Chapter 12

Key Establishment Protocols

Contents in Brief

12.1	Introduction	489
12.2	Classification and framework	490
12.3	Key transport based on symmetric encryption	497
12.4	Key agreement based on symmetric techniques	505
12.5	Key transport based on public-key encryption	506
12.6	Key agreement based on asymmetric techniques	515
12.7	Secret sharing	524
12.8	Conference keying	528
12.9	Analysis of key establishment protocols	530
12.10	Notes and further references	534

12.1 Introduction

This chapter considers key establishment protocols and related cryptographic techniques which provide shared secrets between two or more parties, typically for subsequent use as symmetric keys for a variety of cryptographic purposes including encryption, message authentication, and entity authentication. The main focus is two-party key establishment, with the aid of a trusted third party in some cases. While many concepts extend naturally to multi-party key establishment including conference keying protocols, such protocols rapidly become more complex, and are considered here only briefly, as is the related area of secret sharing. Broader aspects of key management, including distribution of public keys, certificates, and key life cycle issues, are deferred to Chapter 13.

Relationships to other cryptographic techniques. Key establishment techniques known as key transport mechanisms directly employ symmetric encryption (Chapter 7) or public-key encryption (Chapter 8). Authenticated key transport may be considered a special case of message authentication (Chapter 9) with privacy, where the message includes a cryptographic key. Many key establishment protocols based on public-key techniques employ digital signatures (Chapter 11) for authentication. Others are closely related to techniques for identification (Chapter 10).

Chapter outline

The remainder of this chapter is organized as follows. §12.2 provides background material including a general classification, basic definitions and concepts, and a discussion of objectives. §12.3 and §12.4 discuss key transport and agreement protocols, respectively, based on symmetric techniques; the former includes several protocols involving an on-line trusted third party. §12.5 and §12.6 discuss key transport and agreement protocols, respectively, based on asymmetric techniques; the former includes protocols based on public-key encryption, some of which also employ digital signatures, while the latter includes selected variations of Diffie-Hellman key agreement. §12.7 and §12.8 consider secret sharing and conference keying, respectively. §12.9 addresses the analysis of key establishment protocols and standard attacks which must be countered. §12.10 contains chapter notes with references.

The particular protocols discussed provide a representative subset of the large number of practical key establishment protocols proposed to date, selected according to a number of criteria including historical significance, distinguishing merits, and practical utility, with particular emphasis on the latter.

12.2 Classification and framework

12.2.1 General classification and fundamental concepts

12.1 Definition A *protocol* is a multi-party algorithm, defined by a sequence of steps precisely specifying the actions required of two or more parties in order to achieve a specified objective.

12.2 Definition *Key establishment* is a process or protocol whereby a shared secret becomes available to two or more parties, for subsequent cryptographic use.

Key establishment may be broadly subdivided into *key transport* and *key agreement*, as defined below and illustrated in Figure 12.1.

12.3 Definition A *key transport* protocol or mechanism is a key establishment technique where one party creates or otherwise obtains a secret value, and securely transfers it to the other(s).

12.4 Definition A *key agreement* protocol or mechanism is a key establishment technique in which a shared secret is derived by two (or more) parties as a function of information contributed by, or associated with, each of these, (ideally) such that no party can predetermine the resulting value.

Additional variations beyond key transport and key agreement exist, including various forms of *key update*, such as *key derivation* in §12.3.1.

Key establishment protocols involving authentication typically require a set-up phase whereby authentic and possibly secret initial keying material is distributed. Most protocols have as an objective the creation of distinct keys on each protocol execution. In some cases, the initial keying material pre-defines a fixed key which will result every time the protocol is executed by a given pair or group of users. Systems involving such static keys are insecure under known-key attacks (Definition 12.17).

12.5 Definition *Key pre-distribution* schemes are key establishment protocols whereby the resulting established keys are completely determined *a priori* by initial keying material. In contrast, *dynamic key establishment* schemes are those whereby the key established by a fixed pair (or group) of users varies on subsequent executions.

Dynamic key establishment is also referred to as *session key establishment*. In this case the session keys are dynamic, and it is usually intended that the protocols are immune to known-key attacks.

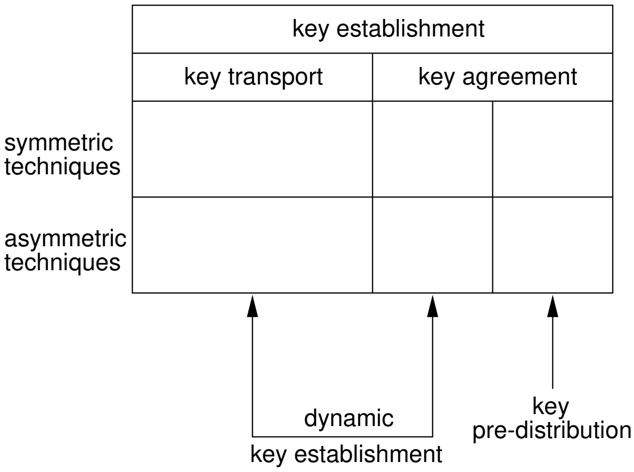

Figure 12.1: *Simplified classification of key establishment techniques.*

Use of trusted servers

Many key establishment protocols involve a centralized or trusted party, for either or both initial system setup and on-line actions (i.e., involving real-time participation). This party is referred to by a variety of names depending on the role played, including: *trusted third party*, *trusted server*, *authentication server*, *key distribution center* (KDC), *key translation center* (KTC), and *certification authority (CA)*. The various roles and functions of such trusted parties are discussed in greater detail in Chapter 13. In the present chapter, discussion is limited to the actions required of such parties in specific key establishment protocols.

Entity authentication, key authentication, and key confirmation

It is generally desired that each party in a key establishment protocol be able to determine the true identity of the other(s) which could possibly gain access to the resulting key, implying preclusion of any unauthorized additional parties from deducing the same key. In this case, the technique is said (informally) to provide *secure key establishment*. This requires both secrecy of the key, and identification of those parties with access to it. Furthermore, the identification requirement differs subtly, but in a very important manner, from that of entity authentication – here the requirement is knowledge of the identity of parties which may gain access to the key, rather than corroboration that actual communication has been established with such parties. Table 12.1 distinguishes various such related concepts, which are highlighted by the definitions which follow.

While *authentication* may be informally defined as the process of verifying that an identity is as claimed, there are many aspects to consider, including who, what, and when. *Entity authentication* is defined in Chapter 10 (Definition 10.1), which presents protocols providing entity authentication alone. *Data origin authentication* is defined in Chapter 9 (Definition 9.76), and is quite distinct.

Authentication term	Central focus
authentication	depends on context of usage
entity authentication	identity of a party, and aliveness at a given instant
data origin authentication	identity of the source of data
(implicit) key authentication	identity of party which may possibly share a key
key confirmation	evidence that a key is possessed by some party
explicit key authentication	evidence an identified party possesses a given key

Table 12.1: *Authentication summary – various terms and related concepts.*

12.6 Definition *Key authentication* is the property whereby one party is assured that no other party aside from a specifically identified second party (and possibly additional identified trusted parties) may gain access to a particular secret key.

Key authentication is independent of the actual possession of such key by the second party, or knowledge of such actual possession by the first party; in fact, it need not involve any action whatsoever by the second party. For this reason, it is sometimes referred to more precisely as *(implicit) key authentication*.

12.7 Definition *Key confirmation* is the property whereby one party is assured that a second (possibly unidentified) party actually has possession of a particular secret key.

12.8 Definition *Explicit key authentication* is the property obtained when both (implicit) key authentication and key confirmation hold.

In the case of explicit key authentication, an identified party is known to actually possess a specified key, a conclusion which cannot otherwise be drawn. Encryption applications utilizing key establishment protocols which offer only implicit key authentication often begin encryption with an initial known data unit serving as an integrity check-word, thus moving the burden of key confirmation from the establishment mechanism to the application.

The focus in key authentication is the identity of the second party rather than the value of the key, whereas in key confirmation the opposite is true. Key confirmation typically involves one party receiving a message from a second containing evidence demonstrating the latter's possession of the key. In practice, possession of a key may be demonstrated by various means, including producing a one-way hash of the key itself, use of the key in a (keyed) hash function, and encryption of a known quantity using the key. These techniques may reveal some information (albeit possibly of no practical consequence) about the value of the key itself; in contrast, methods using zero-knowledge techniques (cf. §10.4.1) allow demonstration of possession of a key while providing no additional information (beyond that previously known) regarding its value.

Entity authentication is not a requirement in all protocols. Some key establishment protocols (such as unauthenticated Diffie-Hellman key agreement) provide *none* of entity authentication, key authentication, and key confirmation. Unilateral key confirmation may always be added e.g., by including a one-way hash of the derived key in a final message.

12.9 Definition An *authenticated key establishment* protocol is a key establishment protocol (Definition 12.2) which provides key authentication (Definition 12.6).

12.10 Remark (*combining entity authentication and key establishment*) In a key establishment protocol which involves entity authentication, it is critical that the protocol be constructed to guarantee that the party whose identity is thereby corroborated is the same party with which the key is established. When this is not so, an adversary may enlist the aid of an unsuspecting authorized party to carry out the authentication aspect, and then impersonate that party in key establishment (and subsequent communications).

Identity-based and non-interactive protocols

Motivation for identity-based systems is provided in §13.4.3.

12.11 Definition A key establishment protocol is said to be *identity-based* if identity information (e.g., name and address, or an identifying index) of the party involved is used as the party's public key. A related idea (see §13.4.4) involves use of identity information as an input to the function which determines the established key.

Identity-based authentication protocols may be defined similarly.

12.12 Definition A two-party key establishment protocol is said to be *message-independent* if the messages sent by each party are independent of any per-session time-variant data (dynamic data) received from other parties.

Message-independent protocols which furthermore involve no dynamic data in the key computation are simply key pre-distribution schemes (Definition 12.5). In general, dynamic data (e.g., that received from another party) is involved in the key computation, even in message-independent protocols.

12.13 Remark (*message-independent vs. non-interactive*) Message-independent protocols include non-interactive protocols (zero-pass and one-pass protocols, i.e., those involving zero or one message but no reply), as well as some two-pass protocols. Regarding inter-party communications, some specification (explicit or otherwise) of the parties involved in key establishment is necessary even in zero-pass protocols. More subtlely, in protocols involving t users identified by a vector $(i_1, \ldots , i_t)$, the ordering of indices may determine distinct keys. In other protocols (e.g., basic Diffie-Hellman key agreement or Protocol 12.53), the cryptographic data in one party's message is independent of both dynamic data in other parties' messages and of all party-specific data including public keys and identity information.

12.2.2 Objectives and properties

Cryptographic protocols involving message exchanges require precise definition of both the messages to be exchanged and the actions to be taken by each party. The following types of protocols may be distinguished, based on objectives as indicated:

1. *authentication protocol* – to provide to one party some degree of assurance regarding the identity of another with which it is purportedly communicating;
2. *key establishment protocol* – to establish a shared secret;
3. *authenticated key establishment protocol* – to establish a shared secret with a party whose identity has been (or can be) corroborated.

Motivation for use of session keys

Key establishment protocols result in shared secrets which are typically called, or used to derive, *session keys*. Ideally, a session key is an *ephemeral* secret, i.e., one whose use is restricted to a short time period such as a single telecommunications connection (or session), after which all trace of it is eliminated. Motivation for ephemeral keys includes the following:

1. to limit available ciphertext (under a fixed key) for cryptanalytic attack;
2. to limit exposure, with respect to both time period and quantity of data, in the event of (session) key compromise;
3. to avoid long-term storage of a large number of distinct secret keys (in the case where one terminal communicates with a large number of others), by creating keys only when actually required;
4. to create independence across communications sessions or applications.

It is also desirable in practice to avoid the requirement of maintaining state information across sessions.

Types of assurances and distinguishing protocol characteristics

When designing or selecting a key establishment technique for use, it is important to consider what assurances and properties an intended application requires. Distinction should be made between functionality provided to a user, and technical characteristics which distinguish mechanisms at the implementation level. (The latter are typically of little interest to the user, aside from cost and performance implications.) Characteristics which differentiate key establishment techniques include:

1. *nature* of the authentication. Any combination of the following may be provided: entity authentication, key authentication, and key confirmation.
2. *reciprocity* of authentication. When provided, each of entity authentication, key authentication, and key confirmation may be *unilateral* or *mutual* (provided to one or both parties, respectively).
3. *key freshness*. A key is *fresh* (from the viewpoint of one party) if it can be guaranteed to be new, as opposed to possibly an old key being reused through actions of either an adversary or authorized party. This is related to key control (below).
4. *key control*. In some protocols (key transport), one party chooses a key value. In others (key agreement), the key is derived from joint information, and it may be desirable that neither party be able to control or predict the value of the key.
5. *efficiency*. Considerations include:
   (a) number of message exchanges (*passes*) required between parties;
   (b) bandwidth required by messages (total number of bits transmitted);
   (c) complexity of computations by each party (as it affects execution time); and
   (d) possibility of precomputation to reduce on-line computational complexity.
6. *third party requirements*. Considerations include (see §13.2.4):
   (a) requirement of an on-line (real-time), off-line, or no third party;
   (b) degree of trust required in a third party (e.g., trusted to certify public keys vs. trusted not to disclose long-term secret keys).
7. *type of certificate used, if any*. More generally, one may consider the manner by which initial keying material is distributed, which may be related to third party requirements. (This is often not of direct concern to a user, being an implementation detail typically providing no additional functionality.)

8. *non-repudiation*. A protocol may provide some type of receipt that keying material has been exchanged.

12.14 Remark (*efficiency vs. security*) The efficiency and security of cryptographic techniques are often related. For example, in some protocols a basic step is executed repeatedly, and security increases with the number of repetitions; in this case, the level of security attainable given a fixed amount of time depends on the efficiency of the basic step.

In the description of protocol messages, it is assumed that when the claimed source identity or source network address of a message is not explicitly included as a message field, these are known by context or otherwise available to the recipient, possibly by (unspecified) additional cleartext fields.

12.2.3 Assumptions and adversaries in key establishment protocols

To clarify the threats protocols may be subject to, and to motivate the need for specific protocol characteristics, one requires (as a minimum) an informal model for key establishment protocols, including an understanding of underlying assumptions. Attention here is restricted to two-party protocols, although the definitions and models may be generalized.

Adversaries in key establishment protocols

Communicating parties or entities in key establishment protocols are formally called *principals*, and assumed to have unique names. In addition to legitimate parties, the presence of an unauthorized "third" party is hypothesized, which is given many names under various circumstances, including: *adversary*, *intruder*, *opponent*, *enemy*, *attacker*, *eavesdropper*, and *impersonator*.

When examining the security of protocols, it is assumed that the underlying cryptographic mechanisms used, such as encryption algorithms and digital signatures schemes, are secure. If otherwise, then there is no hope of a secure protocol. An adversary is hypothesized to be not a *cryptanalyst* attacking the underlying mechanisms directly, but rather one attempting to subvert the protocol objectives by defeating the manner in which such mechanisms are combined, i.e., attacking the protocol itself.

12.15 Definition A *passive attack* involves an adversary who attempts to defeat a cryptographic technique by simply recording data and thereafter analyzing it (e.g., in key establishment, to determine the session key). An *active attack* involves an adversary who modifies or injects messages.

It is typically assumed that protocol messages are transmitted over *unprotected* (*open*) networks, modeled by an adversary able to completely control the data therein, with the ability to record, alter, delete, insert, redirect, reorder, and reuse past or current messages, and inject new messages. To emphasize this, legitimate parties are modeled as receiving messages exclusively via intervening adversaries (on every communication path, or on some subset of t of n paths), which have the option of either relaying messages unaltered to the intended recipients, or carrying out (with no noticeable delay) any of the above actions. An adversary may also be assumed capable of engaging unsuspecting authorized parties by initiating new protocol executions.

An adversary in a key establishment protocol may pursue many strategies, including attempting to:

1. deduce a session key using information gained by eavesdropping;
2. participate covertly in a protocol initiated by one party with another, and influence it, e.g., by altering messages so as to be able to deduce the key;
3. initiate one or more protocol executions (possibly simultaneously), and combine (*interleave*) messages from one with another, so as to masquerade as some party or carry out one of the above attacks;
4. without being able to deduce the session key itself, deceive a legitimate party regarding the identity of the party with which it shares a key. A protocol susceptible to such an attack is not resilient (see Definition 12.82).

In unauthenticated key establishment, impersonation is (by definition) possible. In entity authentication, where there is no session key to attack, an adversary's objective is to arrange that one party receives messages which satisfy that party that the protocol has been run successfully with a party other than the adversary.

Distinction is sometimes made between adversaries based on the type of information available to them. An *outsider* is an adversary with no special knowledge beyond that generally available, e.g., by eavesdropping on protocol messages over open channels. An *insider* is an adversary with access to additional information (e.g., session keys or secret partial information), obtained by some privileged means (e.g., physical access to private computer resources, conspiracy, etc.). A *one-time insider* obtains such information at one point in time for use at a subsequent time; a *permanent insider* has continual access to privileged information.

Perfect forward secrecy and known-key attacks

In analyzing key establishment protocols, the potential impact of compromise of various types of keying material should be considered, even if such compromise is not normally expected. In particular, the effect of the following is often considered:

1. compromise of long-term secret (symmetric or asymmetric) keys, if any;
2. compromise of past session keys.

12.16 Definition A protocol is said to have *perfect forward secrecy* if compromise of long-term keys does not compromise past session keys.

The idea of perfect forward secrecy (sometimes called *break-backward protection*) is that previous traffic is locked securely in the past. It may be provided by generating session keys by Diffie-Hellman key agreement (e.g., Protocol 12.57), wherein the Diffie-Hellman exponentials are based on short-term keys. If long-term secret keys are compromised, future sessions are nonetheless subject to impersonation by an active adversary.

12.17 Definition A protocol is said to be vulnerable to a *known-key attack* if compromise of past session keys allows either a passive adversary to compromise future session keys, or impersonation by an active adversary in the future.

Known-key attacks on key establishment protocols are analogous to known-plaintext attacks on encryption algorithms. One motivation for their consideration is that in some environments (e.g., due to implementation and engineering decisions), the probability of compromise of session keys may be greater than that of long-term keys. A second motivation is that when using cryptographic techniques of only moderate strength, the possibility exists that over time extensive cryptanalytic effort may uncover past session keys. Finally, in some systems, past session keys may be deliberately uncovered for various reasons (e.g., after authentication, to possibly detect use of the authentication channel as a covert or hidden channel).

12.3 Key transport based on symmetric encryption

This section presents a selection of key establishment protocols based on key transport (i.e., transfer of a specific key chosen *a priori* by one party) using symmetric encryption. Related techniques involving non-reversible functions are also presented. Discussion is subdivided into protocols with and without the use of a trusted server, as summarized in Table 12.2. Some of these use time-variant parameters (timestamps, sequence numbers, or random numbers) or nonces as discussed in §10.3.1.

