

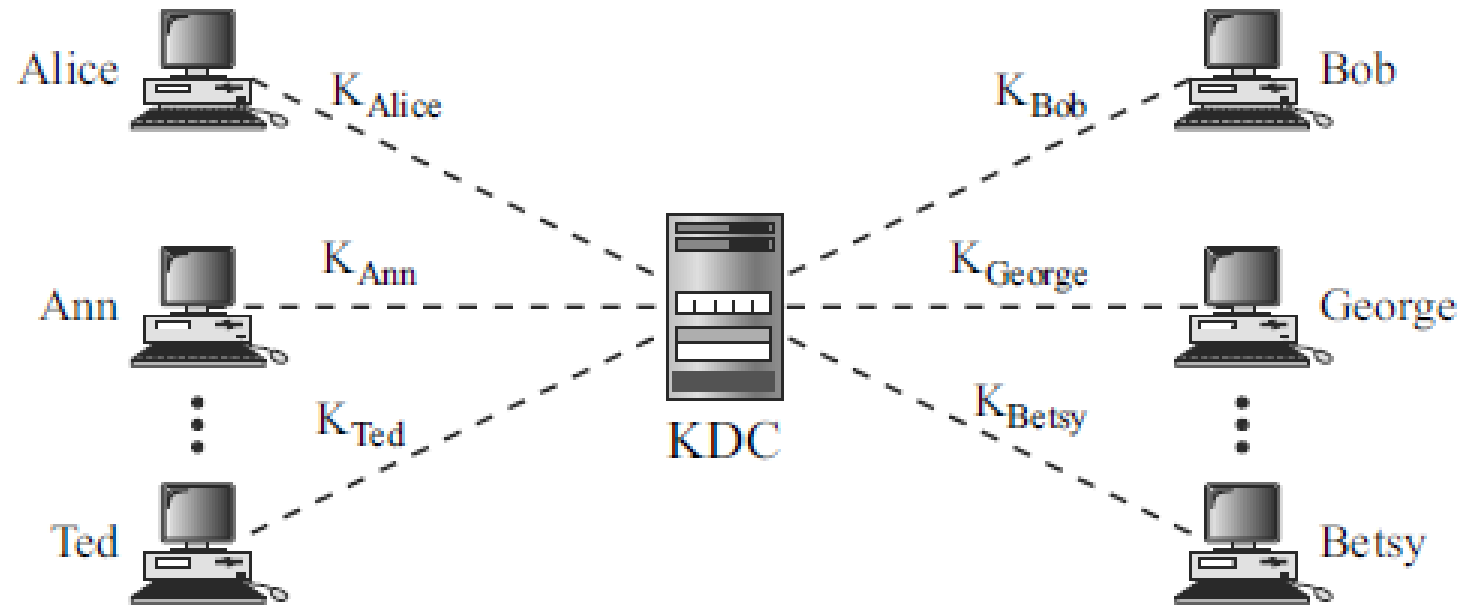
Key Management

SYMMETRIC-KEY DISTRIBUTION

- Symmetric-key cryptography is more efficient than asymmetric-key cryptography for enciphering large messages.
needs a shared secret key between two parties.
- If Alice needs to exchange confidential messages with N people, she needs N different keys.
- if N people need to communicate with each other- A total of $N(N - 1)$ keys is needed if we require that Alice and Bob use two keys for bidirectional communication;
- only $N(N - 1)/2$ keys are needed if we allow a key to be used for both directions.
- This is normally referred to as the N^2 problem because the number of required keys for N entities is N^2
- If Alice and Bob want to communicate, they need a way to exchange a secret key; if Alice wants to communicate with one million people, how can she exchange one million keys with one million people
- Need an efficient way to maintain and distribute secret keys.

Key-Distribution Center: KDC

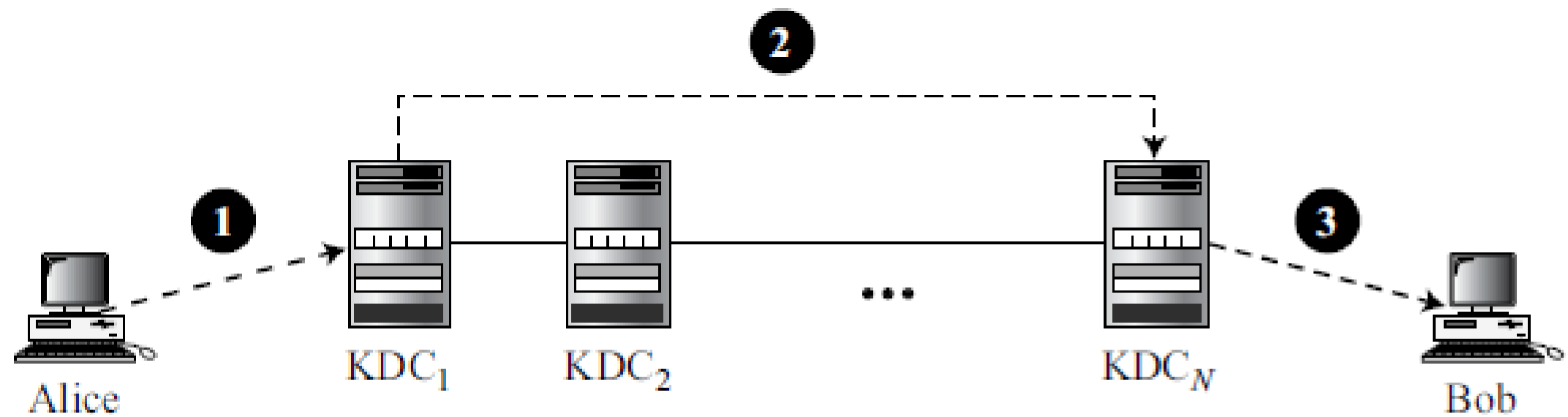
- use of a trusted third party, referred to as a key-distribution center (KDC).
- To reduce the number of keys, each person establishes a shared secret key with the KDC



- A secret key is established between the KDC and each member.
- Alice has a secretkey with the KDC, which we refer to as K_{Alice} ; Bob has a secret key with the KDC, which we refer to as K_{Bob} ; and so on..
- The process is as follows:
 1. Alice sends a request to the KDC stating that she needs a session (temporary)secret key between herself and Bob.
 2. The KDC informs Bob about Alice's request.
 3. If Bob agrees, a session key is created between the two
- The secret key between Alice and Bob that is established with the KDC is used to authenticate Alice and Bob to the KDC and to prevent Eve from impersonating either of them.

Flat Multiple KDCs

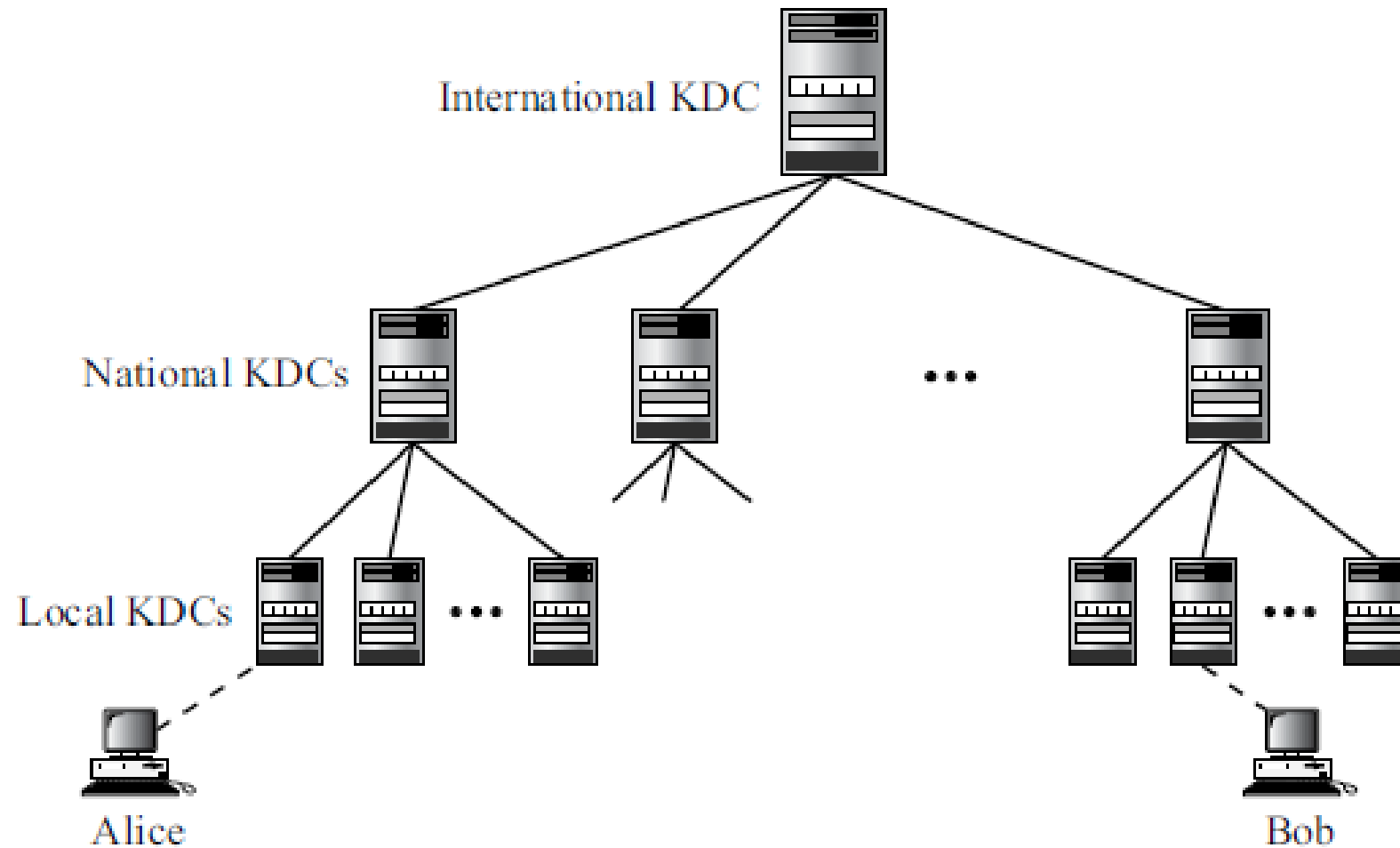
- When the number of people using a KDC increases, the system becomes unmanageable and a bottleneck can result.
- To solve the problem, need to have multiple KDCsWe
- can divide the world into domains.
- Each domain can have one or more KDCs (for redundancy in case of failure).
- if Alice wants to send a confidential message to Bob, who belongs to another domain, Alice contacts her KDC, which in turn contacts the KDC in Bob's domain.
- The two KDCs can create a secret key between Alice and Bob.



Hierarchical Multiple KDCs

- hierarchical system of KDCs, with one or more KDCs at the top of the hierarchy.
- For example, there can be local KDCs, national KDCs, and international KDCs.
- When Alice needs to communicate with Bob, who lives in another country, she sends her request to a local KDC; the local KDC relays the request to the national KDC; the national KDC relays the request to an international KDC.
- The request is then relayed all the way down to the local KDC where Bob lives.

Hierarchical Multiple KDCs



Session Keys


- A KDC creates a secret key for each member. This secret key can be used only between the member and the KDC, not between two members.
- If Alice needs to communicate secretly with Bob, she needs a secret key between herself and Bob.
- A KDC can create a session key between Alice and Bob, using their keys with the center.
- The keys of Alice and Bob are used to authenticate Alice and Bob to the center and to each other before the session key is established.
- After communication is terminated, the session key is no longer useful.


