UNIT 2
Key Management and Distribution

UNIT 2 - Key Management and Distribution

- Symmetric Key Distribution Using Symmetric Encryption
- Symmetric Key Distribution Using Asymmetric Encryption
- Distribution of Public Keys
- X.509 Certificates
- Public Key Infrastructure

Key Management and Distribution

Key Management and Distribution

Key distribution is the function that delivers a key to two parties who wish to exchange secure encrypted data. Some sort of mechanism or protocol is needed to provide for the secure distribution of keys.

Public-key encryption schemes are secure only if the authenticity of the public key is assured. A public-key certificate scheme provides the necessary security.

X.509 defines the format for public-key certificates. This format is widely used in a variety of applications.

A public-key infrastructure (PKI) is defined as the set of hardware, software, people, policies, and procedures needed to create, manage, store, distribute, and revoke digital certificates based on asymmetric cryptography.

Typically, PKI implementations make use of X.509 certificates.

How to share the Key ?

4 Ways to share keys

1. A can select a key and physically deliver it to B.
2. A third party can select the key and physically deliver it to A and B.
3. If A and B have previously and recently used a key, one party can transmit the new key to the other, encrypted using the old key.
4. If A and B each has an encrypted connection to a third party C, C can deliver a key on the encrypted links to A and B.

Factors affecting End-to-end encryption

1. No. of Hosts at IP level

If N no of hosts then the no of keys required is

[N(N-1)]/2

2. No of Hosts and processes at Application Level

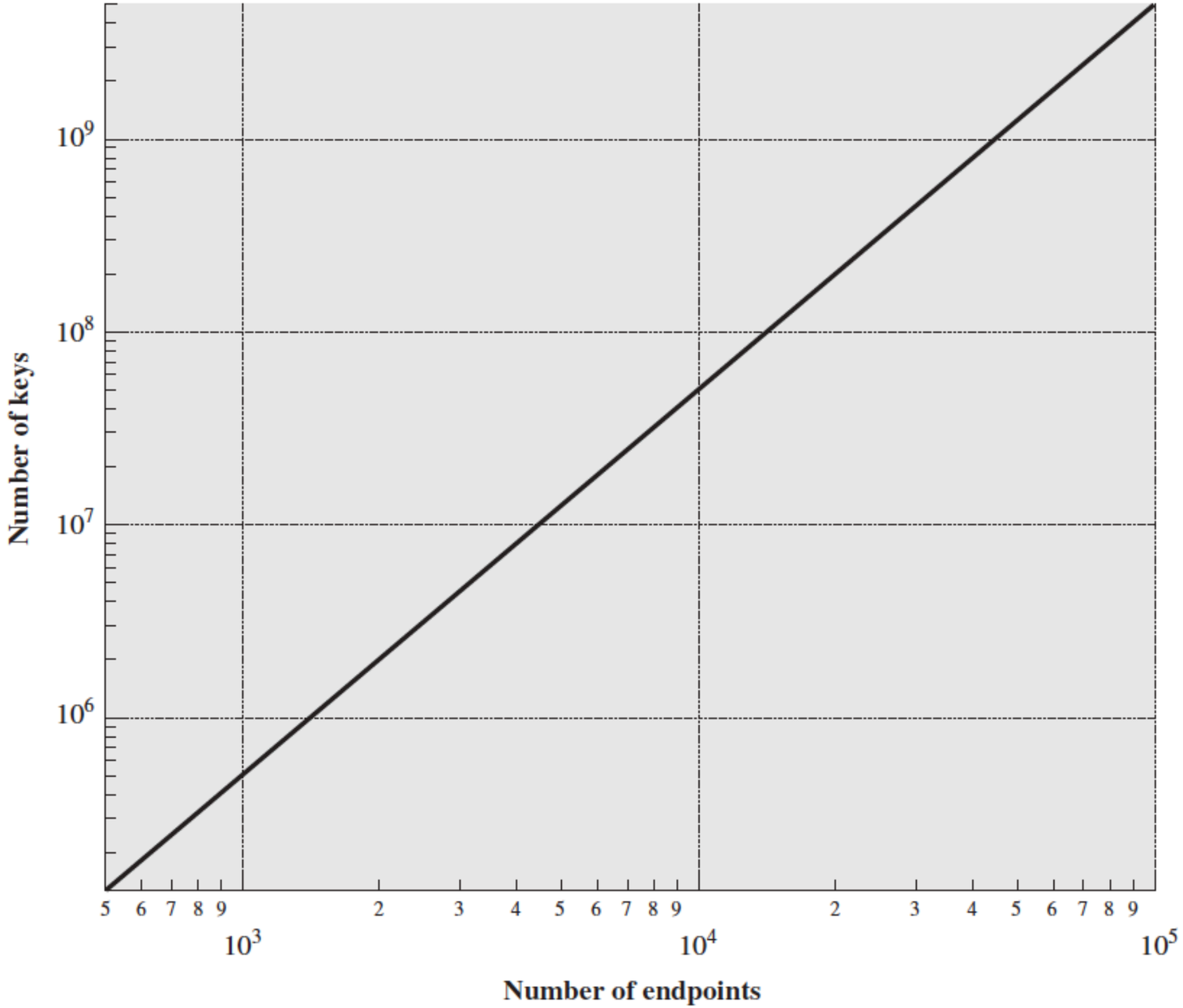


Figure 14.1 Number of Keys Required to Support Arbitrary Connections between Endpoints