→ Properties ↓ Protocol	server type	use of timestamps	number of messages
point-to-point key update	none	optional	1-3
Shamir's no-key protocol	none	no	3
Kerberos	KDC	yes	4
Needham-Schroeder shared-key	KDC	no	5
Otway-Rees	KDC	no	4
Protocol 13.12	KTC	no	3

Table 12.2: *Key transport protocols based on symmetric encryption.*

12.3.1 Symmetric key transport and derivation without a server

Server-less key transport based on symmetric techniques may either require that the two parties in the protocol initially share a long-term pairwise secret or not, respectively illustrated below by point-to-point key update techniques and Shamir's no-key algorithm. Other illustrative techniques are also given.

(i) Point-to-point key update using symmetric encryption

Point-to-point key update techniques based on symmetric encryption make use of a long-term symmetric key K shared *a priori* by two parties A and B. This key, initially distributed over a secure channel or resulting from a key pre-distribution scheme (e.g., see Note 12.48), is used repeatedly to establish new session keys W. Representative examples of point-to-point key transport techniques follow.

Notation: r_A, t_A, and n_A, respectively, denote a random number, timestamp, and sequence number generated by A (see §10.3.1). E denotes a symmetric encryption algorithm (see Remark 12.19). Optional message fields are denoted by an asterisk (*).

1. *key transport with one pass*:

$$A \to B : E_K(r_A) \quad (1)$$

The session key used is $W = r_A$, and both A and B obtain implicit key authentication. Additional optional fields which might be transferred in the encrypted portion include: a timestamp or sequence number to provide a freshness guarantee to B (see Remark 12.18); a field containing redundancy, to provide explicit key authentication to B or facilitate message modification detection (see Remark 12.19); and a target identifier to prevent undetectable message replay back on A immediately. Thus:

$$A \to B : E_K(r_A, t_A{}^*, B^*) \quad (1')$$

If it is desired that both parties contribute to the session key, B may send A an analogous message, with the session key computed as $f(r_A, r_B)$. Choosing f to be a one-way function precludes control of the final key value by either party, or an adversary who acquires one of r_A, r_B.

2. *key transport with challenge-response*:

$$\begin{array}{ll} A \leftarrow B : n_B & (1) \\ A \to B : E_K(r_A, n_B, B^*) & (2) \end{array}$$

If a freshness guarantee is desired but reliance on timestamps is not, a random number or sequence number, denoted n_B here, may be used to replace the timestamp in the one-pass technique; the cost is an additional message. The session key is again $W = r_A$.
If it is required that the session key W be a function of inputs from both parties, A may insert a nonce n_A preceding n_B in (2), and a third message may be added as below. (Here r_A, r_B are random numbers serving as keying material, while n_A, n_B are nonces for freshness.)

$$\begin{array}{ll} A \leftarrow B : n_B & (1) \\ A \to B : E_K(r_A, n_A, n_B, B^*) & (2) \\ A \leftarrow B : E_K(r_B, n_B, n_A, A^*) & (3) \end{array}$$

12.18 Remark (*key update vulnerabilities*) The key update techniques above do not offer perfect forward secrecy, and fail completely if the long-term key K is compromised. For this reason they may be inappropriate for many applications. The one-pass protocol is also subject to replay unless a timestamp is used.

12.19 Remark (*integrity guarantees within encryption*) Many authentication protocols which employ encryption, including the above key update protocols and Protocols 12.24, 12.26, and 12.29, require for security reasons that the encryption function has a built-in data integrity mechanism (see Figure 9.8(b) for an example, and Definition §9.75) to detect message modification.

(ii) Point-to-point key update by key derivation and non-reversible functions

Key update may be achieved by key transport as above, or by *key derivation* wherein the derived session key is based on per-session random input provided by one party. In this case, there is also a single message:

$$A \to B : r_A \quad (1)$$

The session key is computed as $W = E_K(r_A)$. The technique provides to both A and B implicit key authentication. It is, however, susceptible to known-key attacks; Remark 12.18 similarly applies. The random number r_A here may be replaced by other time-variant parameters; for example, a timestamp t_A validated by the recipient by comparison to its local clock provides an implicit key freshness property, provided the long-term key is not compromised.

Here A could control the value of W, forcing it to be x by choosing $r_A = D_K(x)$. Since the technique itself does not require decryption, E may be replaced by an appropriate keyed pseudorandom function h_K, in which case the session key may be computed as $W = h_K(r_A)$, with r_A a time-variant parameter as noted above.

In the other techniques of §12.3.1(i) employing an encryption function E, the confidentiality itself of the encrypted fields other than the session key W is not critical. A key derivation protocol which entirely avoids the use of an encryption function may offer potential advantages with respect to export restrictions. Protocol 12.20 is such a technique, which also provides authentication guarantees as stated. It uses two distinct functions h and h' (generating outputs of different bitlengths), respectively, for message authentication and key derivation.

12.20 Protocol Authenticated Key Exchange Protocol 2 (AKEP2)

SUMMARY: A and B exchange 3 messages to derive a session key W.
RESULT: mutual entity authentication, and implicit key authentication of W.

1. *Setup*: A and B share long-term symmetric keys K, K' (these should differ but need not be independent). h_K is a MAC (keyed hash function) used for entity authentication. $h'_{K'}$ is a pseudorandom permutation or keyed one-way function used for key derivation.
2. *Protocol messages*. Define $T = (B, A, r_A, r_B)$.

$$
\begin{array}{lll}
A \to B: & r_A & (1) \\
A \leftarrow B: & T,\ h_K(T) & (2) \\
A \to B: & (A, r_B),\ h_K(A, r_B) & (3) \\
W = h'_{K'}(r_B) & &
\end{array}
$$

3. *Protocol actions*. Perform the following steps for each shared key required.
   (a) A selects and sends to B a random number r_A.
   (b) B selects a random number r_B and sends to A the values (B, A, r_A, r_B), along with a MAC over these quantities generated using h with key K.
   (c) Upon receiving message (2), A checks the identities are proper, that the r_A received matches that in (1), and verifies the MAC.
   (d) A then sends to B the values (A, r_B), along with a MAC thereon.
   (e) Upon receiving (3), B verifies that the MAC is correct, and that the received value r_B matches that sent earlier.
   (f) Both A and B compute the session key as $W = h'_{K'}(r_B)$.

12.21 Note (*AKEP1 variant of Protocol 12.20*) The following modification of AKEP2 results in AKEP1 (Authenticated Key Exchange Protocol 1). B explicitly generates a random session key W and probabilistically encrypts it using h' under K' and random number r. The quantity $(r, W \oplus h'_{K'}(r))$ is now included as a final extra field within T and $h_K(T)$ in (2), and from which A may recover W. As an optimization, $r = r_B$.

(iii) Key transport without a priori shared keys

Shamir's no-key algorithm (Protocol 12.22) is a key transport protocol which, using only symmetric techniques (although involving modular exponentiation), allows key establishment over an open channel without requiring either shared or public keys. Each party has only its own local symmetric key. The protocol provides protection from passive adversaries only; it does not provide authentication. It thus solves the same problem as basic

Diffie-Hellman (Protocol 12.47) – two parties sharing no *a priori* keying material end up with a shared secret key, secure against passive adversaries – although differences include that it uses three messages rather than two, and provides key transport.

12.22 Protocol Shamir's no-key protocol

SUMMARY: users A and B exchange 3 messages over a public channel.
RESULT: secret K is transferred with privacy (but no authentication) from A to B.

1. *One-time setup (definition and publication of system parameters).*
   (a) Select and publish for common use a prime p chosen such that computation of discrete logarithms modulo p is infeasible (see Chapter 3).
   (b) A and B choose respective secret random numbers a, b, with $1 \leq a, b \leq p-2$, each coprime to $p-1$. They respectively compute a^{-1} and $b^{-1} \bmod p-1$.
2. *Protocol messages.*

$$\begin{array}{ll} A \rightarrow B : K^a \bmod p & (1) \\ A \leftarrow B : (K^a)^b \bmod p & (2) \\ A \rightarrow B : (K^{ab})^{a^{-1}} \bmod p & (3) \end{array}$$

3. *Protocol actions.* Perform the following steps for each shared key required.
   (a) A chooses a random key K for transport to B, $1 \leq K \leq p-1$. A computes $K^a \bmod p$ and sends B message (1).
   (b) B exponentiates (mod p) the received value by b, and sends A message (2).
   (c) A exponentiates (mod p) the received value by $a^{-1} \bmod p-1$, effectively "undoing" its previous exponentiation and yielding $K^b \bmod p$. A sends the result to B as message (3).
   (d) B exponentiates (mod p) the received value by $b^{-1} \bmod p-1$, yielding the newly shared key $K \bmod p$.

Use of ElGamal encryption for key transport (as per §12.5.1) with an uncertified public key sent in a first message (which would by definition be safe from passive attack) achieves in two passes the same goals as the above three-pass algorithm. In this case, the key is transported *from* the recipient of the first message *to* the originator.

12.23 Remark (*choice of cipher in Protocol 12.22*) While it might appear that any commutative cipher (i.e., cipher wherein the order of encryption and decryption is interchangeable) would suffice in place of modular exponentiation in Protocol 12.22, caution is advised. For example, use of the Vernam cipher (§1.5.4) would be totally insecure here, as the XOR of the three exchanged messages would equal the key itself.

12.3.2 Kerberos and related server-based protocols

The key transport protocols discussed in this section are based on symmetric encryption, and involve two communicating parties, A and B, and a trusted server with which they share long-term pairwise secret keys *a priori*. In such protocols, the server either plays the role of a *key distribution center* (KDC) and itself supplies the session key, or serves as a *key translation center* (KTC), and makes a key chosen by one party available to the other, by re-encrypting (translating) it under a key shared with the latter. KDCs and KTCs are discussed further in §13.2.3.

(i) Kerberos authentication protocol

Kerberos is the name given to all of the following: the distributed authentication service originating from MIT's Project Athena, which includes specifications for data integrity and encryption; the software which implements it, and the processes executing such software; and the specific authentication protocol used therein. Focus here, and use of the term "Kerberos", is restricted to the protocol itself, which supports both entity authentication and key establishment using symmetric techniques and a third party.

The basic Kerberos protocol involves A (the *client*), B (the *server* and *verifier*), and a trusted server T (the Kerberos *authentication server*). At the outset A and B share no secret, while T shares a secret with each (e.g., a user password, transformed into a cryptographic key by an appropriate function). The primary objective is for B to verify A's identity; the establishment of a shared key is a side effect. Options include a final message providing mutual entity authentication and establishment of an additional secret shared by A and B (a *subsession key* not chosen by T).

The protocol proceeds as follows. A requests from T appropriate *credentials* (data items) to allow it to authenticate itself to B. T plays the role of a KDC, returning to A a session key encrypted for A and a *ticket* encrypted for B. The ticket, which A forwards on to B, contains the session key and A's identity; this allows authentication of A to B when accompanied by an appropriate message (the *authenticator*) created by A containing a timestamp recently encrypted under that session key.

12.24 Protocol Basic Kerberos authentication protocol (simplified)[1]

SUMMARY: A interacts with trusted server T and party B.
RESULT: entity authentication of A to B (optionally mutual), with key establishment.

1. *Notation*. Optional items are denoted by an asterisk (*).
   E is a symmetric encryption algorithm (see Remark 12.19).
   N_A is a nonce chosen by A; T_A is a timestamp from A's local clock.
   k is the session-key chosen by T, to be shared by A and B.
   L indicates a validity period (called the "lifetime").
2. *One-time setup*. A and T share a key K_{AT}; similarly, B and T share K_{BT}. Define $\text{ticket}_B \stackrel{\text{def}}{=} E_{K_{BT}}(k, A, L)$; $\text{authenticator} \stackrel{\text{def}}{=} E_k(A, T_A, A^*_{\text{subkey}})$.
3. *Protocol messages*.

$$\begin{array}{lll} A \rightarrow T: & A, B, N_A & (1) \\ A \leftarrow T: & \text{ticket}_B, E_{K_{AT}}(k, N_A, L, B) & (2) \\ A \rightarrow B: & \text{ticket}_B, \text{authenticator} & (3) \\ A \leftarrow B: & E_k(T_A, B^*_{\text{subkey}}) & (4) \end{array}$$

4. *Protocol actions*. Algorithm E includes a built-in integrity mechanism, and protocol failure results if any decryption yields an integrity check failure.
   (a) A generates a nonce N_A and sends to T message (1).
   (b) T generates a new session key k, and defines a validity period (lifetime L) for the ticket, consisting of an ending time and optional starting time. T encrypts k, the received nonce, lifetime, and received identifier (B) using A's key. T also creates a ticket secured using B's key containing k, received identifier (A), and lifetime. T sends to A message (2).

[1]The basic Kerberos (version 5) protocol between client and authentication server is given, with messages simplified (some non-cryptographic fields omitted) to allow focus on cryptographic aspects.

(c) A decrypts the non-ticket part of message (2) using K_{AT} to recover: k, N_A, lifetime L, and the identifier of the party for which the ticket was actually created. A verifies that this identifier and N_A match those sent in message (1), and saves L for reference. A takes its own identifier and fresh timestamp T_A, optionally generates a secret A_{subkey}, and encrypts these using k to form the authenticator. A sends to B message (3).

(d) B receives message (3), decrypts the ticket using K_{BT} yielding k to allow decryption of the authenticator. B checks that:

i. the identifier fields (A) in the ticket and authenticator match;
ii. the timestamp T_A in the authenticator is valid (see §10.3.1); and
iii. B's local time is within the lifetime L specified in the ticket.

If all checks pass, B declares authentication of A successful, and saves A_{subkey} (if present) as required.

(e) (Optionally for mutual entity authentication:) B constructs and sends to A message (4) containing A's timestamp from the authenticator (specifically excluding the identifier A, to distinguish it from the authenticator), encrypted using k. B optionally includes a subkey to allow negotiation of a subsession key.

(f) (Optionally for mutual entity authentication:) A decrypts message (4). If the timestamp within matches that sent in message (3), A declares authentication of B successful and saves B_{subkey} (if present) as required.

12.25 Note (*security and options in Kerberos protocol*)

(i) Since timestamps are used, the hosts on which this protocol runs must provide both secure and synchronized clocks (see §10.3.1).

(ii) If, as is the case in actual implementations, the initial shared keys are password-derived, then the protocol is no more secure than the secrecy of such passwords or their resistance to password-guessing attacks.

(iii) Optional parameters A_{subkey} and B_{subkey} allow transfer of a key (other than k) from A to B or vice-versa, or the computation of a combined key using some function $f(A_{\text{subkey}}, B_{\text{subkey}})$.

(iv) The lifetime within the ticket is intended to allow A to re-use the ticket over a limited time period for multiple authentications to B without additional interaction with T, thus eliminating messages (1) and (2). For each such re-use, A creates a new authenticator with a fresh timestamp and the same session key k; the optional subkey field is of greater use in this case.

(ii) Needham-Schroeder shared-key protocol

The Needham-Schroeder shared-key protocol is important primarily for historical reasons. It is the basis for many of the server-based authentication and key distribution protocols proposed since 1978, including Kerberos and Otway-Rees. It is an example of a protocol independent of timestamps, providing both entity authentication assurances and key establishment with key confirmation. However, it is no longer recommended (see Remark 12.28).

12.26 Protocol Needham-Schroeder shared-key protocol

SUMMARY: A interacts with trusted server T and party B.
RESULT: entity authentication (A with B); key establishment with key confirmation.

1. *Notation*. E is a symmetric encryption algorithm (see Remark 12.19).
   N_A and N_B are nonces chosen by A and B, respectively.
   k is a session key chosen by the trusted server T for A and B to share.
2. *One-time setup*. A and T share a symmetric key K_{AT}; B and T share K_{BT}.
3. *Protocol messages*.

$$\begin{array}{lll} A \to T: & A, B, N_A & (1) \\ A \leftarrow T: & E_{K_{AT}}(N_A, B, k, E_{K_{BT}}(k, A)) & (2) \\ A \to B: & E_{K_{BT}}(k, A) & (3) \\ A \leftarrow B: & E_k(N_B) & (4) \\ A \to B: & E_k(N_B - 1) & (5) \end{array}$$

4. *Protocol actions*. Aside from verification of nonces, actions are essentially analogous to those in Kerberos (Protocol 12.24), and are not detailed here.

12.27 Note (*functionality and options in Needham-Schroeder shared-key protocol*)

(i) The protocol provides A and B with a shared key k with key authentication (due to the trusted server).
(ii) Messages (4) and (5) provide entity authentication of A to B; entity authentication of B to A can be obtained provided A can carry out some redundancy check on N_B upon decrypting message (4).
(iii) If it is acceptable for A to re-use a key k with B, A may securely cache the data sent in message (3) along with k. Upon subsequent re-use, messages (1) and (2) may then be omitted, but now to prevent replay of old messages (4), an encrypted nonce $E_k(N_A{}')$ should be appended to message (3), and message (4) should be replaced by $E_k(N_A{}' - 1, N_B)$ allowing A to verify B's current knowledge of k (thereby providing entity authentication).

12.28 Remark (*Needham-Schroeder weakness vs. Kerberos*) The essential differences between Protocol 12.26 and Kerberos (Protocol 12.24) are as follows: the Kerberos lifetime parameter is not present; the data of message (3), which corresponds to the Kerberos ticket, is unnecessarily double-encrypted in message (2) here; and authentication here employs nonces rather than timestamps. A weakness of the Needham-Schroeder protocol is that since B has no way of knowing if the key k is fresh, should a session key k ever be compromised, any party knowing it may both resend message (3) and compute a correct message (5) to impersonate A to B. This situation is ameliorated in Kerberos by the lifetime parameter which limits exposure to a fixed time interval.

(iii) Otway-Rees protocol

The Otway-Rees protocol is a server-based protocol providing authenticated key transport (with key authentication and key freshness assurances) in only 4 messages – the same as Kerberos, but here without the requirement of timestamps. It does not, however, provide entity authentication or key confirmation.

12.29 Protocol Otway-Rees protocol

SUMMARY: B interacts with trusted server T and party A.
RESULT: establishment of fresh shared secret K between A and B.

