication trees, public-key certificates, the use of identity-based systems, and implicitly-certified keys. §13.5 presents techniques for controlling the use of keying material, including key notarization and control vectors. §13.6 considers methods for establishing trust in systems involving multiple domains, certification authority trust models, and certification chains. The key management life cycle is summarized in §13.7, while §13.8 discusses selected specialized third party services, including trusted timestamping and notary services supporting non-repudiation of digital signatures, and key escrow. Notes and sources for further information are provided in §13.9.

13.2 Background and basic concepts

A *keying relationship* is the state wherein communicating entities share common data (*keying material*) to facilitate cryptographic techniques. This data may include public or secret keys, initialization values, and additional non-secret parameters.

13.1 Definition *Key management* is the set of techniques and procedures supporting the establishment and maintenance of keying relationships between authorized parties.

Key management encompasses techniques and procedures supporting:

1. initialization of system users within a domain;
2. generation, distribution, and installation of keying material;
3. controlling the use of keying material;
4. update, revocation, and destruction of keying material; and
5. storage, backup/recovery, and archival of keying material.

13.2.1 Classifying keys by algorithm type and intended use

The terminology of Table 13.1 is used in reference to keying material. A *symmetric cryptographic system* is a system involving two transformations – one for the originator and one for the recipient – both of which make use of either the same secret key (symmetric key) or two keys easily computed from each other. An *asymmetric cryptographic system* is a system involving two related transformations – one defined by a public key (the public transformation), and another defined by a private key (the private transformation) – with the property that it is computationally infeasible to determine the private transformation from the public transformation.

Term	Meaning
private key, public key	paired keys in an asymmetric cryptographic system
symmetric key	key in a symmetric (single-key) cryptographic system
secret	adjective used to describe private or symmetric key

Table 13.1: *Private, public, symmetric, and secret keys.*

Table 13.2 indicates various types of algorithms commonly used to achieve the specified cryptographic objectives. Keys associated with these algorithms may be correspondingly classified, for the purpose of controlling key usage (§13.5). The classification given requires specification of both the type of algorithm (e.g., encryption vs. signature) and the intended use (e.g., confidentiality vs. entity authentication).

↓ Cryptographic objective (usage)	Algorithm type	
	public-key	symmetric-key
confidentiality†	encryption	encryption
data origin authentication‡	signature	MAC
key agreement	Diffie-Hellman	various methods
entity authentication (by challenge-response protocols)	1. signature 2. decryption 3. customized	1. MAC 2. encryption

Table 13.2: *Types of algorithms commonly used to meet specified objectives.*
†May include data integrity, and includes key transport; see also §13.3.1.
‡Includes data integrity; and in the public-key case, non-repudiation.

13.2.2 Key management objectives, threats, and policy

Key management plays a fundamental role in cryptography as the basis for securing cryptographic techniques providing confidentiality, entity authentication, data origin authentication, data integrity, and digital signatures. The goal of a good cryptographic design is to reduce more complex problems to the proper management and safe-keeping of a small number of cryptographic keys, ultimately secured through trust in hardware or software by physical isolation or procedural controls. Reliance on physical and procedural security (e.g., secured rooms with isolated equipment), tamper-resistant hardware, and trust in a large number of individuals is minimized by concentrating trust in a small number of easily monitored, controlled, and trustworthy elements.

Keying relationships in a communications environment involve at least two parties (a sender and a receiver) in real-time. In a storage environment, there may be only a single party, which stores and retrieves data at distinct points in time.

The objective of key management is to maintain keying relationships and keying material in a manner which counters relevant threats, such as:

1. compromise of confidentiality of secret keys.
2. compromise of authenticity of secret or public keys. Authenticity requirements include knowledge or verifiability of the true identity of the party a key is shared or associated with.
3. unauthorized use of secret or public keys. Examples include using a key which is no longer valid, or for other than an intended purpose (see Remark 13.32).

In practice, an additional objective is conformance to a relevant security policy.

Security policy and key management

Key management is usually provided within the context of a specific *security policy*. A security policy explicitly or implicitly defines the threats a system is intended to address. The policy may affect the stringency of cryptographic requirements, depending on the susceptibility of the environment in question to various types of attack. Security policies typically also specify:

1. practices and procedures to be followed in carrying out technical and administrative aspects of key management, both automated and manual;
2. the responsibilities and accountability of each party involved; and
3. the types of records (*audit trail information*) to be kept, to support subsequent reports or reviews of security-related events.

13.2.3 Simple key establishment models

The following key distribution problem motivates more efficient key establishment models.

The n^2 key distribution problem

In a system with n users involving symmetric-key techniques, if each pair of users may potentially need to communicate securely, then each pair must share a distinct secret key. In this case, each party must have $n - 1$ secret keys; the overall number of keys in the system, which may need to be centrally backed up, is then $n(n-1)/2$, or approximately n^2. As the size of a system increases, this number becomes unacceptably large.

In systems based on symmetric-key techniques, the solution is to use centralized key servers: a star-like or spoked-wheel network is set up, with a trusted third party at the center or hub of communications (see Remark 13.3). This addresses the n^2 key distribution problem, at the cost of the requirement of an on-line trusted server, and additional communications with it. Public-key techniques offer an alternate solution.

Point-to-point and centralized key management

Point-to-point communications and *centralized key management*, using key distribution centers or key translation centers, are examples of simple key distribution (communications) models relevant to symmetric-key systems. Here "simple" implies involving at most one third party. These are illustrated in Figure 13.1 and described below, where K_{XY} denotes a symmetric key shared by X and Y.

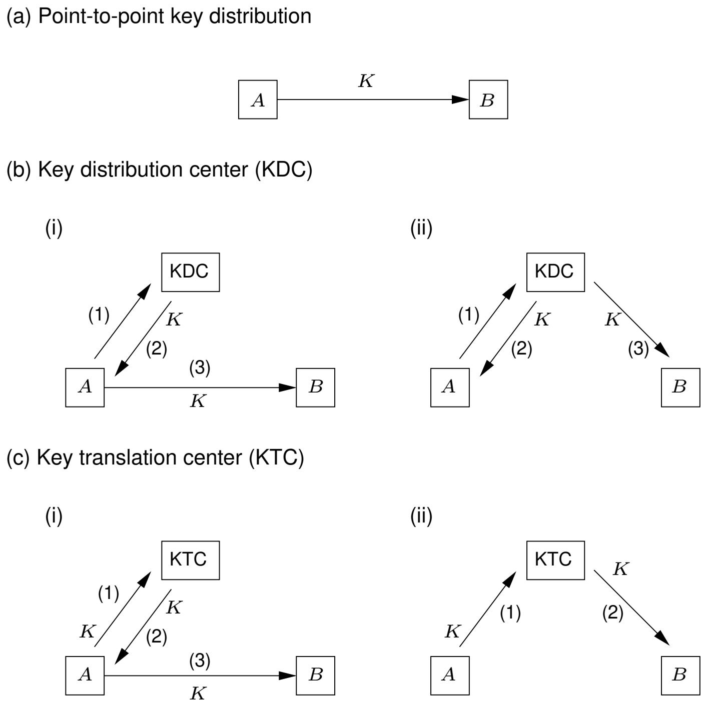

Figure 13.1: *Simple key distribution models (symmetric-key).*

1. *point-to-point* mechanisms. These involve two parties communicating directly (see §12.3.1).
2. *key distribution centers* (KDCs). KDCs are used to distribute keys between users which share distinct keys with the KDC, but not with each other.
   A basic KDC protocol proceeds as follows.[1] Upon request from A to share a key with B, the KDC T generates or otherwise acquires a key K, then sends it encrypted under K_{AT} to A, along with a copy of K (for B) encrypted under K_{BT}. Alternatively, T may communicate K (secured under K_{BT}) to B directly.
3. *key translation centers* (KTCs). The assumptions and objectives of KTCs are as for KDCs above, but here one of the parties (e.g., A) supplies the session key rather than the trusted center.
   A basic KTC protocol proceeds as follows.[2] A sends a key K to the KTC T encrypted under K_{AT}. The KTC deciphers and re-enciphers K under K_{BT}, then returns this to A (to relay to B) or sends it to B directly.

KDCs provide centralized key generation, while KTCs allow distributed key generation. Both are centralized techniques in that they involve an on-line trusted server.

13.2 Note (*initial keying requirements*) Point-to-point mechanisms require that A and B share a secret key *a priori*. Centralized key management involving a trusted party T requires that A and B each share a secret key with T. These shared long-term keys are initially established by non-cryptographic, out-of-band techniques providing confidentiality and authenticity (e.g., in person, or by trusted courier). By comparison, with public keys confidentiality is not required; initial distribution of these need only guarantee authenticity.

13.3 Remark (*centralized key management – pros and cons*) Centralized key management involving third parties (KDCs or KTCs) offers the advantage of key-storage efficiency: each party need maintain only one long-term secret key with the trusted third party (rather than one for each potential communications partner). Potential disadvantages include: vulnerability to loss of overall system security if the central node is compromised (providing an attractive target to adversaries); a performance bottleneck if the central node becomes overloaded; loss of service if the central node fails (a critical reliability point); and the requirement of an on-line trusted server.

13.2.4 Roles of third parties

Below, trusted third parties (TTPs) are first classified based on their real-time interactions with other entities. Key management functions provided by third parties are then discussed.

(i) In-line, on-line, and off-line third parties

From a communications viewpoint, three categories of third parties T can be distinguished based on relative location to and interaction with the communicating parties A and B (see Figure 13.2):

1. *in-line*: T is an intermediary, serving as the real-time means of communication between A and B.
2. *on-line*: T is involved in real-time during each protocol instance (communicating with A or B or both), but A and B communicate directly rather than through T.

[1] For specific examples of such protocols including Kerberos (Protocol 12.24), see §12.3.2.
[2] A specific example is the message-translation protocol, Protocol 13.12, with $M = K$.

3. *off-line*: T is not involved in the protocol in real-time, but prepares information *a priori*, which is available to A or B or both and used during protocol execution.

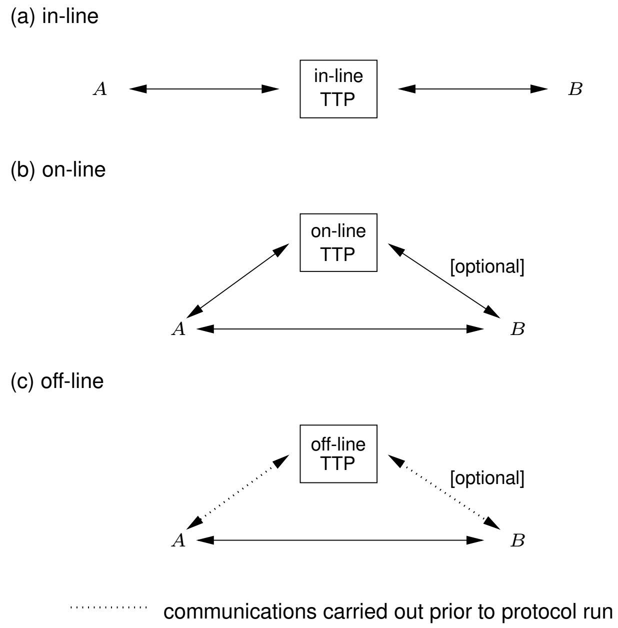

Figure 13.2: *In-line, on-line, and off-line third parties.*

In-line third parties are of particular interest when A and B belong to different security domains or cannot otherwise interact directly due to non-interoperable security mechanisms. Examples of an in-line third party include a KDC or KTC which provides the communications path between A and B, as in Figure 13.1(b)(ii) or (c)(ii). Parts (b)(i) and (c)(i) illustrate examples of on-line third parties which are not in-line. An example of an off-line third party is a certification authority producing public-key certificates and placing them in a public directory; here, the directory may be an on-line third party, but the certification authority is not.

13.4 Remark (*pros and cons: in-line, on-line, off-line*) Protocols with off-line third parties usually involve fewer real-time message exchanges, and do not require real-time availability of third parties. Revocation of privileges (e.g., if a secret key is compromised) is more easily handled by in-line or on-line third parties.

(ii) Third party functions related to public-key certificates

Potential roles played by third parties within a key management system involving public-key certificates (§13.4.2) are listed below. Their relationship is illustrated in Figure 13.3.

1. *certification authority* (CA) – responsible for establishing and vouching for the authenticity of public keys. In certificate-based systems (§13.4.2), this includes binding public keys to distinguished names through signed certificates, managing certificate serial numbers, and certificate revocation.[3]

[3]Certificate creation requires verification of the authenticity of the entity to be associated with the public key. This authentication may be delegated to a registration authority. The CA may carry out the combined functions of a registration authority, name server, and key generation facility; such a combined facility is called either a CA or a *key management facility*.

2. *name server* – responsible for managing a name space of unique user names (e.g., unique relative to a CA).
3. *registration authority* – responsible for authorizing entities, distinguished by unique names, as members of a security domain. User registration usually involves associating keying material with the entity.
4. *key generator* – creates public/private key pairs (and symmetric keys or passwords). This may be part of the user entity, part of the CA, or an independent trusted system component.
5. *certificate directory* – a certificate database or server accessible for read-access by users. The CA may supply certificates to (and maintain) the database, or users may manage their own database entries (under appropriate access control).

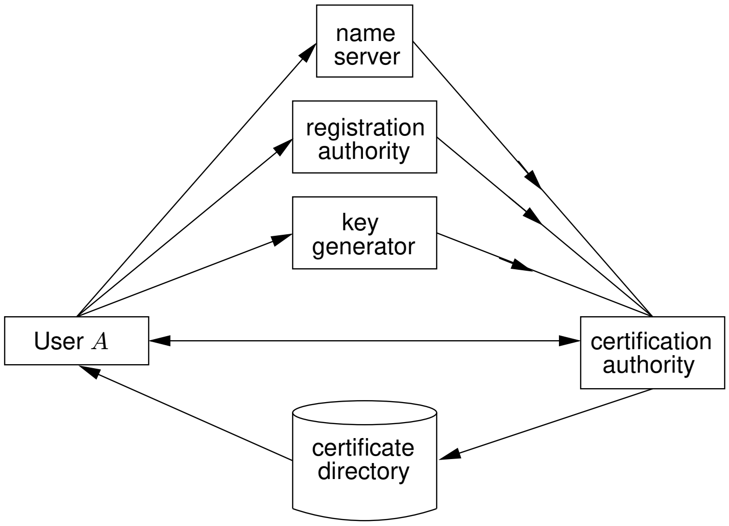

Figure 13.3: *Third party services related to public-key certification.*

(iii) Other basic third party functions

Additional basic functions a trusted third party may provide include:

1. *key server* (*authentication server*) – facilitates key establishment between other parties, including for entity authentication. Examples include *KDCs* and *KTCs* (§13.2.3).
2. *key management facility* – provides a number of services including storage and archival of keys, audit collection and reporting tools, and (in conjunction with a certification authority or CA) enforcement of life cycle requirements including updating and revoking keys. The associated key server or certification authority may provide a record (audit trail) of all events related to key generation and update, certificate generation and revocation, etc.

13.5 Note (*key access server*) A key server may be generalized to a *key access server*, providing shared keys under controlled access to individual members of groups of two or more parties, as follows. A key K is securely deposited with the server by party A along with an access control list specifying entities authorized to access it. The server stores the key and the associated list. Subsequently, entities contact the server and request the key by referencing a key identifier supplied by A. Upon entity authentication, the server grants access to the keying material (using KTC-like functionality) if the entity is authorized.

13.6 Note (*digital enveloping of files*) A key access server may be employed to store a key K used to symmetrically encrypt a file. The source party A may make the (encrypted) file available by attaching it to the encrypted key, posting it to a public site, or communicating it independently over a distinct (unsecured) channel. Retrieval of the key from the server by an authorized party then allows that party access to the (decrypted) file. The same end goal can be attained by public-key techniques directly, without key access servers, as follows: A encrypts the file under K as above; asymmetrically encrypts K using the intended recipient's public encryption key (or recipients' keys); and includes the encrypted key(s) in a header field preceding the encrypted file.

