

MODULE 3

| Chapters | |
|--------------------------------------|--|
| 1.Key Management | 2.Authentication-I |
| 3.Authentication-II | 4.IPsec-security at the network layer |
| 5.Security at transport layer | |

CHAPTER 1 Key Management

3.1 INTRODUCTION

- Key management is related to the *generation, storage, distribution, and backup of keys*.
- The focus is on the management of *public key—private key pairs*.
- The public key—private key pairs are used for *encryption/decryption, signature generation/verification, and for authentication*.
- To encrypt a session key for use in communication between A and B, A needs to know B's public key.
- The key issue here is "**How does A know B's public key?**"
- **Possibility 1:**
 - ✓ A may frequently communicate with **B** in a secure manner, so she may already have **B's public key**.
 - ✓ First, *B must have securely communicated his public key to A* at some point in the past. A actually receives B's public key and not a public key from someone posing as B.
 - ✓ If at *any time B's private key is compromised*, the confidentiality of messages from A to B using the corresponding public key can no longer be guaranteed.
 - ✓ An individual, with the compromised private key, can decrypt messages encrypted with the old public key.
- **Possibility 2:**
 - ✓ Every entity's public key is securely maintained in a **centralized directory**.
 - ✓ Suppose A wishes to securely communicate with an e-commerce website, B-Mart.
 - ✓ All she has to do to obtain B-Mart's public key is to query the directory for it.
 - ✓ The question here is "**Who would take the responsibility for maintaining such a directory?**"
 - ✓ There are huge scalability problems associated with such a directory, spoofing and denial of service attacks, the non-uniqueness of names.
- **Possibility 3:**
 - ✓ A receives a document signed by a trusted source, C, containing B's public key.

3.2 DIGITAL CERTIFICATES

3.2.1 Certificate Types

- A digital certificate is a signed document used to *bind a public key to the identity of a person.*
- Example such as An individual's identity could be his/her name, national identification number, e-mail .or postal address, employer, etc. or some combination of these.
- **CA:**The entity that issues certificates is **a trusted entity called a certification Authority (CA)certificate authority.**
- Certificates may be issued to individuals, to organizations, or even to servers.
- The most basic type of certificate may be applied for through regular e-mail with the applicant stating his/her public key, name, e-mail address, etc.
- In this case, the CA requires no credentials from the applicant.
- It simply assumes that the applicant is in possession of the (uncompromised) private key corresponding to the Public key contained in the application received via e-mail.
- The verifier of such a certificate should realize that the above certificates are "**Trust at your own risk certificates.**"
- To carry more weight, certificate issuance would require the CA to perform identity verification of the applicant.
- The CA may have to obtain and verify several details of the applicant this task would be delegated by the *CA to the registration Authority (RA)*

3.2.2 X.509 Digital Certificate Format

- X.509 is an ITU standard specifying the format for **public key certificates.**
- The fields of an X.509 certificate together with their meaning are as follows:
 1. **Certificate Serial Number and Version :**Each certificate issued by a given CA will have a unique number.
 2. **Issuer information:** The distinguished name of an entity includes his/her/its "common name," e-mail address, organization, country, etc.
 3. **Certificate signature and associated signing algorithm information:** It is necessary to verify the authenticity of the certificate. For this purpose, it is signed by the issuer. So, the certificate should include the issuer's digital signature and also the algorithm used for signing the certificate.
 4. **Validity period:** There are two date fields that specify the *start date and end date* between which the certificate is valid.
 5. **Subject information :**This includes the distinguished name of the certificate's subject or owner.
 - For example, if a customer intends to communicate with an e-commerce web server at www.B-Mart.com, then the customer's browser will request B-Mart's certificate.

- Client-side software will check whether the "Common Name" in B-Mart's certificate tallies with B-Mart's domain name.
 - Other information, such as the subject's country, state, and organization, may be included.
6. **Subject's public key information:** The public key, the public key algorithm (e.g., RSA or DSA), and the public key parameters (modulus in the case of RSA and modulus + generator in case of Diffie-Hellman).