1. *Notation*. E is a symmetric encryption algorithm (see Remark 12.19). k is a session key T generates for A and B to share. N_A and N_B are nonces chosen by A and B, respectively, to allow verification of key freshness (thereby detecting replay). M is a second nonce chosen by A which serves as a transaction identifier.
2. *One-time setup*. T shares symmetric keys K_{AT} and K_{BT} with A, B, respectively.
3. *Protocol messages*.

$$\begin{array}{lll} A \to B: & M, A, B, E_{K_{AT}}(N_A, M, A, B) & (1) \\ B \to T: & M, A, B, E_{K_{AT}}(N_A, M, A, B), E_{K_{BT}}(N_B, M, A, B) & (2) \\ B \leftarrow T: & E_{K_{AT}}(N_A, k), E_{K_{BT}}(N_B, k) & (3) \\ A \leftarrow B: & E_{K_{AT}}(N_A, k) & (4) \end{array}$$

4. *Protocol actions.* Perform the following steps each time a shared key is required.
   (a) A encrypts data for the server containing two nonces, N_A and M, and the identities of itself and the party B to whom it wishes the server to distribute a key. A sends this and some plaintext to B in message (1).
   (b) B creates its own nonce N_B and an analogous encrypted message (with the same M), and sends this along with A's message to T in message (2).
   (c) T uses the cleartext identifiers in message (2) to retrieve K_{AT} and K_{BT}, then verifies the cleartext (M A, B) matches that recovered upon decrypting both parts of message (2). (Verifying M in particular confirms the encrypted parts are linked.) If so, T inserts a new key k and the respective nonces into distinct messages encrypted for A and B, and sends both to B in message (3).
   (d) B decrypts the second part of message (3), checks N_B matches that sent in message (2), and if so passes the first part on to A in message (4).
   (e) A decrypts message (4) and checks N_A matches that sent in message (1).

If all checks pass, each of A and B are assured that k is fresh (due to their respective nonces), and trust that the other party T shared k with is the party bound to their nonce in message (2). A knows that B is active as verification of message (4) implies B sent message (2) recently; B however has no assurance that A is active until subsequent use of k by A, since B cannot determine if message (1) is fresh.

12.30 Remark (*nonces in Otway-Rees protocol*) The use of two nonces generated by A is redundant (N_A could be eliminated in messages (1) and (2), and replaced by M in (3) and (4)), but nonetheless allows M to serve solely as an administrative transaction identifier, while keeping the format of the encrypted messages of each party identical. (The latter is generally considered desirable from an implementation viewpoint, but dubious from a security viewpoint.)

12.31 Remark (*extension of Otway-Rees protocol*) Protocol 12.29 may be extended to provide both key confirmation and entity authentication in 5 messages. Message (4) could be augmented to both demonstrate B's timely knowledge of k and transfer a nonce to A (e.g., appending $E_k(N_A, N_B)$), with a new fifth message ($A \to B : E_k(N_B)$) providing B reciprocal assurances.

12.4 Key agreement based on symmetric techniques

This section presents ideas related to key agreement based on symmetric techniques. It also presents a key pre-distribution system which is in some ways a symmetric-key analogue to Diffie-Hellman key agreement with fixed exponentials (Note 12.48).

12.32 Definition A *key distribution system* (KDS) is a method whereby, during an initialization stage, a trusted server generates and distributes secret data values (*pieces*) to users, such that any pair of users may subsequently compute a shared key unknown to all others (aside from the server).

For fixed pairwise keys, a KDS is a key pre-distribution scheme. A trivial KDS is as follows: the trusted server chooses distinct keys for each pair among the n users, and by some secure means initially distributes to each user its $n-1$ keys appropriately labeled. This provides unconditional security (perfect security in the information-theoretic sense); an outside adversary can do no better than guess the key. However, due to the large amount of storage required, alternate methods are sought, at the price of losing unconditional security against arbitrarily large groups of colluding users.

12.33 Definition A KDS is said to be j-*secure* if, given a specified pair of users, any coalition of j or fewer users (disjoint from the two), pooling their pieces, can do no better at computing the key shared by the two than a party which guesses the key without any pieces whatsoever.

A j-secure KDS is thus unconditionally secure against coalitions of size j or smaller.

12.34 Fact (*Blom's KDS bound*) In any j-secure KDS providing m-bit pairwise session keys, the secret data stored by each user must be at least $m \cdot (j+1)$ bits.

The trivial KDS described above is optimal with respect to the number of secret key bits stored, assuming collusion by all parties other than the two directly involved. This corresponds to meeting the lower bound of Fact 12.34 for $j = n-2$.

Blom's symmetric key pre-distribution system

Blom's scheme (Mechanism 12.35) is a KDS which can be used to meet the bound of Fact 12.34 for values $j < n-2$. It is non-interactive; each party requires only an index i, $1 \leq i \leq n$, which uniquely identifies the party with which it is to form a joint key (the scheme is identity-based in this regard). Each user is assigned a secret vector of initial keying material (*base key*) from which it is then able to compute a pairwise secret (*derived key*) with each other user.

As outlined in Remark 12.37, the scheme may be engineered to provide unconditional security against coalitions of a specified maximum size. The initial keying material assigned to each user (a row of S, corresponding to k keys) allows computation of a larger number of derived keys (a row of K, providing n keys), one per each other user. Storage savings results from choosing k less than n. The derived keys of different user pairs, however, are not statistically independent.

12.35 Mechanism Blom's symmetric key pre-distribution system

SUMMARY: each of n users is given initial secret keying material and public data.
RESULT: each pair of users U_i, U_j may compute an m-bit pairwise secret key $K_{i,j}$.

1. A $k \times n$ generator matrix G of an (n, k) MDS code over a finite field $\mathbb{F}_q$ of order q is made known to all n system users (see Note 12.36).
2. A trusted party T creates a random secret $k \times k$ symmetric matrix D over $\mathbb{F}_q$.
3. T gives to each user U_i the secret key S_i, defined as row i of the $n \times k$ matrix $S = (DG)^T$. (S_i is a k-tuple over $\mathbb{F}_q$ of $k \cdot \lg(q)$ bits, allowing U_i to compute any entry in row i of $(DG)^T G$.)
4. Users U_i and U_j compute the common secret $K_{i,j} = K_{j,i}$ of bitlength $m = \lg(q)$ as follows. Using S_i and column j of G, U_i computes the (i, j) entry of the $n \times n$ symmetric matrix $K = (DG)^T G$. Using S_j and column i of G, U_j similarly computes the (j, i) entry (which is equal to the (i, j) entry since K is symmetric).

12.36 Note (*background on MDS codes*) The motivation for Mechanism 12.35 arises from well-known concepts in linear error-correcting codes, summarized here. Let $G = [I_k A]$ be a $k \times n$ matrix where each row is an n-tuple over $\mathbb{F}_q$ (for q a prime or prime power). I_k is the $k \times k$ identity matrix. The set of n-tuples obtained by taking all linear combinations (over $\mathbb{F}_q$) of rows of G is the *linear code C*. Each of these q^k n-tuples is a *codeword*, and $C = \{c : \ c = mG, m = (m_1\, m_2\, \ldots\, m_k), m_i \in \mathbb{F}_q\}$. G is a *generator matrix* for the linear (n, k) code C. The *distance* between two codewords c, c' is the number of components they differ in; the distance d of the code is the minimum such distance over all pairs of distinct codewords. A code of distance d can correct $e = \lfloor (d-1)/2 \rfloor$ component errors in a codeword, and for linear codes $d \le n - k + 1$ (the *Singleton bound*). Codes meeting this bound with equality ($d = n - k + 1$) have the largest possible distance for fixed n and k, and are called *maximum distance separable* (MDS) codes.

12.37 Remark (*choice of k in Blom's scheme*) The condition $d = n-k+1$ defining MDS codes can be shown equivalent to the condition that every set of k columns of G is linearly independent. From this, two facts follow about codewords of MDS codes: (i) any k components uniquely define a codeword; and (ii) any $j \le k - 1$ components provide no information about other components. For Mechanism 12.35, the choice of k is governed by the fact that if k or more users conspire, they are able to recover the secret keys of all other users. (k conspirators may compute k rows of K, or equivalently k columns, corresponding to k components in each row. Each row is a codeword in the MDS code generated by G, and corresponds to the key of another user, and by the above remark k components thus define all remaining components of that row.) However, if fewer than k users conspire, they obtain no information whatsoever about the keys of any other user (by similar reasoning). Thus Blom's scheme is *j*-secure for $j \le k-1$, and relative to Fact 12.34, is optimal with respect to the amount of initial keying material required.

12.5 Key transport based on public-key encryption

Key transport based on public-key encryption involves one party choosing a symmetric key, and transferring it to a second, using that party's encryption public key. This provides key authentication to the originator (only the intended recipient has the private key allowing decryption), but the originator itself obtains neither entity authentication nor key confirmation. The second party receives no source authentication. Such additional assurances may be obtained through use of further techniques including: additional messages (§12.5.1); digital signatures (§12.5.2); and symmetric encryption in addition to signatures (§12.5.3).

Authentication assurances can be provided with or without the use of digital signatures, as follows:

1. *entity authentication via public-key decryption* (§12.5.1). The intended recipient authenticates itself by returning some time-variant value which it alone may produce or recover. This may allow authentication of both the entity and a transferred key.
2. *data origin authentication via digital signatures* (§12.5.2). Public-key encryption is combined with a digital signature, providing key transport with source identity assurances.

The distinction between entity authentication and data origin authentication is that the former provides a timeliness assurance, whereas the latter need not. Table 12.3 summarizes the protocols presented.

→ Properties ↓ Protocol	signatures required‡	entity authentication	number of messages
basic PK encryption (1-pass)	no	no	1
Needham-Schroeder PK	no	mutual	3
encrypting signed keys	yes	data origin only†	1
separate signing, encrypting	yes	data origin only†	1
signing encrypted keys	yes	data origin only†	1
X.509 (2-pass) – timestamps	yes	mutual	2
X.509 (3-pass) – random #'s	yes	mutual	3
Beller-Yacobi (4-pass)	yes	mutual	4
Beller-Yacobi (2-pass)	yes	unilateral	2

Table 12.3: *Selected key transport protocols based on public-key encryption.*

†Unilateral entity authentication may be achieved if timestamps are included.

‡Schemes using public keys transported by certificates require signatures for verification thereof, but signatures are not required within protocol messages.

12.5.1 Key transport using PK encryption without signatures

One-pass key transport by public-key encryption

One-pass protocols are appropriate for one-way communications and store-and-forward applications such as electronic mail and fax. Basic key transport using public-key encryption can be achieved in a one-pass protocol, assuming the originator A possesses *a priori* an authentic copy of the encryption public key of the intended recipient B. Using B's public encryption key, A encrypts a randomly generated key k, and sends the result $P_B(k)$ to B. Public-key encryption schemes P_B of practical interest here include RSA encryption, Rabin encryption, and ElGamal encryption (see Chapter 8).

The originator A obtains no entity authentication of the intended recipient B (and indeed, does not know if B even receives the message), but is assured of implicit key authentication – no one aside from B could possibly recover the key. On the other hand, B has no assurances regarding the source of the key, which remains true even in the case

$A \to B: \ P_B(k, A)$. A timeliness guarantee may be provided using timestamps, for example, $A \to B: \ P_B(k, T_A)$. This is necessary if security against known-key attacks is required, as this technique is otherwise vulnerable to message replay (cf. Remark 12.18).

Maintaining the restriction of using public-key encryption alone (i.e., without signatures), assurances in addition to unilateral key authentication, namely, mutual entity authentication, and mutual key authentication, may be obtained through additional messages as illustrated by Protocol 12.38 below.

Needham-Schroeder public-key protocol

The Needham-Schroeder public-key protocol provides mutual entity authentication and mutual key transport (A and B each transfer a symmetric key to the other). The transported keys may serve both as nonces for entity authentication and secret keys for further use. Combination of the resulting shared keys allows computation of a joint key to which both parties contribute.

12.38 Protocol Needham-Schroeder public-key protocol

SUMMARY: A and B exchange 3 messages.
RESULT: entity authentication, key authentication, and key transport (all mutual).

1. *Notation.* $P_X(Y)$ denotes public-key encryption (e.g., RSA) of data Y using party X's public key; $P_X(Y_1, Y_2)$ denotes the encryption of the concatenation of Y_1 and Y_2. k_1, k_2 are secret symmetric session keys chosen by A, B, respectively.
2. *One-time setup.* Assume A, B possess each other's authentic public-key. (If this is not the case, but each party has a certificate carrying its own public key, then one additional message is required for certificate transport.)
3. *Protocol messages.*

$$\begin{array}{lll} A \to B: & P_B(k_1, A) & (1) \\ A \leftarrow B: & P_A(k_1, k_2) & (2) \\ A \to B: & P_B(k_2) & (3) \end{array}$$

4. *Protocol actions.*
   (a) A sends B message (1).
   (b) B recovers k_1 upon receiving message (1), and returns to A message (2).
   (c) Upon decrypting message (2), A checks the key k_1 recovered agrees with that sent in message (1). (Provided k_1 has never been previously used, this gives A both entity authentication of B and assurance that B knows this key.) A sends B message (3).
   (d) Upon decrypting message (3), B checks the key k_2 recovered agrees with that sent in message (2). The session key may be computed as $f(k_1, k_2)$ using an appropriate publicly known non-reversible function f.

12.39 Note (*modification of Needham-Schroeder protocol*) Protocol 12.38 may be modified to eliminate encryption in the third message. Let r_1 and r_2 be random numbers generated respectively by A and B. Then, with checks analogous to those in the basic protocol, the messages in the modified protocol are:

$$\begin{array}{lll} A \to B: & P_B(k_1, A, r_1) & (1') \\ A \leftarrow B: & P_A(k_2, r_1, r_2) & (2') \\ A \to B: & r_2 & (3') \end{array}$$

12.5.2 Protocols combining PK encryption and signatures

While privacy of keying material is a requirement in key transport protocols, source authentication is also typically needed. Encryption and signature primitives may respectively be used to provide these properties. Key transport protocols involving both public-key encryption and signatures include:

1. those which sign the key, then public-key encrypt the signed key;
2. those which sign the key, and separately public-key encrypt the (unsigned) key;
3. those which public-key encrypt the key, then sign the encrypted key; and
4. those using symmetric encryption in addition to public-key encryption and signatures.

The first three types are discussed in this subsection (as noted in §12.5.2(ii), the second is secure only in certain circumstances); the fourth is discussed in §12.5.3. The signature schemes S_A of greatest practical interest are RSA, Rabin signatures, and ElGamal-family signatures (see Chapter 11). The public-key encryption schemes P_B of greatest practical interest are RSA, Rabin encryption, and ElGamal encryption (see Chapter 8).

Notation. For data input y, in what follows, $S_A(y)$ and $P_B(y)$ denote the data values resulting, respectively, from the signature operation on y using A's signature private key, and the encryption operation on y using B's encryption public key. As a default, it is assumed that the signature scheme does not provide message recovery, i.e., the input y cannot be recovered from the signature $S_A(y)$, and y must be sent explicitly in addition to $S_A(y)$ to allow signature verification. (This is the case for DSA, or RSA following input hashing; see Chapter 11. However, in the case of encrypting and signing separately, any secret data y must remain confidential.) If y consists of multiple data values $y = (y_1, \ldots, y_n)$, then the input is taken to be the bitwise concatenation of these multiple values.

(i) Encrypting signed keys

One option for combining signatures and public-key encryption is to encrypt signed blocks:

$$A \rightarrow B: \quad P_B(k,\ t_A{}^*,\ S_A(B, k, t_A{}^*))$$

The asterisk denotes that the timestamp t_A of A is optional; inclusion facilitates entity authentication of A to B and provides a freshness property. The identifier B within the scope of the signature prevents B from sending the signed key on to another party and impersonating A. A disadvantage of this method over the "signing encrypted keys" alternative (§12.5.2(iii)) is that here the data to be public-key encrypted is larger, implying the possible requirement of adjusting the block size of the public-key encryption scheme, or the use of techniques such as cipher-block-chaining. In the case of signature schemes with message recovery (e.g., ordinary RSA), the above can be simplified to:

$$A \rightarrow B: \quad P_B(S_A(B, k, t_A{}^*))$$

(ii) Encrypting and signing separately

For signature schemes without message recovery, a variation of the above option is to sign the key and encrypt the key, but not to encrypt the signature itself. This is acceptable only if the signature scheme is such that no information regarding plaintext data can be deduced from the signature itself on that data (e.g., when the signature operation involves preliminary one-way hashing). This is critical because, in general, data may be recovered from a signature on it (e.g., RSA without hashing). A summary of this case is then as follows:

$$A \rightarrow B: \quad P_B(k,\ t_A{}^*),\ S_A(B, k, t_A{}^*)$$

If the key k is used solely to encrypt a data file y, then the signature S_A may be over y instead of k. This is suitable in *store-and-forward* environments. The encrypted file may then be transferred along with the key establishment information, in which case y is first recovered by using k to decrypt the file, and then the signature on y is verified.

(iii) Signing encrypted keys

In contrast to encrypting signed keys, one may sign encrypted keys:

$$A \to B: \quad t_A{}^*,\ P_B(A, k),\ S_A(B, t_A{}^*, P_B(A, k))$$

The asterisk denotes that the timestamp t_A of A is optional; inclusion facilitates entity authentication of A to B. The parameter A within the scope of the public-key encryption prevents *signature stripping* – simply signing a publicly-encrypted key, e.g., $S_A(P_B(k))$ is vulnerable to a third party C extracting the encrypted quantity $P_B(k)$ and then oversigning with its own key, thus defeating authentication (cf. Note 12.42). Furthermore, the encryption mechanism must ensure that an adversary C without access to k, cannot change $P_B(A, k)$ to $P_B(C, k)$; see Remark 12.19. It is desirable and assumed that the combined length of the parameters A and k not exceed the blocklength of the public-key encryption scheme, to limit computation to a single block encryption.

Mutual entity authentication using timestamps. The message format given above can be used for key establishment in a one-pass protocol, although this provides no entity authentication of the recipient to the originator. For mutual entity authentication, two messages of this form may be used, yielding essentially X.509 strong two-way authentication (Protocol 12.40).

Mutual entity authentication using challenge-response. The 2-pass key transport protocol discussed in the previous paragraph requires the use of timestamps, in which case security relies on the assumption of secure, synchronized clocks. This requirement can be eliminated by using a 3-pass protocol with random numbers for challenge-response (essentially the X.509 strong three-way authentication protocol; cf. Protocol 12.43):

$$\begin{array}{ll} A \to B: & r_A \\ A \leftarrow B: & r_B,\ P_A(B, k_1),\ S_B(r_B, r_A, A, P_A(B, k_1)) \\ A \to B: & P_B(A, k_2),\ S_A(r_A, r_B, B, P_B(A, k_2)) \end{array}$$

A and B may compute a joint key k as some function of k_1 and k_2; alternately, one of $P_A(B, k_1)$ and $P_B(A, k_2)$ may be omitted from the second or third message. The identifiers within the scope of the encryption blocks remain necessary as above; the identifiers within the scope of (only) the signature are, however, redundant, both here and in the case of signing encrypted keys above – it may be assumed they must match those corresponding to the public-key encryption.

(iv) X.509 strong authentication protocols

This subsection considers in greater detail a fully-specified protocol involving public-key transport using the general technique of §12.5.2(iii), namely, signing encrypted keys.

The X.509 recommendation defines both "strong two-way" and "strong three-way" authentication protocols, providing mutual entity authentication with optional key transport. Here *strong* distinguishes these from simpler password-based methods, and *two-* and *three-way* refers to protocols with two and three passes (message exchanges), using timestamps and challenge-response based on random numbers, respectively.