Key Distribution Scenario

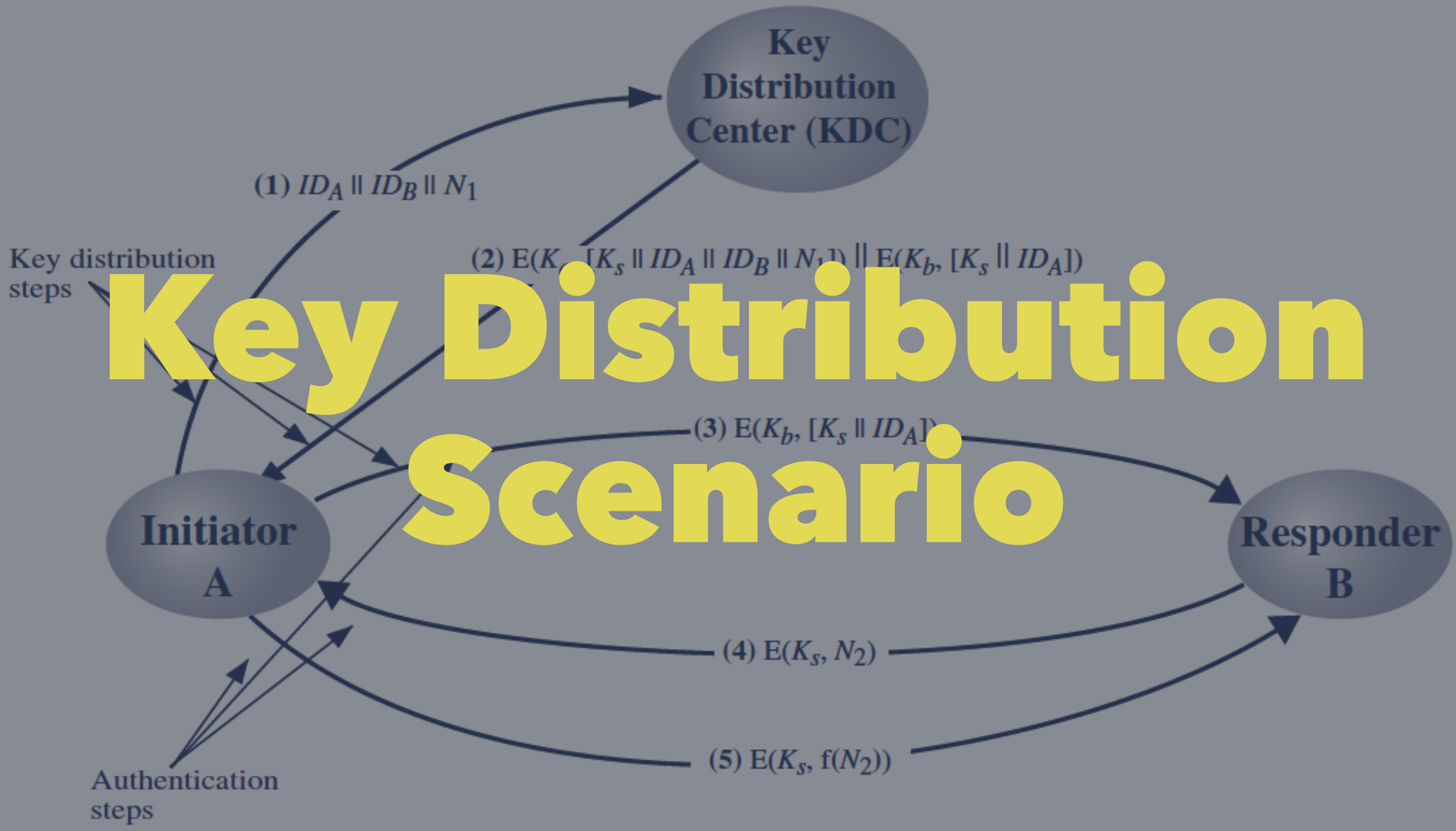


Figure 14.3 Key Distribution Scenario

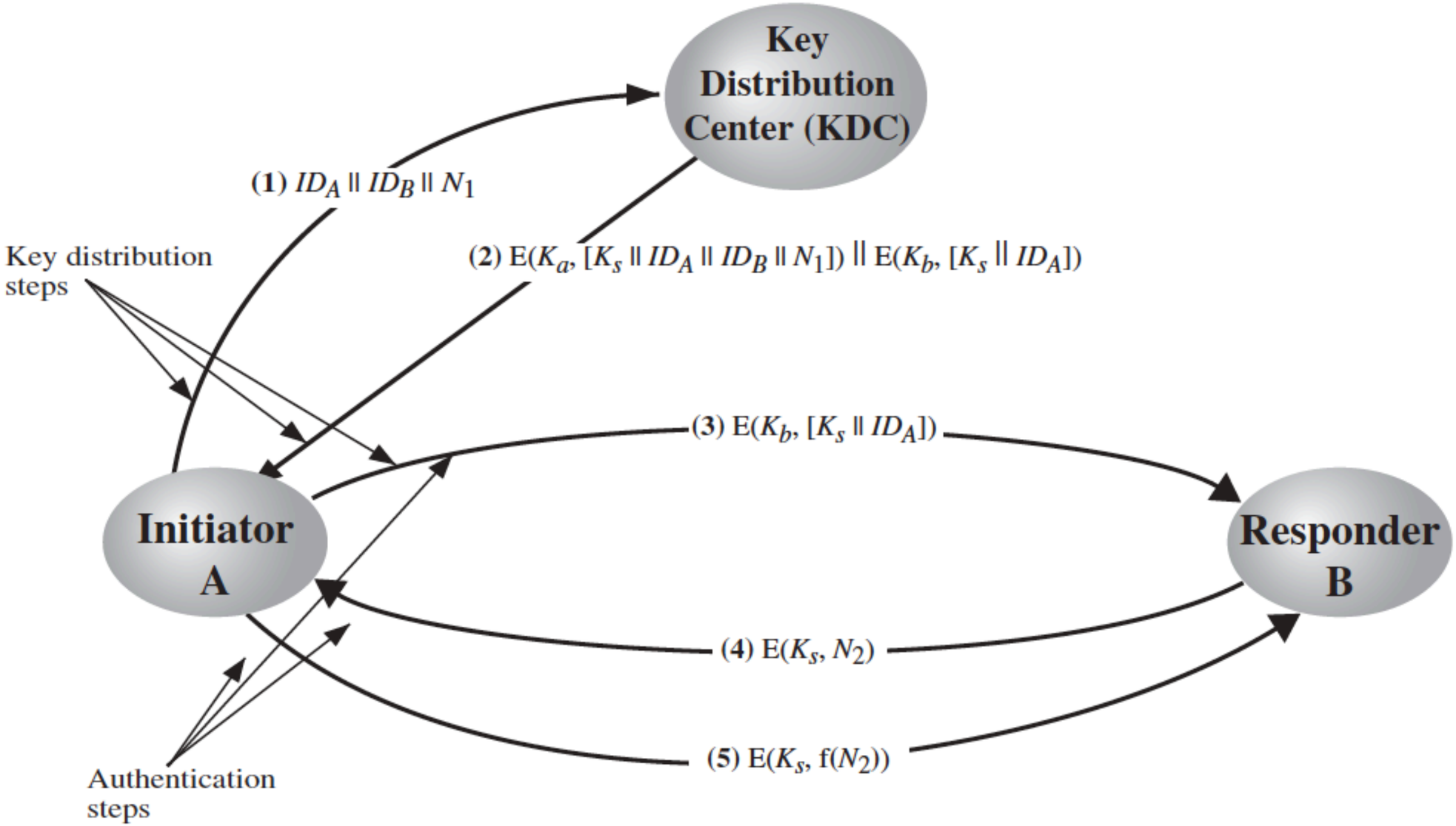


Figure 14.3 Key Distribution Scenario

Assumptions

- Let us assume that user A wishes to establish a logical connection with B and requires a one-time session key to protect the data transmitted over the connection.
- A has a master key, Ka , known only to itself and the KDC; similarly, B shares the master key Kb with the KDC.

Steps Overview

Step 1

A sends areques to IDa || IDb || N1 to KDC where
IDa -> Identitiy of A
IDb -> Identitiy of B
N1 -> Unique Identifier (Nonce)

Step 2

KDC Responds back with a message encrypted with Ka

- Message Intended for A

Ks -> One time session key

Original Message Sent by A

- Message Intended for B

Ks -> One time session key

IDa -> An identifier of A

- Message for B is encrypted with Kb

Step 3

A sends the message intended for B sent by KDC

E(Kb, [Ks || IDa])

Kb -> Master key of B

Ks -> Session Key

IDa -> Unique Identifier of A

Step 4

Using the newly minted session key for encryption, B sends a nonce, N2, to A.

Step 5

Using *Ks*,A responds with *f(N2)*, where f is a function that performs some transformation on *N2*.

Steps Detail

1. A issues a request to the KDC for a session key to protect a logical connection to B.The message includes the identity of A and B and a unique identifier, *N1*, for this transaction, which we refer to as a nonce. The nonce may be a timestamp, a counter, or a random number; the minimum requirement is that it differs with each request. Also, to prevent masquerade, it should be difficult for an opponent to guess the nonce.Thus, a random number is a good choice for a nonce.
2. The KDC responds with a message encrypted using *Ka*. Thus,A is the only one who can successfully read the message, and A knows that it originated at the KDC.The message includes two items intended for A:

• The one-time session key *Ks*, , to be used for the session

• The original request message, including the nonce, to enable A to match this response with the appropriate request

Thus,A can verify that its original request was not altered before reception by the KDC and, because of the nonce, that this is not a replay of some previous request.

In addition, the message includes two items intended for B:

• The one-time session key, *Ks*, to be used for the session

• An identifier of A (e.g., its network address), *IDa*

These last two items are encrypted with *Kb* (the master key that the KDC shares with B).They are to be sent to B to establish the connection and prove A's identity.

3. A stores the session key for use in the upcoming session and forwards to B the information that originated at the KDC for B, namely, *E(Kb, [Ks || IDa])* Because this information is encrypted with *Kb*, it is protected from eavesdropping.

B now knows the session key *Ks*, knows that the other party is A (from *IDa*), and knows that the information originated at the KDC (because it is encrypted using *Kb*).

At this point, a session key has been securely delivered to A and B, and

they may begin their protected exchange. However, two additional steps are

desirable:

4. Using the newly minted session key for encryption, B sends a nonce, *N2*, to A.
5. Also, using *Ks*,A responds with *f(N2)*, where f is a function that performs some transformation on *N2*(e.g., adding one).

Hierarchical Key Control

- Implementation of Multiple KDCs
- Local KDC -> Responsible for small domain
- Global KDC -> Responsible for connecting Local KDCs

Session Key Lifetime

- For connection-oriented protocols
- For a connectionless protocols

For connection-oriented protocols

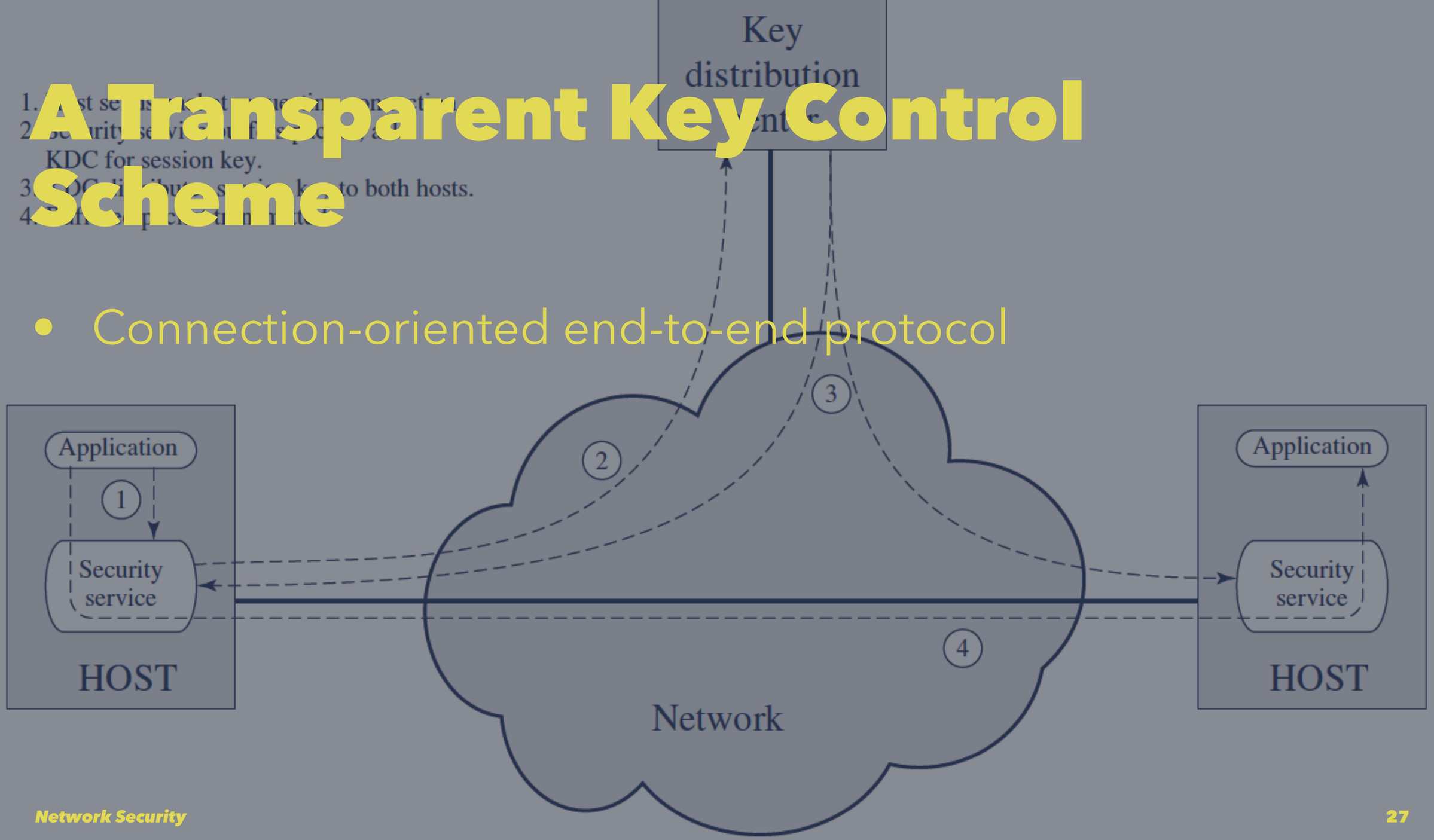
one obvious choice is to use the same session key for the length of time that the connection is open, using a new session key for each new session. If a logical connection has a very long lifetime, then it would be prudent to change the session key periodically, perhaps every time the PDU (protocol data unit) sequence number cycles.