Different approaches have been proposed to create the session key

- A Simple Protocol Using a KDC
- Needham-Schroeder Protocol
- Otway-Rees Protocol

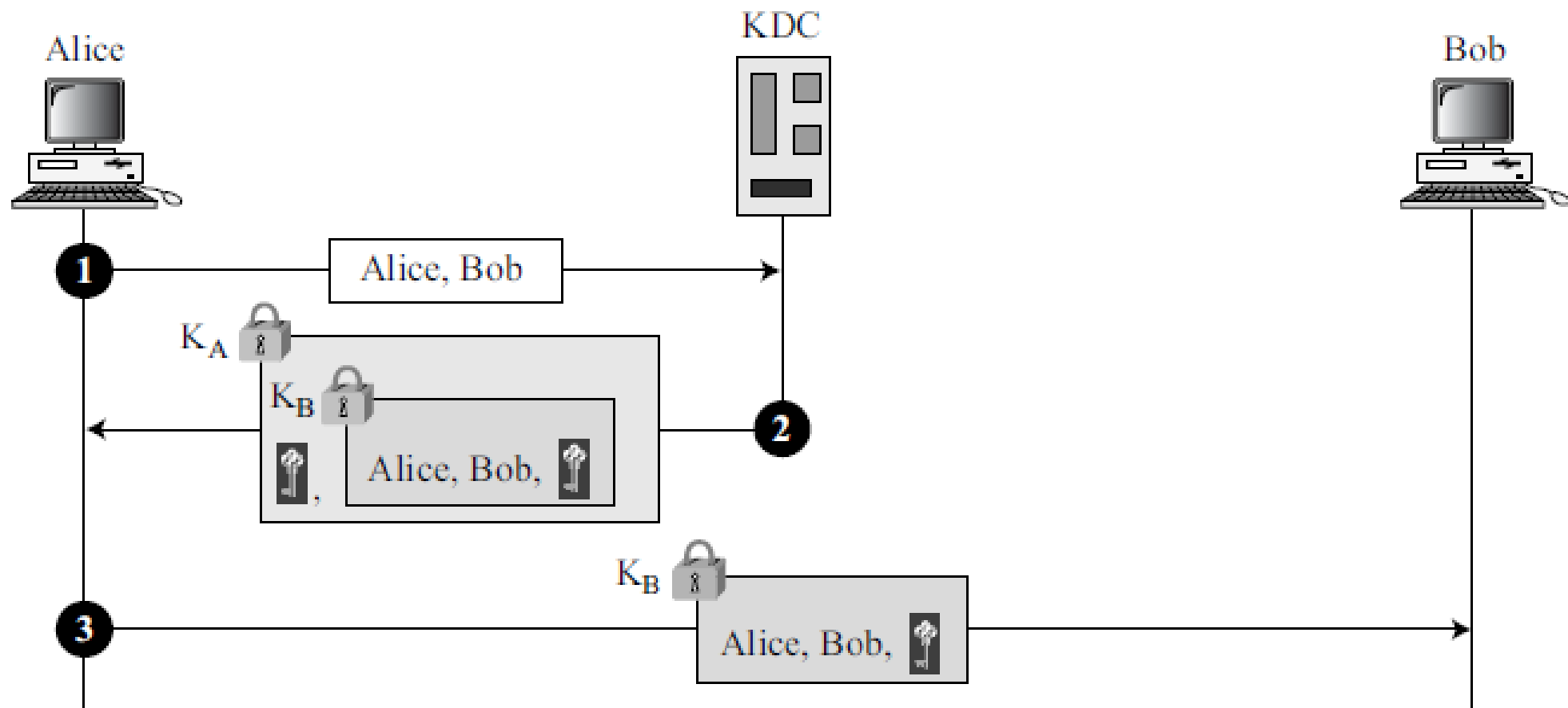
A Simple Protocol Using a KDC

K_A  Encrypted with Alice-KDC secret key

 Session key between Alice and Bob

K_B  Encrypted with Bob-KDC secret key

KDC: Key-distribution center



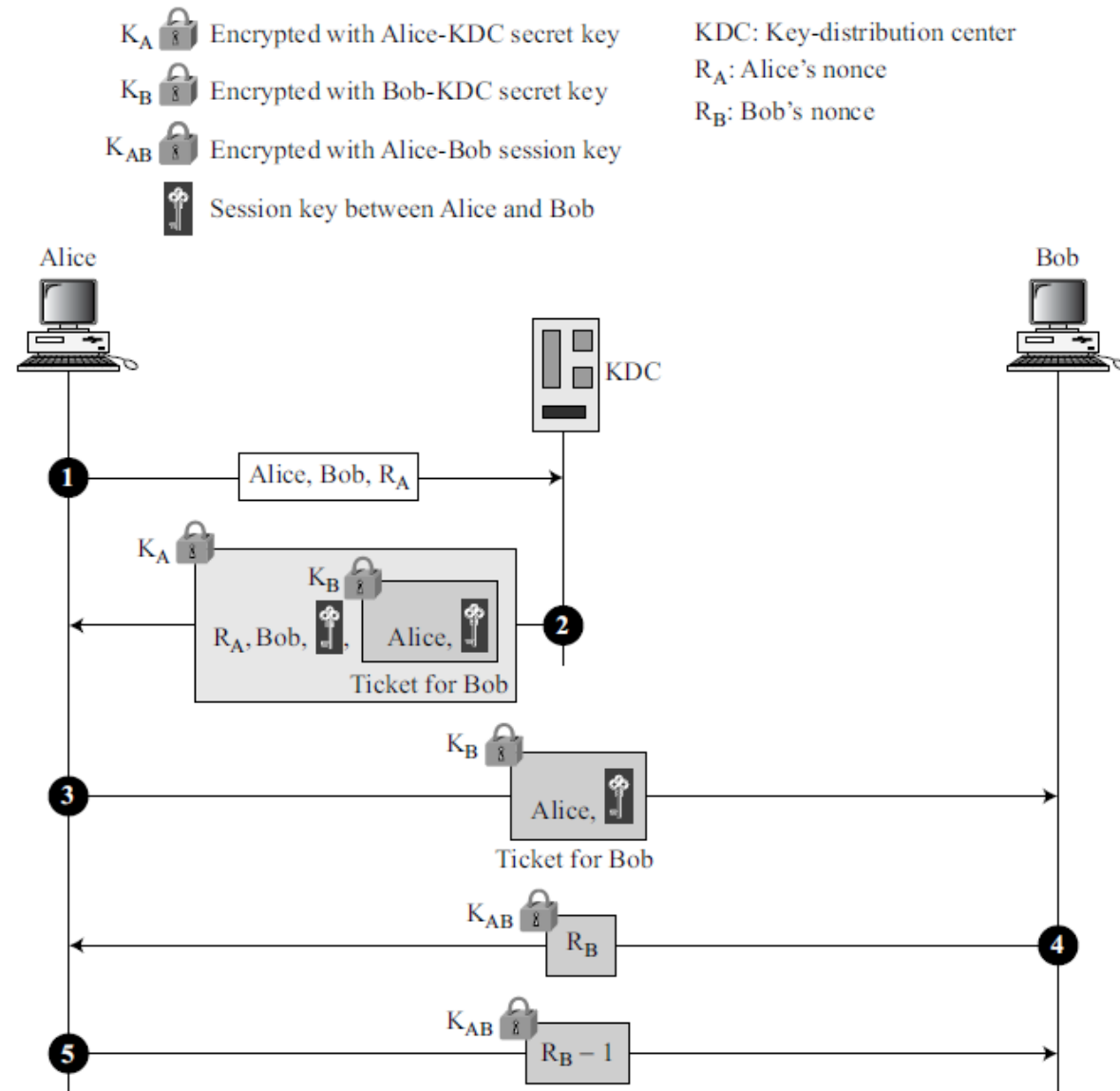
1. Alice sends a plaintext message to the KDC to obtain a symmetric session key between Bob and herself. The message contains her registered identity and the identity of Bob. This message is not encrypted, it is public. The KDC does not care.
2. The KDC receives the message and creates what is called a ticket. The ticket is encrypted using Bob's key (K_B). The ticket contains the identities of Alice and Bob and the session key (K_{AB}). The ticket with a copy of the session key is sent to Alice. Alice receives the message, decrypts it, and extracts the session key. She cannot decrypt Bob's ticket; the ticket is for Bob, not for Alice. Note that this message contains a double encryption; the ticket is encrypted, and the entire message is also encrypted. In the second message, Alice is actually authenticated to the KDC, because only Alice can open the whole message using her secret key with KDC.
3. Alice sends the ticket to Bob. Bob opens the ticket and knows that Alice needs to send messages to him using K_{AB} as the session key. Note that in this message, Bob is authenticated to the KDC because only Bob can open the ticket. Because Bob is authenticated to the KDC, he is also authenticated to Alice, who trusts the KDC. In the same way, Alice is also authenticated to Bob, because Bob trusts the KDC and the KDC has sent Bob the ticket that includes the identity of Alice.

simple protocol has a flaw. Eve can use the replay attack,- she can save the message in step 3 and replay it later.

Needham-Schroeder Protocol

- uses multiple challenge-response interactions between parties to achieve a flawless protocol.
 - Needham and Schroeder uses two nonces: RA and RB.
1. Alice sends a message to the KDC that includes her nonce, RA, her identity, and Bob's identity.
 2. The KDC sends an encrypted message to Alice that includes Alice's nonce, Bob's identity, the session key, and an encrypted ticket for Bob. The whole message is encrypted with Alice's key.
 3. Alice sends Bob's ticket to him.
 4. Bob sends his challenge to Alice (RB), encrypted with the session key.
 5. Alice responds to Bob's challenge. Note that the response carries RB- 1 instead of RB.





Needham-Schroeder Protocol



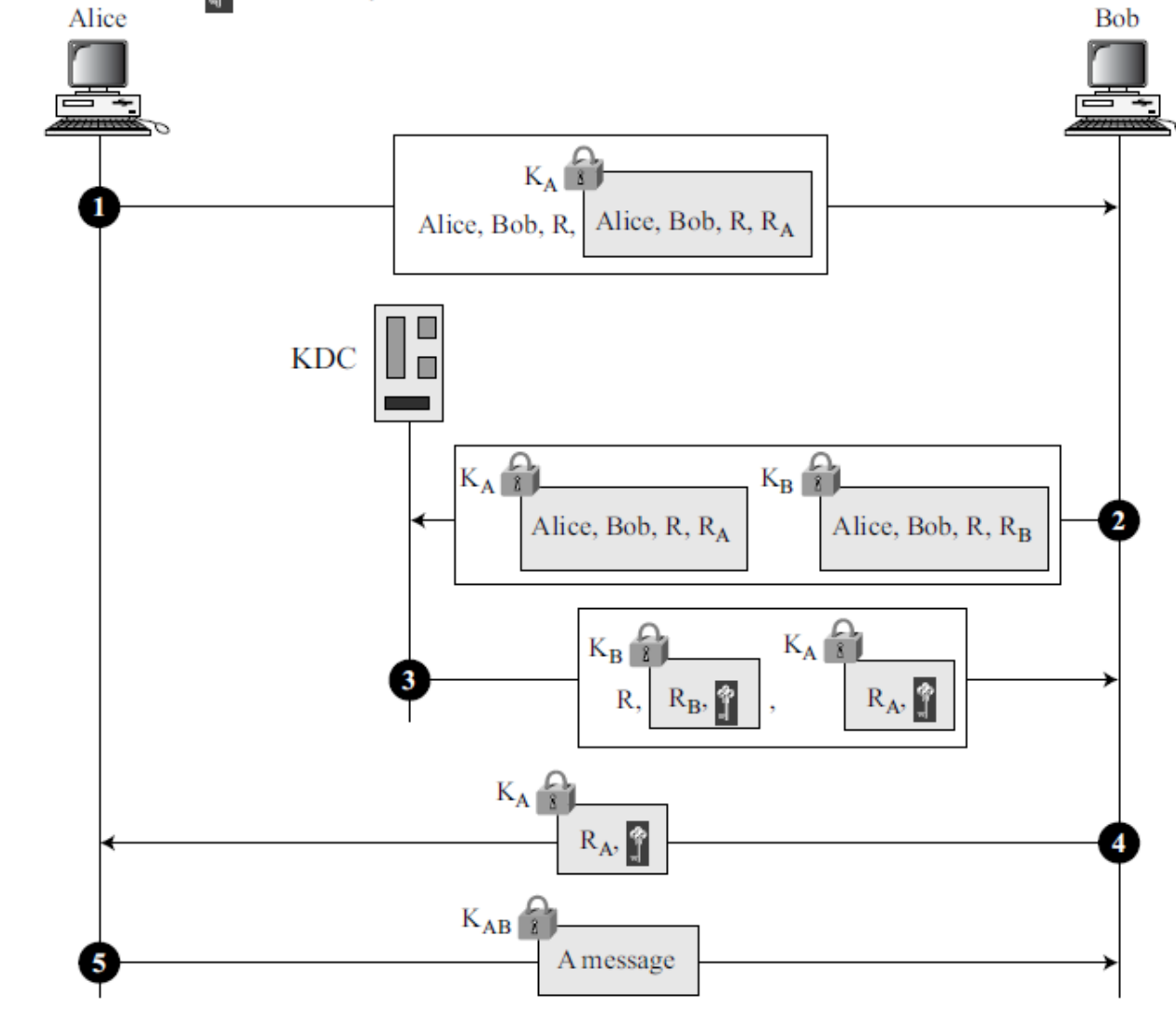
Otway-Rees Protocol

1. Alice sends a message to Bob that includes a common nonce, R , the identities of Alice and Bob, and a ticket for KDC that includes Alice's nonce R_A (a challenge for the KDC to use), a copy of the common nonce, R , and the identities of Alice and Bob.
2. Bob creates the same type of ticket, but with his own nonce R_B . Both tickets are sent to the KDC.
3. The KDC creates a message that contains R , the common nonce, a ticket for Alice and a ticket for Bob; the message is sent to Bob. The tickets contain the corresponding nonce, R_A or R_B , and the session key, K_{AB} .
4. Bob sends Alice her ticket.
5. Alice sends a short message encrypted with her session key K_{AB} to show that she has the session key.