Both protocols were designed to provide the assurances listed below to the responder B (and reciprocal assurances intended for the originator A); here *token* refers to cryptographically protected data:

1. the identity of A, and that the token received by B was constructed by A (and not thereafter altered);
2. that the token received by B was specifically intended for B;
3. that the token received by B has "freshness" (has not been used previously, and originated within an acceptably recent timeframe);
4. the mutual secrecy of the transferred key.

12.40 Protocol X.509 strong two-way authentication (two-pass)

SUMMARY: A sends B one message, and B responds with one message.
RESULT: mutual entity authentication and key transport with key authentication.

1. *Notation.*
   $P_X(y)$ denotes the result of applying X's encryption public key to data y.
   $S_X(y)$ denotes the result of applying X's signature private key to y.
   r_A, r_B are never re-used numbers (to detect replay and impersonation).
   $cert_X$ is a certificate binding party X to a public key suitable for both encryption and signature verification (see Remark 12.41).
2. *System setup.*
   (a) Each party has its public key pair for signatures and encryption.
   (b) A must acquire (and authenticate) the encryption public key of B *a priori*. (This may require additional messages and computation.)
3. *Protocol messages.* (An asterisk denotes items are optional.)
   Let $D_A = (t_A, r_A, B, \text{data}_1{}^*, P_B(k_1)^*)$, $D_B = (t_B, r_B, A, r_A, \text{data}_2{}^*, P_A(k_2)^*)$.

$$A \to B: \quad cert_A, D_A, S_A(D_A) \quad (1)$$
$$A \leftarrow B: \quad cert_B, D_B, S_B(D_B) \quad (2)$$

4. *Protocol actions.*
   (a) A obtains a timestamp t_A indicating an expiry time, generates r_A, optionally obtains a symmetric key k_1 and sends to B message (1). (data_1 is optional data for which data origin authentication is desired.)
   (b) B verifies the authenticity of $cert_A$ (checking the signature thereon, expiry date, etc.), extracts A's signature public key, and verifies A's signature on the data block D_A. B then checks that the identifier in message (1) specifies itself as intended recipient, that the timestamp is valid, and checks that r_A has not been replayed. (r_A includes a sequential component which B checks, against locally maintained state information, for uniqueness within the validity period defined by t_A.)
   (c) If all checks succeed, B declares the authentication of A successful, decrypts k_1 using its private decryption key, and saves this now-shared key. (This terminates the protocol if only unilateral authentication is desired.) B then obtains timestamp t_B, generates r_B, and sends A message (2). (data_2 is optional data, and k_2 is an optional symmetric key provided for A.)
   (d) A carries out actions analogous to those carried out by B. If all checks succeed, A declares the authentication of B successful, and saves key k_2 for subsequent use. A and B share mutual secrets k_1 and k_2.

12.41 Remark (*separate keys in X.509*) The X.509 standard assumes a public-key scheme such as RSA, whereby the same key pair may be used for both encryption and signature functionality. The protocol, however, is easily adapted for separate signature and encryption keys, and, indeed, it is prudent to use separate keys. See also Remark 13.32.

12.42 Note (*criticism of X.509 protocol*) Since Protocol 12.40 does not specify inclusion of an identifier (e.g., A) within the scope of the encryption P_B within D_A, one cannot guarantee that the signing party actually knows (or was the source of) the plaintext key.

12.43 Protocol X.509 strong three-way authentication (three-pass)

SUMMARY: A and B exchange 3 messages.
RESULT: as in Protocol 12.40, without requiring timestamps.
The protocol differs from Protocol 12.40 as follows:

1. Timestamps t_A and t_B may be set to zero, and need not be checked.
2. Upon receiving (2), A checks the received r_A matches that in message (1).
3. A third message is sent from A to B:

$$A \to B: \quad (r_B, B),\ S_A(r_B, B) \quad (3)$$

4. Upon receiving (3), B verifies the signature matches the received plaintext, that plaintext identifier B is correct, and that plaintext r_B received matches that in (2).

12.5.3 Hybrid key transport protocols using PK encryption

In contrast to the preceding key transport protocols, the Beller-Yacobi protocol uses symmetric encryption in addition to both PK encryption and digital signatures. Such protocols using both asymmetric and symmetric techniques are called *hybrid protocols*.

Beller-Yacobi protocol (4-pass)

The key transport protocol of Beller and Yacobi, which provides mutual entity authentication and explicit key authentication, was designed specifically for applications where there is an imbalance in processing power between two parties; the goal is to minimize the computational requirements of the weaker party. (Candidate applications include transactions involving chipcards, and wireless communications involving a low-power telephone handset.) Another feature of the protocol is that the identity of one of the parties (the weaker, here A) remains concealed from eavesdroppers.

Essentially, A authenticates itself to B by signing a random challenge m, while B authenticates itself to A by demonstrating knowledge of a key K only B itself could recover. For simplicity of exposition, the protocol is described using RSA with public exponent 3, although Rabin's scheme is more efficient and recommended in practice (but see Note 8.13 regarding chosen-ciphertext attack).

12.44 Protocol Beller-Yacobi key transport (4-pass)

SUMMARY: A transfers key K to B in a 4-pass protocol.
RESULT: mutual entity authentication and mutual explicit key authentication.

1. *Notation.*
   $E_K(y)$ denotes symmetric encryption of y using key K and algorithm E.
   $P_X(y)$ denotes the result of applying X's public-key function to y.
   $S_X(y)$ denotes the result of applying X's private-key function to y.
   I_X denotes an identifying string for party X.
   $h(y)$ denotes the hash of y, used in association with the signature scheme.
   If $y = (y_1, \ldots, y_n)$, the input is the concatenation of these multiple values.
2. *System setup.*
   (a) *Selection of system parameters.* An appropriate prime n_S and generator α for the multiplicative group of integers modulo n_S are fixed as ElGamal system parameters. A trusted server T chooses appropriate primes p and q yielding public modulus $n_T = pq$ for RSA signatures, then for public exponent $e_T = 3$ computes a private key d_T satisfying: $e_T d_T \equiv 1 \bmod (p-1)(q-1)$.
   (b) *Distribution of system parameters.* Each party (A and B) is given an authentic copy of T's public key and the system parameters: n_T, (n_S, α). T assigns to each party X a unique *distinguished name* or identifying string I_X (e.g., X's name and address).
   (c) *Initialization of terminal.* Each party playing the role of A (*terminal*) selects a random integer a, $1 < a \le n_S - 2$, and computes its ElGamal signature public key $u_A = \alpha^a \bmod n_S$. A keeps its corresponding private key a secret, and transfers an authentic copy of u_A to T, identifying itself to T by out-of-band means (e.g., in person). T constructs and returns to A the public-key certificate: $cert_A = (I_A,\ u_A,\ G_A)$. (The certificate contains A's identity and ElGamal signature public key, plus T's RSA signature G_A over these: $G_A = S_T(I_A,\ u_A) = (h(I_A,\ u_A))^{d_T} \bmod n_T$.)
   (d) *Initialization of server.* Each party playing the role of B (*server*) creates an encryption private key and corresponding public key based on RSA with public exponent $e_B = 3$. B chooses a public-key modulus n_B as the product of two appropriate secret primes, and itself computes the corresponding RSA private key d_B. B transfers n_B to T, identifying itself to T by out-of-band means. T then constructs and returns to B the public-key certificate: $cert_B = (I_B,\ n_B,\ G_B)$. (The certificate contains B's identity and RSA encryption public key n_B, plus T's RSA signature over these: $G_B = S_T(I_B,\ n_B) = (h(I_B,\ n_B))^{d_T} \bmod n_T$.)
3. *Protocol messages.*

$$\begin{array}{lll} A \leftarrow B: & cert_B = (I_B,\ n_B,\ G_B) & (1) \\ A \rightarrow B: & P_B(K) = K^3 \bmod n_B & (2) \\ A \leftarrow B: & E_K(m,\ \{0\}^t) & (3) \\ A \rightarrow B: & E_K((v,w),\ cert_A) & (4) \end{array}$$

4. *Protocol actions.* The following steps are performed each time a shared key is required. The protocol is aborted (with result of failure) if any check fails.
   (a) *Precomputation by terminal.* A selects a random x, $1 \le x \le n_S - 2$, and computes three values: $v = \alpha^x \bmod n_S$; $x^{-1} \bmod (n_S - 1)$; and $av \bmod (n_S - 1)$. (For the security of ElGamal signatures, x must be new for each signature, and be co-prime to $n_S - 1$ to ensure x^{-1} exists.)

(b) B sends to A message (1).

(c) A checks the authenticity of n_B by confirming: $h(I_B,\ n_B) = {G_B}^3 \bmod n_T$. A chooses a random key $1 < K < n_B - 1$ and sends B message (2), where $Y = P_B(K)$.

(d) B recovers $K = S_B(Y) = Y^{d_B} \bmod n_B$. (The final two messages will be encrypted using K.) B chooses a random integer m as a challenge, extends it with t (say $t \approx 50$) least significant zeros, symmetrically encrypts this using key K, and sends A message (3).

(e) A decrypts the received message, and checks it has t trailing zeros; if so, A accepts that it originated from B and that B knows key K. A takes the decrypted challenge m, concatenates it to the identity I_B of the party whose public key it used to share K in message (2), forming the concatenated quantity $M = (m,\ I_B)$, then computes w satisfying: $w \equiv (M - av) \cdot x^{-1} \bmod (n_S - 1)$, and sends B message (4). (Here (v, w) is A's ElGamal signature on M, and $cert_A = (I_A,\ u_A,\ G_A)$. The identity I_B in M is essential to preclude an intruder-in-the-middle attack – see §12.9.)

(f) B decrypts the received message, and verifies the authenticity of u_A by checking that: $h(I_A,\ u_A) = {G_A}^3 \bmod n_T$. Finally, B constructs the concatenated quantity $M = (m,\ I_B)$ from the challenge m remembered from message (3) and its own identity, then verifies A's signature on the challenge by checking that: $\alpha^M \equiv {u_A}^v \cdot v^w \bmod n_S$. If all checks succeed, B accepts the party A associated with identity I_A as the source of key K.

12.45 Note (*on Beller-Yacobi key transport protocol*)

(i) To achieve mutual authentication here requires that each party carry out at least one private-key operation (showing knowledge of its private key), and one or two public-key operations (related to verifying the other's identity, and its public key if not known *a priori*).

(ii) The novelty here is careful selection of two separate public-key schemes, each requiring only an inexpensive computation by the computationally limited party, in this case A. Choosing RSA with exponent 3 or Rabin with exponent 2 results in an inexpensive public-key operation (2 or 1 modular multiplications, respectively), for encryption and signature verification. Choosing ElGamal-family signatures, the private-key operation is inexpensive (a single modular multiplication, assuming precomputation).

(iii) DSA signatures (Chapter 11) or others with similar properties could be used in place of ElGamal signatures.

12.46 Remark (*signature scheme used to certify public keys*) Protocol 12.44 requires an ElGamal public key be certified using an RSA signature. This is done for reasons of efficiency, and highlights an advantage of allowing signature public keys from one system to be certified using signatures of a different type.

Beller-Yacobi protocol (2-pass)

Protocol 12.44 can be modified to yield a 2-pass protocol as illustrated in Figure 12.2. The modified protocol is obtained by essentially combining the pair of messages each party sends into a single message, as now described using notation as in Protocol 12.44.

B generates a random challenge m and sends to A: m, $cert_B$. A computes its ElGamal signature (v, w) on the concatenation $M = (m,\ I_B)$, and using part v of the signature as the session key $K = v$,[2] sends to B: $P_B(v)$, $E_v(cert_A,\ w)$. B recovers v $(= K)$ via public-key decryption, uses it to recover $cert_A$ and w, then verifies $cert_A$ and A's signature (v, w) on $M = (m,\ I_B)$.

The 2-pass protocol has slightly weaker authentication assurances: B obtains entity authentication of A and obtains a key K that A alone knows, while A has key authentication with respect to B. For A to obtain explicit key authentication of B (implying entity authentication also), a third message may be added whereby B exhibits knowledge through use of K on a challenge or standard message (e.g., $\{0\}^t$). All three of A's asymmetric operations remain inexpensive.

terminal A		server B
precompute x, $v = \alpha^x \bmod n_S$		select random challenge m
verify $cert_B$ via $P_T(G_B)$	$\longleftarrow$	send $m, cert_B$
compute $(v, w) = S_A(m, I_B)$		$cert_B = (I_B,\ n_B,\ G_B)$
send $P_B(v), E_v(cert_A, w)$	$\longrightarrow$	recover v, set $K = v$
$cert_A = (I_A,\ u_A,\ G_A)$		verify $cert_A$, signature (v, w)

Figure 12.2: *Summary of Beller-Yacobi protocol (2-pass).*

In Figure 12.2, an alternative to using $K = v$ as the session key is to set $K = w$. This results in the property that both parties influence the value of K (as w is a function of both m and x).

12.6 Key agreement based on asymmetric techniques

Diffie-Hellman key agreement (also called *exponential key exchange*) is a fundamental technique providing unauthenticated key agreement. This section discusses key establishment protocols based on exponential key agreement, as well as the concept of implicitly-certified public keys and their use in Diffie-Hellman protocols.

12.6.1 Diffie-Hellman and related key agreement protocols

This section considers the basic Diffie-Hellman protocol and related protocols providing various authentication assurances (see Table 12.4).

(i) Diffie-Hellman key agreement

Diffie-Hellman key agreement provided the first practical solution to the key distribution problem, allowing two parties, never having met in advance or shared keying material, to establish a shared secret by exchanging messages over an open channel. The security rests on the intractability of the Diffie-Hellman problem and the related problem of computing discrete logarithms (§3.6). The basic version (Protocol 12.47) provides protection in the form of secrecy of the resulting key from passive adversaries (eavesdroppers), but not from

[2] A side effect of using $K = v$ is that A no longer directly controls the key value, transforming the key transport protocol into a key agreement. Alternately, a random x could be chosen by A and used as key $K = x$, and x could be sent encrypted alongside w.

$\rightarrow$ Properties $\downarrow$ Protocol	key authentication	entity authentication	number of messages
Diffie-Hellman	none	none	2
ElGamal key agreement	unilateral	none	1
MTI/A0	mutual – implicit	none	2
Günther (see Remark 12.63)	mutual – implicit	none	2
STS	mutual – explicit	mutual	3

Table 12.4: *Selected key agreement protocols.*

active adversaries capable of intercepting, modifying, or injecting messages. Neither party has assurances of the source identity of the incoming message or the identity of the party which may know the resulting key, i.e., entity authentication or key authentication.

12.47 Protocol Diffie-Hellman key agreement (basic version)

SUMMARY: A and B each send the other one message over an open channel.
RESULT: shared secret K known to both parties A and B.

1. *One-time setup.* An appropriate prime p and generator α of $\mathbb{Z}_p^*$ $(2 \leq \alpha \leq p-2)$ are selected and published.
2. *Protocol messages.*

$$A \rightarrow B : \alpha^x \bmod p \quad (1)$$
$$A \leftarrow B : \alpha^y \bmod p \quad (2)$$

3. *Protocol actions.* Perform the following steps each time a shared key is required.
   (a) A chooses a random secret x, $1 \leq x \leq p-2$, and sends B message (1).
   (b) B chooses a random secret y, $1 \leq y \leq p-2$, and sends A message (2).
   (c) B receives α^x and computes the shared key as $K = (\alpha^x)^y \bmod p$.
   (d) A receives α^y and computes the shared key as $K = (\alpha^y)^x \bmod p$.

12.48 Note (*Diffie-Hellman with fixed exponentials*) A variation of Protocol 12.47 provides mutual key authentication. Fix α^x and $\alpha^y \bmod p$ as long-term public keys of the respective parties, and distribute these using signed certificates, thus fixing the long-term shared key for this user pair to $K = \alpha^{xy}$. If such certificates are available *a priori*, this becomes a zero-pass key agreement (no cryptographic messages need be exchanged). The time-invariant nature of this key K, however, is a drawback; Protocol 12.53 provides one resolution. A second solution involves use of key update techniques as in §12.3.1(ii).

12.49 Remark (*Diffie-Hellman in other groups*) The Diffie-Hellman protocol, and those based on it, can be carried out in any group in which both the discrete logarithm problem is hard and exponentiation is efficient. The most common examples of such groups used in practice are the multiplicative group $\mathbb{Z}_p^*$ of $\mathbb{Z}_p$, the analogous multiplicative group of $\mathbb{F}_{2^m}$, and the group of points defined by an elliptic curve over a finite field.

12.50 Note (*control over Diffie-Hellman key*) While it may appear as though Diffie-Hellman key agreement allows each party to guarantee key freshness and preclude key control, use of an exponential with small multiplicative order restricts the order (and thereby value) of the overall key. The most degenerate case for $\mathbb{Z}_p$ would be selection of 0 as private exponent, yielding an exponential with order 1 and the multiplicative identity itself as the resulting key. Thus, either participant may force the resulting key into a subset of the original (naively assumed) range set. Relatedly, some variants of Diffie-Hellman involving unauthenticated exponentials are vulnerable to the following active attack. Assume α generates $\mathbb{Z}_p^*$ where $p = Rq + 1$ (consider $R = 2$ and q prime). Then $\beta = \alpha^q = \alpha^{(p-1)/R}$ has order R ($\beta = -1$ for $R = 2$). If A and B exchange unauthenticated short-term exponentials α^x and α^y, an adversary may replace these by $(\alpha^x)^q$ and $(\alpha^y)^q$, forcing the shared key to be $K = \alpha^{xyq} = \beta^{xy}$, which takes one of only R values ($+1$ or -1 for $R = 2$). K may thus be found by exhaustive trial of R values. A more direct attack involves simply replacing the exchanged exponentials by $+1$ or $p - 1 = -1$. This general class of attacks may be prevented by authenticating the exchanged exponentials, e.g., by a digital signature.

(ii) ElGamal key agreement in one-pass

ElGamal key agreement is a Diffie-Hellman variant providing a one-pass protocol with unilateral key authentication (of the intended recipient to the originator), provided the public key of the recipient is known to the originator *a priori*. While related to ElGamal encryption (§8.4), the protocol is more simply Diffie-Hellman key agreement wherein the public exponential of the recipient is fixed and has verifiable authenticity (e.g., is embedded in a certificate).

12.51 Protocol ElGamal key agreement (half-certified Diffie-Hellman)

SUMMARY: A sends to B a single message allowing one-pass key agreement.
RESULT: shared secret K known to both parties A and B.

1. *One-time setup (key generation and publication).* Each user B does the following:
   Pick an appropriate prime p and generator α of $\mathbb{Z}_p^*$.
   Select a random integer b, $1 \le b \le p-2$, and compute $\alpha^b \bmod p$.
   B publishes its public key (p, α, α^b), keeping private key b secret.
2. *Protocol messages.*

$$A \to B: \quad \alpha^x \bmod p \quad (1)$$

3. *Protocol actions.* Perform the following steps each time a shared key is required.
   (a) A obtains an authentic copy of B's public key (p, α, α^b).
   A chooses a random integer x, $1 \le x \le p-2$, and sends B message (1).
   A computes the key as $K = (\alpha^b)^x \bmod p$.
   (b) B computes the same key on receipt of message (1) as $K = (\alpha^x)^b \bmod p$.