For a connectionless protocols

such as a transaction-oriented protocol, there is no explicit connection initiation or termination. Thus, it is not obvious how often one needs to change the session key.The most secure approach is to use a new session key for each exchange. However, this negates one of the principal benefits of connectionless protocols, which is minimum overhead and delay for each transaction. A better strategy is to use a given session key for a certain fixed period only or for a certain number of transactions.

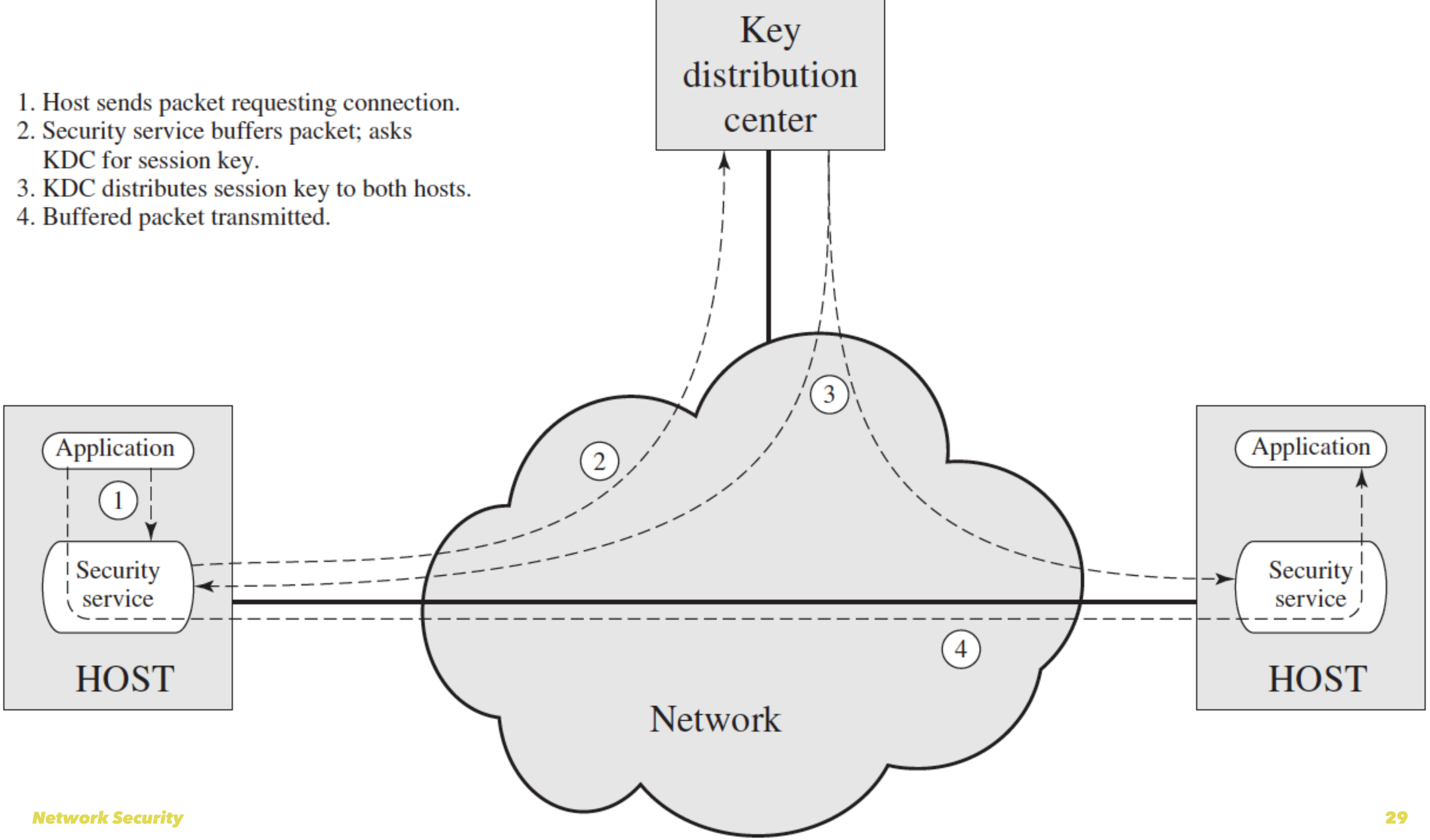
A Transparent Key Control Scheme

- Connection-oriented end-to-end protocol



1. Host sends packet requesting connection.
2. Security service buffers packet; asks KDC for session key.
3. KDC distributes session key to both hosts.
4. Buffered packet transmitted.

1. Host sends packet requesting connection.
2. Security service buffers packet; asks KDC for session key.
3. KDC distributes session key to both hosts.
4. Buffered packet transmitted.



Decentralized Key Control

- Discourages the use of KDC
- Useful within a local context

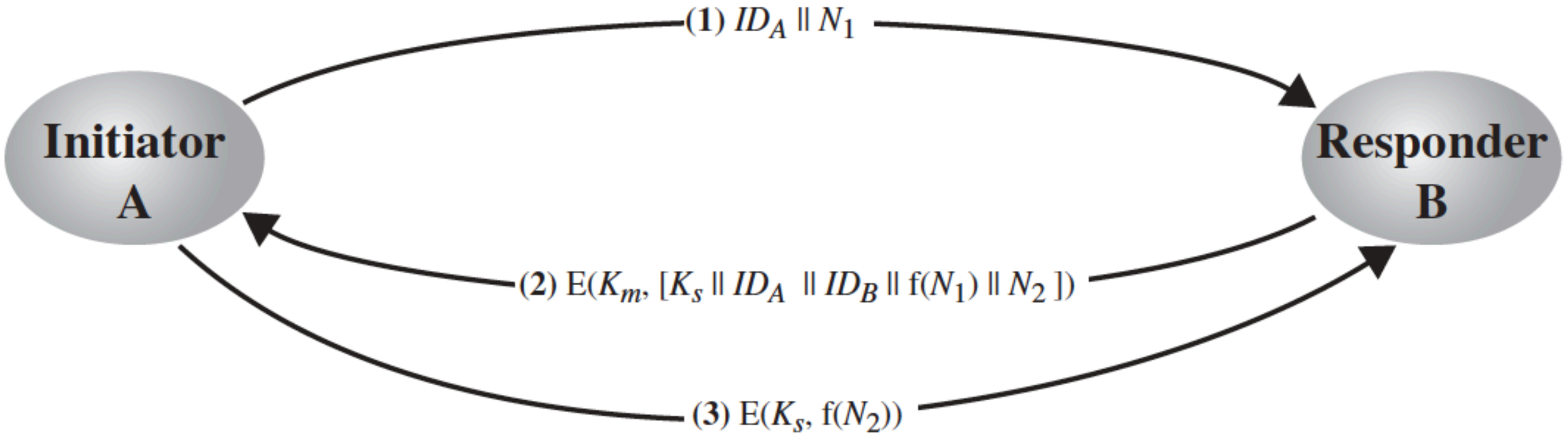
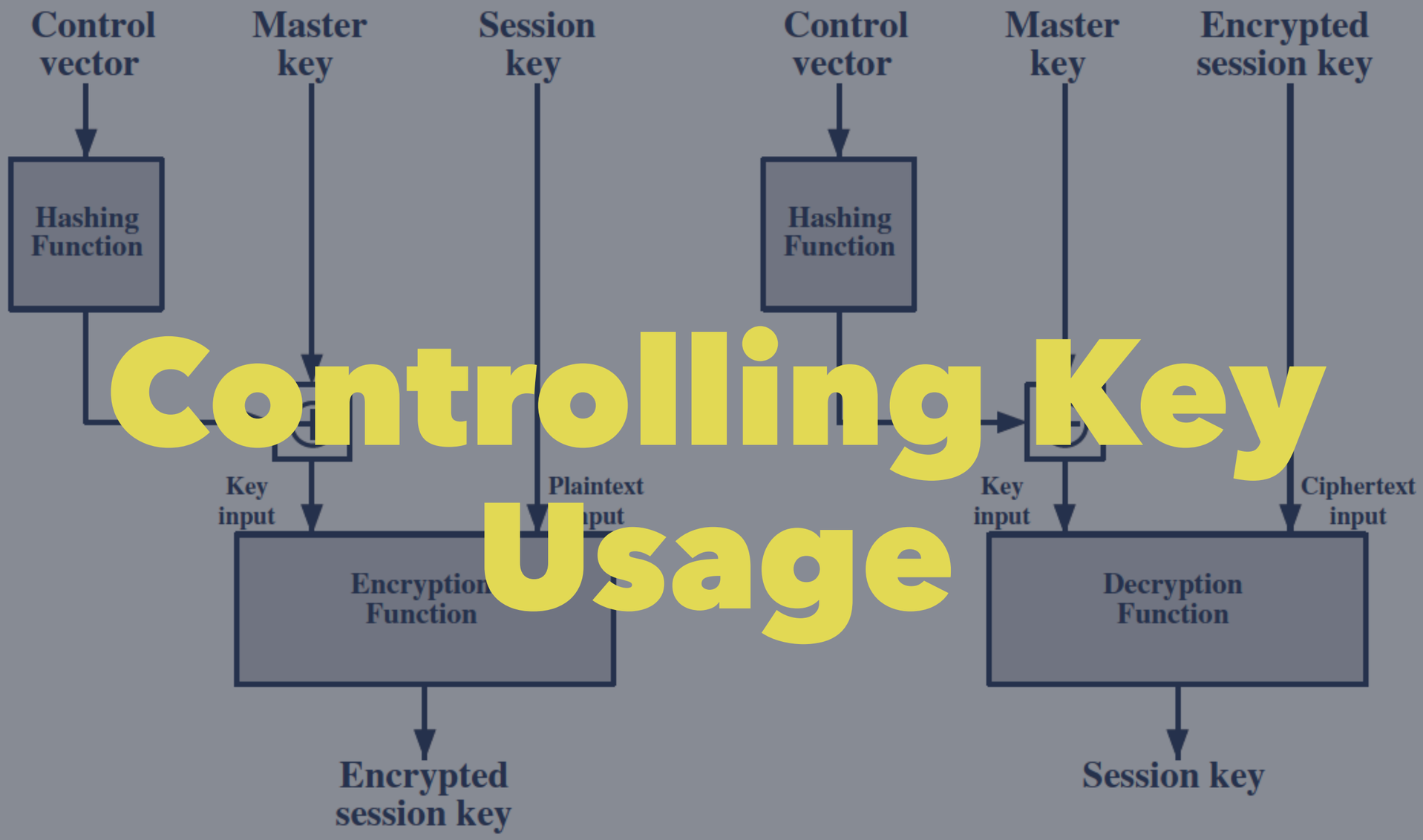


Figure 14.5 Decentralized Key Distribution

1. A issues a request to B for a session key and includes a nonce, N_1.
2. B responds with a message that is encrypted using the shared master key. The response includes the session key selected by B, an identifier of B, the value $f(N_1)$, and another nonce, N_2.
3. Using the new session key, A returns $f(N_2)$ to B.