13.7 Remark (*levels of trust vs. competency*) Various third party services require different types of trust and competency in the third party. For example, a third party possessing secret decryption keys (or entity authentication keys) must be trusted not to disclose encrypted information (or impersonate users). A third party required (only) to bind an encryption public key to an identity must still be trusted not to create false associations and thereafter impersonate an entity. In general, three levels of trust in a third party T responsible for certifying credentials for users may be distinguished. Level 1: T knows each user's secret key. Level 2: T does not know users' secret keys, but can create false credentials without detection. Level 3: T does not know users' secret keys, and generation of false credentials is detectable.

(iv) Advanced third party functions

Advanced service roles which may be provided by trusted third parties, discussed further in §13.8, include:

1. *timestamp agent* – used to assert the existence of a specified document at a certain point in time, or affix a trusted date to a transaction or digital message.
2. *notary agent* – used to verify digital signatures at a given point in time to support non-repudiation, or more generally establish the truth of any statement (which it is trusted on or granted jurisdiction over) at a given point in time.
3. *key escrow agent* – used to provide third-party access to users' secret keys under special circumstances. Here distinction is usually made between key types; for example, encryption private keys may need to be escrowed but not signature private keys (cf. Remark 13.32).

13.2.5 Tradeoffs among key establishment protocols

A vast number of key establishment protocols are available (Chapter 12). To choose from among these for a particular application, many factors aside from cryptographic security may be relevant. §12.2.2 discusses different types of assurances provided, and characteristics useful in comparing protocols.

In selected key management applications, hybrid protocols involving both symmetric and asymmetric techniques offer the best alternative (e.g., Protocol 12.44; see also Note 13.6). More generally, the optimal use of available techniques generally involves combining symmetric techniques for bulk encryption and data integrity with public-key techniques for signatures and key management.

Public-key vs. symmetric-key techniques (in key management)

Primary advantages offered by public-key (vs. symmetric-key) techniques for applications related to key management include:

1. *simplified key management*. To encrypt data for another party, only the encryption public key of that party need be obtained. This simplifies key management as only authenticity of public keys is required, not their secrecy. Table 13.3 illustrates the case for encryption keys. The situation is analogous for other types of public-key pairs, e.g., signature key pairs.
2. *on-line trusted server not required*. Public-key techniques allow a trusted on-line server to be replaced by a trusted off-line server plus any means for delivering authentic public keys (e.g., public-key certificates and a public database provided by an untrusted on-line server). For applications where an on-line trusted server is not mandatory, this may make the system more amenable to scaling, to support very large numbers of users.
3. *enhanced functionality*. Public-key cryptography offers functionality which typically cannot be provided cost-effectively by symmetric techniques (without additional on-line trusted third parties or customized secure hardware). The most notable such features are non-repudiation of digital signatures, and true (single-source) data origin authentication.

	Symmetric keys		Asymmetric keys	
	secrecy	authenticity	secrecy	authenticity
encryption key	yes	yes	no	yes
decryption key	yes	yes	yes	yes

Table 13.3: *Key protection requirements: symmetric-key vs. public-key systems.*

Figure 13.4 compares key management for symmetric-key and public-key encryption. The pairwise secure channel in Figure 13.4(a) is often a trusted server with which each party communicates. The pairwise authentic channel in Figure 13.4(b) may be replaced by a public directory through which public keys are available via certificates; the public key in this case is typically used to encrypt a symmetric data key (cf. Note 13.6).

13.3 Techniques for distributing confidential keys

Various techniques and protocols are available to distribute cryptographic keys whose confidentiality must be preserved (both private keys and symmetric keys). These include the use of key layering (§13.3.1) and symmetric-key certificates (§13.3.2).

13.3.1 Key layering and cryptoperiods

Table 13.2 (page 545) may be used to classify keys based on usage. The class "confidentiality" may be sub-classified on the nature of the information being protected: user data vs. keying material. This suggests a natural *key layering* as follows:

1. *master keys* – keys at the highest level in the hierarchy, in that they themselves are not cryptographically protected. They are distributed manually or initially installed and protected by procedural controls and physical or electronic isolation.

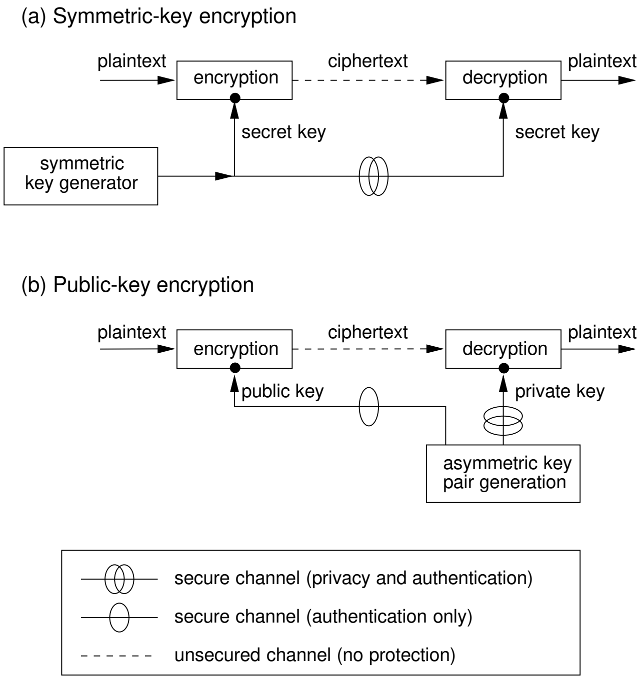

Figure 13.4: *Key management: symmetric-key vs. public-key encryption.*

2. *key-encrypting keys* – symmetric keys or encryption public keys used for key transport or storage of other keys, e.g., in the key transport protocols of Chapter 12. These may also be called *key-transport keys*, and may themselves be secured under other keys.
3. *data keys* – used to provide cryptographic operations on user data (e.g., encryption, authentication). These are generally short-term symmetric keys; however, asymmetric signature private keys may also be considered data keys, and these are usually longer-term keys.

The keys at one layer are used to protect items at a lower level. This constraint is intended to make attacks more difficult, and to limit exposure resulting from compromise of a specific key, as discussed below.

13.8 Note (*protection of key-encrypting keys*) Compromise of a key-encrypting key (and moreover, a master key as a special case thereof) affects all keys protected thereunder. Consequently, special measures are used to protect master keys, including severely limiting access and use, hardware protection, and providing access to the key only under shared control (§12.7.1).

13.9 Example (*key layering with master and terminal keys*) Assume each terminal X from a predefined set shares a key-encrypting key (*terminal key*) K_X with a trusted central node C, and that C stores an encrypted list of all terminal keys under a master key K_M. C may then provide a session key to terminals X and Y as follows. C obtains a random value R (possibly from an external source) and defines the session key to be $S = D_{K_M}(R)$, the decryption of R under K_M. Using K_M, C decrypts the key list to obtain K_X, computes S

from R, then encrypts S under K_X and transmits it to X. S is analogously transmitted to Y, and can be recovered by both X and Y. □

Cryptoperiods, long-term keys, and short-term keys

13.10 Definition The *cryptoperiod* of a key is the time period over which it is valid for use by legitimate parties.

Cryptoperiods may serve to:

1. limit the information (related to a specific key) available for cryptanalysis;
2. limit exposure in the case of compromise of a single key;
3. limit the use of a particular technology to its estimated effective lifetime; and
4. limit the time available for computationally intensive cryptanalytic attacks (in applications where long-term key protection is not required).

In addition to the key layering hierarchy above, keys may be classified based on temporal considerations as follows.

1. *long-term keys*. These include master keys, often key-encrypting keys, and keys used to facilitate key agreement.
2. *short-term* keys. These include keys established by key transport or key agreement, and often used as data keys or *session keys* for a single communications session. See Remark 13.11.

In general, communications applications involve short-term keys, while data storage applications require longer-term keys. Long-term keys typically protect short-term keys. Diffie-Hellman keys are an exception in some cases (see §12.6.1). Cryptoperiods limit the use of keys to fixed periods, after which they must be replaced.

13.11 Remark (*short-term use vs. protection*) The term *short* as used in short-term keys refers to the intended time of the key usage by legitimate parties, rather than the *protection lifetime* (cf. §13.7.1). For example, an encryption key used for only a single session might nonetheless be required to provide protection sufficient to withstand long-term attack (perhaps 20 years), whereas if signatures are verified immediately and never checked again, a signature key may need to provide protection only for a relatively short period of time. The more severe the consequences of a secret key being disclosed, the greater the reward to an adversary for obtaining access to it, and the greater the time or level of effort an adversary will invest to do so. (See also §12.2.2, and §12.2.3 on perfect forward secrecy.)

13.3.2 Key translation centers and symmetric-key certificates

Further to centralized key management discussed in §13.2.3, this section considers techniques involving key translation centers, including use of symmetric-key certificates.

(i) Key translation centers

A key translation center (KTC) T is a trusted server which allows two parties A and B, which do not directly share keying material, to establish secure communications through use of long-term keys K_{AT} and K_{BT} they respectively share with T. A may send a confidential message M to B using Protocol 13.12. If M is a key K, this provides a key transfer protocol (cf. §13.2.3); thus, KTCs provide translation of keys or messages.

13.12 Protocol Message translation protocol using a KTC

SUMMARY: A interacts with a trusted server (KTC) T and party B.
RESULT: A transfers a secret message M (or session key) to B. See Note 13.13.

1. *Notation.* E is a symmetric encryption algorithm. M may be a session key K.
2. *One-time setup.* A and T share key K_{AT}. Similarly B and T share K_{BT}.
3. *Protocol messages.*

$$\begin{array}{lll} A \rightarrow T: & A,\ E_{K_{AT}}(B, M) & (1) \\ A \leftarrow T: & E_{K_{BT}}(M, A) & (2) \\ A \rightarrow B: & E_{K_{BT}}(M, A) & (3) \end{array}$$

4. *Protocol actions.*
   (a) A encrypts M (along with the identifier of the intended recipient) under K_{AT}, and sends this to T with its own identifier (to allow T to look up K_{AT}).
   (b) Upon decrypting the message, T determines it is intended for B, looks up the key (K_{BT}) of the indicated recipient, and re-encrypts M for B.
   (c) T returns the translated message for A to send to (or post in a public site for) B; alternatively, T may send the response to B directly.

Only one of A and B need communicate with T. As an alternative to the protocol as given, A may send the first message to B directly, which B would then relay to T for translation, with T responding directly to B.

13.13 Note (*security of Protocol 13.12*)

(i) The identifier A, corresponding to the key under which message (1) was encrypted, is included in message (2) as a secure indication (to B) of the original source. Key notarization (§13.5.2) offers a more robust method of preventing key substitution.

(ii) A recognizable distinction (e.g., re-ordering the message and identifier fields) between the format of messages (1) and (2) is required to prevent an adversary from reflecting (1) back to A as a message (3) purportedly originating from B.

(iii) Message replay is possible; attacks may be detected through the use of timestamps or sequence numbers within M. The protocol as given provides no entity authentication.

(iv) An integrity check mechanism on the encrypted text should be used to allow T to detect tampering of the cleartext identifier A in (1), as well as in (2) and (3).

(v) A chosen-text attack on key K_{BT} in (2) may be prevented by an encryption mode such as CBC, and inserting an initial field containing a random number.

(ii) Symmetric-key certificates

Symmetric-key certificates provide a means for a KTC to avoid the requirement of either maintaining a secure database of user secrets (or duplicating such a database for multiple servers), or retrieving such keys from a database upon translation requests.

As before, associated with each party B is a key K_{BT} shared with T, which is now embedded in a *symmetric-key certificate* $E_{K_T}(K_{BT}, B)$ encrypted under a symmetric master key K_T known only to T. (A lifetime parameter L could also be included in the certificate as a validity period.) The certificate serves as a memo from T to itself (who alone can open it), and is given to B so that B may subsequently present it back to T precisely when required to access B's symmetric key K_{BT} for message translation. Rather than storing all user keys, T now need securely store only K_T.

Symmetric-key certificates may be used in Protocol 13.12 by changing only the first message as below, where $SCert_A = E_{K_T}(K_{AT}, A)$, $SCert_B = E_{K_T}(K_{BT}, B)$:

$$A \to T: \quad SCert_A,\ E_{K_{AT}}(B, M),\ SCert_B \quad (1)$$

A public database may be established with an entry specifying the name of each user and its corresponding symmetric-key certificate. To construct message (1), A retrieves B's symmetric-key certificate and includes this along with its own. T carries out the translation as before, retrieving K_{AT} and K_{BT} from these certificates, but now also verifies that A's intended recipient B as specified in $E_{K_{AT}}(B, M)$ matches the identifier in the supplied certificate $SCert_B$.

13.14 Remark (*public-key functionality via symmetric techniques*) The trusted third party functionality required when using symmetric-key certificates may be provided by per-user tamper-resistant hardware units keyed with a common (user-inaccessible) master key K_T. The trusted hardware unit H_A of each user A generates a symmetric-key certificate $SCert_A = E_{K_T}(K_{AT}, A)$, which is made available to B when required. H_B decrypts the certificate to recover K_{AT} (inaccessible to B) and the identity A (accessible to B). By design, H_B is constrained to use other users' keys $K_{AT} = K_A$ solely for verification functions (e.g., MAC verification, message decryption). K_A then functions as A's public key (cf. Example 13.36), allowing data origin authentication with non-repudiation; an adjudicator may resolve disputes given a hardware unit containing K_T, a disputed (message, signature) pair, and the authentic value $SCert_A$ from H_A.

13.15 Remark (*symmetric-key vs. public-key certificates*) Symmetric-key certificates differ from public-key certificates as follows: they are symmetric-key encrypted under T's master key (vs. signed using T's private key); the symmetric key within may be extracted only by T (vs. many parties being able to verify a public-key certificate); and T is required to be on-line for key translation (vs. an off-line certification authority). In both cases, certificates may be stored in a public directory.

13.4 Techniques for distributing public keys

Protocols involving public-key cryptography are typically described assuming *a priori* possession of (authentic) public keys of appropriate parties. This allows full generality among various options for acquiring such keys. Alternatives for distributing explicit public keys with guaranteed or verifiable authenticity, including public exponentials for Diffie-Hellman key agreement (or more generally, public parameters), include the following.

1. *Point-to-point delivery over a trusted channel.* Authentic public keys of other users are obtained directly from the associated user by personal exchange, or over a direct channel, originating at that user, and which (procedurally) guarantees integrity and authenticity (e.g., a trusted courier or registered mail). This method is suitable if used infrequently (e.g., one-time user registration), or in small closed systems. A related method is to exchange public keys and associated information over an untrusted electronic channel, and provide authentication of this information by communicating a hash thereof (using a collision-resistant hash function) via an independent, lower-bandwidth authentic channel, such as a registered mail.

Drawbacks of this method include: inconvenience (elapsed time); the requirement of non-automated key acquisition prior to secured communications with each new party (chronological timing); and the cost of the trusted channel.

2. *Direct access to a trusted public file (public-key registry).* A public database, the integrity of which is trusted, may be set up to contain the name and authentic public key of each system user. This may be implemented as a public-key registry operated by a trusted party. Users acquire keys directly from this registry.
   While remote access to the registry over unsecured channels is acceptable against passive adversaries, a secure channel is required for remote access in the presence of active adversaries. One method of authenticating a public file is by tree authentication of public keys (§13.4.1).
3. *Use of an on-line trusted server.* An on-line trusted server provides access to the equivalent of a public file storing authentic public keys, returning requested (individual) public keys in signed transmissions; confidentiality is not required. The requesting party possesses a copy of the server's signature verification public key, allowing verification of the authenticity of such transmissions.
   Disadvantages of this approach include: the trusted server must be on-line; the trusted server may become a bottleneck; and communications links must be established with both the intended communicant and the trusted server.
4. *Use of an off-line server and certificates.* In a one-time process, each party A contacts an off-line trusted party referred to as a *certification authority* (CA), to register its public key and obtain the CA's signature verification public key (allowing verification of other users' certificates). The CA certifies A's public key by binding it to a string identifying A, thereby creating a certificate (§13.4.2). Parties obtain authentic public keys by exchanging certificates or extracting them from a public directory.
5. *Use of systems implicitly guaranteeing authenticity of public parameters.* In such systems, including identity-based systems (§13.4.3) and those using implicitly certified keys (§13.4.4), by algorithmic design, modification of public parameters results in detectable, non-compromising failure of cryptographic techniques (see Remark 13.26).