12.52 Remark (*assurances in one-pass ElGamal*) The recipient in Protocol 12.51 has no corroboration of whom it shares the secret key with, nor any key freshness assurances. Neither party obtains entity authentication or key confirmation.

(iii) MTI two-pass key agreement protocols

The MTI/A0 variant (Protocol 12.53) of Diffie-Hellman key agreement yields, in two messages (neither requiring signatures), time-variant session keys with mutual (implicit) key authentication against passive attacks. As in ElGamal key agreement (Protocol 12.51), A sends to B a single message, resulting in the shared key K. B independently initiates an analogous protocol with A, resulting in the shared key K'. Each of A and B then computes $k = KK' \bmod p$ (p and α are global parameters now). Neither entity authentication nor key confirmation is provided. Although appropriate for applications where only passive attacks are possible, this protocol is vulnerable to certain active attacks (see Note 12.54).

12.53 Protocol MTI/A0 key agreement

SUMMARY: two-pass Diffie-Hellman key agreement secure against passive attacks.
RESULT: shared secret K known to both parties A and B.

1. *One-time setup*. Select and publish (in a manner guaranteeing authenticity) an appropriate system prime p and generator α of $\mathbb{Z}_p^*$, $2 \le \alpha \le p-2$. A selects as a long-term private key a random integer a, $1 \le a \le p-2$, and computes a long-term public key $z_A = \alpha^a \bmod p$. (B has analogous keys b, z_B.) A and B have access to authenticated copies of each other's long-term public key.
2. *Protocol messages*.

$$\begin{array}{lll} A \to B: & \alpha^x \bmod p & (1) \\ A \leftarrow B: & \alpha^y \bmod p & (2) \end{array}$$

3. *Protocol actions*. Perform the following steps each time a shared key is required.
   (a) A chooses a random secret x, $1 \le x \le p-2$, and sends B message (1).
   (b) B chooses a random secret y, $1 \le y \le p-2$, and sends A message (2).
   (c) A computes the key $k = (\alpha^y)^a {z_B}^x \bmod p$.
   (d) B computes the key $k = (\alpha^x)^b {z_A}^y \bmod p$. (Both parties now share the key $k = \alpha^{bx+ay} \bmod p$.)

Table 12.5 summarizes Protocol 12.53 and three related two-pass protocols. All four of these MTI protocols provide mutual key authentication without key confirmation or entity authentication, and are role-symmetric: each party executes directly analogous operations. The protocols are also message-independent per Definition 12.12 (neither party requires receipt of the other's message before sending its own), although three of the four require *a priori* access to the other party's authentic public key. The remaining protocol – MTI/A0 – does not, and requires no additional passes (or communications delays) if this is not true; public keys may be exchanged e.g., via certificates included with the existing protocol messages. Thus in MTI/A0, the content of both messages sent is also independent (e.g., of the identity and public key) of the intended recipient.

↓Protocol	m_{AB}	m_{BA}	K_A	K_B	key K
MTI/A0	α^x	α^y	${m_{BA}}^a {z_B}^x$	${m_{AB}}^b {z_A}^y$	α^{bx+ay}
MTI/B0	${z_B}^x$	${z_A}^y$	${m_{BA}}^{a^{-1}} \alpha^x$	${m_{AB}}^{b^{-1}} \alpha^y$	α^{x+y}
MTI/C0	${z_B}^x$	${z_A}^y$	${m_{BA}}^{a^{-1}x}$	${m_{AB}}^{b^{-1}y}$	α^{xy}
MTI/C1	${z_B}^{xa}$	${z_A}^{yb}$	${m_{BA}}^x$	${m_{AB}}^y$	α^{abxy}

Table 12.5: *Selected MTI key agreement protocols. A and B have long-term secrets a and b, respectively, verifiably authentic corresponding long-term public keys $z_A = \alpha^a$, $z_B = \alpha^b \bmod p$, and random per-session secrets x and y, respectively. m_{AB} denotes the message A sends to B; m_{BA} is analogous. K_A and K_B are the final key K as computed by A and B.*

12.54 Note (*source-substitution attack on MTI/A0*) As a general rule in all public-key protocols (including Table 12.5), prior to accepting the authenticated public key of a party A, a party B should have assurance (either direct or through a trusted third party) that A actually knows the corresponding private key. Otherwise, an adversary C may claim A's public key as its own, allowing possible attacks, such as that on MTI/A0 as follows. Assume that in a particular implementation, A sends to B its certified public key in a certificate appended to message (1). C registers A's public key as its own (legitimately proving its own identity to the certificate-creating party). When A sends B message (1), C replaces A's certificate with its own, effectively changing the source indication (but leaving the exponential α^x sent by A to B unchanged). C forwards B's response α^y to A. B concludes that subsequently received messages encrypted by the key $k = \alpha^{bx+ay}$ originated from C, whereas, in fact, it is only A who knows k and can originate such messages.

A more complicated attack achieves the same, with C's public key differing from A's public key z_A. C selects an integer e, computes $(z_A)^e = \alpha^{ae}$, and registers the public key α^{ae}. C then modifies α^y sent by B in message (2) to $(\alpha^y)^e$. A and B each compute the key $k = \alpha^{aey}\alpha^{xb}$, which A believes is shared with B (and is), while B believes it is shared with C.

In both variations, C is not actually able to compute k itself, but rather causes B to have false beliefs. Such attacks may be prevented by modifying the protocol such that the exponentials are authenticated (cf. Note 12.50), and binding key confirmation evidence to an authenticated source indication, e.g., through a digital signature (cf. Remark 12.58). The MTI protocols are, however, also subject to certain theoretical known-key attacks (see p.538).

12.55 Remark (*implications of message independence*) Protocols such as MTI/A0 "leak" no information about long-term secrets, since the exchanged messages are independent thereof. However, such protocols in which each party's message is independent of the other's, and yet the session key depends on fresh input from each, cannot provide mutual explicit key authentication.

12.56 Remark (*computational complexity of MTI protocols*) The A0 and B0 protocols require 3 exponentiations by each party, whereas the C0 and C1 protocols require only 2. C1 has the additional advantage over B0 and C0 that no inverses are needed; however, these fixed long-term values may be precomputed.

(iv) Station-to-Station protocol (STS)

The following three-pass variation of the basic Diffie-Hellman protocol allows the establishment of a shared secret key between two parties with mutual entity authentication and mutual explicit key authentication. The protocol also facilitates anonymity – the identities of A and B may be protected from eavesdroppers. The method employs digital signatures; the description below is for the specific case of RSA signatures.

12.57 Protocol Station-to-Station protocol (STS)

SUMMARY: parties A and B exchange 3 messages.
RESULT: key agreement, mutual entity authentication, explicit key authentication.

1. *Notation.* E is a symmetric encryption algorithm.
   $S_A(m)$ denotes A's signature on m, defined as: $S_A(m) = (H(m))^{d_A} \bmod n_A$ (i.e., RSA preceded by an appropriate one-way hash function H, $H(m) < n_A$).
2. *One-time setup (definition and publication of system parameters).*
   (a) Select and publish an appropriate system prime p and generator α of $\mathbb{Z}_p^*$, $2 \leq \alpha \leq p-2$. (For additional security, each party may have its own unique such parameters as part of its public key.)
   (b) Each user A selects RSA public and private signature keys (e_A, n_A) and d_A, respectively (B has analogous keys). Assume each party has access to authentic

copies of the other's public key (if not, certificates can be included in existing messages (2) and (3)).

3. *Protocol messages.*

$$\begin{array}{lll} A \to B: & \alpha^x \bmod p & (1) \\ A \leftarrow B: & \alpha^y \bmod p,\ E_k(S_B(\alpha^y,\ \alpha^x)) & (2) \\ A \to B: & E_k(S_A(\alpha^x,\ \alpha^y)) & (3) \end{array}$$

4. *Protocol actions.* Perform the following steps each time a shared key is required. The protocol is aborted (with failure) immediately upon any signature failure.
   (a) A generates a secret random x, $1 \le x \le p-2$, and sends B message (1).
   (b) B generates a secret random y, $1 \le y \le p-2$, and computes the shared key $k = (\alpha^x)^y \bmod p$. B signs the concatenation of both exponentials ordered as in (2), encrypts this using the computed key, and sends A message (2).
   (c) A computes the shared key $k = (\alpha^y)^x \bmod p$, decrypts the encrypted data, and uses B's public key to verify the received value as the signature on the hash of the cleartext exponential received and the exponential sent in message (1). Upon successful verification, A accepts that k is actually shared with B, and sends B an analogous message (3).
   (d) B similarly decrypts the received message (3) and verifies A's signature therein. If successful, B accepts that k is actually shared with A.

The attack of Note 12.50 is precluded in the STS protocol due to the signatures over the exchanged exponentials.

12.58 Remark (*key confirmation in STS protocol*) Encryption under key k provides mutual key confirmation plus allows the conclusion that the party knowing the key is that which signed the exponentials. The optimal use of this protocol occurs when all subsequent messages are also to be encrypted under key k; if this is not the case, alternate means of key confirmation avoiding encryption may be preferable. One alternative is to use a MAC in messages (2) and (3), e.g., for $s = S_A(\alpha^x, \alpha^y)$, $A \to B : (s, \text{MAC}_k(s))$. A second alternative is inclusion of a one-way hash of k within the signed messages, e.g., $A \to B : S_A(\alpha^x,\ \alpha^y,\ h(k))$ where here $h(k)$ may be replaced by k alone if the signature process itself employs an appropriate one-way hash.

12.6.2 Implicitly-certified public keys

In contrast both to systems which use public-key certificates (§13.4.2) and to identity-based systems (§13.4.3), an alternate approach to distributing public keys involves *implicitly-certified public keys*, for which a framework is provided in §13.4.4. Use of the word *implicit* here is consistent with that in the term (implicit) key authentication. The current section presents several specific techniques involving implicitly-certified public keys.

(i) Implicitly-certified public keys (of Günther)

Mechanism 12.59 provides a method by which a trusted party may create a Diffie-Hellman public key $r^s \bmod p$ for an entity, with the key being implicitly-certified. Such public keys, which may be reconstructed from public data, may be used in key agreement protocols requiring certified Diffie-Hellman public keys (e.g., z_A in Protocol 12.53) as an alternative to transporting these keys by public-key certificates, or in customized protocols such as Protocol 12.62.

12.59 Mechanism Günther's implicitly-certified (identity-based) public keys

SUMMARY: a trusted party T creates an implicitly-certified, publicly-recoverable Diffie-Hellman public key for A, and transfers to A the corresponding private key.

1. A trusted server T selects an appropriate fixed public prime p and generator α of $\mathbb{Z}_p^*$. T selects a random integer t, with $1 \leq t \leq p-2$ and $\gcd(t, p-1) = 1$ as its private key, and publishes its public key $u = \alpha^t \bmod p$, along with α, p.
2. T assigns to each party A a unique *distinguished name* or identifying string I_A (e.g., name and address), and a random integer k_A with $\gcd(k_A, p-1) = 1$. T then computes $P_A = \alpha^{k_A} \bmod p$. ($P_A$ is A's *reconstruction public data*, allowing other parties to compute $(P_A)^a$ below. The gcd condition ensures that P_A itself is a generator.)
3. Using a suitable hash function h, T solves the following equation for a (restarting with a new k_A if $a = 0$):
$$h(I_A) \equiv t \cdot P_A + k_A \cdot a \pmod{p-1}. \tag{12.1}$$
4. T securely transmits to A the pair $(r,\ s) = (P_A,\ a)$, which is T's ElGamal signature (see Chapter 11) on I_A. (a is A's private key for Diffie-Hellman key-agreement.)
5. Any other party can then reconstruct A's (Diffie-Hellman) public key ${P_A}^a$ $(= \alpha^{k_A a})$ entirely from publicly available information (α, I_A, u, P_A, p) by computing (since $\alpha^{h(I_A)} \equiv u^{P_A} \cdot {P_A}^a$):
$${P_A}^a \equiv \alpha^{h(I_A)} \cdot u^{-P_A} \bmod p. \tag{12.2}$$

The above mechanism can be generalized to be independent of ElGamal signatures, by using any suitable alternate method to generate a pair $(r,\ s)$ where r is used as the reconstruction public data, the secret s is used as a (key-agreement) private key, and whereby the reconstructed public key $r^s \bmod p$ can be computed from public information alone.

12.60 Remark (*optimization of ElGamal signatures*) Equation (12.1) can be replaced by using the following optimization of the ElGamal signature scheme, where $\gcd(t, p-1) = 1$:
$$h(I_A) \equiv t \cdot a + k_A \cdot P_A \pmod{p-1}.$$
To solve for a then requires a one-time inverse computation ($t^{-1} \bmod p-1$) rather than the per-signature inverse computation ($(k_A)^{-1} \bmod p-1$) required by the original signature scheme. With this modification, A's key-agreement public key is u^a $(= \alpha^{ta})$ rather than ${P_A}^a$ $(= \alpha^{k_A a})$, correspondingly recovered by computing
$$\alpha^{h(I_A)} \cdot P_A^{-P_A} \bmod p \; (= \alpha^{ta} \bmod p). \tag{12.3}$$

(ii) Self-certified public keys (of Girault)

Mechanism 12.61, which is employed in several protocols in §12.6.3, presents a technique for creating implicitly-certified public keys. It differs from that of Mechanism 12.59 in that it allows users to "self-certify" the keys, in the sense that the user itself is the only party knowing the private key (as opposed to the trusted party having access to each party's private key).

12.61 Mechanism Girault's self-certified public keys

SUMMARY: a trusted party T creates an implicitly-certified, publicly-recoverable Diffie-Hellman public key for party A, without learning the corresponding private key.

1. A trusted server T selects secret primes p and q for an RSA integer $n = pq$, an element α of maximal order in $\mathbb{Z}_n^*$ (see Algorithm 4.83), and appropriate integers e and d as a (public, private) RSA key pair for n.
2. T assigns to each party A a unique *distinguished name* or identifying string I_A (e.g., name and address).
3. Party A itself chooses a private key a, and provides the public key $\alpha^a \bmod n$ to T in an authenticatable manner. (α^a is A's key-agreement public key.) Moreover, A provides proof to T that it knows the corresponding secret a. (This is necessary to prevent a certain forgery attack by A in some ways analogous to that of Note 12.54, and might be done by A producing for T a Diffie-Hellman key based on α^a and an exponential chosen by T.)
4. T computes A's reconstruction public data (essentially replacing a certificate) as $P_A = (\alpha^a - I_A)^d \bmod n$. (Thus $({P_A}^e + I_A) \bmod n = \alpha^a \bmod n$, and from public information alone, any party can compute A's public key, $\alpha^a \bmod n$.)

12.6.3 Diffie-Hellman protocols using implicitly-certified keys

The authenticity of Diffie-Hellman exponentials used as public keys in authenticated key agreement protocols can be established by distributing them via public-key certificates, or by reconstructing them as implicitly-certified public keys (e.g., using Mechanisms of §12.6.2) from publicly available parameters. Protocol 12.62 is one example of the latter. The idea may be adopted to other Diffie-Hellman based protocols as further illustrated by Examples 12.64, 12.65, and 12.66 respectively corresponding to the fixed-key Diffie-Hellman, ElGamal, and MTI/A0 key agreement protocols of §12.6.1.

12.62 Protocol Günther's key agreement protocol

SUMMARY: Diffie-Hellman based key agreement protocol between A and B.
RESULT: A and B establish shared secret K with key authentication.

1. *One-time setup (definition of global parameters).* Using Mechanism 12.59, a trusted party T constructs ElGamal signatures $(P_A,\ a)$ and $(P_B,\ b)$ on the identities I_A and I_B of A and B, respectively, and gives these signatures respectively to A and B as secrets, along with the following authentic public system parameters as per Mechanism 12.59: a prime p, generator α of $\mathbb{Z}_p^*$, and T's public key u.
2. *Protocol messages.*

$$\begin{array}{lll} A \rightarrow B: & I_A, P_A & (1) \\ A \leftarrow B: & I_B, P_B, (P_A)^y \bmod p & (2) \\ A \rightarrow B: & (P_B)^x \bmod p & (3) \end{array}$$

3. *Protocol actions.* Perform the following steps each time a shared key is required.
   (a) A sends B message (1).
   (b) B generates a random integer y, $1 \le y \le p-2$, and sends A message (2).
   (c) A generates a random integer x, $1 \le x \le p-2$, and sends B message (3).

(d) *Key computation.* As per Mechanism 12.59, A and B respectively construct the other's identity-based public key (equivalent to $(P_B)^b$ and $(P_A)^a \bmod p$, respectively). The common key-agreement key K $(= \alpha^{k_A ya + k_B bx})$ is established as A and B respectively compute $K = ({P_A}^y)^a \cdot ({P_B}^b)^x$, $K = ({P_A}^a)^y \cdot ({P_B}^x)^b \bmod p$.

Protocol 12.62 is subject to theoretical known-key attacks similar to those which apply to the MTI protocols (Note 12.54).

12.63 Remark (*two-pass Günther protocol*) In Protocol 12.62, a party's identity information and long-term public key (respectively, I_A and P_A) are long-term parameters. If these are known to parties *a priori*, then this three-pass protocol reduces to two passes. The reduced protocol provides the same assurances, namely, key agreement with key authentication, as Protocol 12.62 and the two-pass MTI schemes of Table 12.5, and closely resembles MTI/A0 with respect to the logarithm of the final key.