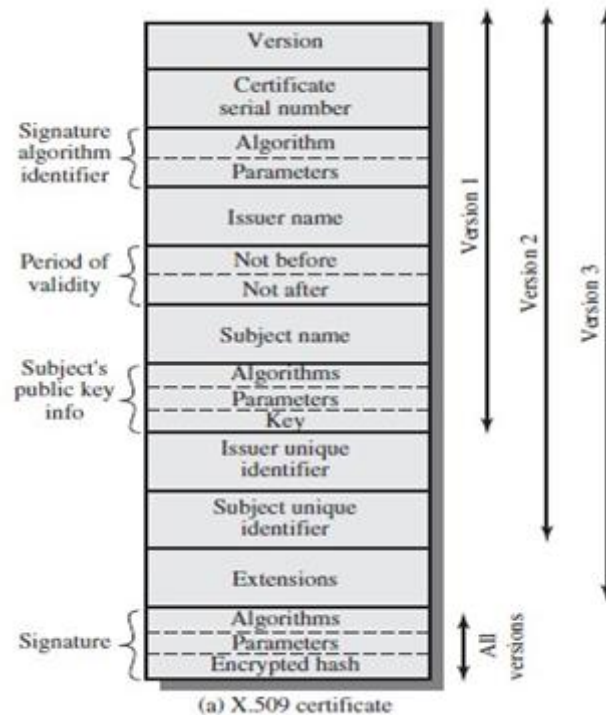


Figure 3.1 A digital certificate

3.2.3 Digital Certificates in Action

- Assume that A needs to securely *transmit a session key* to B.
- So, she encrypts it with B's public key.
- A will need to retrieve the public key from B's certificate.
- A may already have B's certificate or she may send a message to B requesting it.
- There are a number of checks that A will have to perform on B's certificate prior to using B's public key.
 - ✓ Is this indeed B's certificate?
 - ✓ This can be determined by checking whether the certificate contains B's name. But the "common name" field alone may be inadequate (since there are probably many John Browns, for example).

- ✓ It may be necessary to check other fields in the certificate such as the subject's web page URL or e-mail address.
- ✓ A should check if the certificate is still valid. Since the validity period is contained in the certificate, this is easily done.
- ✓ Finally, the certificate must be signed by a CA or RA.
- ✓ A should verify the signature contained in the certificate.
- ✓ A requires the CA's public key for signature verification.
- ✓ The CA may be globally known or may be known to the community that A and B belong.
- ✓ In this case A has access to the CA's public key.

3.3 PUBLIC KEY INFRASTRUCTURE

3.3.1 FUNCTIONS OF A PKI

- A public key infrastructure includes the CA's ,the **physical infrastructure**(encryption technologies, hardware etc.), and the formulation and enforcement of policies/procedure.
- It includes the following services:
 - ✓ **Certificate creation,issuance,storage and archival**
 - ✓ **Key generation and key escrow**
 - ✓ **Certificate/key updation**
 - ✓ **Certificate revocation**
- There are crucial differences in the support required for private keys used for decryption versus those used for signing.
- In the case of encryption/decryption, it is often necessary to have ***a back-up of the decryption key.***
- If not, an employee who loses his decryption key will be unable to decrypt the archives of sensitive data he may have accumulated.
- For this reason, the PKI within an organization, for example, might hold the private keys in **escrow, i.e., they may be securely backed up and made available to the owner or to a trusted authority** (such as a law enforcement agency) under special circumstances.
- On the other hand, there ***is no need to back up a private key used for digital signing.***
- If such a key is lost, the owner could inform the CA or PKI administrator (within an organization).
- He/she could then obtain a new signing key and receive a new certificate carrying the corresponding public key.
- An important function of the PKI is to provide a safe archival facility for all issued certificates.