The following subsections discuss the above techniques in greater detail. Figure 13.7 (page 564) provides a comparison of the certificate-based approach, identity-based systems, and the use of implicitly-certified public keys.

13.4.1 Authentication trees

Authentication trees provide a method for making public data available with verifiable authenticity, by using a tree structure in conjunction with a suitable hash function, and authenticating the root value. Applications include:

1. *authentication of public keys* (as an alternative to public-key certificates). An authentication tree created by a trusted third party, containing users' public keys, allows authentication of a large number of such keys.
2. *trusted timestamping service*. Creation of an authentication tree by a trusted third party, in a similar way, facilitates a trusted timestamping service (see §13.8.1).
3. *authentication of user validation parameters*. Creation of a tree by a single user allows that user to publish, with verifiable authenticity, a large number of its own public validation parameters, such as required in one-time signature schemes (see §11.6.3).

To facilitate discussion of authentication trees, binary trees are first introduced.

Binary trees

A *binary tree* is a structure consisting of vertices and directed edges. The vertices are divided into three types:

1. a *root vertex*. The root has two edges directed towards it, a left and a right edge.
2. *internal vertices*. Each internal vertex has three edges incident to it – an upper edge directed away from it, and left and right edges directed towards it.
3. *leaves*. Each leaf vertex has one edge incident to it, and directed away from it.

The vertices incident with the left and right edges of an internal vertex (or the root) are called the *children* of the internal vertex. The internal (or root) vertex is called the *parent* of the associated children. Figure 13.5 illustrates a binary tree with 7 vertices and 6 edges.

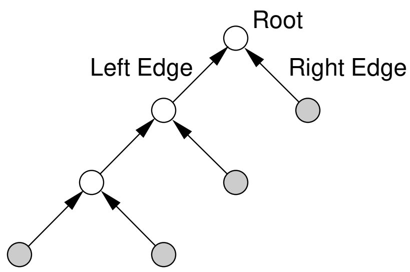

Figure 13.5: *A binary tree (with 4 shaded leaves and 3 internal vertices).*

13.16 Fact There is a unique directed path from any non-root vertex in a binary tree to the root vertex.

Constructing and using authentication trees

Consider a binary tree T which has t leaves. Let h be a collision-resistant hash function. T can be used to authenticate t public values, $Y_1, Y_2, \dots, Y_t$, by constructing an *authentication tree* T^* as follows.

1. Label each of the t leaves by a unique public value Y_i.
2. On the edge directed away from the leaf labeled Y_i, put the label $h(Y_i)$.
3. If the left and right edge of an internal vertex are labeled h_1 and h_2, respectively, label the upper edge of the vertex $h(h_1 \| h_2)$.
4. If the edges directed toward the root vertex are labeled u_1 and u_2, label the root vertex $h(u_1 \| u_2)$.

Once the public values are assigned to leaves of the binary tree, such a labeling is well-defined. Figure 13.6 illustrates an authentication tree with 4 leaves. Assuming some means to authenticate the label on the root vertex, an authentication tree provides a means to authenticate any of the t public leaf values Y_i, as follows. For each public value Y_i, there is a unique path (the *authentication path*) from Y_i to the root. Each edge on the path is a left or right edge of an internal vertex or the root. If e is such an edge directed towards vertex x, record the label on the other edge (not e) directed toward x. This sequence of labels (the *authentication path values*) used in the correct order provides the authentication of Y_i, as illustrated by Example 13.17. Note that if a single leaf value (e.g., Y_1) is altered, maliciously or otherwise, then authentication of that value will fail.

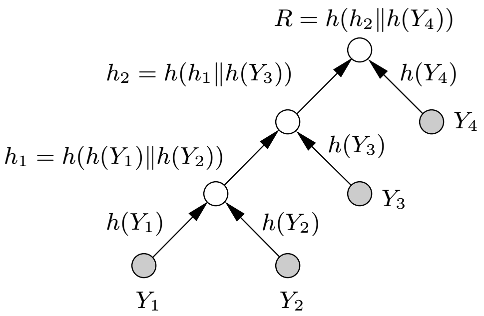

Figure 13.6: *An authentication tree.*

13.17 Example (*key verification using authentication trees*) Refer to Figure 13.6. The public value Y_1 can be authenticated by providing the sequence of labels $h(Y_2)$, $h(Y_3)$, $h(Y_4)$. The authentication proceeds as follows: compute $h(Y_1)$; next compute $h_1 = h(h(Y_1))\|h(Y_2))$; then compute $h_2 = h(h_1\|h(Y_3))$; finally, accept Y_1 as authentic if $h(h_2\|h(Y_4)) = R$, where the root value R is known to be authentic. □

The advantage of authentication trees is evident by considering the storage required to allow authentication of t public values using the following (very simple) alternate approach: an entity A authenticates t public values $Y_1, Y_2, \ldots, Y_t$ by registering each with a trusted third party. This approach requires registration of t public values, which may raise storage issues at the third party when t is large. In contrast, an authentication tree requires only a single value be registered with the third party.

If a public key Y_i of an entity A is the value corresponding to a leaf in an authentication tree, and A wishes to provide B with information allowing B to verify the authenticity of Y_i, then A must (store and) provide to B both Y_i and all hash values associated with the authentication path from Y_i to the root; in addition, B must have prior knowledge and trust in the authenticity of the root value R. These values collectively guarantee authenticity, analogous to the signature on a public-key certificate. The number of values each party must store (and provide to others to allow verification of its public key) is $\lg(t)$, as per Fact 13.19.

13.18 Fact (*depth of a binary tree*) Consider the length of (or number of edges in) the path from each leaf to the root in a binary tree. The length of the longest such path is minimized when the tree is *balanced*, i.e., when the tree is constructed such that all such paths differ in length by at most one. The length of the path from a leaf to the root in a balanced binary tree containing t leaves is about $\lg(t)$.

13.19 Fact (*length of authentication paths*) Using a balanced binary tree (Fact 13.18) as an authentication tree with t public values as leaves, authenticating a public value therein may be achieved by hashing $\lg(t)$ values along the path to the root.

13.20 Remark (*time-space tradeoff*) Authentication trees require only a single value (the root value) in a tree be registered as authentic, but verification of the authenticity of any particular leaf value requires access to and hashing of all values along the authentication path from leaf to root.

13.21 Remark (*changing leaf values*) To change a public (leaf) value or add more values to an authentication tree requires recomputation of the label on the root vertex. For large balanced trees, this may involve a substantial computation. In all cases, re-establishing trust of all users in this new root value (i.e., its authenticity) is necessary.

The computational cost involved in adding more values to a tree (Remark 13.21) may motivate constructing the new tree as an unbalanced tree with the new leaf value (or a subtree of such values) being the right child of the root, and the old tree, the left. Another motivation for allowing unbalanced trees arises when some leaf values are referenced far more frequently than others.

13.4.2 Public-key certificates

Public-key certificates are a vehicle by which public keys may be stored, distributed or forwarded over unsecured media without danger of undetectable manipulation. The objective is to make one entity's public key available to others such that its authenticity (i.e., its status as the true public key of that entity) and validity are verifiable. In practice, X.509 certificates are commonly used (see page 587). Further details regarding public-key certificates follow.

13.22 Definition A *public-key certificate* is a data structure consisting of a *data part* and a *signature part*. The data part contains cleartext data including, as a minimum, a public key and a string identifying the party (*subject entity*) to be associated therewith. The signature part consists of the digital signature of a certification authority over the data part, thereby binding the subject entity's identity to the specified public key.

The *Certification Authority* (CA) is a trusted third party whose signature on the certificate vouches for the authenticity of the public key bound to the subject entity. The significance of this binding (e.g., what the key may be used for) must be provided by additional means, such as an attribute certificate or policy statement. Within the certificate, the string which identifies the subject entity must be a unique name within the system (*distinguished name*), which the CA typically associates with a real-world entity. The CA requires its own signature key pair, the authentic public key of which is made available to each party upon registering as an authorized system user. This CA public key allows any system user, through certificate acquisition and verification, to transitively acquire trust in the authenticity of the public key in any certificate signed by that CA.

Certificates are a means for transferring trust, as opposed to establishing trust originally. The authenticity of the CA's public key may be originally provided by non-cryptographic means including personal acquisition, or through trusted couriers; authenticity is required, but not secrecy.

Examples of additional information which the certificate data part might contain include:

1. a validity period of the public key;
2. a serial number or key identifier identifying the certificate or key;
3. additional information about the subject entity (e.g., street or network address);
4. additional information about the key (e.g., algorithm and intended use);
5. quality measures related to the identification of the subject entity, the generation of the key pair, or other policy issues;
6. information facilitating verification of the signature (e.g., a signature algorithm identifier, and issuing CA's name);
7. the status of the public key (cf. revocation certificates, §13.6.3).

(i) Creation of public-key certificates

Before creating a public-key certificate for a subject entity A, the certification authority should take appropriate measures (relative to the security level required, and customary business practices), typically non-cryptographic in nature, to verify the claimed identity of A and the fact that the public key to be certified is actually that of A. Two cases may be distinguished.

Case 1: trusted party creates key pair. The trusted party creates a public-key pair, assigns it to a specific entity, and includes the public key and the identity of that entity in the certificate. The entity obtains a copy of the corresponding private key over a secure (authentic and private) channel after proving its identity (e.g., by showing a passport or trusted photo-id, in person). All parties subsequently using this certificate essentially delegate trust to this prior verification of identity by the trusted party.

Case 2: entity creates own key pair. The entity creates its own public-key pair, and securely transfers the public key to the trusted party in a manner which preserves authenticity (e.g., over a trusted channel, or in person). Upon verification of the authenticity (source) of the public key, the trusted party creates the public-key certificate as above.

13.23 Remark (*proof of knowledge of private key*) In Case 2 above, the certification authority should require proof of knowledge of the corresponding private key, to preclude (among other possible attacks) an otherwise legitimate party from obtaining, for malicious purposes, a public-key certificate binding its name to the public key of another party. For the case of signature public keys, this might be done by the party providing its own signature on a subset of the data part of the certificate; or by responding to a challenge r_1 randomized by the party itself e.g., signing $h(r_1||r_2)$ for an appropriate hash function h and a random number r_2 chosen by the signer.

(ii) Use and verification of public-key certificates

The overall process whereby a party B uses a public-key certificate to obtain the authentic public key of a party A may be summarized as follows:

1. (One-time) acquire the authentic public key of the certification authority.
2. Obtain an identifying string which uniquely identifies the intended party A.
3. Acquire over some unsecured channel (e.g. from a central public database of certificates, or from A directly), a public-key certificate corresponding to subject entity A and agreeing with the previous identifying string.
4. (a) Verify the current date and time against the validity period (if any) in the certificate, relying on a local trusted time/day-clock;
   (b) Verify the current validity of the CA's public key itself;
   (c) Verify the signature on A's certificate, using the CA's public key;
   (d) Verify that the certificate has not been revoked (§13.6.3).
5. If all checks succeed, accept the public key in the certificate as A's authentic key.

13.24 Remark (*life cycle reasons for single-key certificates*) Due to differing life cycle requirements for different types of keys (e.g., differing cryptoperiods, backup, archival, and other lifetime protection requirements – see §13.7), separate certificates are recommended for separate keys, as opposed to including several keys in a single certificate. See also Remark 13.32.

(iii) Attribute certificates

Public-key certificates bind a public key and an identity, and include additional data fields necessary to clarify this binding, but are not intended for certifying additional information. *Attribute certificates* are similar to public-key certificates, but specifically intended to allow specification of information (*attributes*) other than public keys (but related to a CA, entity, or public key), such that it may also be conveyed in a trusted (verifiable) manner. Attribute certificates may be associated with a specific public key by binding the attribute information to the key by the method by which the key is identified, e.g., by the serial number of a corresponding public-key certificate, or to a hash-value of the public key or certificate.

Attribute certificates may be signed by an *attribute certification authority*, created in conjunction with an *attribute registration authority*, and distributed in conjunction with an *attribute directory service* (cf. Figure 13.3). More generally, any party with a signature key and appropriate recognizable authority may create an attribute certificate. One application is to certify authorization information related to a public key. More specifically, this may be used, for example, to limit liability resulting from a digital signature, or to constrain the use of a public key (e.g., to transactions of limited values, certain types, or during certain hours).

13.4.3 Identity-based systems

Identity-based systems resemble ordinary public-key systems, involving a private transformation and a public transformation, but users do not have explicit public keys as before. Instead, the public key is effectively replaced by (or constructed from) a user's publicly available identity information (e.g., name and network or street address). Any publicly available information which uniquely identifies a user and can be undeniably associated with the user, may serve as the identity information.

13.25 Definition An identity-based cryptographic system (*ID-based system*) is an asymmetric system wherein an entity's public identification information (unique name) plays the role of its public key, and is used as input by a trusted authority T (along with T's private key) to compute the entity's corresponding private key.

After computing it, T transfers the entity's private key to the entity over a secure (authentic and private) channel. This private key is computed from not only the entity's identity information, but must also be a function of some privileged information known only to T (T's private key). This is necessary to prevent forgery and impersonation – it is essential that only T be able to create valid private keys corresponding to given identification information. Corresponding (authentic) publicly available system data must be incorporated in the cryptographic transformations of the ID-based system, analogous to the certification authority's public key in certificate-based systems. Figure 13.7(b) on page 564 illustrates the design of an identity-based system. In some cases, additional system-defined public data D_A must be associated with each user A in addition to its *a priori* identity ID_A (see Remark 13.27); such systems are no longer "purely" identity-based, although neither the authenticity of D_A nor ID_A need be explicitly verified.

13.26 Remark (*authenticity in ID-based systems*) ID-based systems differ from public-key systems in that the authenticity of user-specific public data is not (and need not be) explicitly verified, as is necessary for user public keys in certificate-based systems. The inherent redundancy of user public data in ID-based systems (derived through the dependence of the corresponding private key thereon), together with the use of authentic public system data, implicitly protects against forgery; if incorrect user public data is used, the cryptographic transformations simply fail. More specifically: signature verification fails, entity authentication fails, public-key encryption results in undecipherable text, and key-agreement results in parties establishing different keys, respectively, for (properly constructed) identity-based signature, authentication, encryption, and key establishment mechanisms.

The motivation behind ID-based systems is to create a cryptographic system modeling an ideal mail system wherein knowledge of a person's name alone suffices to allow mail to be sent which that person alone can read, and to allow verification of signatures that person alone could have produced. In such an ideal cryptographic system:

1. users need exchange neither symmetric keys nor public keys;
2. public directories (files of public keys or certificates) need not be kept; and
3. the services of a trusted authority are needed solely during a set-up phase (during which users acquire authentic public system parameters, to be maintained).

13.27 Remark (*ideal vs. actual ID-based systems*) A drawback in many concrete proposals of ID-based systems is that the required user-specific identity data includes additional data (an integer or public data value), denoted D_A in Figure 13.7(b), beyond an *a priori* identity ID_A. For example, see Note 10.29(ii) on Feige-Fiat-Shamir identification. Ideally, D_A is not required, as a primary motivation for identity-based schemes is to eliminate the need to transmit public keys, to allow truly non-interactive protocols with identity information itself sufficing as an authentic public key. The issue is less significant in signature and identification schemes where the public key of a claimant is not required until receiving a message from that claimant (in this case D_A is easily provided); but in this case, the advantage of identity-based schemes diminishes. It is more critical in key agreement and public-key encryption applications where another party's public key is needed at the outset. See also Remark 13.31.