12.64 Example (*Protocol G0*) Fixed-key Diffie-Hellman key-agreement (Note 12.48) may be modified to use implicitly-certified keys as follows. Using the setup and notation as in Girault's self-certified public keys (Mechanism 12.61), A and B establish the time-invariant joint key K by respectively computing $(P_B)^e + I_B \bmod n$ $(= \alpha^b)$ and $(P_A)^e + I_A \bmod n$ $(= \alpha^a)$, from which they effectively compute

$$K = (\alpha^b)^a \quad \text{and} \quad K = (\alpha^a)^b \quad \bmod n. \tag{12.4}$$

Alternatively, the same protocol may be modified to use Günther's ID-based public keys assuming the setup and notation as in Mechanism 12.59 with modified ElGamal signatures as per Remark 12.60. In this case, A and B respectively compute the other's key-agreement public keys α^{tb} and α^{ta} by (12.3), in place of α^b and α^a in (12.4). □

12.65 Example (*Protocol G1*) The one-pass ElGamal key agreement of Protocol 12.51 may be modified to use implicitly-certified keys as follows. Using the setup and notation as in Girault's self-certified public keys (Mechanism 12.61), A chooses a random integer x and sends to B: $\alpha^x \bmod n$. A computes ${P_B}^e + I_B \bmod n$ $(= \alpha^b)$. A and B establish the time-variant joint key $K = \alpha^{bx} \bmod n$, by respectively computing, effectively,

$$K = (\alpha^b)^x \quad \text{and} \quad K = (\alpha^x)^b \quad \bmod n. \tag{12.5}$$

The protocol may be modified to use Günther's ID-based public keys as follows: rather than sending $\alpha^x \bmod n$ to B, A sends ${P_B}^x \bmod p$, with P_B (and p, b, u, etc.) defined as in Mechanism 12.59. B then computes $K = ({P_B}^x)^b \bmod p$, while A effectively computes $K = ({P_B}^b)^x \bmod p$, having reconstructed ${P_B}^b$ via equation (12.2) on page 521. The resulting protocol is essentially one-half of the Günther key agreement of Protocol 12.62. A related modification utilizing Remark 12.60 involves A sending to B $u^x \bmod p$ in place of ${P_B}^x$, the joint key now being $K = u^{bx} \bmod p$, computed by A as $K = (u^b)^x$ with u^b computed per (12.3), and B computing $K = (u^x)^b \bmod p$. This final protocol then resembles (one-half of) Protocol MTI/A0 in that, since the message A sends is independent of the recipient B, it may be computed ahead of time before the recipient is determined. □

12.66 Example (*Protocol G2*) The two-pass MTI/A0 key agreement (Protocol 12.53) may be modified to use implicitly-certified keys as follows. Using the setup and notation as in Girault's self-certified public keys (Mechanism 12.61), A chooses a random integer x and sends to B: $\alpha^x \bmod n$. Analogously, B chooses a random integer y and sends to A: α^y

$\bmod n$. A computes ${P_B}^e + I_B \bmod n\ (= \alpha^b)$; B computes ${P_A}^e + I_A \bmod n\ (= \alpha^a)$. A and B then establish the time-variant common key $K = \alpha^{ay+bx} \pmod n$ by respectively computing $K = (\alpha^y)^a({P_B}^e + I_B)^x$ and $K = (\alpha^x)^b({P_A}^e + I_A)^y \bmod n$. Alternatively, this protocol may be modified to use Günther's ID-based public keys in a manner directly analogous to that of Example 12.64. □

12.67 Example (*self-certified version of Günther's ID-based keys*) The following modification of Mechanism 12.59 transforms it into a "self-certified" public-key scheme (i.e., one in which the third party does not learn users' private keys). A chooses a secret random v, $1 \le v \le p-1$ with $\gcd(v, p-1) = 1$, computes $w = \alpha^v \bmod p$, and gives w to T. While v is not given to T, A should demonstrate knowledge of v to T (cf. Note 12.54). T chooses k_A as before but computes $P_A = w^{k_A} \bmod p$ (instead of: $P_A = \alpha^{k_A}$). T solves equation (12.1) for a as before (using the new P_A) and again gives A the pair $(r, s) = (P_A, a)$. A then calculates $a' = a \cdot v^{-1} \bmod (p-1)$; it follows that (P_A, a') is now T's ElGamal signature on I_A (it is easily verified that $u^{P_A} \cdot {P_A}^{a'} \equiv \alpha^{h(I_A)}$), and T does not know a'. □

12.7 Secret sharing

Secret sharing schemes are multi-party protocols related to key establishment. The original motivation for secret sharing was the following. To safeguard cryptographic keys from loss, it is desirable to create backup copies. The greater the number of copies made, the greater the risk of security exposure; the smaller the number, the greater the risk that all are lost. Secret sharing schemes address this issue by allowing enhanced reliability without increased risk. They also facilitate distributed trust or shared control for critical activities (e.g., signing corporate cheques; opening bank vaults), by gating the critical action on cooperation by t of n users.

The idea of *secret sharing* is to start with a secret, and divide it into pieces called *shares* which are distributed amongst users such that the pooled shares of specific subsets of users allow reconstruction of the original secret. This may be viewed as a key pre-distribution technique, facilitating one-time key establishment, wherein the recovered key is pre-determined (static), and, in the basic case, the same for all groups.

A secret sharing scheme may serve as a *shared control scheme* if inputs (shares) from two or more users are required to enable a critical action (perhaps the recovered key allows this action to trigger, or the recovery itself is the critical action). In what follows, simple shared-control schemes introduced in §12.7.1 are a subset of threshold schemes discussed in §12.7.2, which are themselves a subclass of generalized secret sharing schemes as described in §12.7.3.

12.7.1 Simple shared control schemes

(i) Dual control by modular addition

If a secret number S, $0 \le S \le m-1$ for some integer m, must be entered into a device (e.g., a seed key), but for operational reasons, it is undesirable that any single individual (other than a trusted party) know this number, the following scheme may be used. A trusted party T generates a random number $1 \le S_1 \le m-1$, and gives the values S_1 and $S - S_1 \bmod m$ to two parties A and B, respectively. A and B then separately enter their values into the device, which sums them modulo m to recover S. If A and B are trusted not to collude, then neither one has any information about S, since the value each possesses is a random number between 0 and $m-1$. This is an example of a *split-knowledge* scheme – knowledge of the secret S is split among two people. Any action requiring S is said to be under *dual control* – two people are required to trigger it.

(ii) Unanimous consent control by modular addition

The dual control scheme above is easily generalized so that the secret S may be divided among t users, all of whom are required in order to recover S, as follows: T generates $t-1$ independent random numbers S_i, $0 \leq S_i \leq m-1$, $1 \leq i \leq t-1$. Parties P_1 through P_{t-1} are given S_i, while P_t is given $S_t = S - \sum_{i=1}^{t-1} S_i \bmod m$. The secret is recovered as $S = \sum_{i=1}^{t} S_i \bmod m$. Both here and in the dual control scheme above, modulo m operations may be replaced by exclusive-OR, using data values S and S_i of fixed bit-length $\lg(m)$.

12.68 Remark (*technique for splitting keys*) The individual key components in a split control scheme should be full-length. This provides greater security than partitioning an r-bit key into t pieces of r/t bits each. For example, for $r = 56$ and $t = 2$, if two parties are each given 28 bits of the key, exhaustive search by one party requires only 2^{28} trials, while if each party is given a 56-bit piece, 2^{56} trials are necessary.

12.7.2 Threshold schemes

12.69 Definition A (t, n) *threshold scheme* ($t \leq n$) is a method by which a trusted party computes secret *shares* S_i, $1 \leq i \leq n$ from an initial secret S, and securely distributes S_i to user P_i, such that the following is true: any t or more users who pool their shares may easily recover S, but any group knowing only $t-1$ or fewer shares may not. A *perfect* threshold scheme is a threshold scheme in which knowing only $t-1$ or fewer shares provide no advantage (no information about S whatsoever, in the information-theoretic sense) to an opponent over knowing no pieces.

The split-knowledge scheme of §12.7.1(i) is an example of a $(2, 2)$ threshold scheme, while the unanimous consent control of §12.7.1(ii) is a (t, t) threshold scheme.

12.70 Remark (*use of threshold schemes*) If a threshold scheme is to be reused without decreased security, controls are necessary to prevent participants from deducing the shares of other users. One method is to prevent group members themselves from accessing the value of the recovered secret, as may be done by using a trusted combining device. This is appropriate for systems where the objective is shared control, and participants need only see that an action is triggered, rather than have access to the key itself. For example, each share might be stored on a chipcard, and each user might swipe its card through a trusted card reader which computes the secret, thereby enabling the critical action of opening an access door.

Shamir's threshold scheme

Shamir's threshold scheme is based on polynomial interpolation, and the fact that a univariate polynomial $y = f(x)$ of degree $t-1$ is uniquely defined by t points (x_i, y_i) with distinct x_i (since these define t linearly independent equations in t unknowns).

12.71 Mechanism Shamir's (t, n) threshold scheme

SUMMARY: a trusted party distributes shares of a secret S to n users.
RESULT: any group of t users which pool their shares can recover S.

1. *Setup*. The trusted party T begins with a secret integer $S \geq 0$ it wishes to distribute among n users.
   (a) T chooses a prime $p > \max(S, n)$, and defines $a_0 = S$.
   (b) T selects $t-1$ random, independent coefficients $a_1, \ldots, a_{t-1}$, $0 \leq a_j \leq p-1$, defining the random polynomial over $\mathbb{Z}_p$, $f(x) = \sum_{j=0}^{t-1} a_j x^j$.
   (c) T computes $S_i = f(i) \bmod p$, $1 \leq i \leq n$ (or for any n distinct points i, $1 \leq i \leq p-1$), and securely transfers the share S_i to user P_i, along with public index i.
2. *Pooling of shares*. Any group of t or more users pool their shares (see Remark 12.70). Their shares provide t distinct points $(x, y) = (i, S_i)$ allowing computation of the coefficients a_j, $1 \leq j \leq t-1$ of $f(x)$ by Lagrange interpolation (see below). The secret is recovered by noting $f(0) = a_0 = S$.

The coefficients of an unknown polynomial $f(x)$ of degree less than t, defined by points (x_i, y_i), $1 \leq i \leq t$, are given by the Lagrange interpolation formula:

$$f(x) = \sum_{i=1}^{t} y_i \prod_{1 \leq j \leq t, j \neq i} \frac{x - x_j}{x_i - x_j}.$$

Since $f(0) = a_0 = S$, the shared secret may be expressed as:

$$S = \sum_{i=1}^{t} c_i y_i \;, \quad \text{where } c_i = \prod_{1 \leq j \leq t, j \neq i} \frac{x_j}{x_j - x_i}.$$

Thus each group member may compute S as a linear combination of t shares y_i, since the c_i are non-secret constants (which for a fixed group of t users may be pre-computed).

12.72 Note (*properties of Shamir's threshold scheme*) Properties of Mechanism 12.71 include:

1. *perfect*. Given knowledge of any $t-1$ or fewer shares, all values $0 \leq S \leq p-1$ of the shared secret remain equally probable (see Definition 12.69).
2. *ideal*. The size of one share is the size of the secret (see Definition 12.76).
3. *extendable for new users*. New shares (for new users) may be computed and distributed without affecting shares of existing users.
4. *varying levels of control possible*. Providing a single user with multiple shares bestows more control upon that individual. (In the terminology of §12.7.3, this corresponds to changing the access structure.)
5. *no unproven assumptions*. Unlike many cryptographic schemes, its security does not rely on any unproven assumptions (e.g., about the difficulty of number-theoretic problems).

12.7.3 Generalized secret sharing

The idea of a threshold scheme may be broadened to a *generalized secret sharing scheme* as follows. Given a set P of users, define $\mathcal{A}$ (the *access structure*) to be a set of subsets, called the *authorized subsets* of P. Shares are computed and distributed such that the pooling of shares corresponding to any authorized subset $A \in \mathcal{A}$ allows recovery of the secret S, but the pooling of shares corresponding to any unauthorized subset $B \subseteq P, B \notin \mathcal{A}$ does not.

Threshold schemes are a special class of generalized secret sharing schemes, in which the access structure consists of precisely all t-subsets of users. An access structure is called *monotone* if, whenever a particular subset A of users is an authorized subset, then any subset of P containing A is also authorized. Monotone access structures are a requirement in many applications, and most natural schemes are monotone. Perfect secret sharing schemes have a monotone access structure as a consequence of the entropy formulation in Definition 12.73.

12.73 Definition A secret sharing scheme is *perfect* if the shares corresponding to each unauthorized subset provide absolutely no information, in the information-theoretic sense, about the shared secret (cf. Definition 12.69). More formally, where H denotes entropy (see §2.2.1), and A, B are sets of users using the above notation: $H(S|A) = 0$ for any $A \in \mathcal{A}$, while $H(S|B) = H(S)$ for any $B \notin \mathcal{A}$.

The efficiency of a secret sharing scheme is measured by its information rate.

12.74 Definition For secret sharing schemes, the *information rate* for a particular user is the bitsize ratio (size of the shared secret)/(size of that user's share). The *information rate* for a secret sharing scheme itself is the minimum such rate over all users.

12.75 Fact (*perfect share bound*) In any perfect secret sharing scheme the following holds for all user shares: (size of a user share) $\geq$ (size of the shared secret). Consequently, all perfect secret sharing schemes must have information rate ≤ 1.

Justification. If any user P_i had a share of bit-size less than that of the secret, knowledge of the shares (excepting that of P_i) corresponding to any authorized set to which P_i belonged, would reduce the uncertainty in the secret to at most that in P_i's share. Thus by definition, the scheme would not be perfect.

12.76 Definition Secret sharing schemes of rate 1 (see Definition 12.74) are called *ideal*.

As per Note 12.72, Shamir's threshold scheme is an example of an ideal secret sharing scheme. Examples of access structures are known for which it has been proven that ideal schemes do not exist.

Secret sharing schemes with extended capabilities

Secret sharing schemes with a variety of extended capabilities exist, including:

1. *pre-positioned secret sharing schemes*. All necessary secret information is put in place excepting a single (constant) share which must later be communicated, e.g., by broadcast, to activate the scheme.
2. *dynamic secret sharing schemes*. These are pre-positioned schemes wherein the secrets reconstructed by various authorized subsets vary with the value of communicated activating shares.
3. *multi-secret threshold schemes*. In these secret sharing schemes different secrets are associated with different authorized subsets.
4. *detection of cheaters*, and *verifiable secret sharing*. These schemes respectively address *cheating* by one or more group members, and the distributor of the shares.

5. *secret sharing with disenrollment*. These schemes address the issue that when a secret share of a (t, n) threshold scheme is made public, it becomes a $(t-1, n)$ scheme.

12.8 Conference keying

12.77 Definition A *conference keying protocol* is a generalization of two-party key establishment to provide three or more parties with a shared secret key.

Despite superficial resemblance, conference keying protocols differ from dynamic secret sharing schemes in fundamental aspects. General requirements for conference keying include that distinct groups recover distinct keys (session keys); that session keys are dynamic (excepting key pre-distribution schemes); that the information exchanged between parties is non-secret and transferred over open channels; and that each party individually computes the session key (vs. pooling shares in a black box). A typical application is telephone conference calls. The group able to compute a session key is called the *privileged subset*. When a central point enables members of a (typically large) privileged subset to share a key by broadcasting one or more messages, the process resembles pre-positioned secret sharing somewhat and is called *broadcast encryption*.

An obvious method to establish a conference key K for a set of $t \geq 3$ parties is to arrange that each party share a unique symmetric key with a common trusted party. Thereafter the trusted party may choose a new random key and distribute it by symmetric key transport individually to each member of the conference group. Disadvantages of this approach include the requirement of an on-line trusted third party, and the communication and computational burden on this party.

A related approach not requiring a trusted party involves a designated group member (the *chair*) choosing a key K, computing pairwise Diffie-Hellman keys with each other group member, and using such keys to securely send K individually to each. A drawback of this approach is the communication and computational burden on the chair, and the lack of protocol symmetry (balance). Protocol 12.78 offers an efficient alternative, albeit more complex in design.

Burmester-Desmedt conference keying protocol

The following background is of use in understanding Protocol 12.78. t users U_0 through U_{t-1} with individual Diffie-Hellman exponentials $z_i = \alpha^{r_i}$ will form a conference key $K = \alpha^{r_0r_1+r_1r_2+r_2r_3+\cdots+r_{t-1}r_0}$. Define $A_j = \alpha^{r_jr_{j+1}} = z_j^{r_{j+1}}$ and $X_j = \alpha^{r_{j+1}r_j - r_jr_{j-1}}$. Noting $A_j = A_{j-1}X_j$, K may equivalently be written as (with subscripts taken modulo t)

$$\begin{aligned} K_i = \;& A_0A_1\cdots A_{t-1} = A_{i-1}A_iA_{i+1}\cdots A_{i+(t-2)} \\ = \;& A_{i-1}\cdot(A_{i-1}X_i)\cdot(A_{i-1}X_iX_{i+1})\cdots(A_{i-1}X_iX_{i+1}\cdots X_{i+(t-2)}). \end{aligned}$$

Noting $A_{i-1}{}^t = (z_{i-1})^{tr_i}$, this is seen to be equivalent to K_i as in equation (12.6) of Protocol 12.78.

12.78 Protocol Burmester-Desmedt conference keying

SUMMARY: $t \geq 2$ users derive a common conference key K.
RESULT: K is secure from attack by passive adversaries.

1. *One-time setup*. An appropriate prime p and generator α of $\mathbb{Z}_p^*$ are selected, and authentic copies of these are provided to each of n system users.

2. *Conference key generation*. Any group of $t \le n$ users (typically $t \ll n$), derive a common conference key K as follows. (Without loss of generality, the users are labeled U_0 through U_{t-1}, and all indices j indicating users are taken modulo t.)
   (a) Each U_i selects a random integer r_i, $1 \le r_i \le p-2$, computes $z_i = \alpha^{r_i} \bmod p$, and sends z_i to each of the other $t-1$ group members. (Assume that U_i has been notified *a priori*, of the indices j identifying other conference members.)
   (b) Each U_i, after receiving z_{i-1} and z_{i+1}, computes $X_i = (z_{i+1}/z_{i-1})^{r_i} \bmod p$ (note $X_i = \alpha^{r_{i+1}r_i - r_i r_{i-1}}$), and sends X_i to each of the other $t-1$ group members.
   (c) After receiving X_j, $1 \le j \le t$ excluding $j = i$, U_i computes $K = K_i$ as

$$K_i = (z_{i-1})^{tr_i} \cdot {X_i}^{t-1} \cdot {X_{i+1}}^{t-2} \cdots {X_{i+(t-3)}}^{2} \cdot {X_{i+(t-2)}}^{1} \bmod p \qquad (12.6)$$

For small conferences (small t), the computation required by each party is small, since all but one exponentiation in equation (12.6) involves an exponent between 1 and t. The protocol requires an order be established among users in the privileged subset (for indexing). For $t = 2$, the resulting key is $K = (\alpha^{r_1 r_2})^2$, the square of the standard Diffie-Hellman key. It is provably as difficult for a passive adversary to deduce the conference key K in Protocol 12.78 as to solve the Diffie-Hellman problem.

Attention above has been restricted to unauthenticated conference keying; additional measures are required to provide authentication in the presence of active adversaries. Protocol 12.78 as presented assumes a broadcast model (each user exchanges messages with all others); it may also be adapted for a bi-directional ring (wherein each user transmits only to two neighbors).

Unconditionally secure conference keying

While conference keying schemes such as Protocol 12.78 provide computational security, protocols with the goal of unconditional security are also of theoretical interest. Related to this, a generalization of Fact 12.34 is given below, for conferences of fixed size (t participants from among n users) which are information-theoretically secure against conspiracies of up to j non-participants. The model for this result is a non-interactive protocol, and more specifically a key pre-distribution scheme: each conference member computes the conference key solely from its own secret data (pre-distributed by a server) and an identity vector specifying (an ordered sequence of) indices of the other conference members.