3.3.2 PKI Architectures

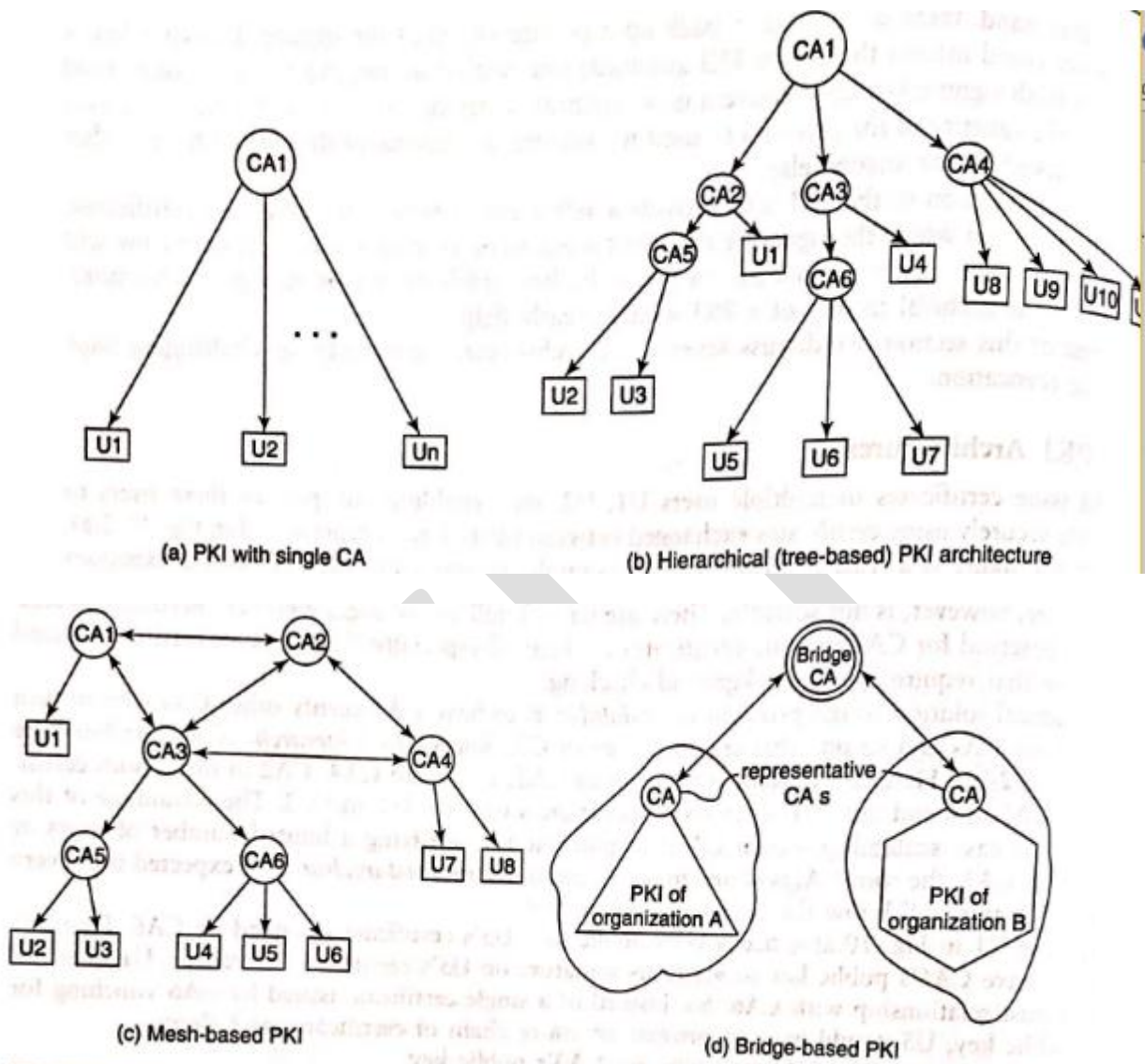


Figure: PKI architecture

1. PKI with single CA:

- CA1 could issue certificates to multiple users U1, U2, etc., enabling any pair of these users to communicate securely using certificates exchanged between them.
- This is represented in above Fig.(a).
- Each arc in the figure is a trust relationship.
- For example, the arc from the CA1 to U2 expresses the fact that CA1 vouches for U2's public key in the certificate issued by the CA1 to U2. Such an architecture, however, is not scalable.
- There are tens of millions of users who may need certificates. It is not practical for CA1 to issue certificates to all.

2. Hierarchical (tree-based PKI architecture)

- A practical solution to the problem of scalability is to have CA1 certify other CAs who in turn certify other CAs and so on.
- This creates a tree of CAs known as a **hierarchical PKI architecture** [see above Fig.(b)].
- Here, CA1 issues certificates to CA2, CA3, and CA4.
- CA2 in turn issues certificates to CA5 and end user U1.
- CA5 issues certificates to users U2 and U3.
- The advantage of this approach is easy scalability — each CA is responsible for certifying a limited number of users or other CAs.
- CA1, the root CA, is sometimes referred to as the trust anchor.
- every node in the tree will know the root CA's public key.
- Suppose U1 in Fig.(b) needs U5's public key.
- U5 would have to provide an entire chain of certificates as follows:
- **(1) Certificate signed by CA1 vouching for CA3's public key**
- **(2) Certificate signed by CA3 vouching for CA6's public key**
- **(3) Certificate signed by CA6 vouching for U5's public key**
- It is assumed that each node has a copy of the root's public key.
- So, upon receiving the above certificate chain, U1 can verify the signature on the first certificate using CA1's (the trust anchor's) public key.