Otway-Rees Protocol

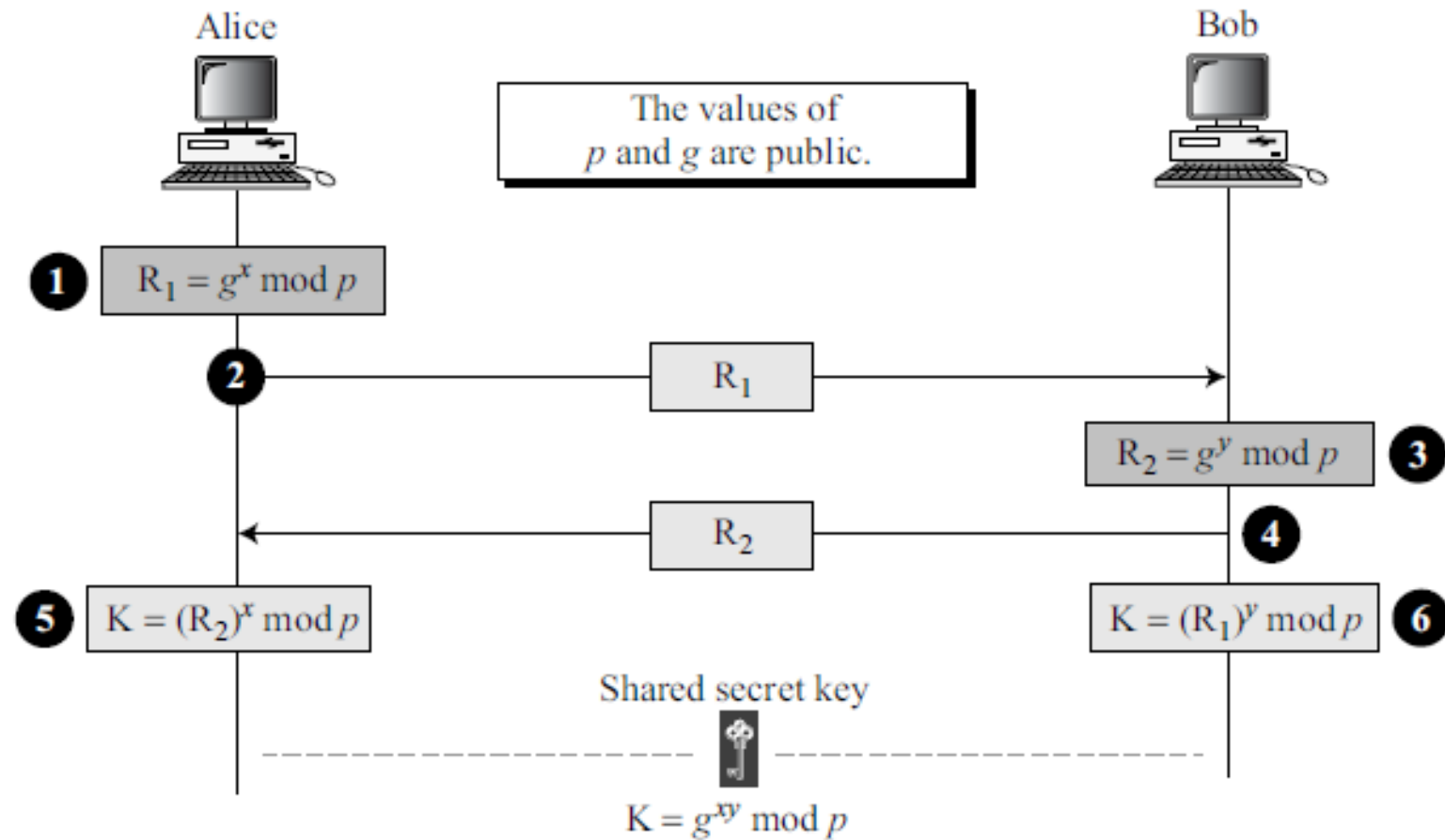
K_A  Encrypted with Alice-KDC secret key
 K_B  Encrypted with Bob-KDC secret key
 K_{AB}  Encrypted with Alice-Bob session key
 Session key between Alice and Bob

KDC: Key-distribution center
 R_A : Nonce from Alice to KDC
 R_B : Nonce from Bob to KDC
 R : Common nonce



SYMMETRIC-KEY AGREEMENT

- Alice and Bob can create a session key between themselves without using a KDC.
- This method of session-key creation is referred to as the symmetric-key agreement.
- Diffie-Hellman Key Agreement



Diffie-Hellman Key Agreement

- Man-in-the-Middle Attack

1. Alice chooses x , calculates $R1 = g^x \bmod p$, and sends $R1$ to Bob.

2. Eve, the intruder, intercepts $R1$. She chooses z , calculates $R2 = g^z \bmod p$, and sends $R2$ to both Alice and Bob.

3. Bob chooses y , calculates $R3 = g^y \bmod p$, and sends $R3$ to Alice. $R3$ is intercepted by Eve and never reaches Alice.

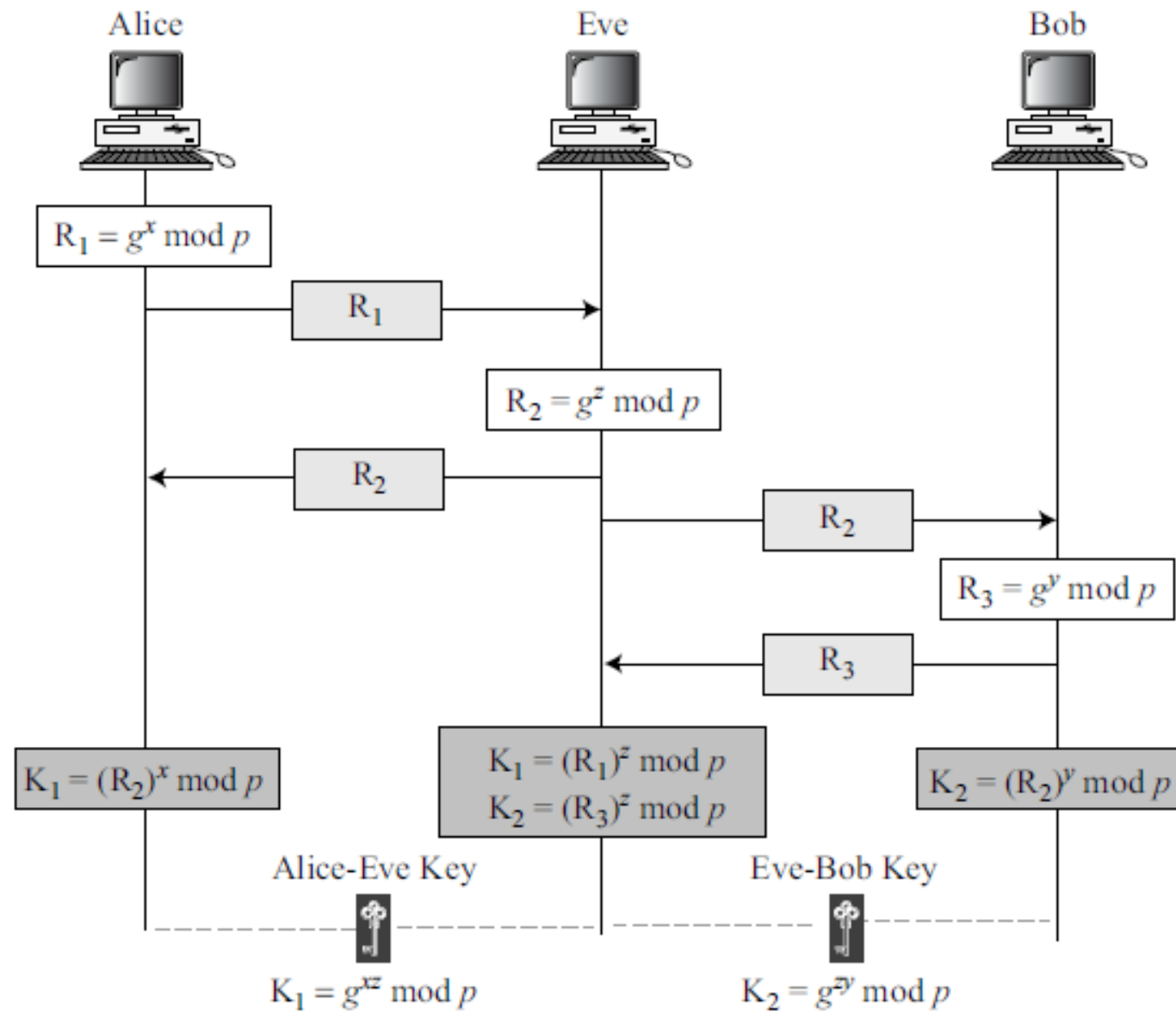
4. Alice and Eve calculate $K1 = g^{xz} \bmod p$, which becomes a shared key between Alice and Eve.

Alice, however, thinks that it is a key shared between Bob and herself.

5. Eve and Bob calculate $K2 = g^{zy} \bmod p$, which becomes a shared key between Eve and Bob. Bob, however, thinks that it is a key shared between Alice and himself

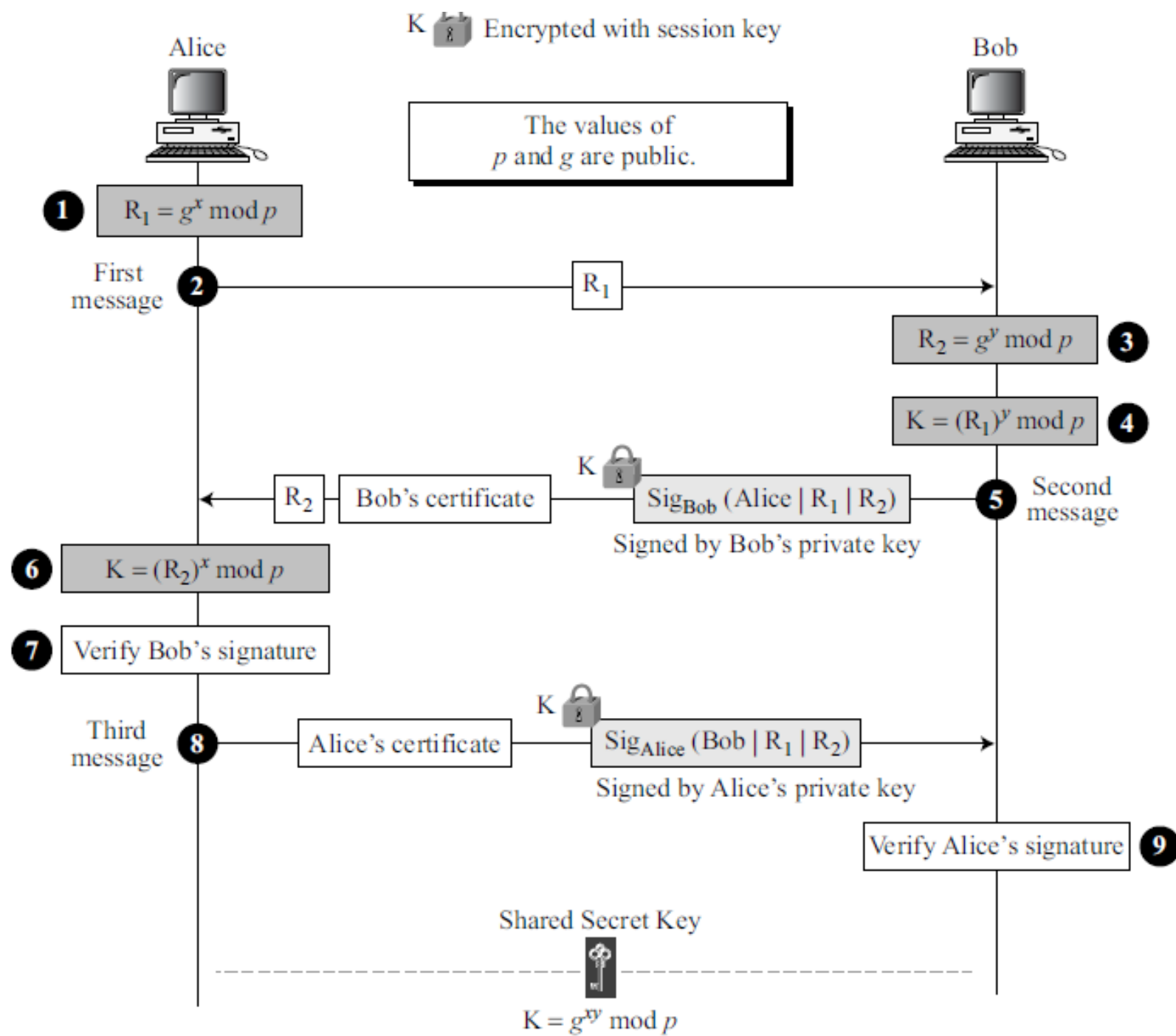
- When Alice sends data to Bob encrypted with $K1$ (shared by Alice and Eve), it can be deciphered and read by Eve.
- Eve can send the message to Bob encrypted by $K2$ (shared key between Eve and Bob); or she can even change the message or send a totally new message.
- Bob is fooled into believing that the message has come from Alice.