12.79 Fact (*Blundo's conference KDS bound*) In any j-secure conference KDS providing m-bit conference keys to privileged subsets of fixed size t, the secret data stored by each user must be at least $m \cdot \binom{j+t-1}{t-1}$ bits.

Fact 12.79 with $t = 2$ and $j = n - 2$ corresponds to the trivial scheme (see p.505) where each user has $n - 1$ shared keys each of m bits, one for each other user. A non-trivial scheme meeting the bound of Fact 12.79 can be constructed as a generalization of Mechanism 12.35 (see p.540).

12.80 Remark (*refinement of Fact 12.79*) A more precise statement of Fact 12.79 requires consideration of entropy; the statement holds if the conference keys in question have m bits of entropy.

12.9 Analysis of key establishment protocols

The main objective of this section is to highlight the delicate nature of authenticated key establishment protocols, and the subtlety of design flaws. Examples of flawed protocols are included to illustrate typical attack strategies, and to discourage protocol design by the novice.

12.9.1 Attack strategies and classic protocol flaws

The study of successful attacks which have uncovered flaws in protocols allows one to learn from previous design errors, understand general attack methods and strategies, and formulate design principles. This both motivates and allows an understanding of various design features of protocols. General attack strategies are discussed in §12.2.3. In the specific examples below, A and B are the legitimate parties, and E is an adversary (enemy). Two of the protocols discussed are, in fact, authentication-only protocols (i.e., do not involve key establishment), but are included in this discussion because common principles apply.

Attack 1: Intruder-in-the-middle

The classic "intruder-in-the-middle" attack on unauthenticated Diffie-Hellman key agreement is as follows.

$$\begin{array}{ccccc} A & & E & & B \\ \rightarrow & \alpha^{x} & \rightarrow & \alpha^{x'} & \rightarrow \\ \leftarrow & \alpha^{y'} & \leftarrow & \alpha^{y} & \leftarrow \end{array}$$

A and B have private keys x and y, respectively. E creates keys x' and y'. E intercepts A's exponential and replaces it by $\alpha^{x'}$; and intercepts B's exponential, replacing it with $\alpha^{y'}$. A forms session key $K_A = \alpha^{xy'}$, while B forms session key $K_B = \alpha^{x'y}$. E is able to compute both these keys. When A subsequently sends a message to B encrypted under K_A, E deciphers it, re-enciphers under K_B, and forwards it to B. Similarly E deciphers messages encrypted by B (for A) under K_B, and re-enciphers them under K_A. A and B believe they communicate securely, while E reads all traffic.

Attack 2: Reflection attack

Suppose A and B share a symmetric key K, and authenticate one another on the basis of demonstrating knowledge of this key by encrypting or decrypting a challenge as follows.

$$\begin{array}{cccc} A & & B & \\ \rightarrow & r_A & & (1) \\ & E_K(r_A, r_B) & \leftarrow & (2) \\ \rightarrow & r_B & & (3) \end{array}$$

An adversary E can impersonate B as follows. Upon A sending (1), E intercepts it, and initiates a new protocol, sending the identical message r_A back to A as message (1) purportedly from B. In this second protocol, A responds with message (2′): $E_K(r_A, {r_A}')$, which E again intercepts and simply replays back on A as the answer (2) in response to the challenge r_A in the original protocol. A then completes the first protocol, and believes it has successfully authenticated B, while in fact B has not been involved in any communications.

$$\begin{array}{llcr}
A & & E & \\
\rightarrow & r_A & & (1) \\
 & r_A & \leftarrow & (1') \\
\rightarrow & E_K(r_A, r_A{}') & & (2') \\
 & E_K(r_A, r_B = r_A{}') & \leftarrow & (2) \\
\rightarrow & r_B & & (3)
\end{array}$$

The attack can be prevented by using distinct keys K and K' for encryptions from A to B and B to A, respectively. An alternate solution is to avoid message symmetry, e.g., by including the identifier of the originating party within the encrypted portion of (2).

Attack 3: Interleaving attack

Consider the following (flawed) authentication protocol, where s_A denotes the signature operation of party A, and it is assumed that all parties have authentic copies of all others' public keys.

$$\begin{array}{llcr}
A & & B & \\
\rightarrow & r_A & & (1) \\
 & r_B, s_B(r_B, r_A, A) & \leftarrow & (2) \\
\rightarrow & r_A{}', s_A(r_A{}', r_B, B) & & (3)
\end{array}$$

The intention is that the random numbers chosen by A and B, respectively, together with the signatures, provide a guarantee of freshness and entity authentication. However, an enemy E can initiate one protocol with B (pretending to be A), and another with A (pretending to be B), as shown below, and use a message from the latter protocol to successfully complete the former, thereby deceiving B into believing E is A (and that A initiated the protocol).

$$\begin{array}{llllcr}
A & & E & & B & \\
 & & \rightarrow & r_A & & (1) \\
 & & & r_B, s_B(r_B, r_A, A) & \leftarrow & (2) \\
 & r_B & \leftarrow & & & (1') \\
\rightarrow & r_A{}', s_A(r_A{}', r_B, B) & & & & (2') \\
 & & \rightarrow & r_A{}', s_A(r_A{}', r_B, B) & & (3)
\end{array}$$

This attack is possible due to the message symmetry of (2) and (3), and may be prevented by making their structures differ, securely binding an identifier to each message indicating a message number, or simply requiring the original r_A take the place of $r_A{}'$ in (3).

The implications of this attack depend on the specific objectives the protocol was assumed to provide. Such specific objectives are, however, (unfortunately) often not explicitly stated.

Attack 4: Misplaced trust in server

The Otway-Rees protocol (Protocol 12.29) has messages as follows:

$$\begin{array}{lllr}
A \rightarrow B: & M, A, B, E_{K_{AT}}(N_A, M, A, B) & & (1) \\
B \rightarrow T: & M, A, B, E_{K_{AT}}(N_A, M, A, B), E_{K_{BT}}(N_B, M, A, B) & & (2) \\
B \leftarrow T: & E_{K_{AT}}(N_A, k), E_{K_{BT}}(N_B, k) & & (3) \\
A \leftarrow B: & E_{K_{AT}}(N_A, k) & & (4)
\end{array}$$

Upon receiving message (2), the server must verify that the encrypted fields (M, A, B) in both parts of (2) match, and in addition that these fields match the cleartext (M, A, B). If the latter check is not carried out, the protocol is open to attack by an enemy E (who is another authorized system user) impersonating B as follows. E modifies (2), replacing cleartext B

by E (but leaving both enciphered versions of both identifiers A and B intact), replacing nonce N_B by its own nonce N_E, and using key K_{ET} (which E shares *a priori* with T) in place of K_{BT}. Based on the cleartext identifier E, T then encrypts part of message (3) under K_{ET} allowing E to recover k; but A believes, as in the original protocol, that k is shared with B. The attack is summarized as follows.

$$\begin{array}{lll}
A \to B: & M, A, B, E_{K_{AT}}(N_A, M, A, B) & (1) \\
B \to E: & M, A, B, E_{K_{AT}}(N_A, M, A, B), E_{K_{BT}}(N_B, M, A, B) & (2) \\
E \to T: & M, A, E, E_{K_{AT}}(N_A, M, A, B), E_{K_{ET}}(N_E, M, A, B) & (2') \\
E \leftarrow T: & E_{K_{AT}}(N_A, k), E_{K_{ET}}(N_E, k) & (3) \\
A \leftarrow E: & E_{K_{AT}}(N_A, k) & (4)
\end{array}$$

The attack is possible due to the subtle manner by which A infers the identity of the other party to which k is made available: in (4), A has no direct indication of the other party to which T has made k available, but relies on the nonce N_A in (4) and its association with the pair (N_A, B) within the protected part of (1). Thus, A relies on (or delegates trust to) the server to make k available only to the party requested by A, and this can be assured only by T making use of the protected fields (M, A, B).

12.9.2 Analysis objectives and methods

The primary aim of protocol analysis is to establish confidence in the cryptographic security of a protocol. The following definitions aid discussion of protocol analysis.

12.81 Definition A key establishment protocol is *operational* (or *compliant*) if, in the absence of active adversaries and communications errors, honest participants who comply with its specification always complete the protocol having computed a common key and knowledge of the identities of the parties with whom the key is shared.

The most obvious objectives and properties of key establishment protocols, namely authenticity and secrecy of keys, are discussed in §12.2.2.

12.82 Definition A key establishment protocol is *resilient* if it is impossible for an active adversary to mislead honest participants as to the final outcome.

Protocol analysis should confirm that a protocol meets all claimed objectives. As a minimum, for a key establishment protocol this should include being operational (note this implies no security guarantees), providing both secrecy and authenticity of the key, and being resilient. Key authenticity implies the identities of the parties sharing the key are understood and corroborated, thus addressing impersonation and substitution. Resilience differs subtlely from authentication, and is a somewhat broader requirement (e.g., see the attack of Note 12.54). Additional objectives beyond authenticated key establishment may include key confirmation, perfect forward secrecy, detection of key re-use, and resistance to known-key attacks (see §12.2.3).

In addition to verifying objectives are met, additional benefits of analysis include:

1. explicit identification of assumptions on which the security of a protocol is based;
2. identification of protocol properties, and precise statement of its objectives (this facilitates comparison with other protocols, and determining appropriateness);
3. examination of protocol efficiency (with respect to bandwidth and computation).

Essentially all protocol analysis methods require the following (implicitly or explicitly):

1. *protocol specification* – an unambiguous specification of protocol messages, when they are sent, and the actions to be taken upon reception thereof;
2. *goals* – an unambiguous statement of claimed assurances upon completion;
3. *assumptions and initial state* – a statement of assumptions and initial conditions;
4. *proof* – some form of argument that, given the assumptions and initial state, the specified protocol steps lead to a final state meeting the claimed goals.

Analysis methods

Common approaches for analyzing cryptographic protocols include the following:

1. *ad hoc and practical analysis*. This approach consists of any variety of convincing arguments that any successful protocol attack requires a resource level (e.g., time or space) greater than the resources of the perceived adversary. Protocols which survive such analysis are said to have *heuristic security*, with security here typically in the computational sense and adversaries assumed to have fixed resources. Arguments often presuppose secure building blocks. Protocols are typically designed to counter standard attacks, and shown to follow accepted principles. Practical arguments (paralleling complexity-theoretic arguments) involving constructions which assemble basic building blocks may justify security claims.
   While perhaps the most commonly used and practical approach, it is in some ways the least satisfying. This approach may uncover protocol flaws thereby establishing that a protocol is bad. However, claims of security may remain questionable, as subtle flaws in cryptographic protocols typically escape ad hoc analysis; unforeseen attacks remain a threat.
2. *reducibility from hard problems*. This technique consists of proving that any successful protocol attack leads directly to the ability to solve a well-studied reference problem (Chapter 3), itself considered computationally infeasible given current knowledge and an adversary with bounded resources. Such analysis yields so-called *provably secure protocols*, although the security is conditional on the reference problem being truly (rather than presumably) difficult.
   A challenge in this approach is to establish that all possible attacks have been taken into account, and can in fact be equated to solving the identified reference problems. This approach is considered by some to be as good a practical analysis technique as exists. Such provably secure protocols belong to the larger class of techniques which are *computationally secure*.
3. *complexity-theoretic analysis*. An appropriate model of computation is defined, and adversaries are modeled as having polynomial computational power (they may mount attacks involving time and space polynomial in the size of appropriate security parameters). A security proof relative to the model is then constructed. The existence of underlying cryptographic primitives with specified properties is typically assumed. An objective is to design cryptographic protocols which require the fewest cryptographic primitives, or the weakest assumptions.
   As the analysis is asymptotic, care is required to determine when proofs have practical significance. Polynomial attacks which are feasible under such a model may nonetheless in practice be computationally infeasible. Asymptotic analysis may be of limited relevance to concrete problems in practice, which have finite size. Despite these issues, complexity-theoretic analysis is invaluable for formulating fundamental principles and confirming intuition.
4. *information-theoretic analysis*. This approach uses mathematical proofs involving entropy relationships to prove protocols are *unconditionally secure*. In some cases,

this includes the case where partial secrets are disclosed (e.g., for unconditional security against coalitions of fixed size). Adversaries are modeled to have unbounded computing resources.

While unconditional security is ultimately desirable, this approach is not applicable to most practical schemes for several reasons. These include: many schemes, such as those based on public-key techniques, can at best be computationally secure; and information-theoretic schemes typically either involve keys of impractically large size, or can only be used once. This approach cannot be combined with computational complexity arguments because it allows unlimited computation.

5. *formal methods*. So-called *formal* analysis and verification methods include logics of authentication (cryptographic protocol logics), term re-writing systems, expert systems, and various other methods which combine algebraic and state-transition techniques. The most popular protocol logic is the Burrows-Abadi-Needham (BAN) logic. Logic-based methods attempt to reason that a protocol is correct by evolving a set of beliefs held by each party, to eventually derive a belief that the protocol goals have been obtained.

   This category of analysis is somewhat disjoint from the first four. Formal methods have proven to be of utility in finding flaws and redundancies in protocols, and some are automatable to varying degrees. On the other hand, the "proofs" provided are proofs within the specified formal system, and cannot be interpreted as absolute proofs of security. A one-sidedness remains: the absence of discovered flaws does not imply the absence of flaws. Some of these techniques are also unwieldy, or applicable only to a subset of protocols or classes of attack. Many require (manually) converting a concrete protocol into a formal specification, a critical process which itself may be subject to subtle flaws.

12.10 Notes and further references

§12.1

While the literature is rife with proposals for key establishment protocols, few comprehensive treatments exist and many proposed protocols are supported only by ad hoc analysis.

§12.2

Much of §12.2 builds on the survey of Rueppel and van Oorschot [1086]. Fumy and Munzert [431] discuss properties and principles for key establishment. While encompassing the majority of key establishment as currently used in practice, Definition 12.2 gives a somewhat restricted view which excludes a rich body of research. More generally, key establishment may be defined as a process or mechanism which provides a shared capability (rather than simply a shared secret) between specified sets of participants, facilitating some operation for which the intention is that other sets of participants cannot execute. This broader definition includes many protocols in the area of *threshold cryptography*, introduced independently by Desmedt [336], Boyd [182], and Croft and Harris [288]; see the comprehensive survey of Desmedt [337].

The term *perfect forward secrecy* (Definition 12.16) was coined by Günther [530]; see also Diffie, van Oorschot, and Wiener [348]. Here "perfect" does not imply any properties of information-theoretic security (cf. Definition 12.73). The concept of known-key attacks (Definition 12.17), developed by Yacobi and Shmuely [1256] (see also Yacobi [1255]), is related to that of Denning and Sacco [330] on the use of timestamps to prevent message replay (see page 535).

Among items not discussed in detail in this chapter is *quantum cryptography*, based on the uncertainty principle of quantum physics, and advanced by Bennett et al. [114] building on the idea of quantum coding first described by Wiesner [1242] *circa* 1970. Although not providing digital signatures or non-repudiation, quantum cryptography allows key distribution (between two parties who share no *a priori* secret keying material), which is provably secure against adversaries with unlimited computing power, provided the parties have access to (aside from the quantum channel) a conventional channel subject to only passive adversaries. For background on the basic quantum channel for key distribution (*quantum key distribution*), see Brassard [192]; Phoenix and Townsend [973] survey developments in this area including experimental implementations.

Mitchell [879] presented a key agreement system based on use of a public broadcast channel transmitting data at a rate so high that an eavesdropper cannot store all data sent over a specified time interval. This is closely related to work of Maurer [815] regarding secret key agreement using only publicly available information, in turn motivated by Wyner's *wire-tap channel* [1254], which addresses the rate at which secret information can be conveyed to a communicating partner with security against a passive eavesdropper whose channel is subject to additional noise.

§12.3

Regarding point-to-point techniques presented, those based on symmetric encryption are essentially from ISO/IEC 11770-2 [617], while AKEP1 and AKEP2 (Note 12.21; Protocol 12.20) are derived from Bellare and Rogaway [94] (see also §12.9 below). The idea of key derivation allowing key establishment by symmetric techniques based on a one-way function (without encryption), was noted briefly by Matsumoto, Takashima and Imai [800]; see also the proposals of Gong [499], and related techniques in the *KryptoKnight* suite [891, 141, 142].

Shamir's no-key protocol (Protocol 12.22; also called *Shamir's three-pass protocol*), including exponentiation-based implementation, is attributed to Shamir by Konheim [705, p.345]. Massey [786, p.35] notes that Omura [792], aware of Shamir's generic protocol, later independently proposed implementing it with an exponentiation-based cipher as per Protocol 12.22. See also Massey and Omura [956] (discussed in Chapter 15).

Version 5 of Kerberos (V5), the development of which began in 1989, was specified by Kohl and Neuman [1041]; for a high-level overview, see Neuman and Ts'o [926] who also note that a typical timestamp window is 5 minutes (centered around the verifier's time). The original design of Kerberos V4 was by Miller and Neuman, with contributions by Saltzer and Schiller [877]; an overview is given by Steiner, Neuman, and Schiller [1171], while V4 issues are noted by Kohl [701] and the critique of Bellovin and Merritt [103]. The basic protocol originates from the shared-key protocol of Needham and Schroeder [923], with timestamps (which Needham and Schroeder explicitly avoided) later proposed by Denning and Sacco [330], reducing the number of messages at the expense of secure and synchronized clocks. Bauer, Berson, and Feiertag [76] addressed symmetric assurances of freshness, recovery from single-key compromise, and reduction of messages through per-participant use of a local counter called an *event marker*; they also extended the Needham-Schroeder setting to multiple security domains (each with a separate KDC) and connectionless environments. Bellare and Rogaway [96] presented an efficient 4-pass server-based key transfer protocol with implicit key authentication, and key freshness properties secure against known-key attacks; significantly, their treatment (the first of its kind) shows the protocol to be provably secure (assuming a pseudorandom function). Advantages and disadvantages of using timestamps are discussed in §10.3.1.

Protocol 12.29 is due to Otway and Rees [961]. Kehne, Schönwälder, and Langendörfer [663] discuss a 5-message nonce-based protocol with the same features as Kerberos (Protocol 12.24), without requiring distributed timeclocks. Excluding the optional re-authentication capability (as per Kerberos), it is essentially that of Mechanism 9 in ISO/IEC DIS 11770-2 [617], and similar to the 5-message Otway-Rees protocol as augmented per Remark 12.30 (with one fewer encryption by each of A and B); but see also the analysis of Neuman and Stubblebine [925]. A 5-message authentication protocol included in ISO/IEC 9798-2 [599] provides key transport using a trusted server, with mutual entity authentication and mutual key confirmation, without timestamps; Needham and Schroeder [924] propose a 7-message protocol with similar properties.