13.28 Example (*ID-based system implemented using chipcards*) A simplified ID-based system based on chipcards may be run as follows. A third party T, acting as a trusted key generation system, is responsible solely for providing each user a chipcard during a set-up phase, containing that party's ID-based private key, after carrying out a thorough identity check. If no further users need be added, T may publish the public system data and cease to exist. Users are responsible for not disclosing their private keys or losing their cards. □

13.4.4 Implicitly-certified public keys

Another variation of public-key systems is asymmetric systems with *implicitly-certified public keys*. Here explicit user public keys exist (see Figure 13.7(c)), but they must be reconstructed rather than transported by public-key certificates as per certificate-based systems. For other advantages, see Remark 13.30. Examples of specific such mechanisms are given in §12.6.2. Systems with implicitly-certified public keys are designed such that:

1. Entities' public keys may be reconstructed (by other parties) from public data (which essentially replace a certificate).
2. The public data from which a public key is reconstructed includes:
   (a) public (i.e., system) data associated with a trusted party T;
   (b) the user entity's identity (or identifying information, e.g., name and address);
   (c) additional per-user public data (*reconstruction public data*).

3. The integrity of a reconstructed public key is not directly verifiable, but a "correct" public key can be recovered only from authentic user public data.

Regarding authenticity of reconstructed public keys, the system design must guarantee:

1. Alteration of either a user's identity or reconstruction public data results in recovery of a corrupted public key, which causes denial of service but not cryptographic exposure (as per Remark 13.26).
2. It is computationally infeasible for an adversary (without knowledge of T's private data) to compute a private key corresponding to any party's public key, or to construct a matching user identity and reconstruction public data for which a corresponding private key may also be computed. Reconstructed public keys are thus implicitly authenticated by construction.

13.29 Remark (*applications of implicitly-certified keys*) Implicitly-certified public keys may be used as an alternate means for distributing public keys (e.g., Diffie-Hellman keys – see §12.6.3) in various key agreement protocols, or in conjunction with identification protocols, digital signature schemes, and public-key encryption schemes.

Classes of implicitly-certified public keys

Two classes of implicitly-certified public keys may be distinguished:

1. *identity-based public keys (Class 1).* The private key of each entity A is computed by a trusted party T, based on A's identifying information and T's private key; it is also a function of A's user-specific reconstruction public data, which is fixed *a priori* by T. A's private key is then securely transferred by T to A. An example is Mechanism 12.59.
2. *self-certified public keys (Class 2).* Each entity A itself computes its private key and corresponding public key. A's reconstruction public data (rather than A's private key, as in Class 1) is computed by T as a function of the public key (transferred to T by A), A's identifying information, and T's private key. An example is Mechanism 12.61.

Class 1 requires more trust in the third party, which therein has access to users' private keys. This differs from Class 2, as emphasized by the term "self" in "self-certified", which refers to the knowledge of this key being restricted to the entity itself.

13.4.5 Comparison of techniques for distributing public keys

§13.4 began with an overview of techniques for addressing authenticity in public key distribution. The basic approaches of §13.4.2, §13.4.3, and §13.4.4 are discussed further here. Figure 13.7 illustrates corresponding classes of asymmetric signature systems, contrasting public-key systems (with explicit public keys), identity-based systems (the public key is a user's identity information), and systems with implicitly-certified public keys (an explicit public key is reconstructed from user public data).[4] The main differences are as follows:

1. Certificate-based public-key systems have explicit public keys, while ID-based systems do not; in implicitly-certified systems explicit public keys are reconstructed. The explicit public key in public-key systems (Figure 13.7(a)) is replaced by:
   (a) the triplet (D_A, ID_A, P_T) for identity-based systems (Figure 13.7(b)). ID_A is an identifying string for A, D_A is additional public data (defined by T and related to ID_A and A's private key), and P_T consists of the trusted public key (or system parameters) of a trusted authority T.

[4] While the figure focuses (for concreteness) on signature systems, concepts carry over analogously for asymmetric entity authentication, key establishment, and encryption systems.

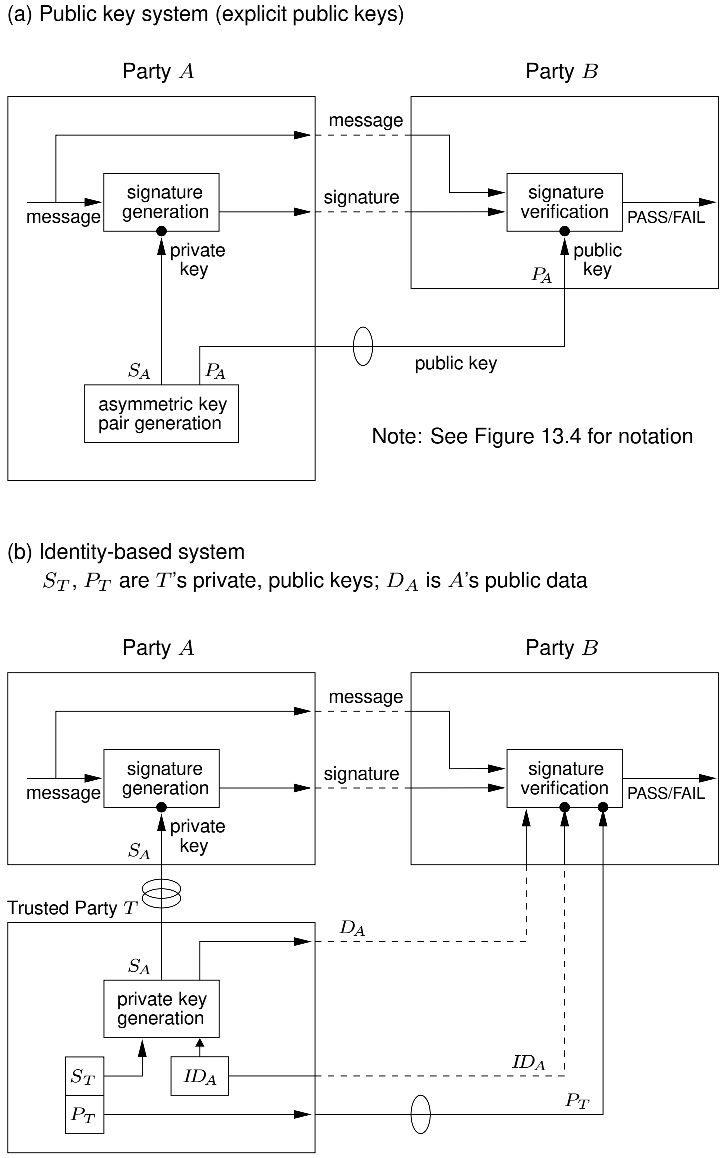

Figure 13.7: *Key management in different classes of asymmetric signature systems.*

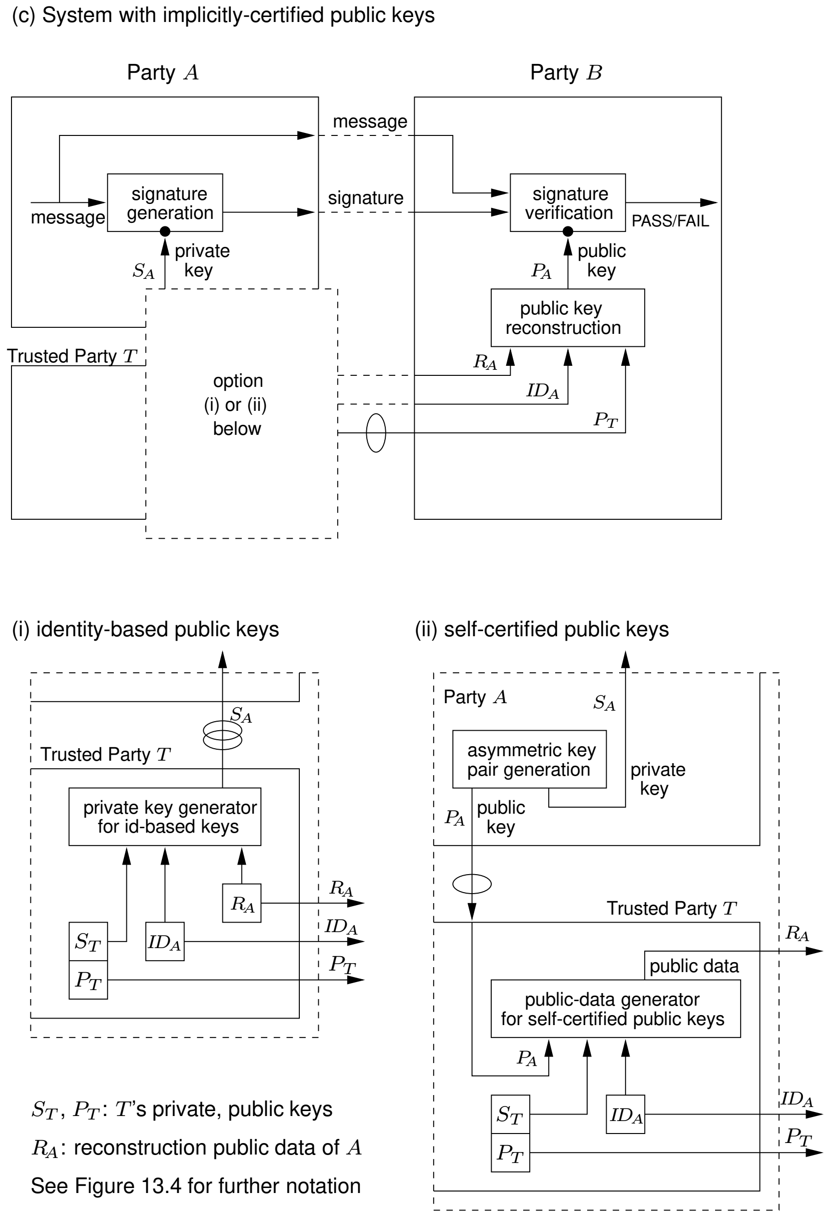

Figure 13.7: *(cont'd) Key management in different classes of asymmetric signature systems.*

   (b) the triplet (R_A, ID_A, P_T) for systems with implicitly-certified public keys (Figure 13.7(c)). In this case, an explicit public key P_A is reconstructed from these parameters. The reconstruction public data R_A plays a role analogous to the public data D_A in Figure 13.7(b).
2. The authenticity of public keys can (and must) be explicitly verified in certificate-based systems, but not (and need not) in ID-based or implicitly-certified systems.
3. The trusted authority need not know users' private keys in certificate-based public-key systems or implicitly-certified systems with self-certified public keys; but does in ID-based systems, and in implicitly-certified systems with ID-based keys.
4. Similar to identity-based systems (§13.4.3), implicitly-certified public keys (of both classes) depend on an entity's identifying information, and in this sense are also "identity-based". However, ID-based systems avoid explicit public keys entirely (a user's identity data is essentially its public key), while implicitly-certified public keys are not restricted to user identities and may be explicitly computed (and thus more easily used in conjunction with ordinary public-key schemes).
5. The two classes of implicitly-certified public keys (Figure 13.7(c)) differ in their relationship between users' reconstruction public data and private keys as follows.
   (a) Class 1: a user's private key is computed as a function of the reconstruction data, and this private key is computed by the trusted authority;
   (b) Class 2: the reconstruction data is computed as a function of the user's public key, and the corresponding private key is computed by the party itself.
6. In all three approaches, at some stage a third party which is trusted to some level (cf. Note 13.7) is required to provide a link transferring trust between users who may have never met each other and may share nothing in common other than authentic system parameters (and possibly knowledge of other users' identities).

13.30 Remark (*implicitly-certified public keys vs. public-key certificates*) Advantages of implicitly-certified public keys over public-key certificates include: possibly reduced space requirements (signed certificates require storage for signatures); possible computational savings (signature verification, as required for certificates, is avoided); and possible communications savings (e.g. if identity-based and the identity is known *a priori*). Countering these points, computation is actually required to reconstruct a public key; and additional reconstruction public data is typically required.

13.31 Remark (*key revocation in ID-based systems*) Revocation of public keys may be addressed in ID-based schemes and systems using implicitly-certified public keys by incorporating information such as a key validity period or serial number into the identification string used to compute an entity's public key (cf. Remark 13.27). The revocation issue is then analogous to that for public-key certificates. Additional information, e.g., pertaining to key usage or an associated security policy, may similarly be incorporated.

13.5 Techniques for controlling key usage

This section considers techniques for restricting keys to pre-authorized uses.

13.5.1 Key separation and constraints on key usage

Information that may be associated with cryptographic keys includes both attributes which restrict their use, and other information of operational use. These include:

1. owner of key
2. validity period (intended cryptoperiod)
3. key identifier (allowing non-cryptographic reference to the key)
4. intended use (see Table 13.2 for a coarse selection)
5. specific algorithm
6. system or environment of intended use, or authorized users of key
7. names of entities associated with key generation, registration, and certification
8. integrity checksum on key (usually part of authenticity requirement)

Key separation and the threat of key misuse

In simple key management systems, information associated with keys, including authorized uses, are inferred by context. For additional clarity or control, information explicitly specifying allowed uses may accompany distributed keys and be enforced by verification, at the time of use, that the attempted uses are authorized. If control information is subject to manipulation, it should be bound to the key by a method which guarantees integrity and authenticity, e.g., through signatures (cf. public-key certificates) or an encryption technique providing data integrity.

The principle of *key separation* is that keys for different purposes should be cryptographically separated (see Remark 13.32). The threat of key misuse may be addressed by techniques which ensure that keys are used only for those purposes pre-authorized at the time of key creation. Restrictions on key usage may be enforced by procedural techniques, physical protection (tamper-resistant hardware), or cryptographic techniques as discussed below.

Discussion of other methods in §13.5.2 includes key tags, which allow key separation with explicitly-defined uses; key variants, which separate keys without explicitly defining authorized uses; and key notarization and control vectors, which bind control information into the process by which keys are derived.

13.32 Remark (*cryptographic reasons for key separation*) A principle of sound cryptographic design is to avoid use of the same cryptographic key for multiple purposes. A key-encrypting key should not be used interchangeably as a data encryption key, since decrypted keys are not generally made available to application programs, whereas decrypted data is. Distinct asymmetric encryption and signature keys are also generally used, due to both differing life cycle requirements and cryptographic prudence. Flaws also potentially arise if: asymmetric keys are used for both signatures and challenge-response entity authentication (Remark 10.40); keys are used for both encryption and challenge-response entity authentication (chosen-text attacks); symmetric keys are used for both encryption and message authentication (Example 9.88). See also Remark 13.24.

13.5.2 Techniques for controlling use of symmetric keys

The main technique discussed below is the use of control vectors. For historical context, key tags/key variants and key notarization are also discussed.

(i) Key tags and key variants

Key tags provide a simplified method for specifying allowed uses of keys (e.g., data-encrypting vs. key-encrypting keys). A key tag is a bit-vector or structured field which accompanies and remains associated with a key over its lifetime. The tag bits are encrypted jointly with the key and thereby bound to it, appearing in plaintext form only when the key is decrypted. If the combination of tag bits and key are sufficiently short to allow encryption in a single block operation (e.g., a 56-bit key with an 8-bit tag for a 64-bit block cipher), then the inherent integrity provided by encryption precludes meaningful manipulation of the tag.

A naive method for providing key separation is to derive separate keys from a single *base key* (or *derivation key*) using additional non-secret parameters and a non-secret function. The resulting keys are called *key variants* or *derived keys*.

One technique for varying keys is *key offsetting*, whereby a key-encrypting key K is modified on a per-use basis by a counter N incremented after each use. This may prevent replay of encrypted keys. The modified key $K \oplus N$ is used to encrypt another (e.g., session) key. The recipient likewise modifies K to decrypt the session key. A second technique, complementing alternate 4-bit blocks of K commencing with the first 4 bits, is a special case of fixed-mask offsetting (Example 13.33).