Man-in-the-Middle Attack



Station-to-Station Key Agreement

- The station-to-station protocol is a method based on Diffie-Hellman.
- It uses digital signatures with public-key certificates to establish a session key between Alice and Bob
- After calculating R_1 , Alice sends R_1 to Bob
- After calculating R_2 and the session key, Bob concatenates Alice's ID, R_1 , and R_2 . He then signs the result with his private key. Bob now sends R_2 , the signature, and his own public-key certificate to Alice. The signature is encrypted with the session key
- After calculating the session key, if Bob's signature is verified, Alice concatenates Bob's ID, R_1 , and R_2 . She then signs the result with her own private key and sends it to Bob. The signature is encrypted with the session key
- If Alice's signature is verified, Bob keeps the session key



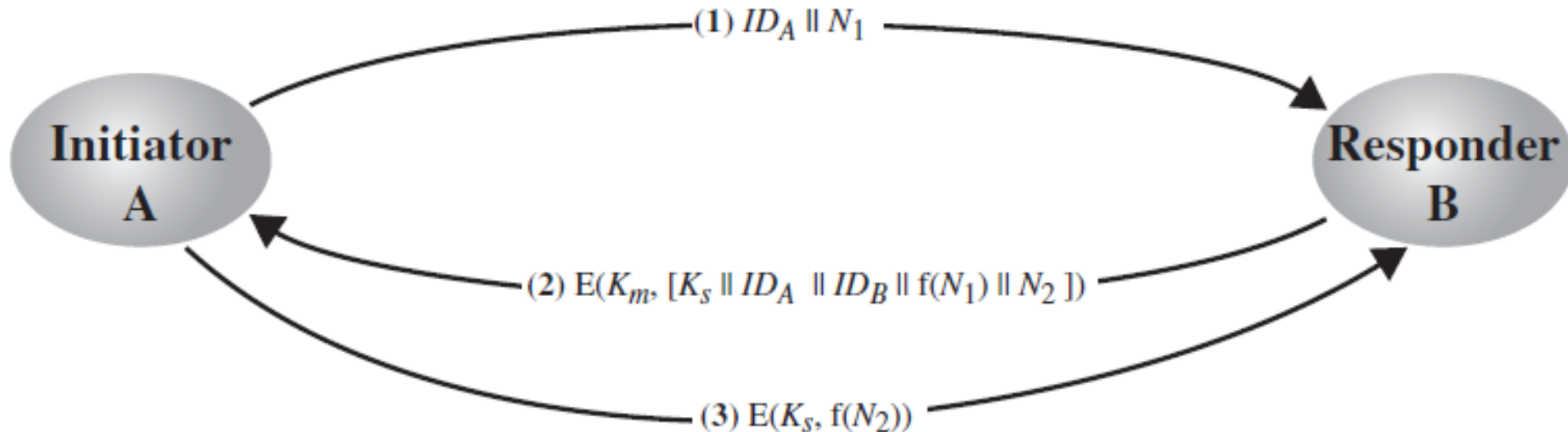
- **Security of Station-to-Station Protocol**
- The station-to-station protocol prevents man-in-the-middle attacks.
- After intercepting R1, Eve cannot send her own R2 to Alice and pretend it is coming from Bob because Eve cannot forge the private key of Bob to create the signature - the signature cannot be verified with Bob's public key defined in the certificate.
- In the same way, Eve cannot forge Alice's private key to sign the third message sent by Alice.
- The certificates, are trusted because they are issued by trusted authorities.

Decentralized Key Control

- The use of a key distribution center imposes the requirement that the KDC be trusted and be protected from subversion.
- This requirement can be avoided if key distribution is fully decentralized.
- Although full decentralization is not practical for larger networks using symmetric encryption only, it may be useful within a local context.
- A decentralized approach requires that each end system be able to communicate in a secure manner with all potential partner end systems for purposes of session key distribution.
- there may need to be as many as $n(n-1)/2$ master keys for a configuration with n end systems
- although each node must maintain at most $(n-1)$ master keys, as many session keys as required may be generated and used.
- Because the messages transferred using the master key are short, cryptanalysis is difficult.
- Session keys are used for only a limited time to protect them.

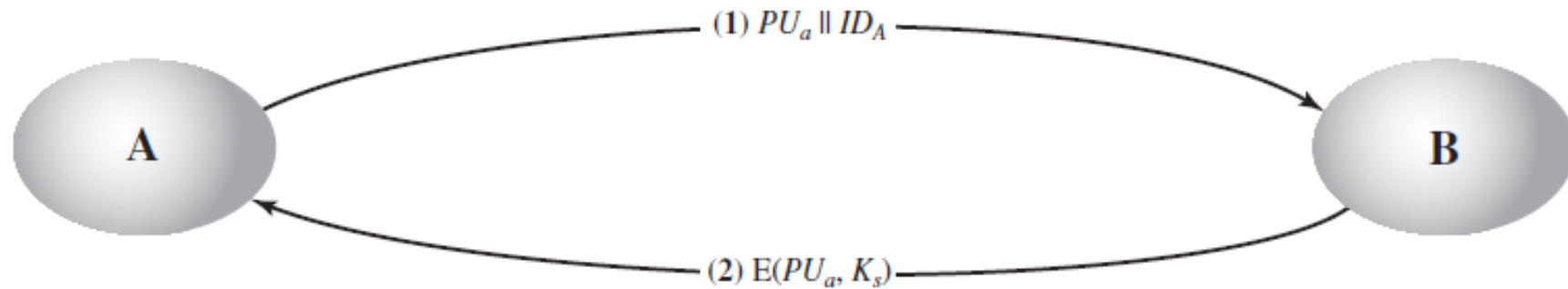
Decentralized Key distribution

- A session key may be established with the following sequence of steps
 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 , and another nonce, N_2
 3. Using the new session key, A returns $f(N_2)$ to B.



SYMMETRIC KEY DISTRIBUTION USING ASYMMETRIC ENCRYPTION

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, IDA
 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 .



SYMMETRIC KEY DISTRIBUTION USING ASYMMETRIC ENCRYPTION Contd..

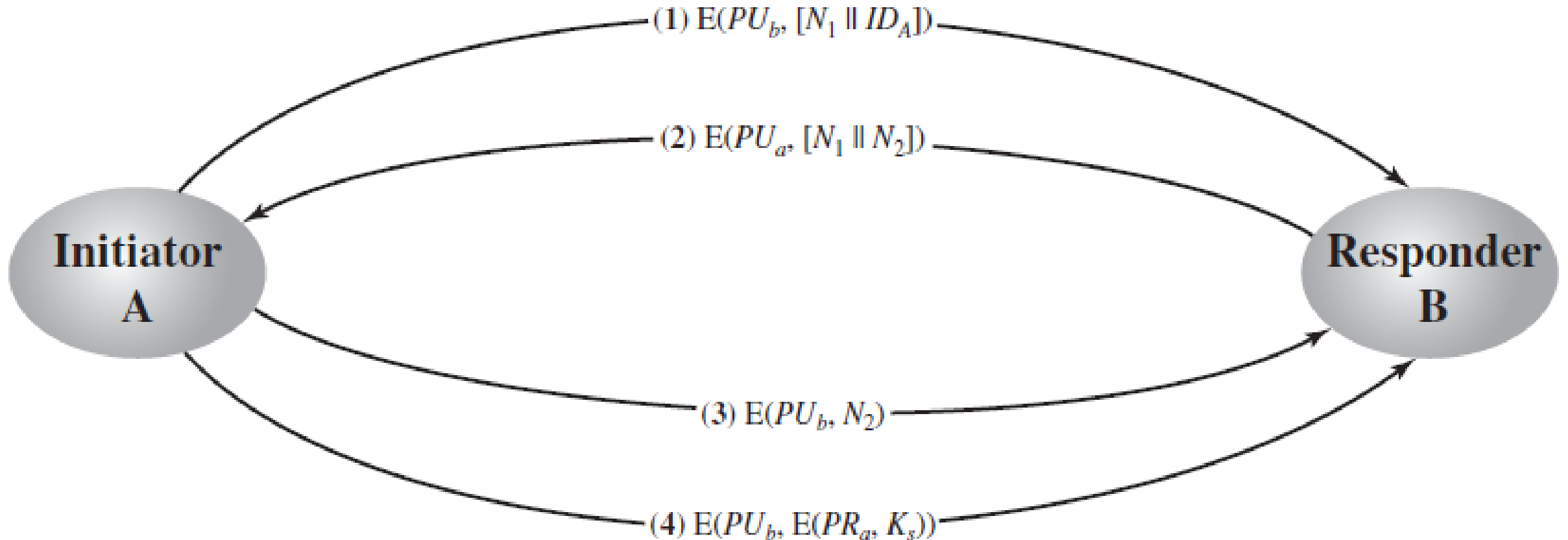
- No keys exist before the start of the communication and none exist after the completion of communication.
 - Thus, the risk of compromise of the keys is minimal.
 - At the same time, the communication is secure from eavesdropping.
 - insecure against an adversary who can intercept messages and then either relay the intercepted message or substitute another message -an attack is known as a **man-in-the-middle attack**
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, IDA
 2. E intercepts the message, creates its own public/private key pair $\{PU_e, PR_e\}$ and transmits $PU_e || IDA$ 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.

man-in-the-middle attack Contd...

- The result is that both A and B know K_s and are unaware that K_s has also been revealed to E.
- A and B can now exchange messages using K_s .
- E no longer actively interferes with the communications channel but simply eavesdrops.
- Knowing K_s , E can decrypt all messages, and both A and B are unaware of the problem.
- Thus, this simple protocol is only useful in an environment where the only threat is eavesdropping.

Secret Key Distribution with Confidentiality and Authentication

- provides protection against both active and passive attacks.
- it is assumed that A and B have exchanged public keys by one of the schemes. The following steps occur.



1. A uses B's public key to encrypt a message to B containing an identifier of $A(IDA)$ and a nonce ($N1$), which is used to identify this transaction uniquely.
 2. B sends a message to A encrypted with PUa and containing A's nonce ($N1$) as well as a new nonce generated by B ($N2$). Because only B could have decrypted message (1), the presence of ($N1$) in message (2) assures A that the correspondent is B.
 3. A returns $N2$, encrypted using B's public key, to assure B that its correspondent is A.
 4. A selects a secret key Ks and sends $M = E(PUb, E(PRa, Ks))$ 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 $D(PUa, D(PRb, M))$ to recover the secret key.
- The result is that this scheme ensures both confidentiality and authentication in the exchange of a secret key.

PUBLIC-KEY DISTRIBUTION

- If Alice wants to send a message to Bob, she only needs to know Bob's public key, which is open to the public and available to everyone.
- If Bob needs to send a message to Alice, he only needs to know Alice's public key, which is also known to everyone.
- In public-key cryptography, everyone shields a private key and advertises a public key.
- Public keys, like secret keys, need to be distributed to be useful
- all these proposals can be grouped into the following general schemes
 - Public announcement
 - Publicly available directory
 - Public-key authority
 - Public-key certificates

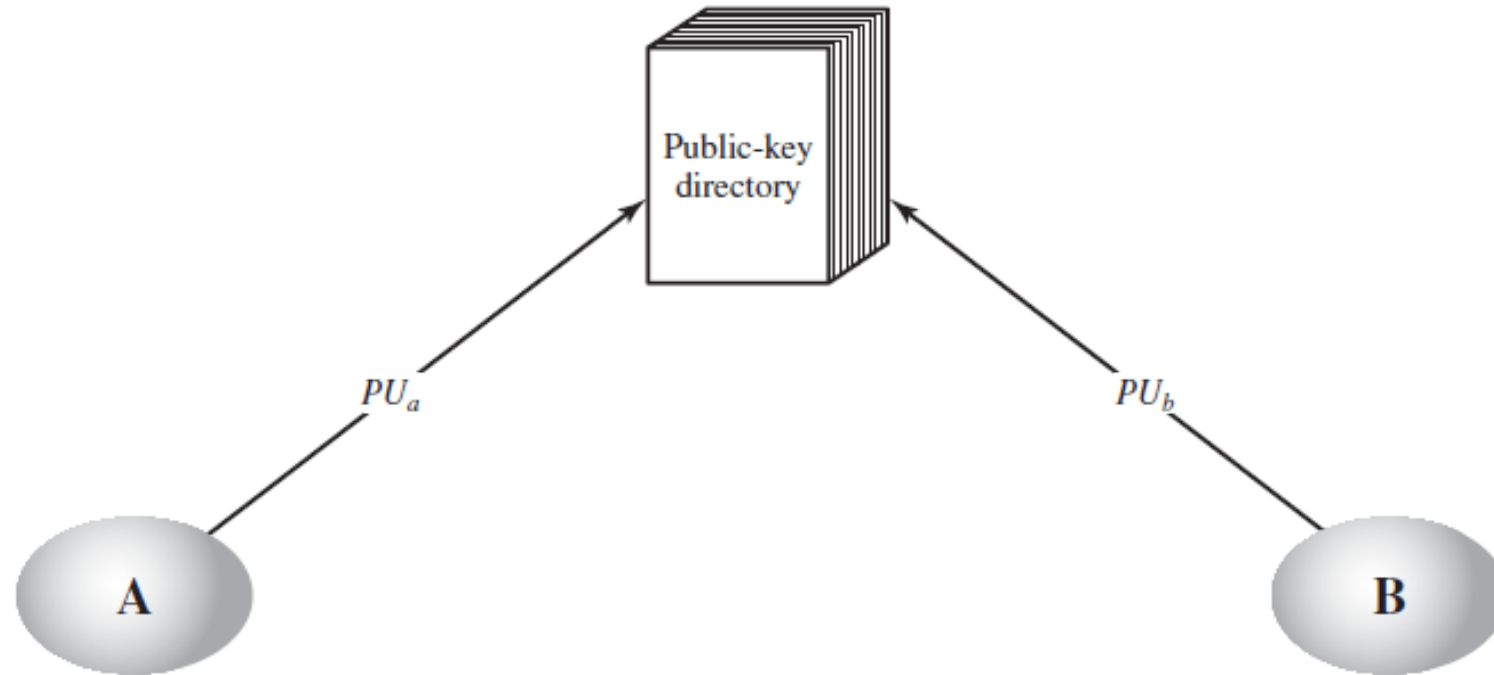
Public Announcement of Public Keys



- Although this approach is convenient, it has a major weakness.
- Anyone can forge such a public announcement. That is, some user could pretend to be user A and send a public key to another participant or broadcast such a public key.
- Until such time as user A discovers the forgery and alerts other participants, the forger is able to read all encrypted messages intended for A and can use the forged keys for authentication

Publicly Available Directory (Trusted Center)

- A greater degree of security can be achieved by maintaining a publicly available dynamic directory of public keys.
- Maintenance and distribution of the public directory would have to be the responsibility of some trusted entity or organization



Publicly Available Directory (Trusted Center) Contd..

Such a scheme would include the following elements

1. The authority maintains a directory with a {name, public key} entry for each participant.
2. Each participant registers a public key with the directory authority.

Registration would have to be in person or by some form of secure authenticated communication.