3. Mesh based PKI

- A more dense web of trust is shown in Fig. (c) and is referred to as a **mesh-based PKI**. This could include mutually trusting CAs — CA1 trusting CA2 and CA2 trusting CA1 shown by a bidirectional arc between CA1 and CA2.
- In tree based PKI, there may be multiple trust paths between two users.
- Example there could be multiple trust paths between user 1 and user 7
 - Path 1: CA1, CA3, and CA 4
 - Path 2: CA1, CA2, and CA 4.

4. Bridge based PKI

- Another PKI architecture, referred to as **bridge-based PKI**, is motivated by the need for secure communications between organizations in a business partnership.
- Suppose that the partnering organizations already have their own PKIs.
- **A bridge CA is introduced that establishes a trust relationship with a representative CA from each organization.**
- This is accomplished by the bridge CA and the organizational representatives issuing certificates to each other.
- The representative CA is one that has a trust path to all (or at least most) of the users in that organization.

- Figure 10.2(d) shows a bridge CA that extends the web of trust between two existing organizational PKIs.

3.3.3 Certificate revocation

Revocation Scenarios

- The validity period of an X.509 certificate is always contained in the certificate.
- However, there are other reasons why a seemingly valid certificate may actually be invalid.

Scenario 1: The certificate subject, Prashant, was issued a certificate valid between Jan 01, 2010, and Dec 31, 2010. However, he quit the organization on April 1, 2010.

- Assume that Prashant's certificate is used for key exchange/authentication and that **he has made a copy of it**.
- The session key itself is then used to encrypt all messages in both directions for the duration of the ensuing session.
- Generally speaking, it is not legal for Prashant to act on behalf of his company beyond the date of his resignation. However, that is precisely what he could do when he attempts to establish official business communication with a customer of his company on say June 10, 2010.
- Based on the expiration date in Prashant's certificate, the customer would deduce that the certificate was valid.
- Moreover, Prashant would be able to authenticate himself or perform unauthorized decryption since he knows the *private key corresponding to the public key in his certificate*. Thus, Prashant might continue to do business on behalf of his company even after resigning.
- Based on Scenario 1, we need a mechanism to revoke a certificate issued by an organization to an employee when he leaves or changes roles.

Scenario 2:

- Consider a single chain in a PKI (Fig. 3.3).
- Suppose that the *private key of CA3* were compromised.
- An attacker with access to the *compromised private key* could then do the following:
 - Generate a *public key, private key pair (X, Y)*.
 - Create a certificate containing the public key X with subject name = U'.
 - Sign the above certificate using the compromised private key of CA3.
- The attacker has thus created a fictitious entity U', masquerading as a legitimate subject, U (see Fig. 3.3).
- Now the attacker can forge the signature of U on any message by signing with the private key, Y.

- The attacker would provide a certificate chain of two certificates — the certificate issued by CA1 vouching for CA3's public key and the above certificate created by him.
- This chain is a valid trust path from the root CA to the subject U.
- Using the public key of CA1 and the certificate chain, the verifier would accept the fraudulent signature generated using Y as an authentic signature of U.
- Scenario 2 is that if a CA's private key is compromised, then any certificate issued by that CA is invalid and it should not be included in any trust path or certificate chain.

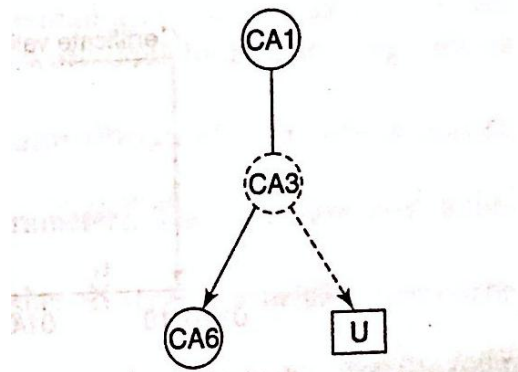


Figure 3.3 Revocation scenario 2

Handling Revocation

Solution 1:

- One possible solution to the problem of certificate revocation is to use an *on-line facility* that provides information on the *current status of digital certificates*.
- For this purpose, a *protocol called On-line Certificate Status Protocol (OCSP)* is employed.