13.33 Example (*key variants using fixed-mask offsets*) Suppose exactly three classes of keys are desired. Construct keys by using variations K_1 and K_2 of a master key K, with $K_1 = K \oplus v_1$, $K_2 = K \oplus v_2$, and v_1, v_2 nonsecret mask values. Using K, K_1, and K_2 to encrypt other keys then allows key separation of the latter into three classes. □

If the derivation process is invertible, the base key can be recovered from the derived key. Ideally, the derivation technique is non-reversible (one-way), implying that compromise of one derived key would not compromise the base key or other derived keys (cf. §13.7.1 on security impacts of related keys). Yet another example of key derivation (see §12.3.1) has this property: compute $K_i = E_K(r_i)$ where r_i is a random number, or replace the encryption function E by a MAC, or simply hash K and r_i using a hash function h with suitable properties.

(ii) Key notarization

Key notarization is a technique intended to prevent key substitution by requiring explicit specification of the identities of parties involved in a keying relationship. A key is authenticated with respect to these identities (preventing impersonation) by modifying a key-encrypting key such that the correct identities must be specified to properly recover the protected key. The key is said to be *sealed* with these identities. Preventing key substitution is a requirement in all (authenticated) key establishment protocols. Notarization requires proper control information for accurate recovery of encrypted keys, providing implicit protection analogous to implicitly-certified public keys (§13.4.4).

The basic technique (*simple key notarization*) involves a trusted server (notary), or one of the parties sharing the key, using a key-encrypting key K to encrypt a session key S, intended for use with the originating party i and the recipient j, as: $E_{K \oplus (i||j)}(S)$. Here i and j are assumed to identify unique entities in the given system. The party intending to recover

S from this must share K and explicitly specify i and j in the correct order, otherwise a random key will be recovered. The analogy to a notary originated from the assumption that the third party properly authenticates the identities of the intended parties, and then provides a session key which may only be recovered by these parties. A more involved process, key notarization with offsetting, is given in Example 13.34

13.34 Example (*key notarization with offsetting*) Let E be a block cipher operating on 64-bit blocks with 64-bit key, $K = K_L||K_R$ be a 128-bit key-encrypting key, N a 64-bit counter, and $i = i_L||i_R$, $j = j_L||j_R$ 128-bit source and destination identifiers. For key notarization with offsetting, compute: $K_1 = E_{K_R \oplus i_L}(j_R) \oplus K_L \oplus N$, $K_2 = E_{K_L \oplus j_L}(i_R) \oplus K_R \oplus N$. The resulting 128-bit *notarized key* (K_1, K_2) then serves as a key-encrypting key in two-key triple-encryption. The leftmost terms $f_1(K_R, i, j)$ and $f_2(K_L, i, j)$ in the computation of K_1, K_2 above are called *notary seals*, which, when combined with K_L and K_R, respectively, result in quantities analogous to those used in simple key notarization (i.e., functions of K, i, j). For K a 64-bit (single-length) key, the process is modified as follows: using $K_L = K_R = K$, compute the notary seals $f_1(K_R, i, j)$, $f_2(K_L, i, j)$ as above, concatenate the leftmost 32 bits of f_1 with the rightmost of f_2 to obtain f, then compute $f \oplus K \oplus N$ as the notarized key. □

(iii) Control vectors

While key notarization may be viewed as a mechanism for establishing authenticated keys, *control vectors* provide a method for controlling the use of keys, by combining the idea of key tags with the mechanism of simple key notarization. Associated with each key S is a control vector C, which is a data field (similar to a key tag) defining the authorized uses of the key (effectively *typing* the key). It is bound to S by varying a key-encrypting key K before encryption: $E_{K \oplus C}(S)$.

Key decryption thus requires the control vector be properly specified, as well as the correct key-encrypting key; if the combined quantity $K \oplus C$ is incorrect, a spurious key of no advantage to an adversary is recovered. Cryptographically binding the control vector C to S at the time of key generation prevents unauthorized manipulation of C, assuming only authorized parties have access to the key-encrypting key K.

Control vectors may encompass key notarization by using one or more fields in C to specify identities. In relation to standard models for access control (Note 13.35), a control vector may be used to specify a subject's identity (S_i) and privileges ($A_{i,j}$) regarding the use of a key (K_j).

At time of use for a specific cryptographic operation, the control vector is input as well as the protected key. At this time, a check is made that the requested operation complies with the control vector; if so, the key is decrypted using the control vector. If the control vector does not match that bound to the protected key (or if K is incorrect), the recovered key $S' \neq S$ will be spurious. Security here is dependent on the assumption that checking is inseparable from use, and done within a trusted subsystem.

If the bitsize of the control vector C differs from that of the key K, a collision-resistant hash function may be used prior to coupling. This allows arbitrary length control vectors. Thus a 128-bit key K and a hash function h with 128-bit output may be used to encrypt S as: $E_{K \oplus h(C)}(S)$.

13.35 Note (*models for access control*) Several methods are available to control access to resources. The *access matrix model* uses a 2-dimensional matrix $A_{i \times j}$ with a row for each subject (S_i) and a column for each object (O_j), and relies on proper identification of subjects S_i. Each access record $A_{i,j}$ specifies the privileges entity S_i has on object O_j (e.g., an application program may have read, write, modify, or execute privileges on a file). Column j may alternately serve as an *access list* for object O_j, having entries (S_i, P_{ij}) where $P_{ij} = A_{i,j}$ specifies privileges. Another method of resource protection uses the idea of capabilities: a *capability* (O, P) specifies an object O and privilege set P related to O, and functions as a *ticket* – possession of capability (O, P) grants the holder the specified privileges, without further validation or ticket-holder identification.

13.36 Example (*sample uses of control vectors*) Control vectors may be used to provide a public-key like functionality as follows (cf. Remark 13.14). Two copies of a symmetric key are distributed, one typed to allow encryption only (or MAC generation), and a second allowing decryption only (or MAC verification). Other sample uses of control fields include: allowing random number generation; allowing ciphertext translation (e.g., in KTCs); distinguishing data encryption and key encryption keys; or incorporation of any field within a public-key certificate. □

13.37 Remark (*key verification and preventing replay*) Replay of keys distributed by key transport protocols may be countered by the same techniques used to provide uniqueness/timeliness and prevent replay of messages – sequence numbers, timestamps, and challenge-response techniques (§10.3.1). Before a key resulting from a key derivation, notarization, or control vector technique is actually used, verification of its integrity may be desirable (cf. key confirmation, §12.2). This can be achieved using standard techniques for data integrity (Figure 9.8). A simple method involves the originator sending the encryption (under the key in question) of a data item which the recipient can recognize.

13.6 Key management involving multiple domains

This section considers key management models for systems involving multiple domains or authorities, as opposed to the simpler single-domain models of §13.2.3.

13.38 Definition A *security domain* (domain) is defined as a (sub)system under the control of a single authority which the entities therein trust. The security policy in place over a domain is defined either implicitly or explicitly by its authority.

The trust that each entity in a domain has in its authority originates from, and is maintained through, an entity-specific shared secret key or password (in the symmetric case), or possession of the authority's authentic public key (in the asymmetric case). This allows secure communications channels (with guaranteed authenticity and/or confidentiality) to be established between the entity and authority, or between two entities in the same domain. Security domains may be organized (e.g., hierarchically) to form larger domains.

13.6.1 Trust between two domains

Two parties A and B, belonging to distinct security domains D_A and D_B with respective trusted authorities T_A and T_B, may wish to communicate securely (or A may wish to access resources from a distinct domain D_B). This can be reduced to the requirement that A and B either:

1. (*share a symmetric key*) establish a shared secret key K_{AB} which both trust as being known only to the other (and possibly trusted authorities); or
2. (*share trusted public keys*) acquire trust in one or more common public keys which may be used to bridge trust between the domains, e.g., allowing verification of the authenticity of messages purportedly from the other, or ensure the confidentiality of messages sent to the other.

Either of these is possible provided T_A and T_B have an existing trust relationship, based on either trusted public keys or shared secret keys.

If T_A and T_B do have an existing trust relationship, either requirement may be met by using this and other initial pairwise trust relationships, which allow secure communications channels between the pairs (A, T_A), (T_A, T_B), and (T_B, B), to be successively used to establish the objective trust relationship (A, B). This may be done by A and B essentially delegating to their respective authorities the task of acquiring trust in an entity under the other authority (as detailed below).

If T_A and T_B do not share an existing trust relationship directly, a third authority T_C, in which they both do trust, may be used as an intermediary to achieve the same end result. This is analogous to a *chain of trust* in the public-key case (§13.6.2). The two numbered options beginning this subsection are now discussed in further detail.

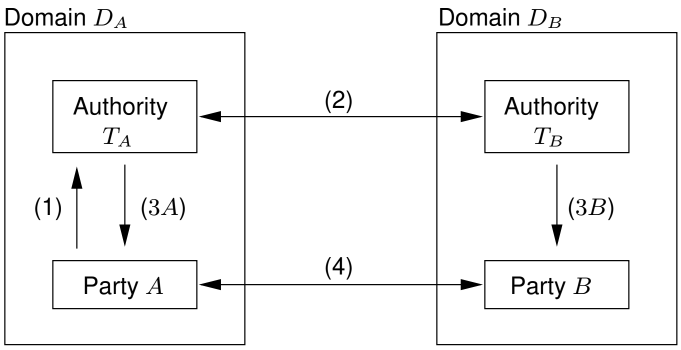

Figure 13.8: *Establishing trust between users in distinct domains.*

1. *trusted symmetric key*: Trust in a shared secret key may be acquired through a variety of authenticated key establishment techniques (see §12.3 for detailed protocols). An outline of steps by which parties A and B above may do so follows, with reference to Figure 13.8.
   (a) A makes a request to T_A to obtain a key to share with B (1).
   (b) T_A and T_B establish a short-term secret key K_{AB} (2).
   (c) T_A and T_B, respectively, distribute K_{AB} to A and B, guaranteeing secrecy and authenticity (3A, 3B).
   (d) A uses K_{AB} for secure direct communications with B (4). Message (3B) may be eliminated if its contents are relayed by T_B to A via T_A as part of the existing messages (2), (3A).

   In this case, from A's viewpoint the composition of T_A, T_B and the trust relationship (T_A, T_B) may be seen as a single (composite) authority, which A communicates with through T_A, and which plays the role of the (simple) authority in the standard case of a KDC or KTC (see §13.2.3).

2. *trusted public key*: Trust in a public key may be acquired, based on existing trust relationships, through data origin authentication by standard techniques such as digital signatures or message authentication codes. A may acquire the trusted public key of party B described above as follows (cf. Figure 13.8).
   (a) A requests from T_A the trusted public key of user B (1).
   (b) T_A acquires this from T_B, with guaranteed authenticity (2).
   (c) T_A transfers this public key to A, with guaranteed authenticity (3A).
   (d) A uses this public key to secure direct communications with B (4).

13.39 Definition A *cross-certificate* (or *CA-certificate*) is a certificate created by one certification authority (CA), certifying the public key of another CA.

13.40 Remark (*user-specific vs. domain cross-trust*) Method 2 above transfers to A trust specifically in the public key of B; this may be called a user-specific transfer of trust. Alternatively, a general transfer of trust between domains is possible as follows, assuming T_B has created a certificate C_B containing the identity and public key of B. In this case, T_A creates a cross-certificate containing the identity and public key of T_B. A, possessing the trusted signature verification key of T_A, may verify the signature on this latter certificate, thereby acquiring trust in T_B's signature verification key, and allowing A to verify and thereby trust B's public key within C_B (or the public key in any other certificate signed by T_B). Thus, user A from domain D_A (with authority T_A) acquires trust in public keys certified in D_B by T_B.

13.6.2 Trust models involving multiple certification authorities

Many alternatives exist for organizing trust relationships between certification authorities (CAs) in public-key systems involving multiple CAs. These are called *trust models* or *certification topologies*, and are logically distinct from (although possibly coincident with) communications models. (In particular, a communications link does not imply a trust relationship.) Trust relationships between CAs determine how certificates issued by one CA may be utilized or verified by entities certified by distinct CAs (in other domains). Before discussing various trust models, certificate chains are first introduced.

(i) Certificate chains and certification paths

Public-key certificates provide a means for obtaining authenticated public keys, provided the verifier has a trusted verification public key of the CA which signed the certificate. In the case of multiple certification authorities, a verifier may wish to obtain an authentic public key by verifying a certificate signed by a CA other than one for which it (originally) possesses a trusted public key. In this case, the verifier may still do so provided a *chain of certificates* can be constructed which corresponds to an unbroken chain of trust from the CA public key which the verifier does trust, to the public key it wishes to obtain trust in.

Certificate chains correspond to directed paths in the graphical representation of a CA trust model (see Figure 13.9). The goal is to find a sequence of certificates corresponding to a directed path (*certification path*) starting at the node corresponding to the CA whose public key a verifier trusts *a priori*, and ending at the CA which has signed the certificate of the public key to be verified.

13.41 Example (*illustration of certificate chain*) Consider Figure 13.9(e) on page 574. Suppose an entity A in possession of the public key P_5 of CA_5 wishes to verify the certificate of an entity B signed by CA_3, and thereby obtain trust in P_B. A directed path (CA_5, CA_4, CA_3) exists. Let $CA_5\{CA_4\}$ denote a certificate signed by CA_5 binding the name CA_4 to the public key P_4. Then the certificate chain $(CA_5\{CA_4\},\ CA_4\{CA_3\})$, along with initial trust in P_5, allows A to verify the signature on $CA_5\{CA_4\}$ to extract a trusted copy of P_4, use P_4 to verify the signature on $CA_4\{CA_3\}$ to extract a trusted copy of P_3, and then use P_3 to verify the authenticity of (the certificate containing) P_B. □

Given an initial trusted public key and a certificate to be verified, if a certificate chain is not provided to the verifier, a method is required to find (build) the appropriate chain from publicly available data, prior to actual cryptographic chain verification. This non-cryptographic task resembles that of routing in standard communications networks.

13.42 Example (*building certificate chains using cross-certificate pairs*) One search technique for finding the certification path given in Example 13.41 involves cross-certificate pairs. In a public directory, in the directory entry for each CA X, for every CA Y that either cross-certifies X or that X cross-certifies, store the certificate pair (forward, reverse) = $(CA_Y\{CA_X\},\ CA_X\{CA_Y\})$, called a *cross-certificate pair*. Here notation is as in Example 13.41, the pair consists of the forward and reverse certificates of CA_X (see page 575), and at least one of the two certificates is present. In the absence of more advanced techniques or routing tables, any existent certification path could be found by depth-first or breadth-first search of the reverse certificates in cross-certificate pairs starting at the CA whose public key the verifier possesses initially. □

As part of signature verification with certificate chains, verification of cross-certificates requires checking they themselves have not been revoked (see §13.6.3).

(ii) Trust with separate domains

Figure 13.9 illustrates a number of possible trust models for certification, which are discussed below, beginning with the case of separated domains.

Simple public-key systems involve a single certification authority (CA). Larger systems involve two or more CAs. In this case, a trust relationship between CAs must be specified in order for users under different CAs to interoperate cryptographically. By default, two distinct CAs define separate security domains as in Figure 13.9(a), with no trust relationship between domains. Users in one domain are unable to verify the authenticity of certificates originating in a separate domain.

(iii) Strict hierarchical trust model

The first solution to the lack of cryptographic interoperability between separate domains is the idea of a strict hierarchy, illustrated by Figure 13.9(b). Each entity starts with the public key of the root node – e.g., entity $E_1^{(1)}$ is now given CA_5's public key at registration, rather than that of CA_1 as in figure (a). This model is called the *rooted chain* model, as all trust chains begin at the root. It is a *centralized trust* model.

Several such rooted trees, each being a strict hierarchy, may be combined in a trust model supporting *multiple rooted trees* as in Figure 13.9(c). In this case, a cross-certificate is allowed between the roots of the trees, illustrated by a bi-directional arrow between roots. The arrow directed from CA_X to CA_Y denotes a certificate for the public key of CA_Y created by CA_X. This allows users in the tree under CA_X to obtain trust in certificates under CA_Y through certificate chains which start at CA_X and cross over to CA_Y.