3. A participant may replace the existing key with a new one at any time, either because of the desire to replace a public key that has already been used for a large amount of data, or because the corresponding private key has been compromised in some way

4. Participants could also access the directory electronically. For this purpose, secure, authenticated communication from the authority to the participant is mandatory.

- This scheme is more secure than individual public announcements
- has vulnerabilities- If an adversary succeeds in obtaining or computing the private key of the directory authority, the adversary could authoritatively pass out counterfeit public keys and subsequently impersonate any participant and eavesdrop on messages sent to any participant.
- the adversary can tamper with the records kept by the authority.

Public-Key Authority

- Stronger security for public-key distribution can be achieved by providing tighter control over the distribution of public keys from the directory.
- assumes that a central authority maintains a dynamic directory of public keys of all participants.
- In addition, each participant reliably knows a public key for the authority, with only the authority knowing the corresponding private key. The following steps occur.

1. A sends a timestamped message to the public-key authority containing a request for the current public key of B.

2. The authority responds with a message that is encrypted using the authority's private key, PR_{auth} . A is able to decrypt the message using the authority's public key. Therefore, A is assured that the message originated with the authority. The message includes the following:

- B's public key, PUB , which A can use to encrypt messages destined for B
- The original request used to enable A to match this response with the corresponding earlier request and to verify that the original request was not altered before reception by the authority
- The original timestamp given so A can determine that this is not an old message from the authority containing a key other than B's current public key

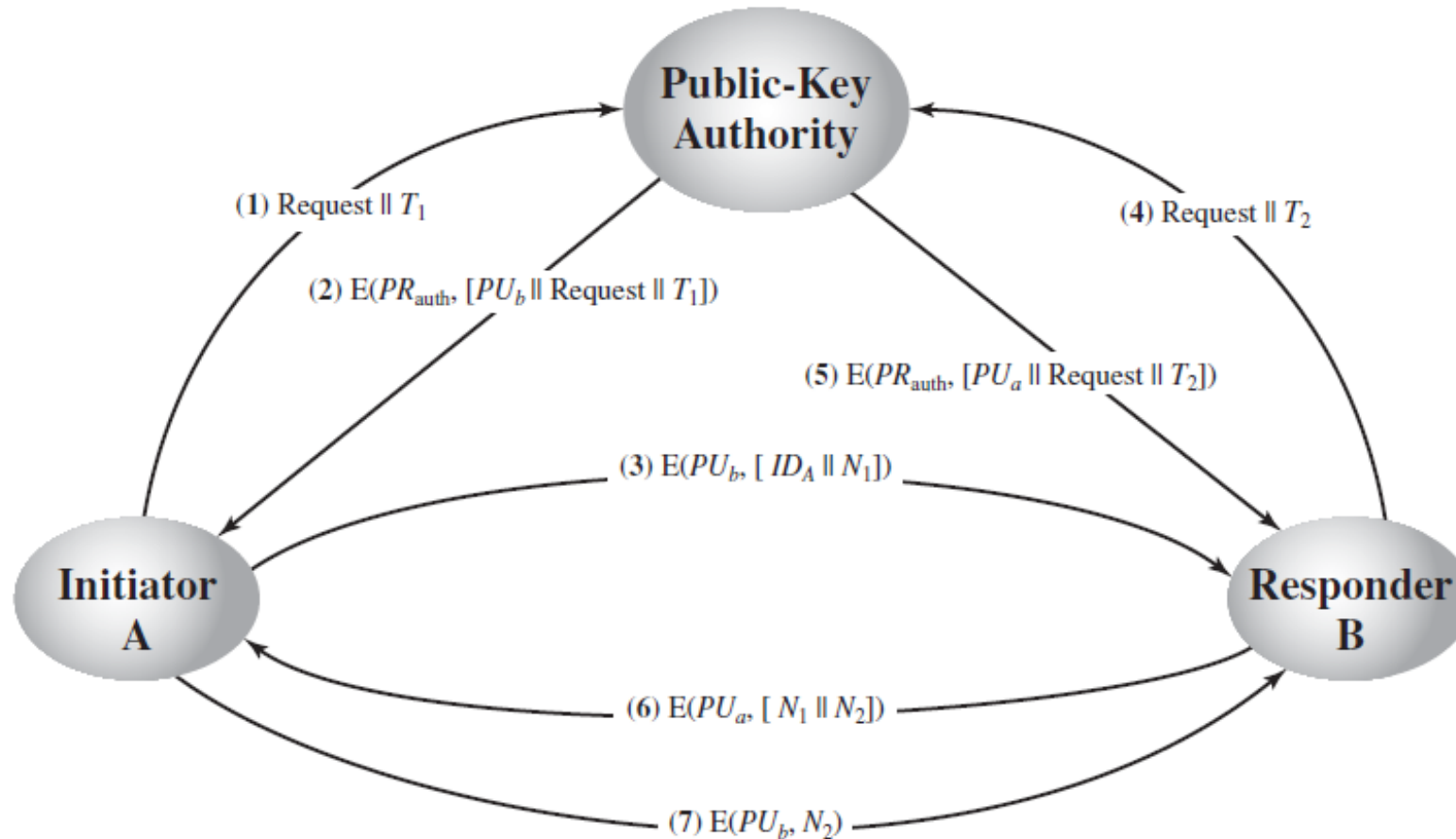
3. A stores B's public key and also uses it to encrypt a message to B containing an identifier of A (*IDA*) and a nonce (*N1*), which is used to identify this transaction uniquely.

4, 5. B retrieves A's public key from the authority in the same manner as A retrieved B's public key. At this point, public keys have been securely delivered to A and B, and they may begin their protected exchange. However, two additional steps are desirable:

6. B sends a message to A encrypted with *PUa* and containing A's nonce as (*N1*) well as a new nonce generated by B (*N2*) . Because only B could have decrypted message (3), the presence of *N1* in message (6) assures A that the correspondent is B.

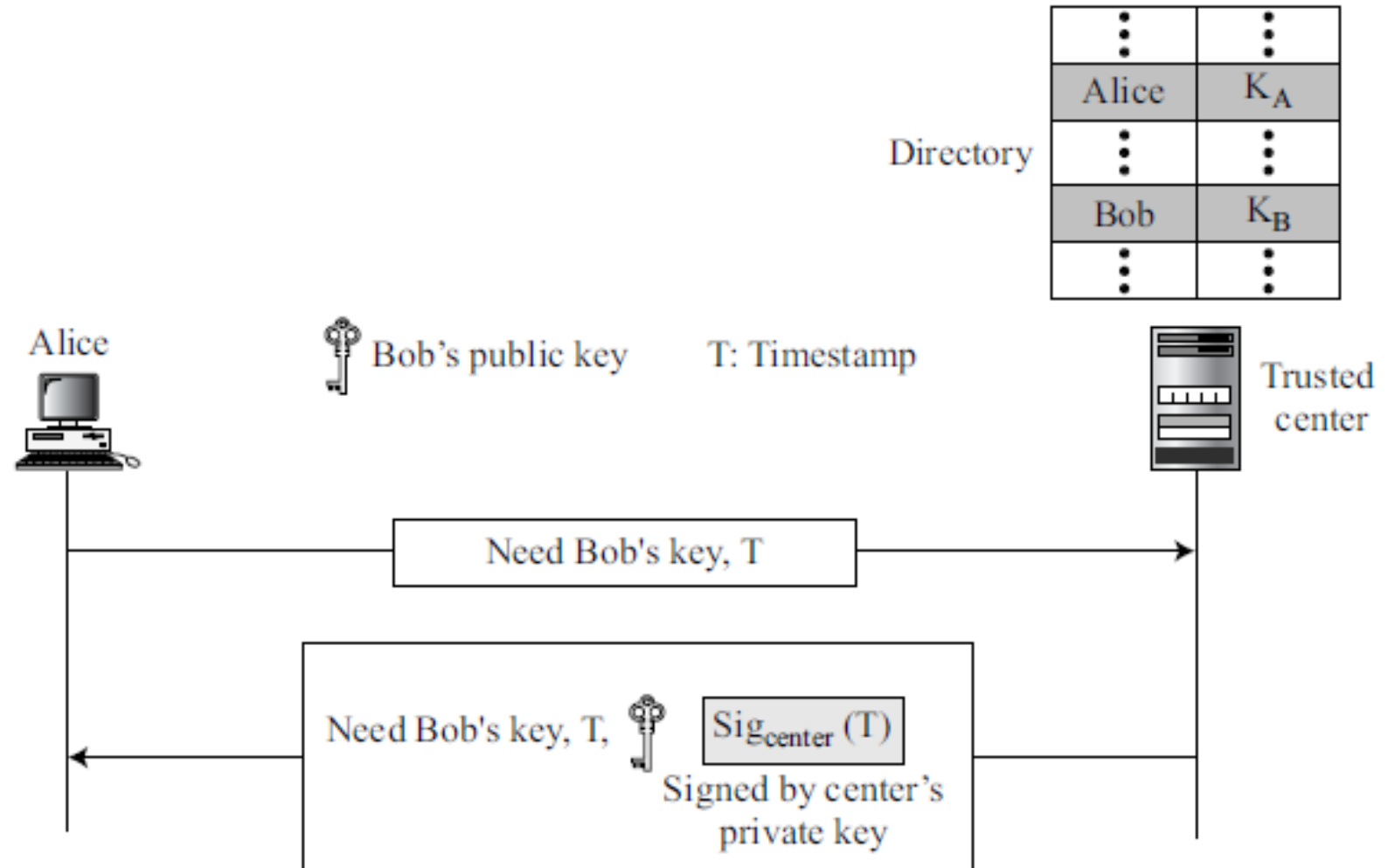
7. A returns , *N2* , which is encrypted using B's public key, to assure B that its correspondent is A.

Public-Key Distribution Scenario



- a total of seven messages are required.
- the initial four messages need be used only infrequently because both A and B can save the other's public key for future use—a technique known as caching.
- Periodically, a user should request fresh copies of the public keys of its correspondents to ensure currency.

Controlled trusted center



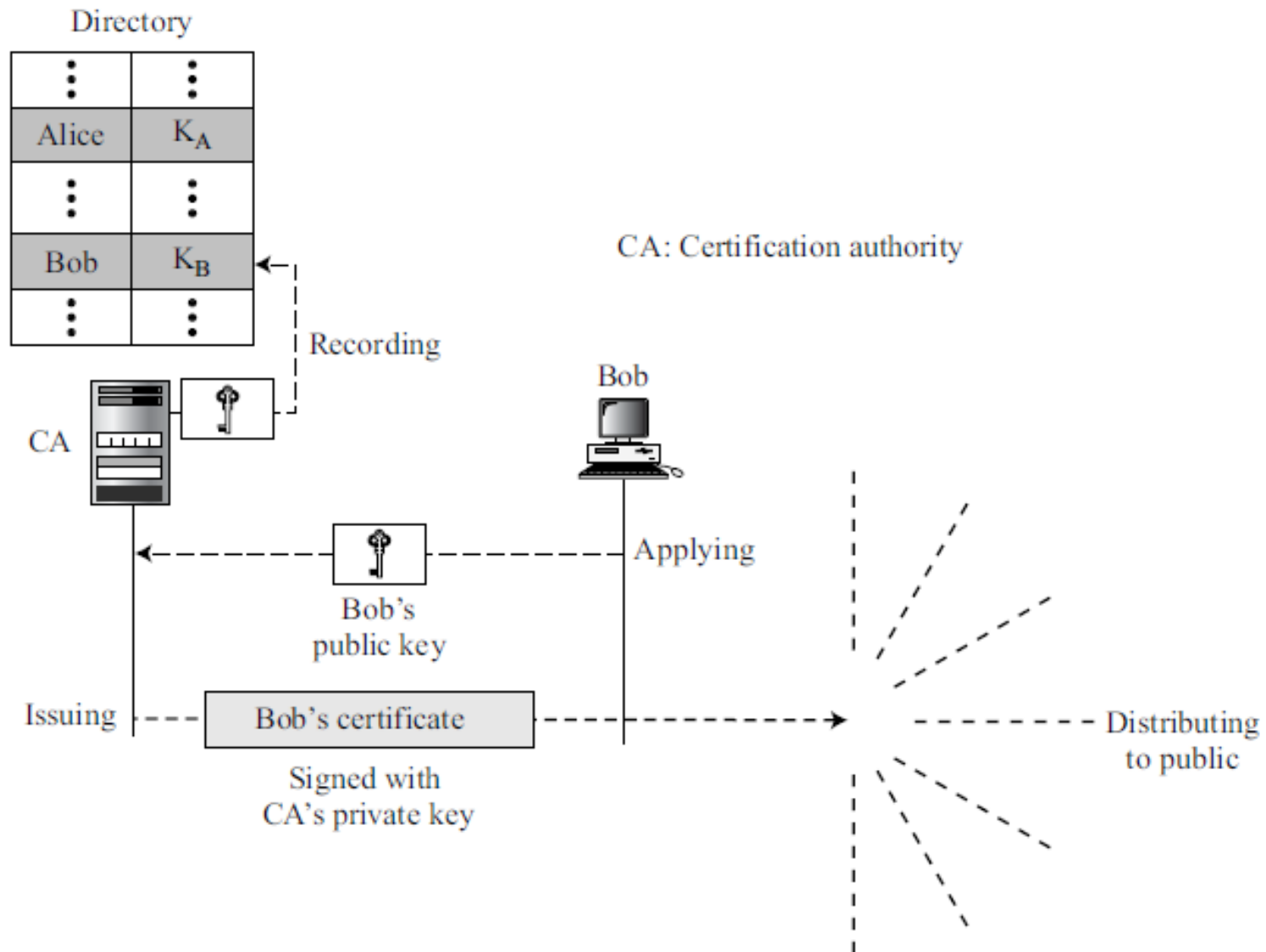
Public-Key Certificates

- The public-key authority could be somewhat of a bottleneck in the system, for a user must appeal to the authority for a public key for every other user that it wishes to contact.
- As before, the directory of names and public keys maintained by the authority is vulnerable to tampering
- An alternative approach, is to use **certificates** that can be used by participants to exchange keys without contacting a public-key authority, in a way that is as reliable as if the keys were obtained directly from a public-key authority.
- In essence, a certificate consists of a public key, an identifier of the key owner, and the whole block signed by a trusted third party.