Controlling Key Usage



(a) Control vector encryption

(b) Control vector decryption

Figure 14.6 Control Vector Encryption and Decryption

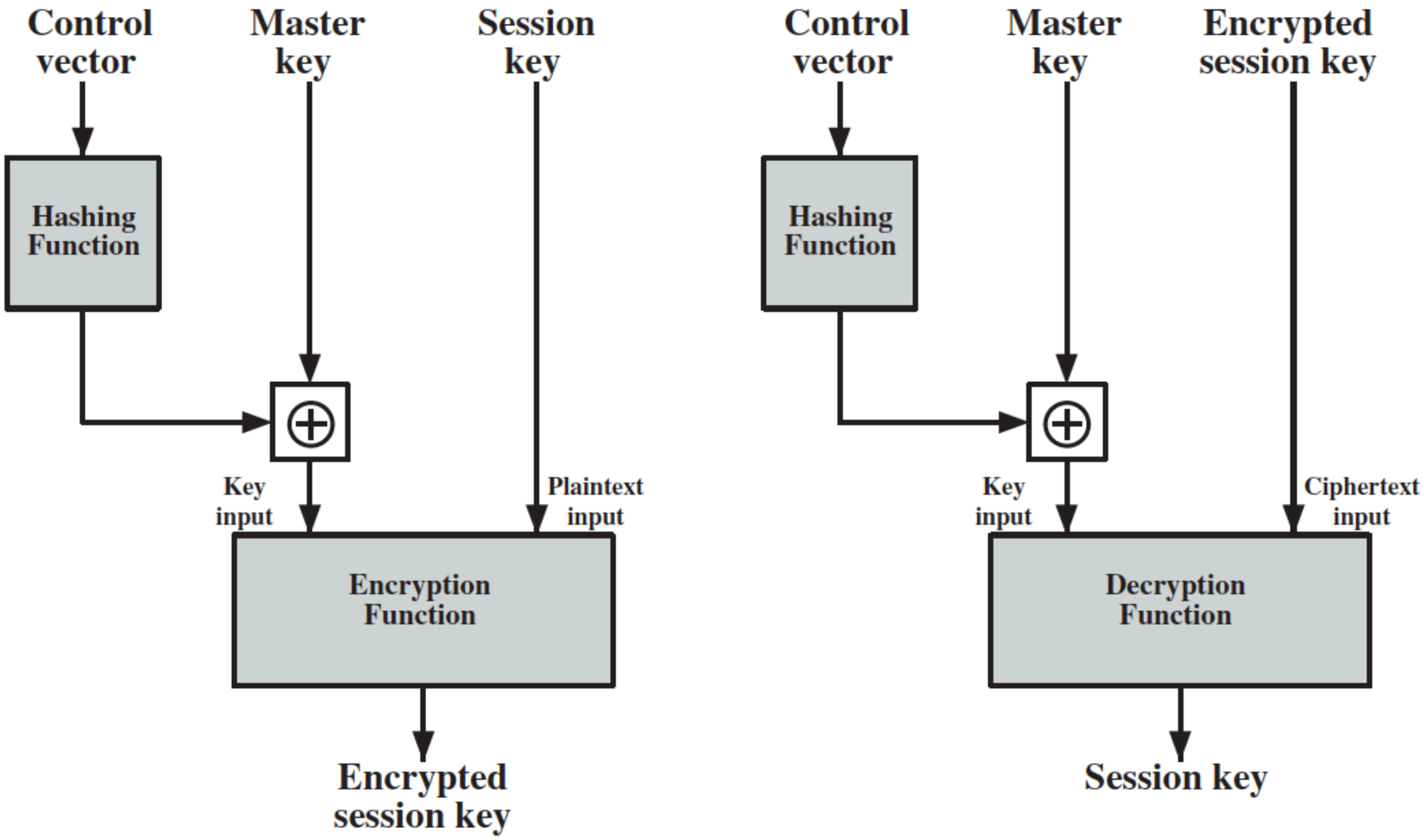


Figure 14.6 Control Vector Encryption and Decryption

Symmetric Key Distribution using asymmetric encryptions

- Simple Secret Key Distribution
- Secret Key Distribution with Confidentiality and Authentication
- A Hybrid Scheme

Simple Secret Key Distribution

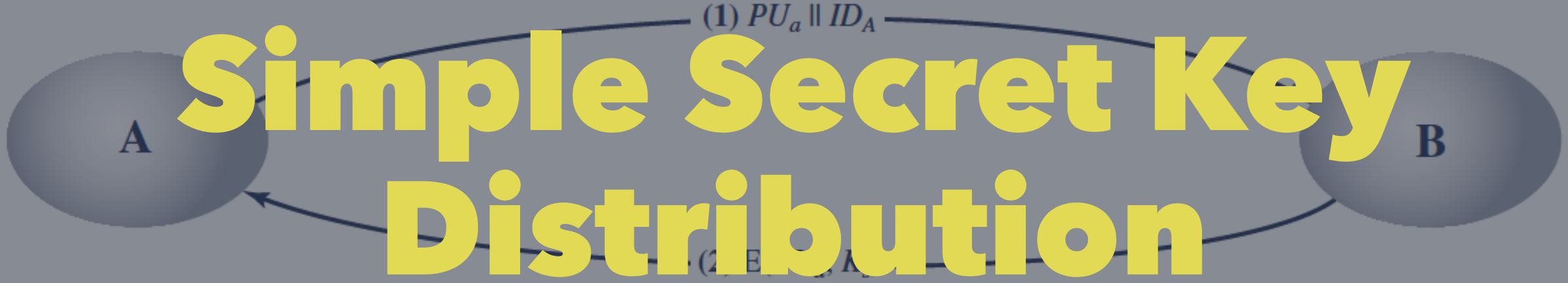


Figure 14.7 Simple Use of Public-Key Encryption to Establish a Session Key

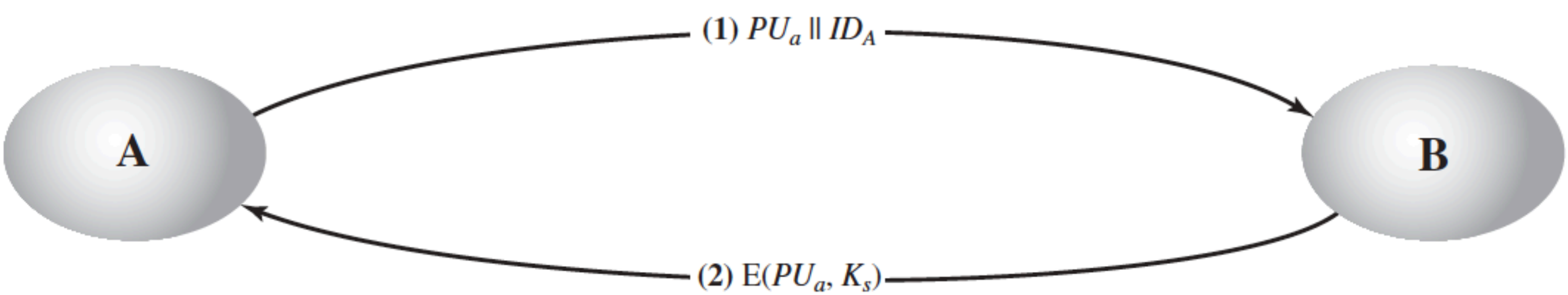


Figure 14.7 Simple Use of Public-Key Encryption to Establish a Session Key

1. A generates a public/private key pair $\{PU_a, PR_a\}$ and transmits a message to B consisting of PU_a and an identifier of A, ID_A.
2. B generates a secret key, K_s, and transmits it to A, which is encrypted with A's public key.
3. A computes $D(PR_a, E(PU_a, K_s))$ to recover the secret key. Because only A can decrypt the message, only A and B will know the identity of K_s.
4. A discards PU_a and PR_a and B discards PU_a.

A and B can now securely communicate using conventional encryption and the session key K_s. At the completion of the exchange, both A and B discard K_s.

1. A generates a public/private key pair $\{PU_a, PR_a\}$ and transmits a message intended for B consisting of PU_a and an identifier of A, ID_A.
2. E intercepts the [illegible] pair $\{PU_e, PR_e\}$ and transmits $PU_e||I$[illegible]
3. B generates a secret key, K_s, and transmits $E(PU_e, K_s)$.
4. E intercepts the message and learns K_s by computing $D(PR_e, E(PU_e, K_s))$.
5. E transmits $E(PU_a, K_s)$ to A.

MITM Attack

1. A generates a public/private key pair $\{PU_a, PR_a\}$ and transmits a message intended for B consisting of PU_a and an identifier of A, ID_A.
2. E intercepts the message, creates its own public/private key pair $\{PU_e, PR_e\}$ and transmits $PU_e || ID_A$ to B.
3. B generates a secret key, K_s, and transmits $E(PU_e, K_s)$.
4. E intercepts the message and learns K_s by computing $D(PR_e, E(PU_e, K_s))$.
5. E transmits $E(PU_a, K_s)$ to A.