In the strict hierarchical model, all entities are effectively in a single domain (defined by the root). Despite the fact that, for example, CA_1 signs the public-key certificate of $E_1^{(1)}$,

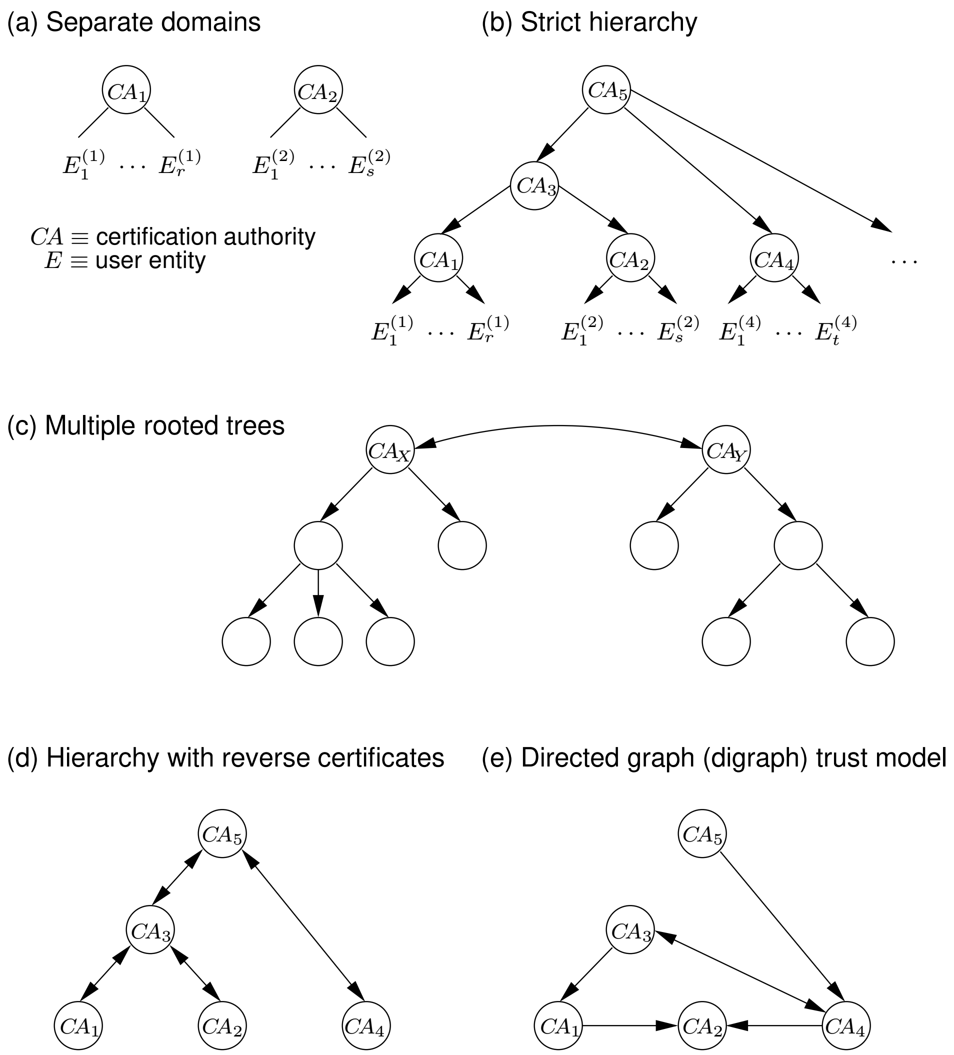

Figure 13.9: *Trust models for certification.*

$E_1^{(1)}$ trusts the root (CA_5) directly but not CA_1. $E_1^{(1)}$ trusts CA_1 only indirectly through the root. Potential drawbacks of this model include:

1. all trust in the system is dependent on the root key
2. certificate chains are required even for two entities under the same CA
3. certificate chains become long in deep hierarchies
4. a more natural model in some organizations is for trust to begin at a local node (the parent CA) rather than a distant node (the root).

(iv) Reverse certificates and the general digraph trust model

A more general hierarchical model, a *hierarchy with reverse certificates*, is illustrated in Figure 13.9(d). This resembles the strict hierarchy of Figure 13.9(b), but now each CA lower in the hierarchy also creates certificates certifying the public keys of its directly superior (*parent*) CA. Two types of certificates may then be distinguished in a hierarchy:

1. *forward certificate*. A forward certificate (relative to CA_X) is created by the CA directly above CA_X signing the public key of CA_X, and illustrated in the hierarchy by a downward arrow towards CA_X.
2. *reverse certificate*. A reverse certificate (relative to CA_X) is created by CA_X signing the public key of its immediately superior CA, and illustrated in the hierarchy by an upward arrow originating from CA_X.

In this model, each entity starts not with the public key of the root, but rather with the public key of the CA which created its own certificate, i.e., its local CA (parent). All trust chains now begin at an entity's local CA. The shortest trust chain from any entity A to any other entity B is now the path in the tree which travels upwards from A to the *least-common-ancestor* of A and B, and downwards from that node on to B.

A drawback of the hierarchical model with reverse certificates is that long certificate chains may arise between entities which are under distinct CAs even if these entities communicate frequently (e.g., consider entities under CA_1 and CA_4 in Figure 13.9(d). This situation can be ameliorated by allowing CA_1 to cross-certify CA_4 directly, even though this edge is not in the hierarchy. This is the most general model, the *directed graph (digraph) trust* model as illustrated in Figure 13.9(e). The analogy to graph theory is as follows: CAs are represented by nodes or vertices in a graph, and trust relationships by directed edges. (The *complete graph* on n vertices, with a directed edge from each vertex to every other, corresponds to complete trust, with each CA cross-certifying every other directly.)

The digraph model is a *distributed trust* model. There is no central node or root, any CA may cross-certify any other, and each user-entity begins with the trusted public key of its local CA. The concept of a hierarchy remains useful as a reference for organizing trust relationships. This model may be used to implement the other trust models discussed above, including strict hierarchies if variation is permitted in the trusted public key(s) end-user entities are provided with initially.

13.43 Remark (*assigning end-users to CAs*) In hierarchical models, one option is to specify that only CAs at the lowest level certify end-users, while internal CAs serve (only) to cross-certify other CAs. In the general digraph model, where all CAs are considered equal, it is more natural to allow every CA to certify end-users.

(v) Constraints in trust models

Trust obtained through certificate chains requires successful verification of each certificate forming a link in the chain. Once a CA (CA_X) cross-certifies the public key of another CA (CA_Y), in the absence of additional constraints, this trust extended by CA_X is transitively granted to all authorities which may be reached by certificate chains originating from

CA_Y. To limit the scope of trust extended by a single cross-certificate, a CA may impose constraints on cross-certificates it signs. Such constraints would be enforced during verification of certificate chains, and might be recorded explicitly through additional certificate fields indicating specific policies, or through attribute certificates (§13.4.2). Examples of simple constraints on cross-certificates include:

1. *limiting chain length*. A constraint may be imposed on the length of the certificate chain which may follow the cross-certificate in question. For example, a CA may limit the extent of trust granted to CAs which it directly cross-certifies by specifying, in all cross-certificates it signs, that that certificate must be the last CA-certificate in any trust chain.
2. *limiting the set of valid domains*. A set of CAs (or domain names) may be specified as valid with respect to a given cross-certificate. All CAs in a certificate chain following the cross-certificate in question may be required to belong to this set.

Certification may also be carried out relative to a *certification policy* specifying the conditions under which certification took place, including e.g., the type of authentication carried out on the certificate subject before certifying a key, and the method used to guarantee unique subject names in certificates.

13.6.3 Certificate distribution and revocation

A certificate directory (cf. §13.2.4) is a database which implements a *pull* model – users extract (pull) certificates from the database as necessary. A different model of certificate distribution, the *push* model, involves certificates being sent out (pushed) to all users upon certificate creation or periodically; this may be suitable for closed systems. Alternatively, individual users may provide their certificates to others when specifically needed, e.g., for signature verification. In certificate-based systems with certificate revocation lists (CRLs – see below), a method for distribution of CRLs as well as certificates is required.

A certificate directory is usually viewed as an *unsecured* third party. While access control to the directory in the form of write and delete protection is necessary to allow maintenance and update without denial of service, certificates themselves are individually secured by the signatures thereon, and need not be transferred over secured channels. An exception is *on-line certificates*, which are created by a certification authority in real-time on request and have no on-going lifetime, or are distributed by a trusted party which guarantees they have not been revoked.

Certificate or CRL *caching* may be used, whereby frequently referenced items are saved in short-term local storage to avoid the cost of repeated retrievals. Cached CRLs must be refreshed sufficiently often to ensure recent revocations are known.

Certificate revocation and CRLs

Upon compromise of a secret key, damage may be minimized by preventing subsequent use of or trust in the associated keying material. (Note the implications differ between signature and encryption keys.) Here *compromise* includes any situation whereby an adversary gains knowledge of secret data. If public keys must be obtained in real-time from a trusted on-line server, the keys in question may be immediately removed or replaced. The situation involving certificates is more difficult, as all distributed copies must be effectively retracted. While (suspected or actual) key compromise may be rare, there may be other reasons a CA will prematurely dissolve its binding of a public key to a user name (i.e., *revoke* the certificate). Reasons for early termination of keying material include the associated entity leaving or changing its role within an organization, or ceasing to require authorization as a user. Techniques for addressing the problem of revoked keys include:

1. *expiration dates within certificates*. Expiration dates limit exposure following compromise. The extreme case of short validity periods resembles on-line certificates which expire essentially immediately. Short-term certificates without CRLs may be compared to long-term certificates with frequently updated CRLs.
2. *manual notification*. All system users are informed of the revoked key by out-of-band means or special channels. This may be feasible in small or closed systems.
3. *public file of revoked keys*. A public file is maintained identifying revoked keys, to be checked by all users before key use. (The authenticity of data extracted from the file may be provided by similar techniques as for public keys – see §13.4.)
4. *certificate revocation lists* (CRLs). A CRL is one method of managing a public file of revoked keys (see below).
5. *revocation certificates*. An alternative to CRLs, these may be viewed as public-key certificates containing a revocation flag and a time of revocation, serving to cancel the corresponding certificate. The original certificate may be removed from the certificate directory and replaced by the revocation certificate.

A CRL is a signed list of entries corresponding to revoked public keys, with each entry indicating the serial number of the associated certificate, the time the revocation was first made, and possibly other information such as the revocation reason. The list signature, guaranteeing its authenticity, is generated by the CA which originally issued the certificates; the CRL typically includes this name also. Inclusion of a date on the overall CRL provides an indication of its freshness. If CRLs are distributed using a pull model (e.g., via a public database), they should be issued at regular intervals (or intervals as advertised within the CRL itself) even if there are no changes, to prevent new CRLs being maliciously replaced by old CRLs.

Revoked cross-certificates may be specified on separate *authority revocation lists* (ARLs), analogous to CRLs (which are then restricted to revoked end-user certificates).

13.44 Note (*CRL segmenting*) For reasons of operational efficiency when large CRLs may arise, an option is to distribute CRLs in pieces. One technique is to use *delta-CRLs*: upon each CRL update, only new entries which have been revoked since the last issued CRL are included. This requires end-users maintain (and update) secured, local images of the current CRL. A second technique is to partition a CRL into segments based on revocation reason. A third is to segment a CRL by pre-assigning each certificate (upon creation) to a specified sub-list, with a limit $n_{\max}$ on the number of certificates pre-assigned to any segment and new segments created as required. In all cases, for each certificate, available information must indicate which CRL segment must be consulted.

13.7 Key life cycle issues

Key management is simplest when all cryptographic keys are fixed for all time. Cryptoperiods necessitate the update of keys. This imposes additional requirements, e.g., on certification authorities which maintain and update user keys. The set of stages through which a key progresses during its existence, referred to as the *life cycle* of keys, is discussed in this section.

13.7.1 Lifetime protection requirements

Controls are necessary to protect keys both during usage (cf. §13.5.2) and storage. Regarding long-term storage of keys, the duration of protection required depends on the cryptographic function (e.g., encryption, signature, data origin authentication/integrity) and the time-sensitivity of the data in question.

Security impact of dependencies in key updates

Keying material should be updated prior to cryptoperiod expiry (see Definition 13.10). Update involves use of existing keying material to establish new keying material, through appropriate key establishment protocols (Chapter 12) and key layering (§13.3.1).

To limit exposure in case of compromise of either long term secret keys or past session keys, dependencies among keying material should be avoided. For example, securing a new session key by encrypting it under the old session key is not recommended (since compromise of the old key compromises the new). See §12.2.3 regarding perfect forward secrecy and known-key attacks.

Lifetime storage requirements for various types of keys

Stored secret keys must be secured so as to provide both confidentiality and authenticity. Stored public keys must be secured such that their authenticity is verifiable. Confidentiality and authenticity guarantees, respectively countering the threats of disclosure and modification, may be provided by cryptographic techniques, procedural (trust-based) techniques, or physical protection (tamper-resistant hardware).

Signature verification public keys may require archival to allow signature verification at future points in time, including possibly after the private key ceases to be used. Some applications may require that signature private keys neither be backed up nor archived: such keys revealed to any party other than the owner potentially invalidates the property of non-repudiation. Note here that loss (without compromise) of a signature private key may be addressed by creation of a new key, and is non-critical as such a private key is not needed for access to past transactions; similarly, public encryption keys need not be archived. On the other hand, decryption private keys may require archival, since past information encrypted thereunder might otherwise be lost.

Keys used for entity authentication need not be backed up or archived. All secret keys used for encryption or data origin authentication should remain secret for as long as the data secured thereunder requires continued protection (the *protection lifetime*), and backup or archival is required to prevent loss of this data or verifiability should the key be lost.

13.7.2 Key management life cycle

Except in simple systems where secret keys remain fixed for all time, cryptoperiods associated with keys require that keys be updated periodically. Key update necessitates additional procedures and protocols, often including communications with third parties in public-key systems. The sequence of states which keying material progresses through over its lifetime is called the *key management life cycle*. Life cycle stages, as illustrated in Figure 13.10, may include:

1. *user registration* – an entity becomes an authorized member of a security domain. This involves acquisition, or creation and exchange, of initial keying material such as shared passwords or PINs by a secure, one-time technique (e.g., personal exchange, registered mail, trusted courier).

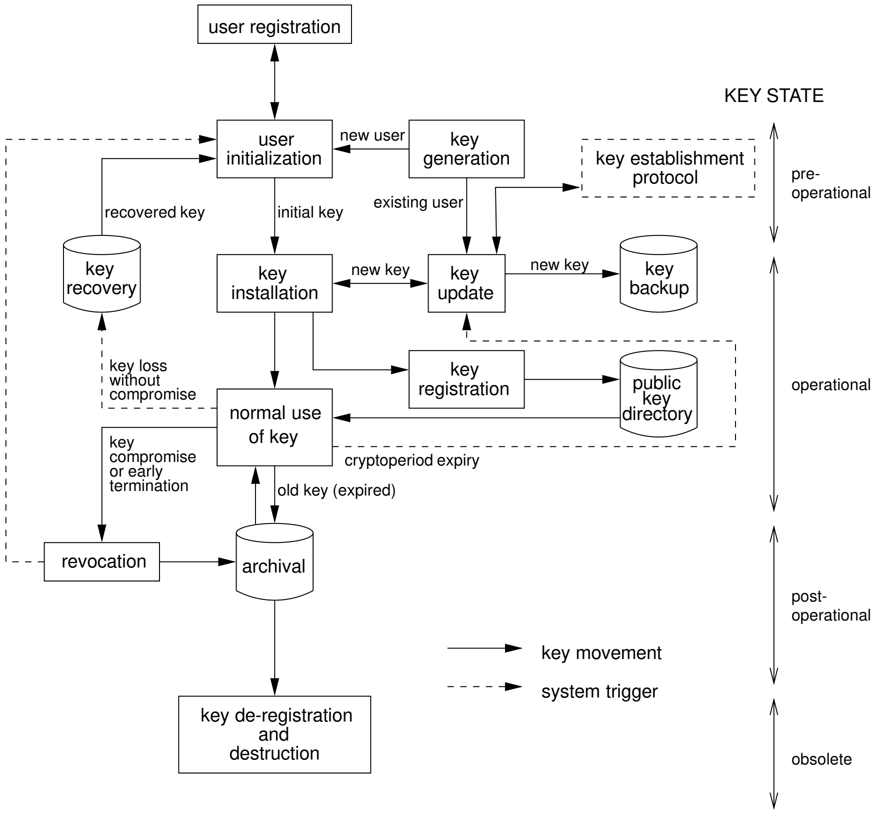

Figure 13.10: *Key management life cycle.*

2. *user initialization* – an entity initializes its cryptographic application (e.g., installs and initializes software or hardware), involving use or installation (see below) of initial keying material obtained during user registration.
3. *key generation* – generation of cryptographic keys should include measures to ensure appropriate properties for the intended application or algorithm and randomness in the sense of being predictable (to adversaries) with negligible probability (see Chapter 5). An entity may generate its own keys, or acquire keys from a trusted system component.
4. *key installation* – keying material is installed for operational use within an entity's software or hardware, by a variety of techniques including one or more of the following: manual entry of a password or PIN, transfer of a disk, read-only-memory device, chipcard or other hardware token or device (e.g., key-loader). The initial keying material may serve to establish a secure on-line session through which working keys are established. During subsequent updates, new keying material is installed to replace that in use, ideally through a secure on-line update technique.
5. *key registration* – in association with key installation, keying material may be officially recorded (by a registration authority) as associated with a unique name which distinguishes an entity. For public keys, public-key certificates may be created by a certification authority (which serves as guarantor of this association), and made available to others through a public directory or other means (see §13.4).