Public-Key Certificates Contd...

- Typically, the third party is a certificate authority, such as a government agency or a financial institution, that is trusted by the user community.
- A user can present his or her public key to the authority in a secure manner and obtain a certificate.
- The user can then publish the certificate.
- Anyone needing this user's public key can obtain the certificate and verify that it is valid by way of the attached trusted signature.
- A participant can also convey its key information to another by transmitting its certificate. Other participants can verify that the certificate was created by the authority

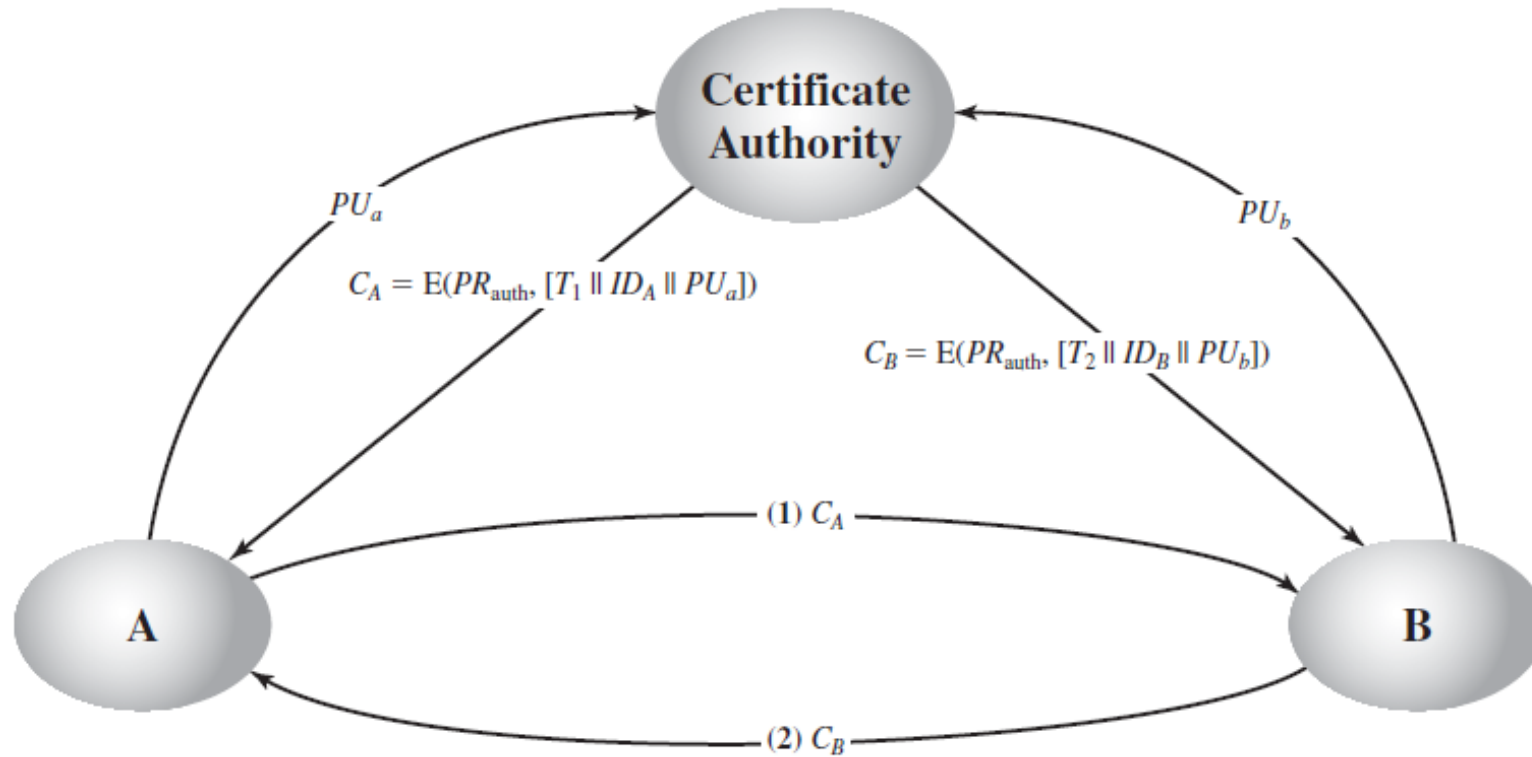
Certification authority



Requirements

1. Any participant can read a certificate to determine the name and public key of the certificate's owner.
 2. Any participant can verify that the certificate originated from the certificate authority and is not counterfeit.
 3. Only the certificate authority can create and update certificates.
- Additional Requirement
- Any participant can verify the currency of the certificate.

Exchange of Public-Key Certificates



- Each participant applies to the certificate authority, supplying a public key and requesting a certificate
- Application must be in person or by some form of secure authenticated communication.
- For participant A, the authority provides a certificate of the form
$$CA = E(PR_{auth}, [T \parallel ID_A \parallel PU_a])$$
- where PR_{auth} is the private key used by the authority and T is a timestamp.

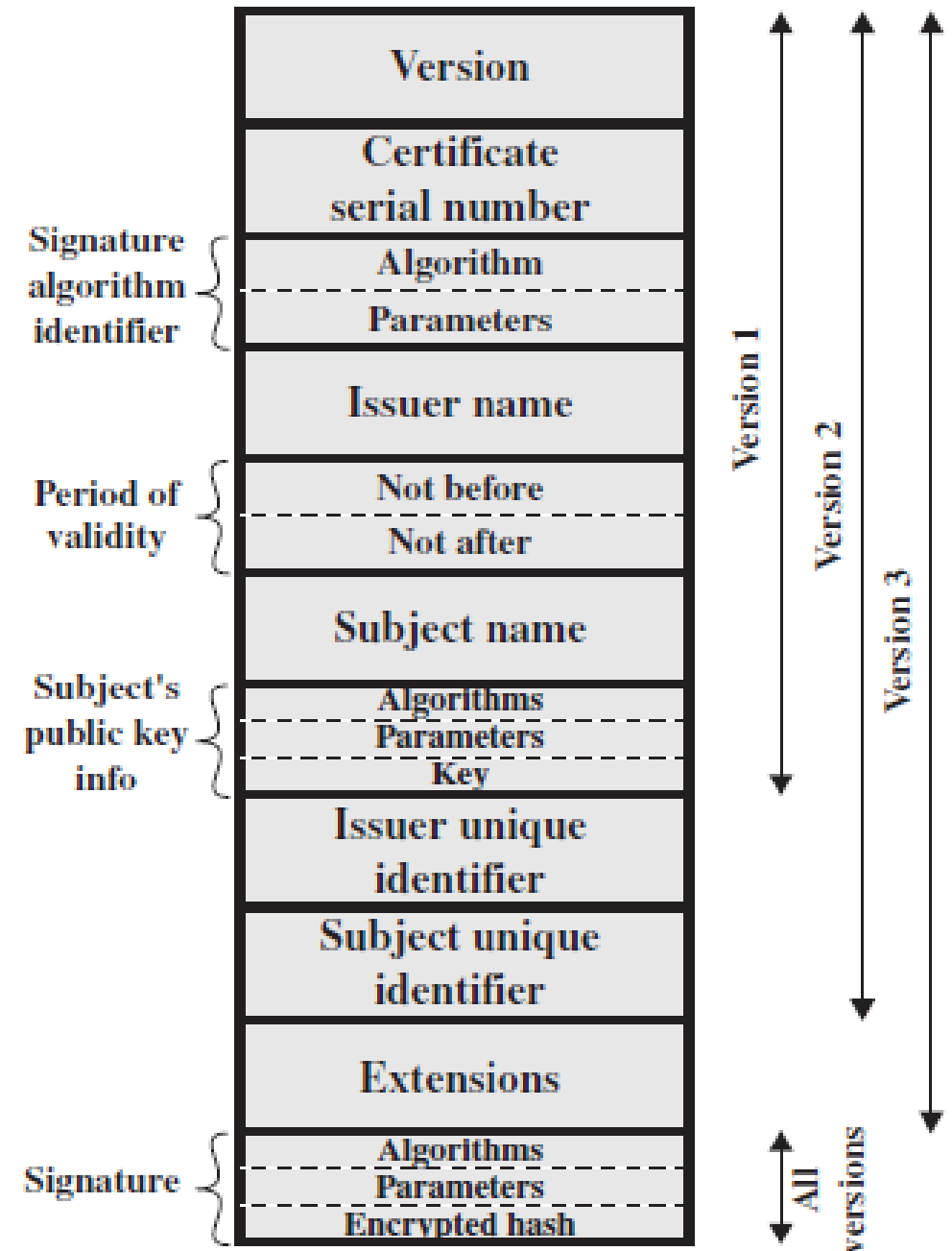
- A may then pass this certificate on to any other participant, who reads and verifies the certificate as follows:

$$D(PU_{\text{auth}}, C_A) = D(PU_{\text{auth}}, E(PR_{\text{auth}}, [T || ID_A || PU_a])) = (T || ID_A || PU_a)$$

- The recipient uses the authority's public key, PU_{auth} , to decrypt the certificate.
- Because the certificate is readable only using the authority's public key, this verifies that the certificate came from the certificate authority.
- The elements ID_A and PU_a provide the recipient with the name and public key of the certificate's holder.
- The timestamp T validates the currency of the certificate.
- The timestamp counters the following scenario.
 - A's private key is learned by an adversary.
 - A generates a new private/public key pair and applies to the certificate authority for a new certificate. Meanwhile, the adversary replays the old certificate to B. If B then encrypts messages using the compromised old public key, the adversary can read those messages.

X.509

- Each certificate may have a different format.
- One certificate may have the public key in one format and another in a different format.
- The public key may be on the first line in one certificate, and on the third line in another.
- Anything that needs to be used universally must have a universal format.
- To remove this side effect, the ITU has designed X.509, a recommendation that has been accepted by the Internet with some changes.
- X.509 is a way to describe the certificate in a structured way.
- It uses a well-known protocol called ASN.1 (Abstract Syntax Notation 1) that defines fields familiar to C programmers.



- **Version:** Differentiates among successive versions of the certificate format; the default is version 1. If the *issuer unique identifier* or *subject unique identifier* are present, the value must be version 2. If one or more extensions are present, the version must be version 3.
- **Serial number:** An integer value unique within the issuing CA that is unambiguously associated with this certificate.
- **Signature algorithm identifier:** The algorithm used to sign the certificate together with any associated parameters. Because this information is repeated in the signature field at the end of the certificate, this field has little, if any, utility.
- **Issuer name:** X.500 is the name of the CA that created and signed this certificate.
- **Period of validity:** Consists of two dates: the first and last on which the certificate is valid.

- **Subject name:** The name of the user to whom this certificate refers. That is, this certificate certifies the public key of the subject who holds the corresponding private key.
- **Subject's public-key information:** The public key of the subject, plus an identifier of the algorithm for which this key is to be used, together with any associated parameters.
- **Issuer unique identifier:** An optional-bit string field used to identify uniquely the issuing CA in the event the X.500 name has been reused for different entities
- **Subject unique identifier:** An optional-bit string field used to identify uniquely the subject in the event the X.500 name has been reused for different entities.
- **Extensions:** A set of one or more extension fields. Extensions were added in version 3 and are discussed later in this section.
- **Signature:** Covers all of the other fields of the certificate; it contains the hash code of the other fields encrypted with the CA's private key. This field includes the signature algorithm identifier.

The standard uses the following notation to define a certificate:

$$CA \ll A \gg = CA \{V, SN, AI, CA, UCA, A, UA, A_p, T^A\}$$

where

$Y \ll X \gg$ = the certificate of user X issued by certification authority Y

$Y \{I\}$ = the signing of I by Y. It consists of I with an encrypted hash code appended

V = version of the certificate

SN = serial number of the certificate

AI = identifier of the algorithm used to sign the certificate

CA = name of certificate authority

UCA = optional unique identifier of the CA

A = name of user A

UA = optional unique identifier of the user A

A_p = public key of user A

T^A = period of validity of the certificate

The CA signs the certificate with its private key. If the corresponding public key is known to a user, then that user can verify that a certificate signed by the CA is valid

OBTAINING A USER'S CERTIFICATE

User certificates generated by a CA have the following characteristics:

- Any user with access to the public key of the CA can verify the user public key that was certified.
- No party other than the certification authority can modify the certificate without this being detected.

Because certificates are unforgeable, they can be placed in a directory without the need for the directory to make special efforts to protect them.

- If all users subscribe to the same CA, then there is a common trust of that CA.
- All user certificates can be placed in the directory for access by all users.
- In addition, a user can transmit his or her certificate directly to other users.
- once B is in possession of A's certificate, B has confidence that messages it encrypts with A's public key will be secure from eavesdropping and that messages signed with A's private key are unforgeable.
- If there is a large community of users, it may not be practical for all users to subscribe to the same CA.
- Because it is the CA that signs certificates, each participating user must have a copy of the CA's own public key to verify signatures.
- This public key must be provided to each user in an absolutely secure (with respect to integrity and authenticity) way
- Thus, with many users, it may be more practical for there to be a number of CAs, each of which securely provides its public key to some fraction of the users.