Secret Key Distribution with Confidentiality and Authentication

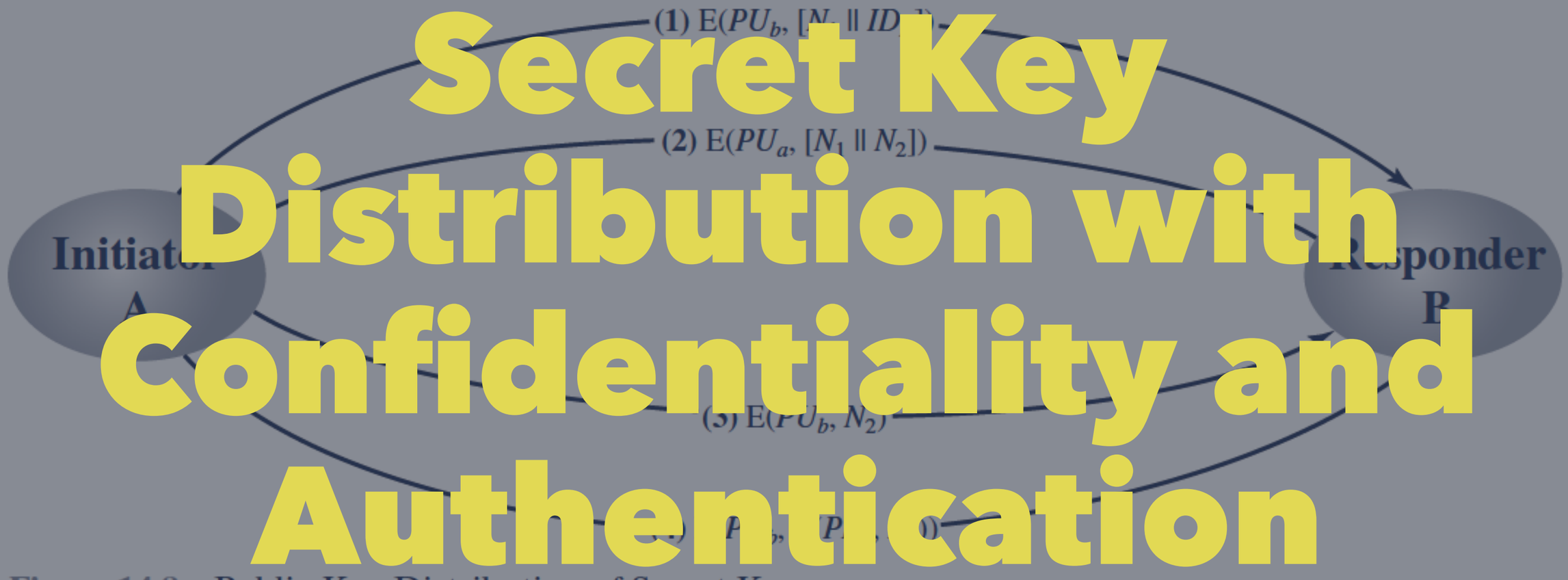


Figure 14.8 Public-Key Distribution of Secret Keys

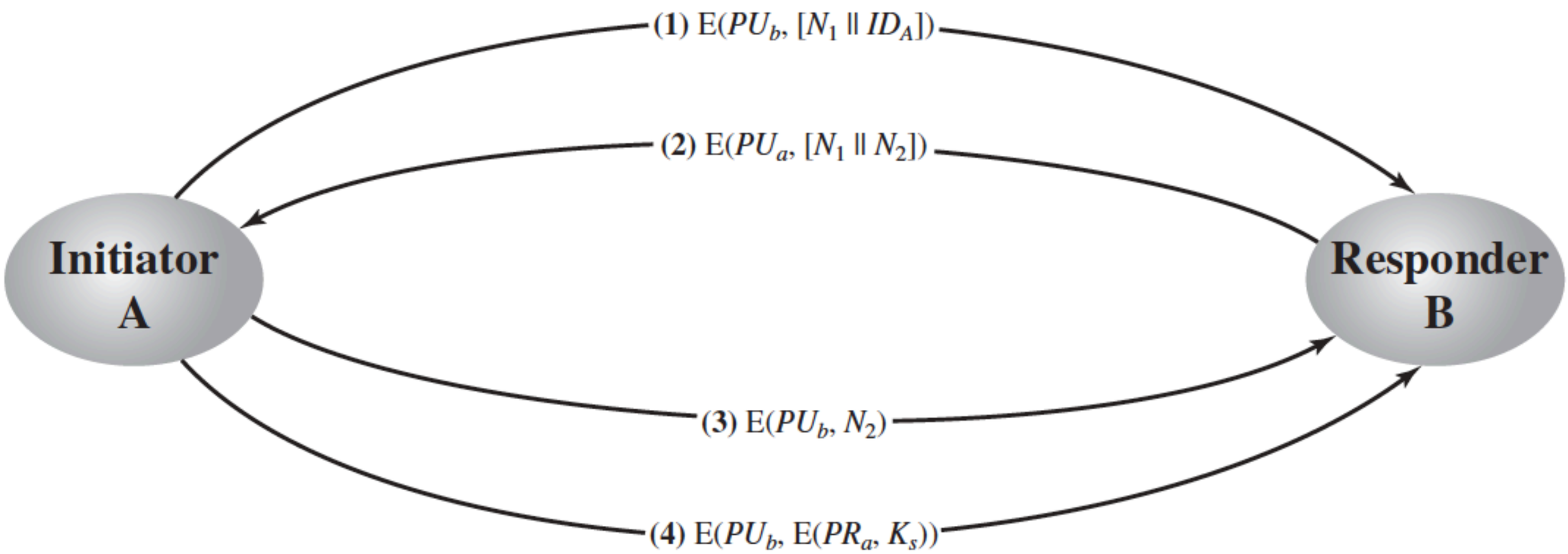


Figure 14.8 Public-Key Distribution of Secret Keys

1. A uses B's public key to encrypt a message to B containing an identifier of A(ID_A) and a nonce (N_1), which is used to identify this transaction uniquely.
2. B sends a message to A encrypted with PU_a and containing A's nonce (N_1) as well as a new nonce generated by B (N_2). Because only B could have decrypted message (1), the presence of N_1 in message (2) assures A that the correspondent is B.
3. A returns N_2, encrypted using B's public key, to assure B that its correspondent is A.
4. A selects a secret key K_s and sends $M = \mathrm{E}(PU_b, \mathrm{E}(PR_a, K_s))$ to B. Encryption of this message with B's public key ensures that only B can read it; encryption with A's private key ensures that only A could have sent it.
5. B computes $\mathrm{D}(PU_a, \mathrm{D}(PR_b, M))$ to recover the secret key.

The result is that this scheme ensures both confidentiality and authentication in the exchange of a secret key.

A Hybrid Scheme

- Performance
- Backward Compatibility

DISTRIBUTION OF PUBLIC KEYS

DISTRIBUTION OF PUBLIC KEYS

- Public announcement
- Publicly available directory
- Public-key authority
- Public-key certificates