6. *normal use* – the objective of the life cycle is to facilitate operational availability of keying material for standard cryptographic purposes (cf. §13.5 regarding control of keys during usage). Under normal circumstances, this state continues until cryptoperiod expiry; it may also be subdivided – e.g., for encryption public-key pairs, a point may exist at which the public key is no longer deemed valid for encryption, but the private key remains in (normal) use for decryption.
7. *key backup* – backup of keying material in independent, secure storage media provides a data source for key recovery (point 11 below). Backup refers to short-term storage during operational use.
8. *key update* – prior to cryptoperiod expiry, operational keying material is replaced by new material. This may involve some combination of key generation, key derivation (§13.5.2), execution of two-party key establishment protocols (Chapter 12), or communications with a trusted third party. For public keys, update and registration of new keys typically involves secure communications protocols with certification authorities.
9. *archival* – keying material no longer in normal use may be archived to provide a source for key retrieval under special circumstances (e.g., settling disputes involving repudiation). Archival refers to off-line long-term storage of post-operational keys.
10. *key de-registration and destruction* – once there are no further requirements for the value of a key or maintaining its association with an entity, the key is de-registered (removed from all official records of existing keys), and all copies of the key are destroyed. In the case of secret keys, all traces are securely erased.
11. *key recovery* – if keying material is lost in a manner free of compromise (e.g., due to equipment failure or forgotten passwords), it may be possible to restore the material from a secure backup copy.
12. *key revocation* – it may be necessary to remove keys from operational use prior to their originally scheduled expiry, for reasons including key compromise. For public keys distributed by certificates, this involves revoking certificates (see §13.6.3).

Of the above stages, all are regularly scheduled, except key recovery and key revocation which arise under special situations.

13.45 Remark (*public-key vs. symmetric-key life cycle*) The life cycle depicted in Figure 13.10 applies mainly to public-key pairs, and involves keying material of only a single party. The life cycle of symmetric keys (including key-encrypting and session keys) is generally less complex; for example, session keys are typically not registered, backed up, revoked, or archived.

Key states within life cycle

The typical events involving keying material over the lifetime of the key define stages of the life cycle. These may be grouped to define a smaller set of *states* for cryptographic keys, related to their availability for use. One classification of key states is as follows (cf. Figure 13.10):

1. *pre-operational*. The key is not yet available for normal cryptographic operations.
2. *operational*. The key is available, and in normal use.
3. *post-operational*. The key is no longer in normal use, but off-line access to it is possible for special purposes.
4. *obsolete*. The key is no longer available. All records of the key value are deleted.

System initialization and key installation

Key management systems require an initial keying relationship to provide an initial secure channel and optionally support the establishment of subsequent working keys (long-term and short-term) by automated techniques. The initialization process typically involves non-cryptographic one-time procedures such as transfer of keying material in person, by trusted courier, or over other trusted channels.

The security of a properly architected system is reduced to the security of keying material, and ultimately to the security of initial key installation. For this reason, initial key installation may involve dual or split control, requiring co-operation of two or more independent trustworthy parties (cf. Note §13.8).

13.8 Advanced trusted third party services

This section provides further details on trusted third party services of a more advanced nature, introduced briefly in §13.2.4.

13.8.1 Trusted timestamping service

A trusted timestamping service provides a user with a dated receipt (upon presentation of a document), which thereafter can be verified by others to confirm the presentation or existence of the document at the (earlier) date of receipt. Specific applications include establishing the time of existence of documents such as signed contracts or lab notes related to patent claims, or to support non-repudiation of digital signatures (§13.8.2).

The basic idea is as follows. A trusted third party T (the *timestamp agent*) appends a timestamp t_1 to a submitted digital document or data file D, signs the composite document (thereby vouching for the time of its existence), and returns the signed document including t_1 to the submitter. Subsequent verification of T's signature then establishes, based on trust in T, the existence of the document at the time t_1.

If the data submitted for timestamping is the hash of a document, then the document content itself need not be disclosed at the time of timestamping. This also provides privacy protection from eavesdroppers in the case of submissions over an unsecured channel, and reduces bandwidth and storage costs for large documents.

13.46 Remark (*non-cryptographic timestamp service*) A similar service may be provided by non-cryptographic techniques as follows. T stores D along with a timestamp t_1, and is trusted to maintain the integrity of this record by procedural techniques. Later some party A submits the document again (now D'), and T compares D' to D on file. If these match, T declares that D' existed at the time t_1 of the retrieved timestamp.

The timestamp agent T is trusted not to disclose its signing key, and also to competently create proper signatures. An additional desirable feature is *prevention of collusion*: T should be unable to successfully collude (with any party) to undetectably back-date a document. This may be ensured using Mechanism 13.47, which combines digital signatures with tree authentication based on hashing.

13.47 Mechanism Trusted timestamping service based on tree authentication

SUMMARY: party A interacts with a trusted timestamping agent T.
RESULT: A obtains a timestamp on a digital document D.

1. A submits the hash value $h(D)$ to T. (h is a collision-resistant hash function.)
2. T notes the date and time t_1 of receipt, digitally signs the concatenation of $h(D)$ and t_1, and returns t_1 and the signature to A. (The signature is called the *certified timestamp*.) A may verify the signature to confirm T's competence.
3. At the end of each fixed period (e.g., one day), or more frequently if there is a large number n of certified timestamps, T:
   (i) computes from these an authentication tree T^* with root label R (see §13.4.1);
   (ii) returns to A the authentication path values to its certified timestamp; and
   (iii) makes the root value R widely available through a means allowing both verifiable authenticity and establishment of the time of creation t_c of T^* (e.g., publishing in a trusted dated medium such as a newspaper).
4. To allow any other party B to verify (with T's verification public key) that D was submitted at time t_1, A produces the certified timestamp. If trust in T itself is challenged (with respect to backdating t_1), A provides the authentication path values from its certified timestamp to the root R, which B may verify (see §13.4.1) against an independently obtained authentic root value R for the period t_c.

To guarantee verifiability, A should itself verify the authentication path upon receiving the path values in step 3.

13.8.2 Non-repudiation and notarization of digital signatures

The timestamping service of §13.8.1 is a document certification or document notarization service. A *notary service* is a more general service capable not only of ascertaining the existence of a document at a certain time, but of vouching for the truth of more general statements at specified points in time. The terminology originates from the dictionary definition of a *notary public* – a public official (usually a solicitor) legally authorized to administer oaths, and attest and certify certain documents. No specific legal connotation is intended in the cryptographic use of this term.

The non-repudiation aspect of digital signatures is a primary advantage of public-key cryptography. By this property, a signer is prevented from signing a document and subsequently being able to successfully deny having done so. A non-repudiation service requires specification of precise details including an adjudication process and adjudicator (judge), what evidence would be submitted to the adjudicator, and what precise process the adjudicator is to follow to render judgement on disputes. The role of an adjudicator is distinct from that of a timestamp agent or notary which generates evidence.

13.48 Remark (*origin authentication vs. non-repudiable signature*) A fundamental distinction exists between a party A being able to convince itself of the validity of a digital signature s at a point in time t_0, and that party being able to convince others at some time $t_1 \geq t_0$ that s was valid at time t_0. The former resembles data origin authentication as typically provided by symmetric-key origin authentication mechanisms, and may be accepted by a verifier as a form of authorization in an environment of mutual trust. This differs from digital signatures which are non-repudiable in the future.

Data origin authentication as provided by a digital signature is valid only while the secrecy of the signer's private key is maintained. A threat which must be addressed is a signer who intentionally discloses his private key, and thereafter claims that a previously valid signature was forged. (A similar problem exists with credit cards and other methods of authorization.) This threat may be addressed by:

1. *preventing direct access to private keys*. Preventing users from obtaining direct access to their own private keys precludes intentional disclosure. As an example, the private keys may be stored in tamper-resistant hardware, and by system design never available outside thereof.
2. *use of a trusted timestamp agent*. The party obtaining a signature on a critical document submits the signature to a timestamp agent, which affixes a timestamp to signature and then signs the concatenation of these. This establishes a time t_1 at which the critical signature may be ascertained to have existed. If the private signature key corresponding to this signature is subsequently compromised, and the compromise occurred after t_1, then the critical signature may still be considered valid relative to t_1. For reasons as given in Remark 13.49, use of a notary agent (below) may be preferable.
3. *use of a trusted notary agent*. The party obtaining a signature on a critical document (or hash thereof) submits the signature (and document or hash thereof) to an agent for *signature notarization*. The agent verifies the signature and notarizes the result by appending a statement (confirming successful signature verification) to the signature, as well as a timestamp, and signing the concatenation of the three. A reasonable period of time (clearance period) may be allowed for declarations of lost private keys, after which the notary's record of verification must be accepted (by all parties who trust the notary and verify its signature) as the truth regarding the validity of the critical signature at that point in time,[5] even should the private key corresponding to the critical signature subsequently be compromised.

For signed messages having short lifetimes (i.e., whose significance does not extend far into the future), non-repudiation is less important, and notarization may be unnecessary. For other messages, the requirement for a party to be able to re-verify signatures at a later point in time (including during or after signature keys have been updated or revoked), as well as the adjudication process related to non-repudiation of signatures, places additional demands on practical key management systems. These may include the storage or archival of keying material (e.g., keys, certificates, CRLs) possibly required as evidence at a future point in time.

A related support service is that of maintaining a record (*audit trail*) of security-related events including registration, certificate generation, key update, and revocation. Audit trails may provide sufficient information to allow resolution of disputed signatures by non-automated procedures.

13.49 Remark (*reconstructing past trust*) Both signature re-verification (relative to a past point in time) and resolution of disputes may require reconstruction of chains of trust from a past point in time. This requires access to keying material and related information for (re)constructing past chains of trust. Direct reconstruction of such past chains is unnecessary if a notarizing agent was used. The original verification of the notary establishes existence of a trust chain at that point in time, and subsequently its record thereof serves as proof of prior validity. It may be of interest (for audit purposes) to record the details of the original trust chain.

[5]More generally, the truth of the appended statement must be accepted, relative to the timestamp.

13.8.3 Key escrow

The objective of a key escrow encryption system is to provide encryption of user traffic (e.g., voice or data) such that the session keys used for traffic encryption are available to properly authorized third parties under special circumstances (“emergency access”). This grants third parties which have monitored user traffic the capability to decrypt such traffic. Wide-scale public interest in such systems arose when law enforcement agencies promoted their use to facilitate legal wiretapping of telephone calls to combat criminal activities. However, other uses in industry include recovery of encrypted data following loss of keying material by a legitimate party, or destruction of keying material due to equipment failure or malicious activities. One example of a key escrow system is given below, followed by more general issues.

(i) The Clipper key escrow system

The Clipper key escrow system involves use of the Clipper chip (or a similar tamper-resistant hardware device – generically referred to below as an escrow chip) in conjunction with certain administrative procedures and controls. The basic idea is to deposit two key components, which jointly determine an encryption key, with two trusted third parties (escrow agents), which subsequently allow (upon proper authorization) recovery of encrypted user data.

More specifically, encryption of telecommunications between two users proceeds as follows. Each party has a telephone combined with a key escrow chip. The users negotiate or otherwise establish a session key K_S which is input to the escrow chip of the party encrypting data (near end). As a function of K_S and an initialization vector (IV), the chip creates by an undisclosed method a data block called a *law enforcement access field* (LEAF). The LEAF and IV are transmitted to the far end during call set-up of a communications session. The near end escrow chip then encrypts the user data D under K_S producing $E_{K_S}(D)$, by a U.S. government classified symmetric algorithm named SKIPJACK. The far end escrow chip decrypts the traffic only if the transmitted LEAF validates properly. Such verification requires that this far end chip has access to a common family key K_F (see below) with the near end chip.

The LEAF (see Figure 13.11) contains a copy of the session key encrypted under a device-specific key K_U. K_U is generated and data-filled into the chip at the time of chip manufacture, but prior to the chip being embedded in a security product. The system meets its objective by providing third party access under proper authorization (as defined by the Key Escrow System) to the device key K_U of targeted individuals.

To derive the key K_U embedded in an escrow chip with identifier UID, two key components (K_{C1}, K_{C2}) are created whose XOR is K_U. Each component is encrypted under a key $K_{CK} = K_{N1} \oplus K_{N2}$, where K_{Ni} is input to the chip programming facility by the first and second trusted *key escrow agent*, respectively. (Used to program a number of chips, K_{Ni} is stored by the escrow agent for subsequent recovery of K_{CK}.) One encrypted key component is then given to each escrow agent, which stores it along with UID to service later requests. Stored data from both agents must subsequently be obtained by an authorized official to allow recovery of K_U (by recovering first K_{CK}, and then K_{C1}, K_{C2}, and $K_U = K_{C1} \oplus K_{C2}$).

Disclosed details of the LEAF are given in Figure 13.11. Each escrow chip contains a 32-bit device unique identifier (UID), an 80-bit device unique key (K_U), and an 80-bit family key (K_F) common to a larger collection of devices. The LEAF contains a copy of the 80-bit session key K_S encrypted under K_U, the UID, and a 16-bit encryption authenticator (EA) created by an undisclosed method; these are then encrypted under K_F. Recovery of K_S from the LEAF thus requires both K_F and K_U. The encryption authenticator is a checksum designed to allow detection of LEAF tampering (e.g., by an adversary attempting to prevent authorized recovery of K_S and thereby D).