- Suppose that A has obtained a certificate from certification authority X1 and B has obtained a certificate from CA X2
- If A does not securely know the public key of X2, then B's certificate, issued by X2, is useless to A.
- A can read B's certificate, but A cannot verify the signature.
- However, if the two CAs have securely exchanged their own public keys, the following procedure will enable A to obtain B's public key.
- **Step 1** A obtains from the directory the certificate of X2 signed by X1. Because A securely knows X1's public key, A can obtain X2's public key from its certificate and verify it by means of X1's signature on the certificate.
- **Step 2** A then goes back to the directory and obtains the certificate of B signed by X2. Because A now has a trusted copy of X2's public key, A can verify the signature and securely obtain B's public key.

- A has used a chain of certificates to obtain B's public key. In the notation of X.509, this chain is expressed as

$$X1\langle\langle X2\rangle\rangle X2\langle\langle B\rangle\rangle$$

In the same fashion, B can obtain A's public key with the reverse chain:

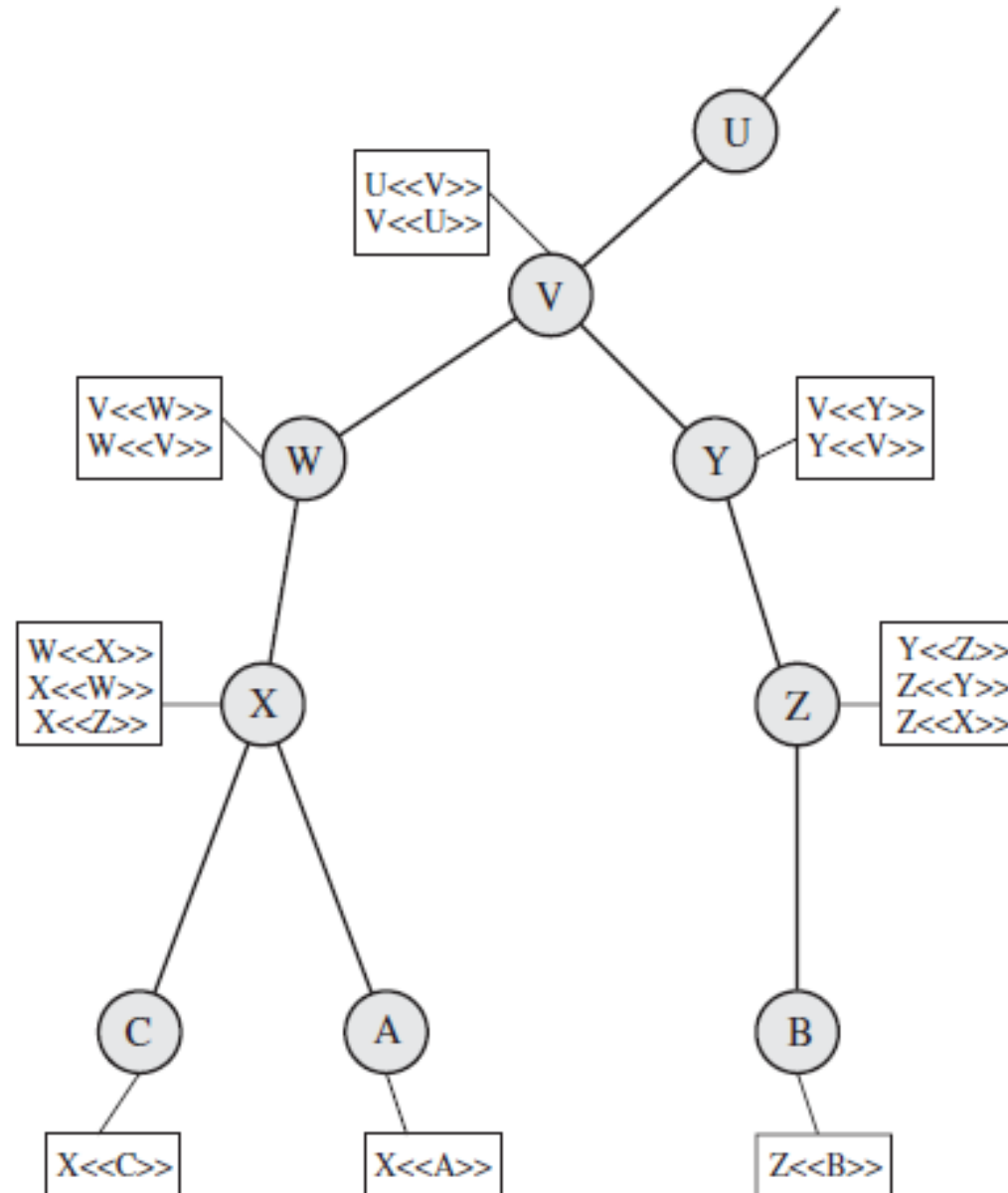
$$X2\langle\langle X1\rangle\rangle X1\langle\langle A\rangle\rangle$$

- This scheme need not be limited to a chain of two certificates.
- An arbitrarily long path of CAs can be followed to produce a chain.
- A chain with elements would be expressed as

$$X1\langle\langle X2\rangle\rangle X2\langle\langle X3\rangle\rangle \dots XN\langle\langle B\rangle\rangle$$

- In this case, each pair of CAs in the chain (X_i, X_{i+1}) must have created certificates for each other.
- All these certificates of CAs by CAs need to appear in the directory, and the user needs to know how they are linked to follow a path to another user's public key certificate.
- X.509 suggests that CAs be arranged in a hierarchy so that navigation is straightforward

X.509 Hierarchy: A Hypothetical Example



A Hypothetical Example

- The connected circles indicate the hierarchical relationship among the CAs;
- the associated boxes indicate certificates maintained in the directory for each CA entry.
- The directory entry for each CA includes two types of certificates:
- **Forward certificates:** Certificates of X generated by other CAs
- **Reverse certificates:** Certificates generated by X that are the certificates of other CAs
- user A can acquire the following certificates from the directory to establish a certification path to B:

$X \ll W \gg W \ll V \gg V \ll Y \gg Y \ll Z \gg Z \ll B \gg$

- When A has obtained these certificates, it can unwrap the certification path in sequence to recover a trusted copy of B's public key. Using this public key, A can send encrypted messages to B. If A wishes to receive encrypted messages back from B, or to sign messages sent to B, then B will require A's public key, which can be obtained from the following certification path:

$Z \ll Y \gg Y \ll V \gg V \ll W \gg W \ll X \gg X \ll A \gg$

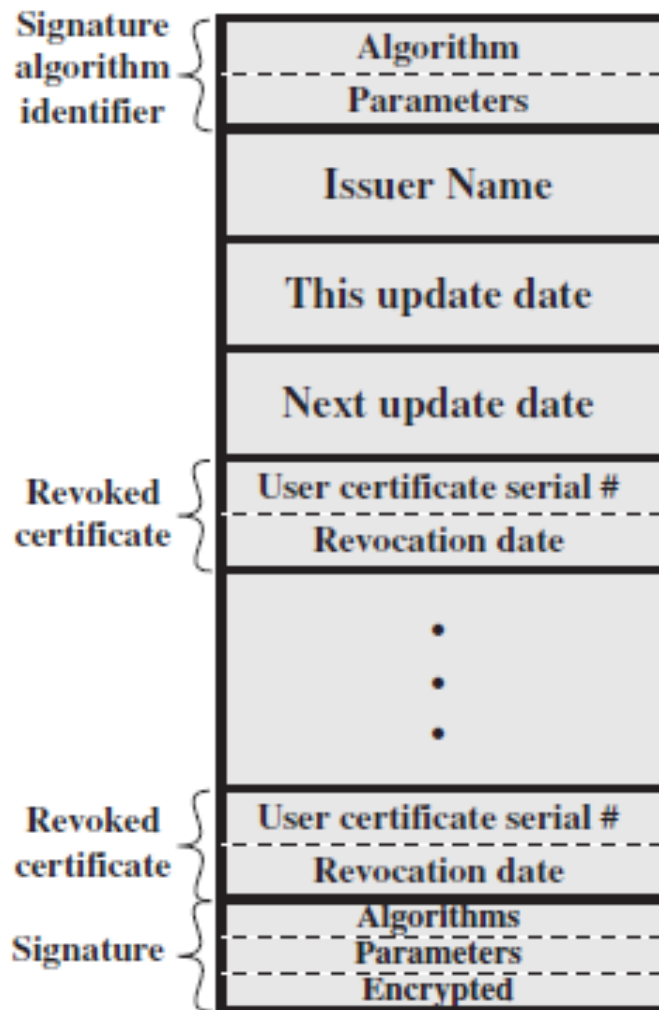
- B can obtain this set of certificates from the directory, or A can provide them as part of its initial message to B

REVOCATION OF CERTIFICATES

- each certificate includes a period of validity
- a new certificate is issued just before the expiration of the old one.
- In addition, it may be desirable on occasion to revoke a certificate before it expires, for one of the following reasons
 1. The user's private key is assumed to be compromised.
 2. The user is no longer certified by this CA. Reasons for this include that the subject's name has changed, the certificate is superseded, or the certificate was not issued in conformance with the CA's policies.
 3. The CA's certificate is assumed to be compromised.

Certificate revocation list (CRL)

- Each CA must maintain a list consisting of all revoked but not expired certificates issued by that CA, including both those issued to users and to other CAs.
- These lists should also be posted on the directory.
- Each certificate revocation list (CRL) posted to the directory is signed by the issuer and includes the issuer's name, the date the list was created, the date the next CRL is scheduled to be issued, and an entry for each revoked certificate.
- Each entry consists of the serial number of a certificate and revocation date for that certificate.
- Because serial numbers are unique within a CA, the serial number is sufficient to identify the certificate.
- When a user receives a certificate in a message, the user must determine whether the certificate has been revoked.
- The user could check the directory each time a certificate is received.
- To avoid the delays (and possible costs) associated with directory searches, it is likely that the user would maintain a local cache of certificates and lists of revoked certificates.



A certificate revocation list has the following fields:

- **Signature algorithm ID.** This field is the same as the one in the certificate.
- **Issuer name.** This field is the same as the one in the certificate.
- **This update date.** This field defines when the list is released.
- **Next update date.** This field defines the next date when the new list will be released.
- **Revoked certificate.** This is a repeated list of all unexpired certificates that have been revoked. Each list contains two sections:
 - user certificate serial number
 - revocation date.
- **Signature.** This field is the same as the one in the certificate list.

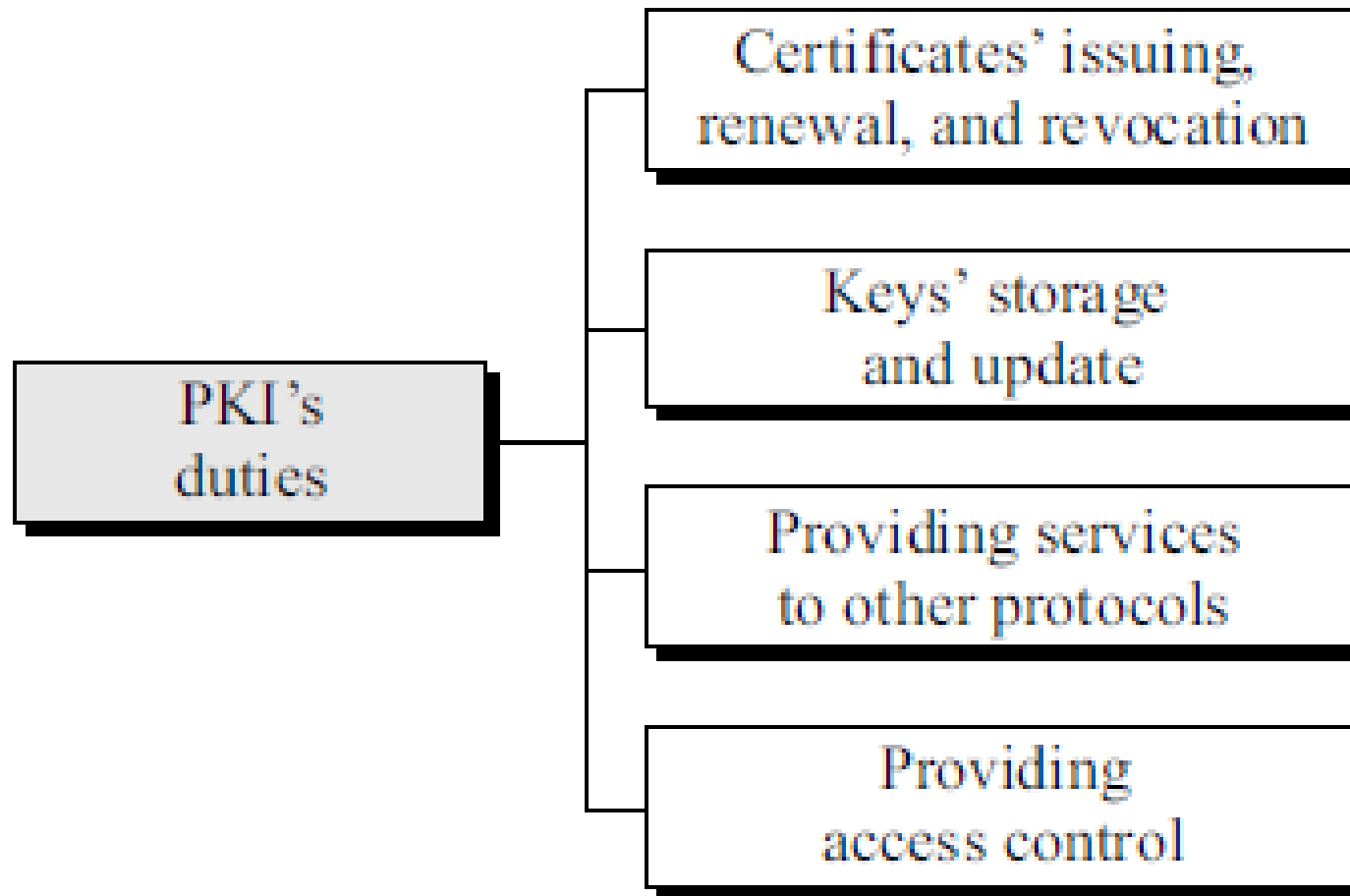
PUBLIC-KEY INFRASTRUCTURE

- Public-Key Infrastructure (PKI) is a model for creating, distributing, and revoking certificates based on the X.509

Several duties have been defined for a PKI.