Figure 14.9 Uncontrolled Public-Key Distribution

Public announcement

Figure 14.9 Uncontrolled Public-Key Distribution

Publicly available directory

Figure 14.10 Public-Key Publication

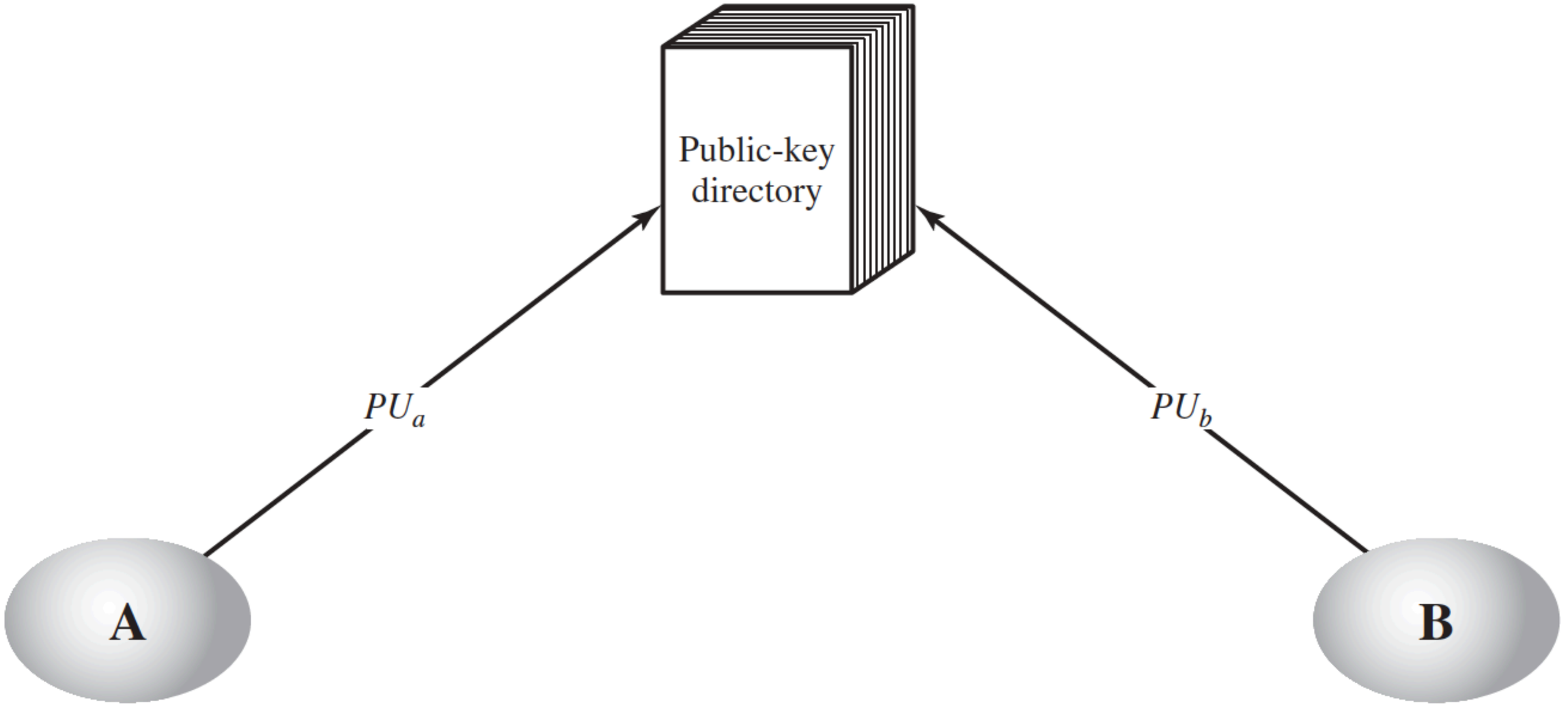
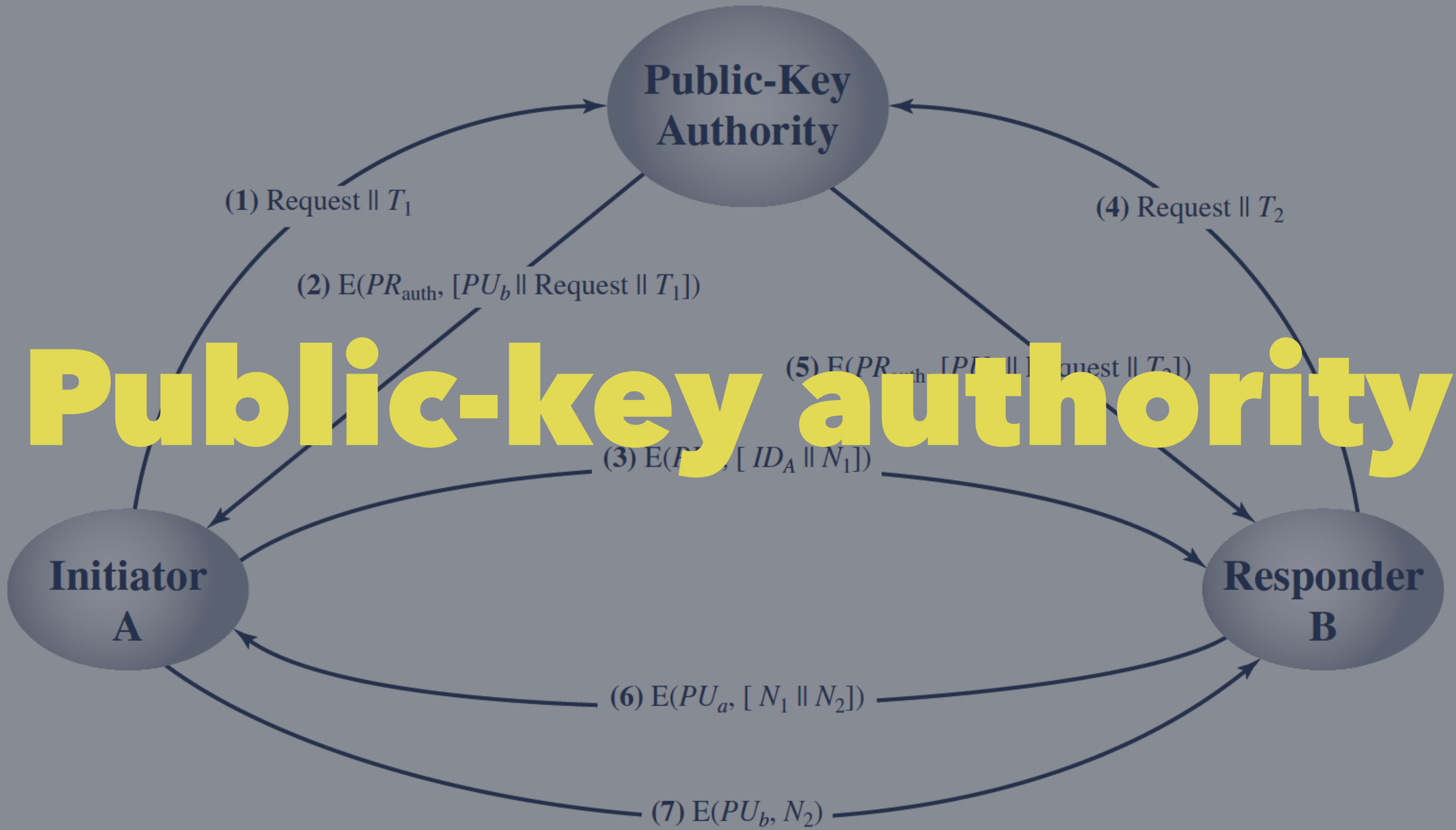