§12.4

Mechanism 12.35 and Fact 12.34 are due to Blom [158]; a simpler polynomial formulation is noted under §12.8 below. For background in coding theory, see MacWilliams and Sloane [778]. Mitchell and Piper [881] consider the use of combinatorial block designs and finite incidence structures called *key distribution patterns* to construct a class of non-interactive KDS. Each user is given a set of secret subkeys (with no algebraic structure as per Blom's scheme), from which each pair of users may compute a common key by combining appropriate subkeys via a public function. The question of reducing key storage was considered earlier by Blom [157], including security against coalitions of fixed size and the use of commutative functions (later generalized to symmetric functions by Blundo et al. [169]; see also §12.8 below). For related work, see Quinn [1014], Gong and Wheeler [506], and §12.7 below.

§12.5

Protocol 12.38, the public-key protocol of Needham and Schroeder [923], was originally specified to include 4 additional messages whereby signed public keys were requested from an on-line certification authority. Asymmetric key transport protocols involving various combinations of encryption and signatures are given in ISO/IEC CD 11770-3 [618]. The three-pass encrypt-then-sign protocol of §12.5.2(iii) originates from ISO/IEC 9798-3 [600]; it is closely related to the STS protocol (Protocol 12.57) which transfers Diffie-Hellman exponentials in place of random numbers. I'Anson and Mitchell [567] critique (e.g., see Note 12.42) the X.509 protocols [595]; see also the formal analysis of Gaarder and Snekkenes [433]. Protocol 12.44 and the related 2-pass key agreement of Figure 12.2 are due to Beller and Yacobi [101, 100], building on work of Beller, Chang, and Yacobi [99, 98, 97].

A two-pass key transport protocol called COMSET, based on public-key encryption, was adopted by the European community RACE Integrity Primitives Evaluation (RIPE) project [178]. Arising from zero-knowledge considerations studied by Brandt et al. [188], it employs Williams' variant of the Rabin public-key encryption (§8.3), and is similar in some aspects to the Needham-Schroeder public-key and Beller-Yacobi protocols. The protocol specified in Note 12.39 combines concepts of COMSET and the Needham-Schroeder protocol.

§12.6

The landmark 1976 paper of Whitfield Diffie and Martin Hellman [345] is the standard reference for both the seminal idea of public-key cryptography and the fundamental technique of exponential key agreement. An earlier conference paper of Diffie and Hellman [344], written in December 1975 and presented in June 1976, conceived the concept of public key agreement and the use of public-key techniques for identification and digital signatures.

Diffie [342] reports that amidst joint work on the problem for some time, Hellman distilled exponential key agreement in May 1976, and this was added to their June 1976 conference presentation (but not the written paper). Preceding this, in the fall of 1974, Merkle independently conceived a particular method for key agreement using the same abstract concepts. Merkle's *puzzle system* [849], submitted for publication in 1975 and appearing in April 1978, is as follows. Alice constructs m *puzzles*, each of which is a cryptogram Bob can solve in n steps (exhaustively trying n keys until a recognizable plaintext is found). Alice sends all m puzzles to Bob over an unsecured channel. Bob picks one of these, solves it (cost: n steps), and treats the plaintext therein as the agreed key, which he then uses to encrypt and send to Alice a known message. The encrypted message, now a puzzle which Alice must solve, takes Alice n steps (by exhaustively trying n keys). For $m \approx n$, each of Alice and Bob require $O(n)$ steps for key agreement, while an opponent requires $O(n^2)$ steps to deduce the key. An appropriate value n is chosen such that n steps is computationally feasible, but n^2 is not.

Rueppel [1078] explores the use of function composition to generalize Diffie-Hellman key agreement. Shmuely [1127] and McCurley [825] consider *composite Diffie-Hellman*, i.e., Diffie-Hellman key agreement with a composite modulus. McCurley presents a variation thereof, with an RSA-like modulus m of specific form and particular base α of high order in $\mathbb{Z}_m^*$, which is provably as secure (under passive attack) as the more difficult of factoring m and solving the discrete logarithm problem modulo the factors of m.

Regarding Diffie-Hellman key agreement, van Oorschot and Wiener [1209] note that use of "short" private exponents in conjunction with a random prime modulus p (e.g., 256-bit exponents with 1024-bit p) makes computation of discrete logarithms easy. They also document the attack of Note 12.50, which is related to issues explored by Simmons [1150] concerning a party's ability to control the resulting Diffie-Hellman key, and more general issues of unfairness in protocols. Waldvogel and Massey [1228] carefully examine the probability distribution and entropy of Diffie-Hellman keys under various assumptions. When private exponents are chosen independently and uniformly at random from the invertible elements of $\mathbb{Z}_{p-1}$, the $\phi(p-1)$ keys which may result are equiprobable. When private exponents are chosen independently and uniformly at random from $\{0, \ldots, p-2\}$ (as is customary in practice), in the best case (when p is a *safe prime*, $p = 2q + 1$, q prime) the most probable Diffie-Hellman key is only 6 times more likely than the least probable, and the key entropy is less than 2 bits shy of the maximum, $\lg(p-1)$; while in the worst case (governed by a particular factorization pattern of $p-1$) the distribution is still sufficiently good to preclude significant cryptanalytic advantage, for p of industrial size or larger.

The one-pass key agreement of Protocol 12.51 was motivated by the work of ElGamal [368]. The MTI protocols of Table 12.5 were published in 1986 by Matsumoto, Takashima, and Imai [800]. MTI/A0 is closely related to a scheme later patented by Goss [519]; in the latter, exclusive-OR is used in place of modular multiplication to combine partial keys. Matsumoto et al. equate the computational complexity of passive attacks (excluding known-key attacks) on selected key agreement protocols to that of one or two Diffie-Hellman problems. Active attacks related to Note 12.54 are considered by Diffie, van Oorschot, and Wiener [348], and Menezes, Qu, and Vanstone [844]. Yacobi and Shmuely [1256] note two time-variant versions of Diffie-Hellman key agreement which are insecure against known-key attack. A similar protocol which falls to known-key attack was discussed by Yacobi [1255], subsequently rediscovered by Alexandris et al. [21], and re-examined by Nyberg and Rueppel [937]. Yacobi [1255] proves that the MTI/A0 protocol with composite-modulus is provably secure (security equivalent to composite Diffie-Hellman) under known-key attack by a passive adversary; Desmedt and Burmester [339], however, note the security is only heuristic under known-key attack by an active adversary. A formal-logic security comparison of the protocols of Goss (essentially Protocol 12.53), Günther (Protocol 12.62), and STS (Protocol 12.57) is given by van Oorschot [1204]. Burmester [220] identifies known-key *triangle attacks* which may be mounted on the former two and related protocols which provide only implicit key authentication (including MTI protocols, cf. Note 12.54). Known-key attacks were also one motivation for Denning and Sacco [330] to modify the Needham-Schroeder protocol as discussed above (cf. p.534).

Protocol 12.57 (STS) evolved from earlier work on ISDN telephone security as outlined by Diffie [342, p.568], who also reports on STU-III telephones. Variations of STS and an informal model for authentication and authenticated key establishment are discussed by Diffie, van Oorschot, and Wiener [348]. Bellovin and Merritt [104, 105] (see also the patent [102]) propose another hybrid protocol (*Encrypted Key Exchange* – EKE), involving exponential key agreement with authentication based on a shared password, designed specifically to protect against password-guessing attacks by precluding easy verification of guessed passwords; Steiner, Tsudik, and Waidner [1172] provide further analysis and extensions. A hybrid protocol with similar goals is given Gong et al. [504], including discussion of its relationship to EKE, and expanding the earlier work of Lomas et al. [771].

Blom [157] was apparently the first to propose an identity-based (or more accurately, index-based) key establishment protocol. Shamir [1115] proposed the more general idea of *identity-based systems* wherein a user's public key may be a commonly known name and address. For further discussion of ID-based schemes, see the chapter notes on §13.4. Self-certified public keys (Mechanism 12.61) are discussed by Girault [459], who credits earlier work by others, and provides the self-certified version of Günther's ID-based keys (Example 12.67). The parenthetical forgery attack mentioned in Mechanism 12.61 is outlined by Stinson [1178]. Key agreement protocols as in Examples 12.64 and 12.65, using both ID-based public keys of Günther [530] (Mechanism 12.59) and modified ElGamal signatures, are given by Horster and Knobloch [562]. The optimization of ElGamal signatures noted in Remark 12.60 is by Agnew, Mullin, and Vanstone [19]. Rabin's signature scheme (Chapter 11) may be used in place of RSA to reduce the computations required in schemes based on Girault's implicitly-certified public keys. Maurer and Yacobi [824] (modifying their earlier proposal [823]) propose an identity-based one-pass key pre-distribution scheme using composite modulus Diffie-Hellman, featuring implicitly-certified public key-agreement keys essentially consisting of a user's identity (or email address); the corresponding private key is the discrete logarithm of this, computed by a trusted authority which, knowing the factorization of an appropriately chosen modulus n, can thereby compute logarithms.

Nyberg and Rueppel [936] note their signature scheme (Chapter 11) may be used to create implicitly certified, identity-based public keys with properties similar to those of Girault (Mechanism 12.61), as well as key agreement protocols; Nyberg [935] presents an improved one-pass key agreement based on these ideas. Okamoto and Tanaka [946] propose identity-based key agreement protocols combining exponential key agreement and RSA, including one using timestamps and providing entity authentication, and a simpler protocol providing (implicit) key authentication.

§12.7

The idea of split control has long been known (e.g., see Sykes [1180]). Shamir [1110] and Blakley [148] independently proposed the idea of threshold schemes, the latter based on vector subspaces. The simplest example of the Blakley's idea is a $(2, n)$ threshold scheme where the shares (here called *shadows*) distributed to parties are non-collinear lines in a common plane; the shared secret of any two parties is the intersection of their lines. For a $(3, n)$ scheme, the shadows consist of non-parallel planes, any two of which intersect in a line, and any three of which intersect in a point. While Shamir's threshold scheme is *perfect*, Blakley's vector scheme is not (the set of possible values of the shared secret narrows as subsequent shares are added). Karnin, Greene, and Hellman [662] discuss the unanimous consent control scheme of §12.7.1; see also Diffie and Hellman [344, p.110].

Generalized secret sharing schemes and the idea of access structures were first studied by Ito, Saito, and Nishizeki [625], who provided a construction illustrating that any monotone access structure can be realized by a perfect secret sharing scheme. Benaloh and Leichter [112] provided more elegant constructions. A comprehensive discussion of secret sharing including adaptations providing shared control capabilities of arbitrary complexity, and many of the extended capabilities including pre-positioned schemes, is given by Simmons [1145, 1141, 1142], mainly with geometric illustration. An exposition by Stinson [1177] addresses information rate in particular. Ingemarsson and Simmons [570] consider secret sharing schemes which do not require a trusted party.

Laih et al. [732] consider dynamic secret sharing schemes. Blundo et al. [168] consider pre-positioned schemes, dynamic secret sharing, and bounds on share sizes and broadcast messages therein; Jackson, Martin, and O'Keefe [629] examine related multi-secret threshold schemes. Blakley et al. [147] consider threshold schemes with disenrollment.

Tompa and Woll [1195] note that an untrustworthy participant U may cheat in Shamir's threshold scheme by submitting a share different than its own, but carefully computed such that pooling of shares provides other participants with no information about the secret S, while allowing U to recover S. They propose modifications which (with high probability) allow detection of cheating, and which prevent a cheater U from actually obtaining the secret.

The related problem of *verifiable secret sharing*, which is of broader interest in secure distributed computation, was introduced by Chor et al. [259]; see also Benaloh [110] and Feldman [390], as well as Rabin and Ben-Or [1028]. Here the trusted party distributing shares might also cheat, and the goal is to verify that all distributed shares are consistent in the sense that appropriate subsets of shares define the same secret. For applications of verifiable secret sharing to key escrow, see Micali [863].

Fact 12.75 is based on the definition of perfect secret sharing and information-theoretic security, as is the majority of research in secret sharing. *Ramp schemes* with shares shorter than the secret were examined by Blakley and Meadows [151]; while trading off perfect security for shorter shares, their examination is nonetheless information-theoretic. In practice, a more appropriate goal may be computationally secure secret sharing; here the objective is that if one or more shares is missing, an opponent has insufficient information to (computationally) recover the shared secret. This idea was elegantly addressed by Krawczyk [715] as follows. To share a large s-bit secret $S = P$ (e.g., a plaintext file) among n users, first encrypt it under a k-bit symmetric key K as $C = E_K(P)$; using a perfect secret sharing scheme such as Shamir's (t, n) scheme, split K into n k-bit shares $K_1, \dots, K_n$; then using Rabin's *information dispersal algorithm* (IDA) [1027] split C into n pieces $C_1, \dots, C_n$ each of (s/t) bits; finally, distribute to user U_i the secret share $S_i = (K_i, C_i)$. Any t participants who pool their shares can then recover K by secret sharing, C by IDA, and $P = S$ by decrypting C using K. By the remarkable property of IDA, the sum of the sizes of the t pieces C_i used is exactly the size of the recovered secret S itself (which cannot be bettered); globally, the only space overhead is that for the short keys K_i, whose size k is independent of the large secret S.

The clever idea of *visual cryptography* to facilitate sharing (or encryption) of pictures is due to Naor and Shamir [919]. The pixels of a (secret) picture are treated as individual secrets to be shared. The picture is split into two or more images each of which contains one share for each original pixel. Each original pixel is split into shares by subdivision into subpixels of appropriate size, with selection of appropriate combinations of subpixel shadings (black and white) such that stacking the images on transparencies reveals the original, while each individual image appears random. Picture recovery requires no computation (it is visual); anyone with all but one of the images still has (provably) no information.

§12.8

An early investigation of conference keying schemes based on Diffie-Hellman key agreement was undertaken by Ingemarsson, Tang and Wong [571]. The protocol of Burmester and Desmedt [222] (Protocol 12.78) is the most efficient of those which have been proposed and are provably secure; their work includes a review of alternate proposals and a thorough bibliography. Research in this area with particular emphasis on digital telephony includes that of Brickell, Lee, and Yacobi [205]; Steer et al. [1169]; and Heiman [547].

Matsumoto and Imai [799] systematically define (symmetric-key) key pre-distribution schemes, based on symmetric functions, for conferences of two or more parties. Their proposals are non-interactive and ID-based, following the original idea of two-party non-interactive ID-based schemes by Blom [157, 158], including consideration of information-theoretic security against coalitions of fixed size. Tsujii and Chao [1197], among many others, propose schemes in a similar setting. Blundo et al. [169] both specialize the work of Matsumoto and Imai, and generalize Blom's symmetric key distribution (Mechanism 12.35) and bounds from two-party key pre-distribution to non-interactive j-secure conference keying schemes of fixed size; prove Fact 12.79; and provide a scheme meeting this bound. Their generalization uses symmetric polynomials in t variables for privileged subsets of size t, yielding in the two-party case ($t = 2$) an equivalent but simpler formulation of Blom's scheme: the trusted party selects an appropriate secret symmetric polynomial $f(x, y)$ and gives party i the secret univariate polynomial $f(i, y)$, allowing parties i and j to share the pairwise key $f(i, j) = f(j, i)$. They also consider an interactive model. Further examination of interactive vs. non-interactive conferencing is undertaken by Beimel and Chor [83]. Fiat and Naor [394] consider j-secure broadcast encryption schemes, and practical schemes requiring less storage; for the former, Blundo and Cresti [167] establish lower bounds on the number of keys held and the size of user secrets.

Berkovits [116] gives constructions for creating *secret broadcasting schemes* (conference keying schemes where all messages are broadcast) from (t, n) threshold schemes. Essentially, for conferences with t members, a new $(t + 1, 2t + 1)$ threshold scheme with secret K is created from the old, and t new shares are publicly broadcast such that each of the t pre-assigned secret shares of the intended conference members serves as share $t+1$, allowing recovery of the conference key K in the new scheme. For related work involving use of polynomial interpolation, key distribution involving a trusted party, and broadcasting keys, see Gong [502] and Just et al. [647].

§12.9

The intruder-in-the-middle attack (Attack 1) is discussed by Rivest and Shamir [1057], who propose an "interlock protocol" to allow its detection; but see also Bellovin and Merritt [106]. The reflection attack (Attack 2) is discussed by Mitchell [880]. Attack 4 on the Otway-Rees protocol is discussed by Boyd and Mao [183] and van Oorschot [1205]. The interleaving attack (Attack 3) is due to Wiener circa June 1991 (document ISO/IEC JTC1/SC27 N313, 2 October 1991), and discussed by Diffie, van Oorschot, and Wiener [348] along with attacks on sundry variations of Diffie-Hellman key agreement. Bird et al. [140] systematically examine interleaving attacks on symmetric-key protocols, consider exhaustive analysis to detect such attacks, and propose a protocol resistant thereto (namely 2PP, included in the IBM prototype *KryptoKnight* [891]; see also [141, 142]).

Bellare and Rogaway [94], building on the work of earlier informal models, present a complexity-theoretic communications model and formal definitions for secure symmetric-key two-party mutual authentication and authenticated key establishment, taking known-key attacks into account. They prove AKEP1 (Note 12.21) and AKEP2 (Protocol 12.20) secure relative to this model, for parameters of appropriate size and assuming h and h' are pseudorandom functions or pseudorandom permutations; they also suggest practical constructions for pseudorandom functions based on DES and MD5. Gong [503] examines the efficiency of various authentication protocols and proposes lower bounds (e.g., on the number of message-passes required).

The examples illustrating attacks on flawed protocols are only a few of countless documented in the literature. Moore [898] provides an excellent survey on protocol failure; see also Anderson and Needham [31] and Abadi and Needham [1] for sound engineering principles. A large number of authenticated key establishment protocols with weaknesses are analyzed using the BAN logic in the highly recommended report of Burrows, Abadi, and Needham [227] (and by the same title: [224, 226, 225]). Gligor et al. [463] discuss the limitations of authentication logics. Syverson [1181] examines the goals of formal logics for protocol analysis and the utility of formal semantics as a reasoning tool. Among the authentication logics evolving from BAN are those of Abadi and Tuttle [2], Gong, Needham, and Yahalom [505], and Syverson and van Oorschot [1183]. The work of Abadi and Tuttle is notable for its model of computation and formal semantics relative to this model. Lampson et al. [740] both provide a theory of authentication in distributed systems (including delegation and revocation) and discuss a practical system based on this theory.

One of the first contributions to formal protocol analysis was that of Dolev and Yao [359], whose formal model, which focuses on two-party protocols for transmitting secret plaintexts, facilitates precise discussion of security issues. This approach was augmented with respect to message authentication and information leakage by Book and Otto [170]. Three general approaches to protocol analysis are discussed by Kemmerer, Meadows, and Millen [664] (see also Simmons [1148]): an algebraic approach, a state transition approach, and a logical approach (which can be given a state-transition semantics). They illustrate several methods on a protocol with known flaws (the infamous TMN protocol of Tatebayashi, Matsuzaki, and Newman [1188]). Other recent surveys on formal methods include that of Meadows [831], and the comprehensive survey of Rubin and Honeyman [1073]. An extensive bibliographic tour of authentication literature is provided by Liebl [765].