- Certificates' issuing, renewal, and revocation.
 - These are duties defined in the X.509. Because the PKIX is based on X.509, it needs to handle all duties related to certificates.
- Keys' storage and update.
 - A PKI should be a storage place for private keys of those members that need to hold their private keys somewhere safe
 - In addition, a PKI is responsible for updating these keys on members' demands.
- Providing services to other protocols.
 - some Internet security protocols, such as IPSec and TLS, are relying on the services by a PKI.
- Providing access control. A PKI can provide different levels of access to the information stored in its database.
 - For example, an organization PKI may provide access to the whole database for the top management, but limited access for employees.

Some duties of a PKI

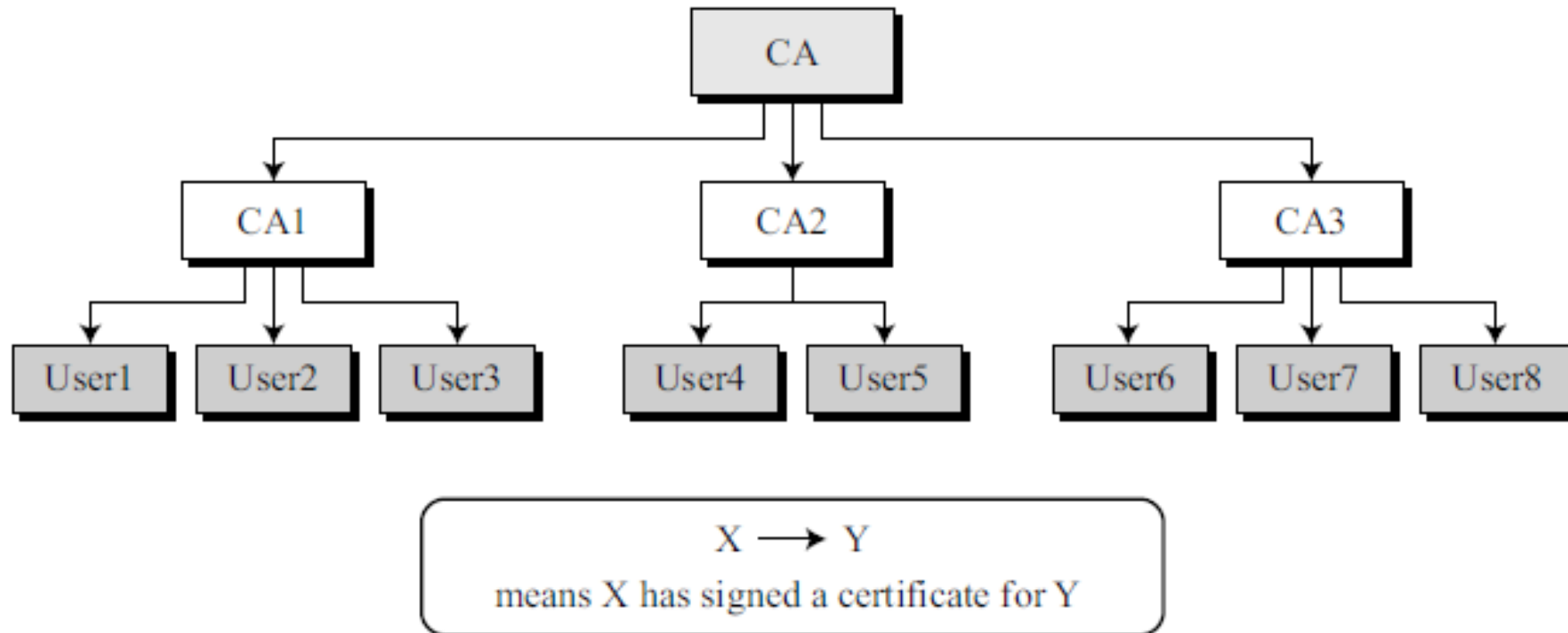


Trust Model

- It is not possible to have just one CA issuing all certificates for all users in the world.
- There should be many CAs, each responsible for creating, storing, issuing, and revoking a limited number of certificates.
- The trust model defines rules that specify how a user can verify a certificate received from a CA.

Hierarchical Model

- In this model, there is a tree-type structure with a root CA.
- The root CA has a self-signed, self-issued certificate; it needs to be trusted by other CAs and users for the system to work.
- Figure shows a trust model of this kind with three hierarchical levels. The number of levels can be more than three in a real situation.



- The figure shows that the CA (the root) has signed certificates for CA1, CA2, and CA3;
- CA1 has signed certificates for User1, User2, and User3; and so on. PKI uses the following notation to mean the certificate issued by authority X for entity Y

$$X\ll Y \gg$$

Show how User1, knowing only the public key of the CA (the root), can obtain a verified copy of User3's public key.

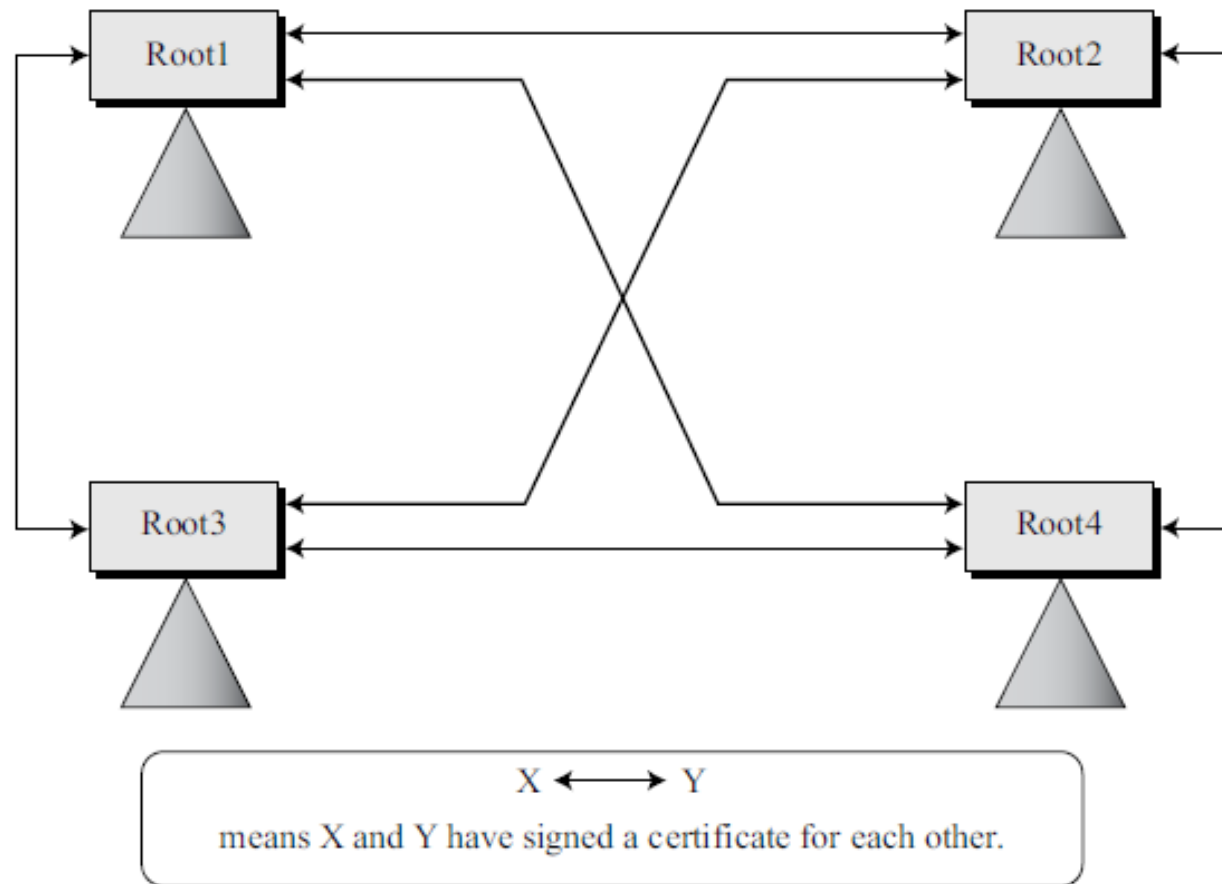
Solution

- User3 sends a chain of certificates, $CA\ll CA1 \gg$ and $CA1\ll User3 \gg$, to User1.
 - a. User1 validates $CA\ll CA1 \gg$ using the public key of CA.
 - b. User1 extracts the public key of CA1 from $CA\ll CA1 \gg$.
 - c. User1 validates $CA1\ll User3 \gg$ using the public key of CA1.
 - d. User1 extracts the public key of User 3 from $CA1\ll User3 \gg$.

Mesh Model

- The hierarchical model may work for an organization or a small community.
- A larger community may need several hierarchical structures connected together.
- One method is to use a mesh model to connect the roots together.
- In this model, each root is connected to every other root.
- Figure shows that the mesh structure connects only roots together;
- each root has its own hierarchical structure, shown by a triangle.
- The certifications between the roots are cross-certificates;
- each root certifies all other roots, which means there are $N(N - 1)$ certificates.
- In Figure there are 4 nodes, so we need $4 \times 3 = 12$ certificates.
- Note that each double-arrow line represents two certificates.

Mesh Model



Alice is under the authority Root1; Bob is under the authority Root4.
Show how Alice can obtain Bob's verified public key.

Solution

- Bob sends a chain of certificates from Root4 to Bob. Alice looks at the directory of Root1 to find Root1<<Root1>> and Root1<< Root4>> certificates. Using the process shown Alice can verify Bob's public key.

Public Key Infrastructure X.509 (PKIX)

- The Internet Engineering Task Force (IETF) Public Key Infrastructure X.509 (PKIX) working group has been the driving force behind setting up a formal (and generic) model based on X.509 that is
- suitable for deploying a certificate-based architecture on the Internet
- Figure shows the interrelationship among the key elements of the PKIX model.
- These elements a
- **End entity:** A generic term used to denote end users, devices (e.g., servers, routers), or any other entity that can be identified in the subject field of a public key certificate. End entities typically consume and/or support PKI-related services.
- **Certification authority (CA):** The issuer of certificates and (usually) certificate revocation lists (CRLs). It may also support a variety of administrative functions, although these are often delegated to one or more Registration Authorities

- **Registration authority (RA):** An optional component that can assume a number of administrative functions from the CA. The RA is often associated with the end entity registration process but can assist in a number of other areas as well.
- **CRL issuer:** An optional component that a CA can delegate to publish CRLs.
- **Repository:** A generic term used to denote any method for storing certificates and CRLs so that they can be retrieved by end entities.

PKIX Management Functions

- **Registration:** This is the process whereby a user first makes itself known to a CA (directly or through an RA), prior to that CA issuing a certificate or certificates for that user. Registration begins the process of enrolling in a PKI. Registration usually involves some offline or online procedure for mutual authentication. Typically, the end entity is issued one or more shared secret keys used for subsequent authentication.
- **Initialization:** Before a client system can operate securely, it is necessary to install key materials that have the appropriate relationship with keys stored elsewhere in the infrastructure. For example, the client needs to be securely initialized with the public key and other assured information of the trusted CA(s), to be used in validating certificate paths.
- **Certification:** This is the process in which a CA issues a certificate for a user's public key, returns that certificate to the user's client system, and/or posts that certificate in a repository.

- **Key pair recovery:** Key pairs can be used to support digital signature creation and verification, encryption and decryption, or both. When a key pair is used for encryption/decryption, it is important to provide a mechanism to recover the necessary decryption keys when normal access to the keying material is no longer possible, otherwise it will not be possible to recover the encrypted data. Loss of access to the decryption key can result from forgotten passwords/PINs, corrupted disk drives, damage to hardware tokens, and so on. Key pair recovery allows end entities to restore their encryption/decryption key pair from an authorized key backup facility (typically, the CA that issued the end entity's certificate).
- **Key pair update:** All key pairs need to be updated regularly (i.e., replaced with a new key pair) and new certificates issued. Update is required when the certificate lifetime expires and as a result of certificate revocation.
- **Revocation request:** An authorized person advises a CA of an abnormal situation requiring certificate revocation. Reasons for revocation include private key compromise, change in affiliation, and name change.
- **Cross certification:** Two CAs exchange information used in establishing a cross-certificate. A cross-certificate is a certificate issued by one CA to another CA that contains a CA signature key used for issuing certificates.