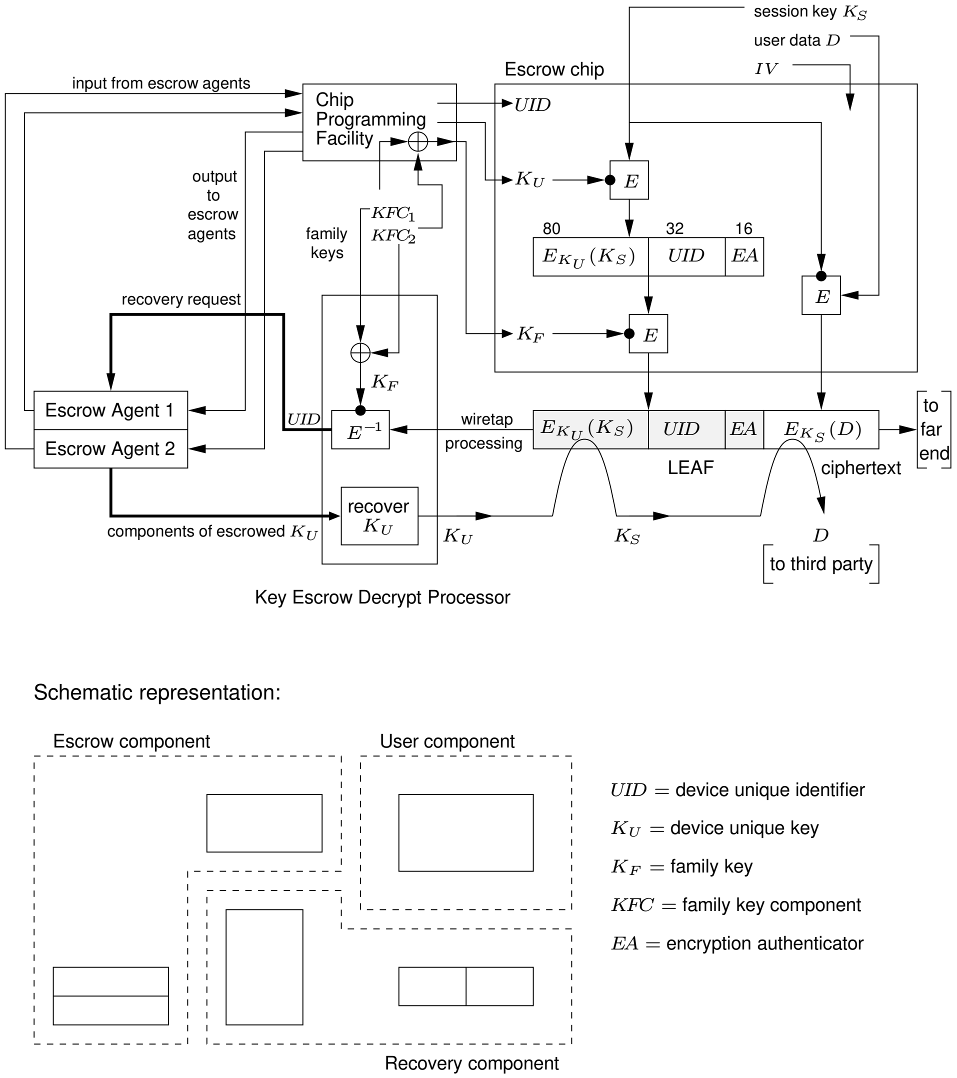

Figure 13.11: *Creation and use of LEAF for key escrow data recovery.*

(ii) Issues related to key escrow

Key escrow encryption systems may serve a wide variety of applications, and a corresponding range of features exists. Distinguishing properties of escrow systems include:

1. applicability to store-and-forward vs. real-time user communications
2. capability of real-time decryption of user traffic
3. requirement of tamper-resistant hardware or hardware with trusted clock
4. capability of user selection of escrow agents

5. user input into value of escrowed key
6. varying trust requirements in escrow agents
7. extent of user data uncovered by one escrow access (e.g., limited to one session or fixed time period) and implications thereof (e.g., hardware replacement necessary).

Threshold systems and shared control systems may be put in place to access escrowed keying information, to limit the chances of unauthorized data recovery. Key escrow systems may be combined with other life cycle functions including key establishment, and key backup and archival (cf. key access servers – Notes 13.5 and 13.6).

13.9 Notes and further references

§13.1

Davies and Price [308] provide a comprehensive treatment of key management, including overviews of ISO 8732 [578] and techniques introduced in several 1978 *IBM Systems Journal* papers [364, 804]. Early work addressing protection in communications networks and/or key management includes that of Feistel, Notz, and Smith [388], Branstad [189], Kent [665], Needham and Schroeder [923], and the surveys of Popek and Kline [998] and Voydock and Kent [1225]. Security issues in electronic funds transfer (EFT) systems for point-of-sale (POS) terminals differ from those for remote banking machines due to the weaker physical security of the former; special key management techniques such as unique (derived) transaction keys reduce the implications of terminal key compromise – see Beker and Walker [85], Davies [305], and Davies and Price [308, Ch.10]. See Meyer and Matyas [859] for general symmetric-key techniques, EFT applications, and PIN management; and Ford [414] for directory services and standards, including the X.500 Directory and X.509 Authentication Framework [626].

For an overview of key management concepts and life cycles aspects, see Fumy and Landrock [429]. Fumy and Leclerc [430] consider placement of key distribution protocols within the ISO Open Systems Interconnection (OSI) architecture. Regarding key management principles, see Abadi and Needham [1], and Anderson and Needham [31]. See Vedder [1220] for security issues and architectures relevant to wireless communications, including European digital cellular (Global System for Mobile Communications – GSM) and the Digital European Cordless Telephone (DECT) system. Regarding key management for security (authentication and encryption) in North American digital cellular systems, see IS-54 Rev B [365]. ISO 11166-1 [586] (see also comments by Rueppel [1082]) specifies key management techniques and life cycle principles for use in banking systems, and is used by the Society for Worldwide Interbank Financial Telecommunications (SWIFT).

§13.2

Various parts of ISO/IEC 11770 [616, 617, 618] contain background material on key management; Figure 13.3 is derived from an early draft of 11770-3. KDCs and KTCs were popularized by ANSI X9.17 [37]. Related to tradeoffs, Needham and Schroeder [923] compare symmetric and public-key techniques; the formalization proposed by Rueppel [1080] allows analysis of security architectures to distinguish complexity-increasing from complexity-reducing techniques.

The Kerberos authentication service (§12.3.2) includes a *ticket-granting service* whereby a client may re-authenticate itself multiple times using its long-term secret only once. The client A first acquires a *ticket-granting-ticket* through a protocol with an Authentication Server (AS). Thereafter, using a variation of Protocol 12.24, A may obtain authentication credentials for a server B from a Ticket-Granting-Server (TGS), extracting a TGS session key from the time-limited ticket to secure protocol messages with the TGS. A's long-term secret (password) need neither be cached for an extended period in memory nor re-entered, reducing the threat of its compromise; compromise of a TGS session key has time-restricted impact. See RFC 1510 [1041] for details.

Ford and Wiener [417] describe key access servers (Note 13.6), effectively an access control mechanism where the resource is a *key package*. Girault [459] mentions the three levels of trust of Remark 13.7. Digital envelopes (Note 13.6) are discussed in PKCS #7 [1072].

§13.3

Example 13.9 is from Tuchman [1198]. Davis and Swick [310] discuss symmetric-key certificates as defined herein under the name *private-key certificates* (crediting Abadi, Burrows, and Lampson) and propose protocols for their use with trusted third parties, including a password-based initial registration protocol. Predating this, Davies and Price [308, p.259] note that tamper-resistant hardware may replace the trusted third party requirement of symmetric-key certificates (Note 13.14). A generalization of Protocol 13.12 appears as Mechanism 11 of ISO/IEC 11770-2 [617], along with related KTC protocols offering additional authenticity guarantees (cf. Note 13.13(iii)); these provide KTC variations of the KDC protocols of §12.3.2).

§13.4

Diffie and Hellman [345] suggested using a trusted public file, maintained by a trusted authority with which each communicant registers once (in person), and from which authentic public keys of other users can be obtained. To secure requests by one party for the public key of another, Rivest, Shamir, and Adleman [1060] and Needham and Schroeder [923] note the trusted authority may respond via signed messages (essentially providing on-line certificates).

Authentication trees were first discussed in Merkle's thesis [851, p.126-131] (see also [852, 853]). For security requirements on hash functions used for tree authentication, see Preneel [1003, p.38]. Public-key certificates were first proposed in the 1978 B.Sc. thesis of Kohnfelder [703]; the overall thesis considers implementation and systems issues related to using RSA in practice. Kohnfelder's original certificate was an ordered triple containing a party's name, public-key information, and an authenticator, with the authenticator a signature over the value resulting from encrypting the name with the public key/algorithm in question.

X.509 certificates [626] were defined in 1988 and modified in 1993 (yielding Version 2 certificates); an *extensions* field was added by a technical corrigendum [627] in 1995 (yielding Version 3 certificates). Standard extensions for Version 3 certificates appear in an amendment to X.509 [628]; these accommodate information related to key identifiers, key usage, certificate policy, alternate names (vs. X.500 names) and name attributes, certification path constraints, and enhancements for certificate revocation including revocation reasons and CRL partitioning. For details, see Ford [416]. ANSI X9.45 [49] addresses attribute certificates. The alternative of including hard-coded attribute fields within public-key certificates is proposed in PKCS #6 [1072]; suggested attributes are listed in PKCS #9 [1072].

In 1984 Shamir [1115] formulated the general idea of asymmetric systems employing user's identities in place of public keys (*identity-based systems*), giving a concrete proposal for an ID-based signature system, and the model for an ID-based encryption scheme. Fiat and Shamir [395] combined this idea with that of zero-knowledge interactive proofs, yielding interactive identification and signature protocols. T. Okamoto [947] (based on a January 1984 paper in Japanese by Okamoto, Shiraishi, and Kawaoka [954]) independently proposed a specific entity-authentication scheme wherein a trusted center T distributes to a claimant A a secret accreditation value computed as a function of T's private key and A's identity (or unique index value). The identity-based key-agreement scheme of Maurer and Yacobi [824] (cf. §12.6 notes on page 538) is an exception to Remark 13.27: extra public data D_A is avoided, as ideally desired.

Günther [530] proposed a protocol for key agreement (Protocol 12.62) wherein users' private keys are constructed by a trusted authority T based on their identities, with corresponding Diffie-Hellman public keys reconstructed from public data provided by T (herein called implicitly-certified public keys, identity-based subclass). The protocol introduced by Girault [459], based on the key agreement protocol of Paillès and Girault [962] (itself updated by Girault and Paillès [461] and Girault [458]) similar to a protocol of Tanaka and E. Okamoto [1184], involved what he christened *self-certified public keys* (herein called implicitly-certified public keys, self-certified subclass); see Mechanism 12.61.

Related to self-certified public keys, Brands [185] has proposed *secret-key certificates* for use in so-called restrictive blind signature schemes. These involve a data triple consisting of a private key, matching public key, and an explicit (secret-key) certificate created by a trusted third party to certify the public key. Users can themselves create pairs of public keys and matching (secret-key) certificates, but cannot create valid triples. As with self-certified keys, performance of a cryptographic action relative to the public key (e.g., signing) implicitly demonstrates that the performing party knows the private key and hence that the corresponding public key was indeed issued by the trusted third party.

§13.5

Key tags are due to Jones [642]. Key separation as in Example 13.33 is based on Ehrsam et al. [364], which outlines the use of master keys, key variants, key- and data-encrypting keys. Smid [1153] introduced key notarization in the Key Notarization System (KNS), a key management system designed by the U.S. National Bureau of Standards (now NIST), and based on a Key Notarization Facility (KNF) – a KTC-like system component trusted to handle master keys, and to generate and notarize symmetric keys. Key notarization with key offsetting (Example 13.34) is from ISO 8732 [578], which is derived from ANSI X9.17 [37].

The generalization of key notarization to control vectors is due to Matyas, Meyer, and Brachtl [806], and described by Matyas [803] (also [802]), including an efficient method for allowing arbitrary length control vectors that does not penalize short vectors. The IBM proposal specifies E as two-key triple-DES, as per ANSI X9.17. Matyas notes that a second approach to implement control vectors, using a MAC computed on the control vector and the key (albeit requiring additional processing), has the property that both the control vector and the recovered key may be authenticated before the key is used. The notion of a capability (Note 13.35) was introduced in 1966 by Dennis and Van Horn [332], who also considered the access matrix model.

§13.6

Key distribution between domains is discussed in ISO/IEC 11770-1 [616]; see also Kohl and Neuman [1041] with respect to Kerberos V5, and Davis and Swick [310]. A Kerberos domain is called a *realm*; authentication of clients in one realm to servers in others is supported in V5 by *inter-realm keys*, with a concept of authentication paths analogous to public-key certification paths.

Kent [666] overviews the design and implementation of *Privacy Enhanced Mail* (PEM) (see RFC 1421-1424 [1036, 1037, 1038, 1039]), a prototyped method for adding security to Internet mail. Encryption and signature capabilities are provided. The PEM infrastructure of

RFC 1422 is based on a strict hierarchy of certification authorities, and includes specification of *Policy Certification Authorities* (PCAs) which define policies with respect to which certificates are issued. Regarding certification paths, see Tarah and Huitema [1185].

The 1988 version of X.509 [626] defines forward and reverse certificates, certificate chains, and cross-certificate pairs, allowing support for the general digraph trust model. The formal analysis of Gaarder and Snekkenes [433] highlights a practical difficulty in verifying the validity of certificates – the requirement of trusted timeclocks. For reports on implementations based on X.509 certificates, see Tardo and Alagappan [1186] and others [660, 72, 839]. Techniques for segmenting CRLs (Note 13.44) are included in the above-cited work on Version 3 certificate extensions [628]. Kohnfelder [703] noted many of the issues regarding certificate revocation in 1978 when use of certificates was first proposed, and suggested techniques to address revocation including manual notification, maintaining a public file identifying revoked keys, and use of certificate expiration dates (cf. Denning [326, p.170]).

§13.7

Matyas and Meyer [804] consider several life cycle aspects. ISO 11770-1 [616] provides a general overview of key management issues including key life cycle. ANSI X9.57 [52] provides broad discussion on certificate management, including trust models, registration, certificate chains, and life cycle aspects. ISO 10202-7 [584] specifies a key management life cycle for chipcards.

§13.8

Davies and Price [308] discuss practical issues related to registries of public keys, non-repudiation, and revocation, including the use of timestamps and notarization; see also the original works of Kohnfelder [703] and Merkle [851], which include discussion of notaries. Haber and Stornetta [535] propose two additional techniques for timestamping digital data (one enhanced by Bayer, Haber, and Stornetta [79]), although tree authentication, due to Merkle [852], appears to be preferable in practice. Benaloh and de Mare [111] introduce one-way accumulators to address the same problem.

Although key backup/archive functionality existed in earlier commercial products, the widespread study of key escrow systems began circa 1992, and combines issues related to secret sharing, key establishment, and key life cycle. For practical aspects including commercial key recovery and backup, see Walker et al. [1229] and Maher [780]. Denning and Branstad [329] provide an excellent overview of the numerous proposals to date, including a taxonomy. Among such proposals and results are those of Micali [863] (see also [862]), Leighton and Micali [745], Beth et al. [125], Desmedt [338] (but see also Knudsen and Pedersen [690]), Jefferies, Mitchell, and Walker [635], Lenstra, Winkler, and Yacobi [755], Kilian and Leighton [671], Frankel and Yung [420], and Micali and Sidney [869]. In some systems, it is required that escrow agents be able to verify that (partial) keys received are authentic, raising issues of verifiable secret sharing (see Chor et al. [259]).

The Clipper chip is a tamper-resistant hardware encryption device compliant with FIPS 185 [405], a voluntary U.S. government standard intended for sensitive but unclassified phone (voice and data) communications. FIPS 185 specifies use of the SKIPJACK encryption algorithm (80-bit key, 64-bit blocks) and LEAF creation method, the details of both of which remain classified. The two initial key escrow agents named by the U.S. Government are the National Institute of Standards and Technology (NIST) and the Department of the Treasury, Automated Systems Division. Denning and Smid [331] describe the operation of an initial key escrow system employing a chip in accordance with FIPS 185. The *Capstone* chip, a more advanced device than Clipper, implements in addition a public key agreement algorithm, DSA, SHA, high-speed general-purpose exponentiation, and a (pure noise source) random number generator; it is used in the U.S. government Multilevel Information Security System Initiative (MISSI) for secure electronic mail and other applications. Blaze [152] demonstrated that a protocol attack is possible on Clipper, requiring at most 2^{16} trial LEAF values to construct a bogus LEAF with a valid EA; Denning and Smid note this is not a threat in practical systems. For a debate on issues related to U.S. digital telephony legislation passed in October 1994 as the Communications Assistance for Law Enforcement Act (CALEA), requiring telephone companies to provide technical assistance facilitating authorized wiretapping, see Denning [328].