Figure 14.10 Public-Key Publication



Public-key authority

Figure 14.11 Public-Key Distribution Scenario

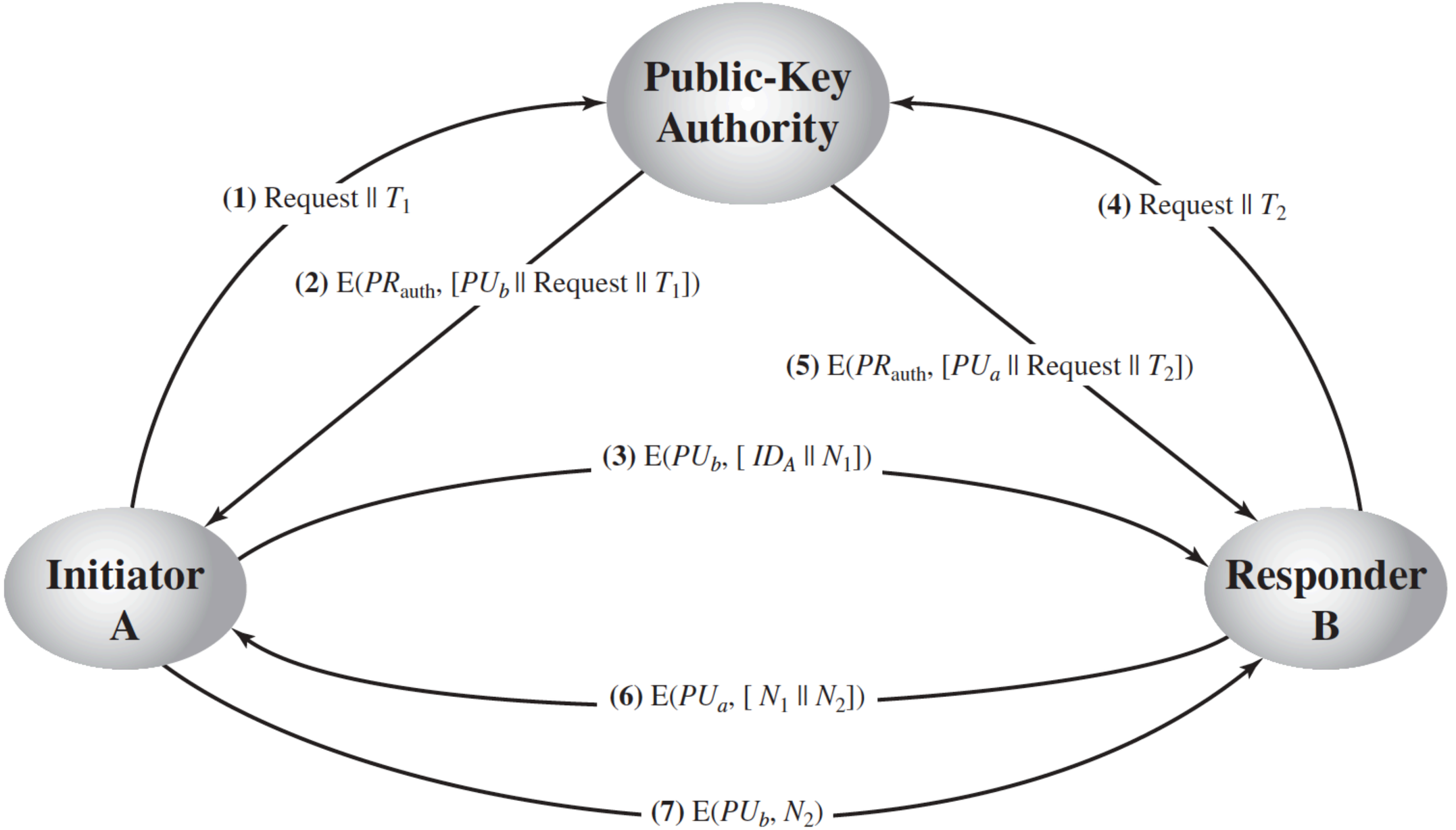


Figure 14.11 Public-Key Distribution Scenario

Public-key certificates

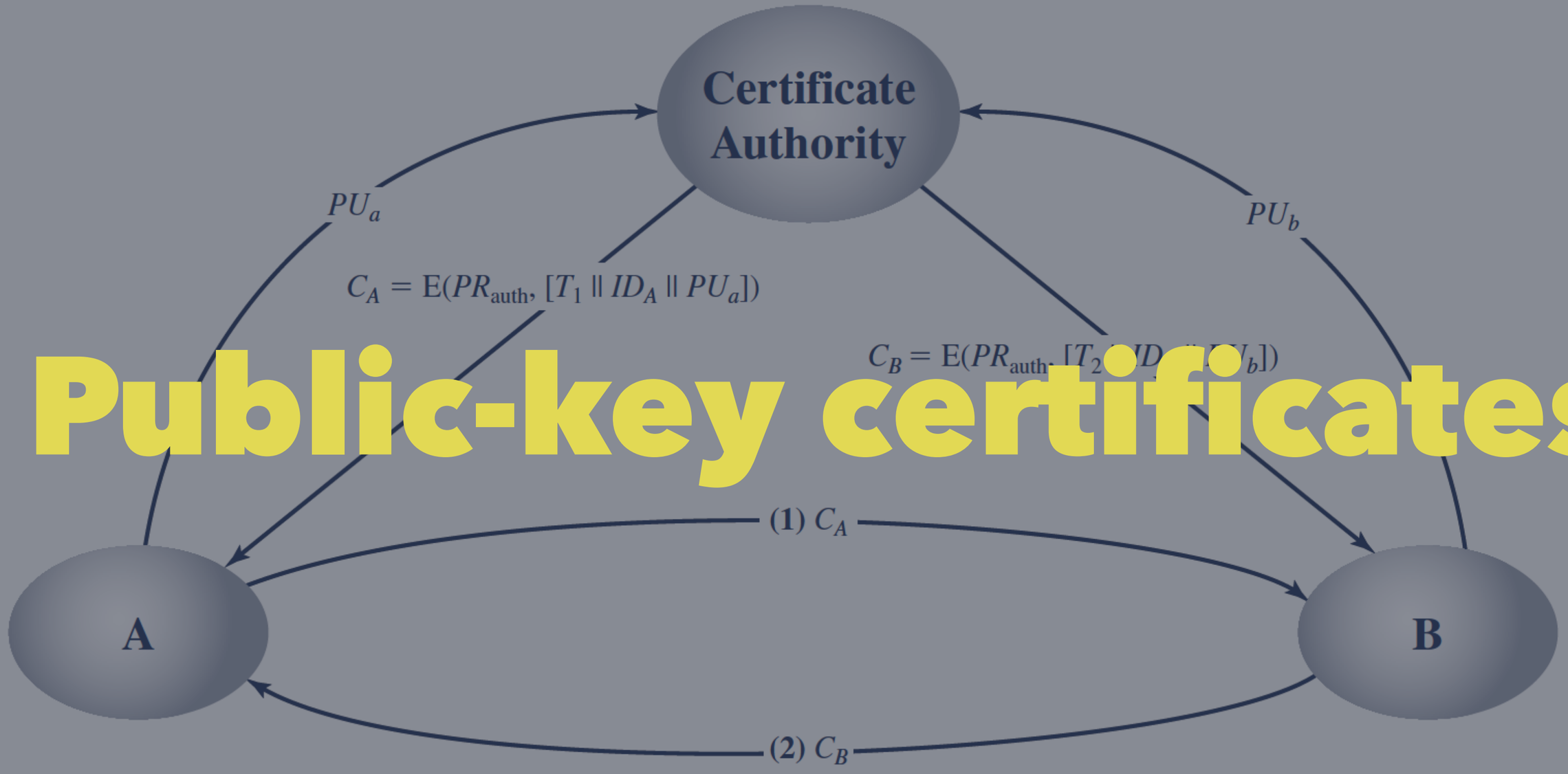


Figure 14.12 Exchange of Public-Key Certificates

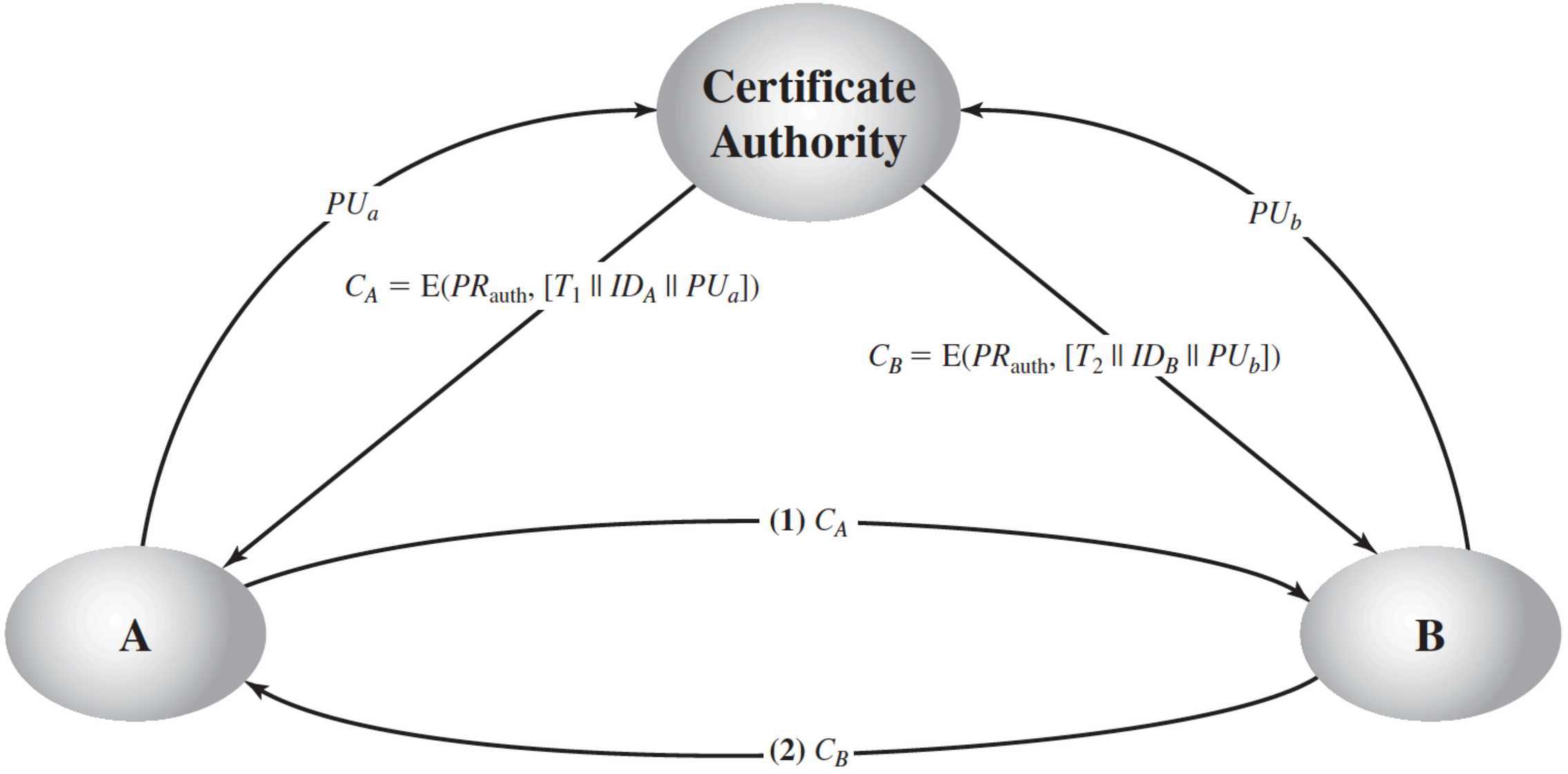


Figure 14.12 Exchange of Public-Key Certificates

X.509 Certificates

- 1988 -> 1990 -> 1993 -> 1995 ->2000

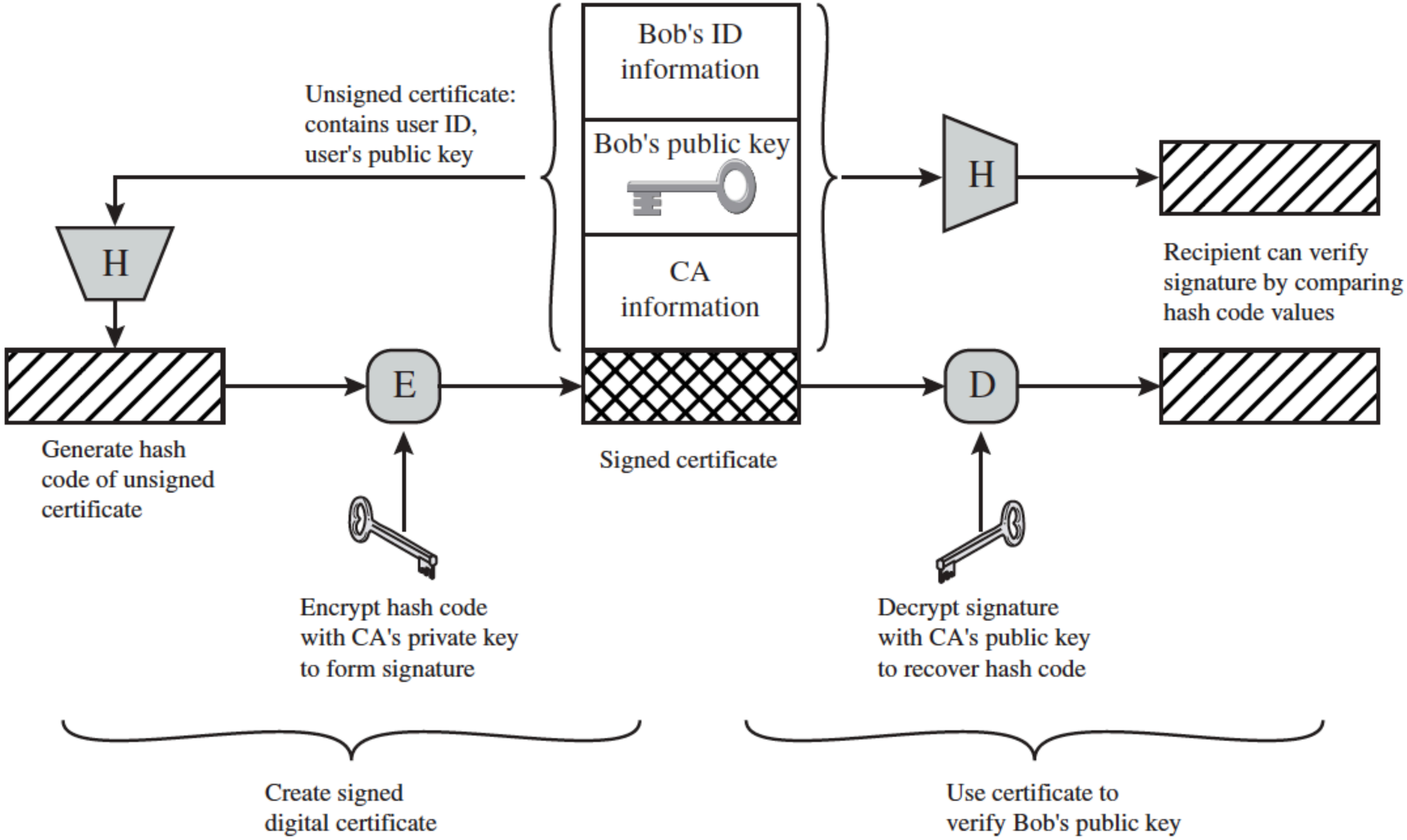
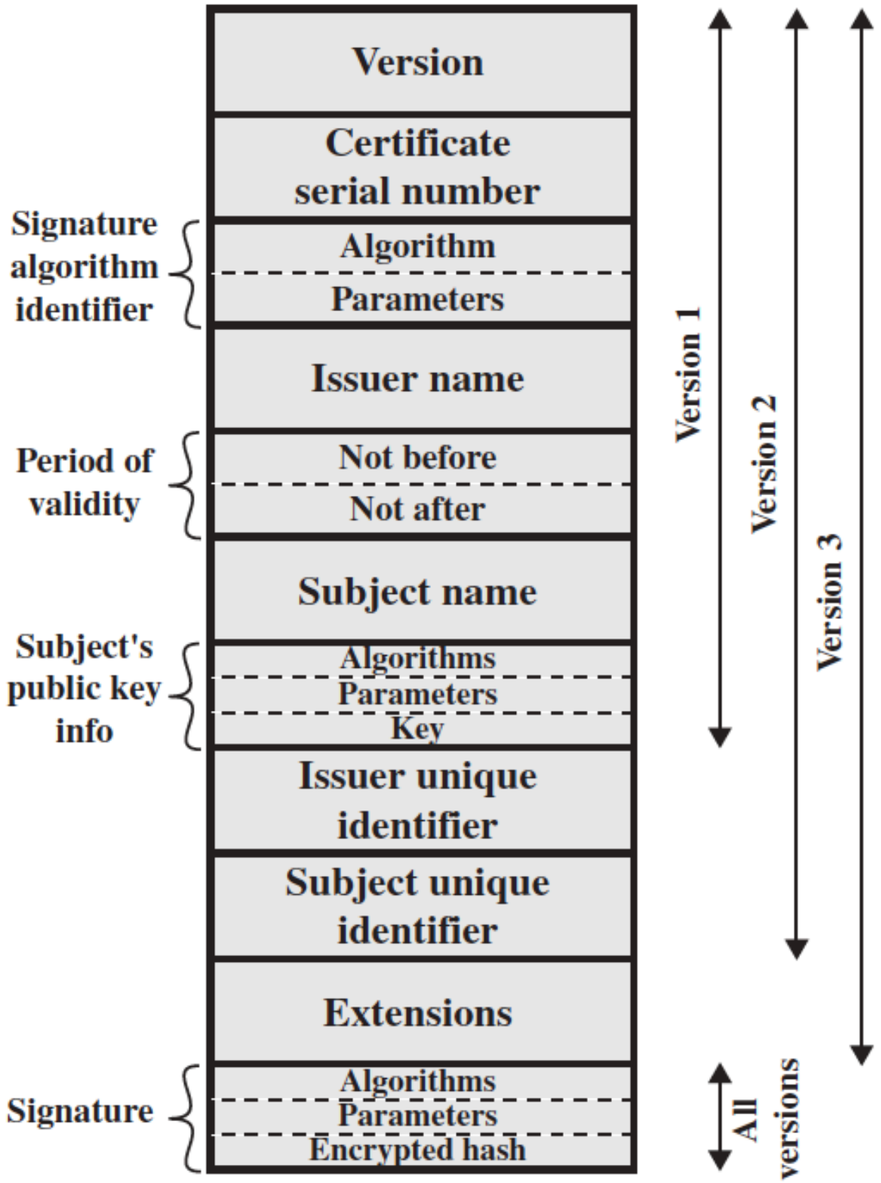
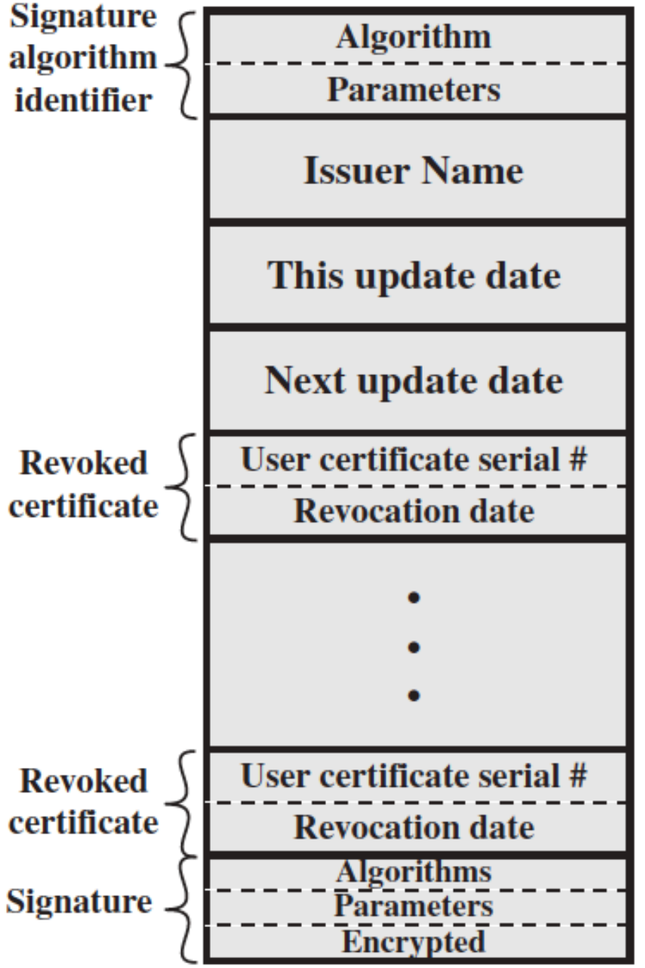


Figure 14.13 Public-Key Certificate Use



(a) X.509 certificate



(b) Certificate revocation list

Figure 14.14 X.509 Formats

PUBLIC-KEY INFRASTRUCTURE

- PKIX Architectural Model
- Elements of PKIX Architectural Model
- PKIX Management Functions

PKIX Architectural Model

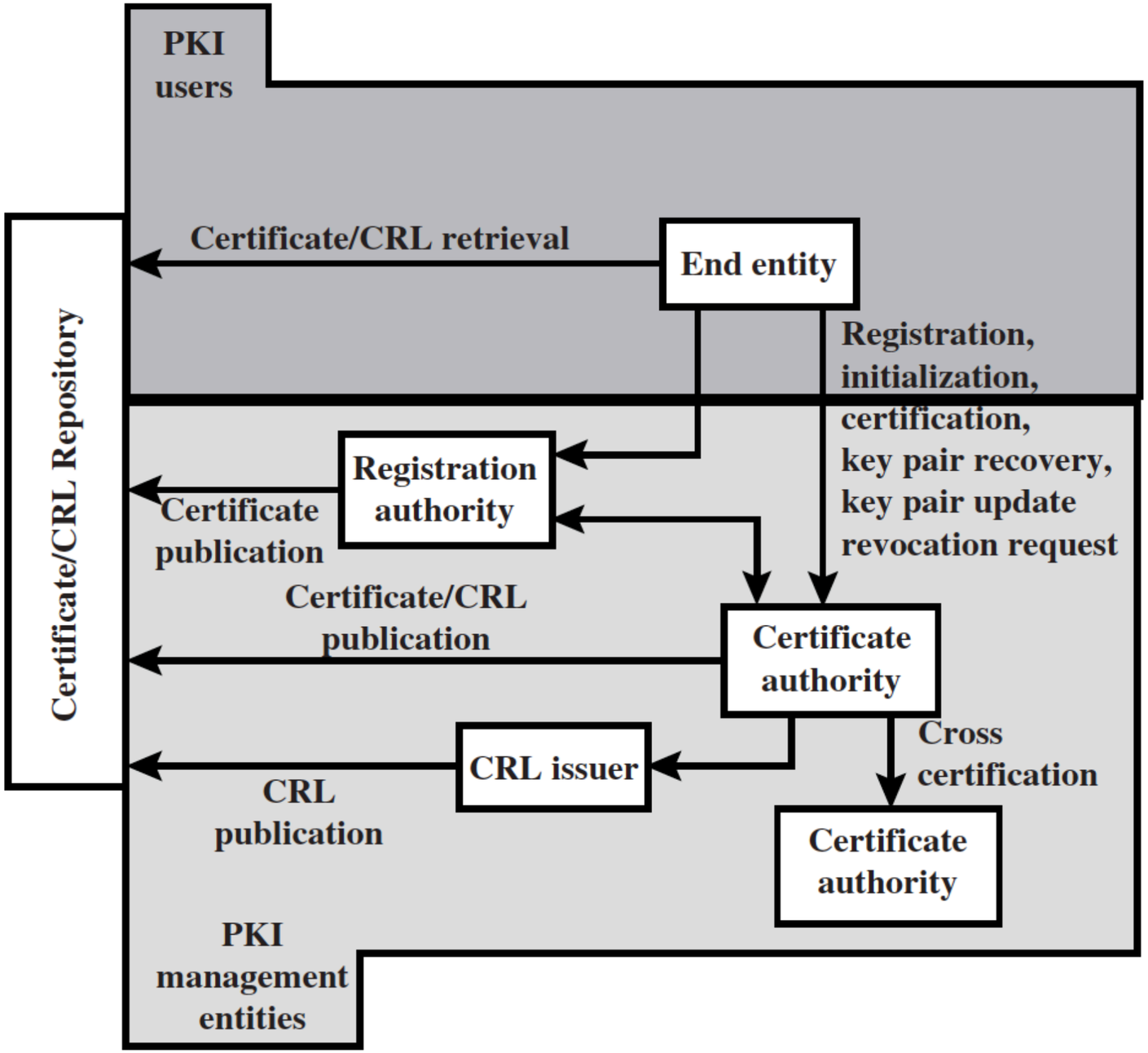


Figure 14.16 PKIX Architectural Model

Elements of PKIX Architectural Model

- End entity
- Certification authority (CA)
- Registration authority (RA)
- CRL issuer
- Repository

PKIX Management Functions

- Registration
- Initialization
- Certification
- Key pair recovery
- Key pair update
- Revocation request
- Cross certification