Solution 2:

- Another proposed solution is to distribute lists of revoked certificates — *Certificate Revocation Lists (CRLs)*. The frequency of list updation is an important consideration.
- If CRLs are distributed, too frequently, they could consume considerable bandwidth.
- On the other hand, if they were distributed infrequently, information on recently revoked certificates may not reach those who need it in a timely fashion.

Solution 3

- Design a system wherein the signer requires the cooperation of a **Trusted Third Party (TTP)** in generating a signature.
- Both, the signer and the TTP have a part of the private key with neither party knowing the other part.
- To sign a document, the signer would contact the TTP.

- Before requesting to sign, the TTP could check whether the signer's certificate has been revoked and participate only if the signer's certificate has not been revoked.
- Indeed, the TTP may itself maintain **certificate revocation information**.
- The TTP may also act as a **timestamp authority** and certify the time at which the document is signed.
- This may be done, for example, by signing a value obtained by **concatenating a timestamp with the hash of the document**.

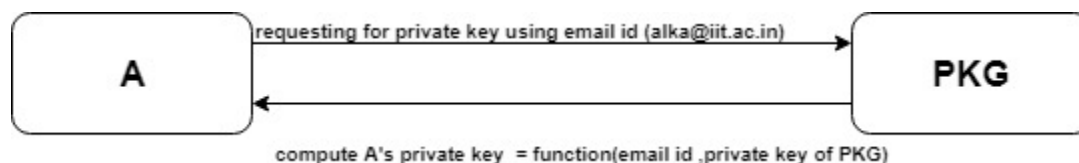
3.4 IDENTITY-BASED ENCRYPTION

3.4.1 Preliminaries

- The digital certificate is a verifiable way of communicating the public key of an entity.
- Certificates are transmitted along with messages for purposes such as *as authentication, signature verification, and encryption*.
- An alternative to digital certificates emerged in 1984 in the form of **Identity-based Encryption (IBE)**.
- Shamir's used a scheme wherein a person's public key could be computed as a function of that **person's unique credential such as his/her e-mail address**. Thus, anyone can reliably compute A's public key only knowing A's e-mail address, for example.
- IBE assumes the use of a **TTP called the Private Key Generator (PKG)**.

Here is how a generic IBE scheme works:

- The PKG has a **private key and associated public key parameters**. (K_{pr} , public key parameters)
- To obtain a private key, A informs the PKG that she wishes to receive a private key corresponding to her ID, say alka@iitb.ac.in
- The PKG makes sure that that the credential does indeed belong to A.
- The PKG also makes sure that this ID is universally unique, i.e., there is no other individual with the same credential (in this case alka@iitb.ac.in).
- If so, it generates a **private key for A**, which is a **function of her ID and the private key of the PKG**.
- The PKG then securely transmits the private key to A.
- **Disadvantage:** With knowledge of the PKG's public parameters and A's unique ID, anyone can compute A's public key



3.5 Bilinear mapping

- A bilinear mapping $B(x,y)$ maps any pair of elements from one given set to an element in a second set.
- The term bilinear follows from the following property mapping:

$$B(k_1 \times u_1 + k_2 \times u_2, v) = k_1 \times B(u_1, v) + k_2 \times B(u_2, v)$$

- Here u_1, u_2 and v are elements of the first set and k_1 and k_2 are integer constants.
- An example of dot product of vectors

Let $u = (2, 4, 1)$ and let $v = (5, 3, 2)$.

Then, $(2, 4, 1) \cdot (5, 3, 2)^T = 24$.

We next verify that the dot product is a bilinear operation.

Now let

$$u_1 = (2, 4, 5), u_2 = (7, 1, 2), k_1 = 3, k_2 = 4$$

So,

$$\begin{aligned} k_1 u_1 + k_2 u_2 &= 3(2, 4, 5) + 4(7, 1, 2) \\ &= (6, 12, 15) + (28, 4, 8) \\ &= (34, 16, 23) \end{aligned}$$

So,

$$\begin{aligned} (k_1 u_1 + k_2 u_2) \cdot v &= (34, 16, 23) \cdot (5, 3, 2)^T \\ &= 264 \end{aligned}$$

Next, consider

$$\begin{aligned} k_1 (u_1 \cdot v) + k_2 (u_2 \cdot v) &= 3 ((2, 4, 5) \cdot (5, 3, 2)^T) + 4 ((7, 1, 2) \cdot (5, 3, 2)^T) \\ &= 3 * 32 + 4 * 42 \\ &= 264 \end{aligned}$$

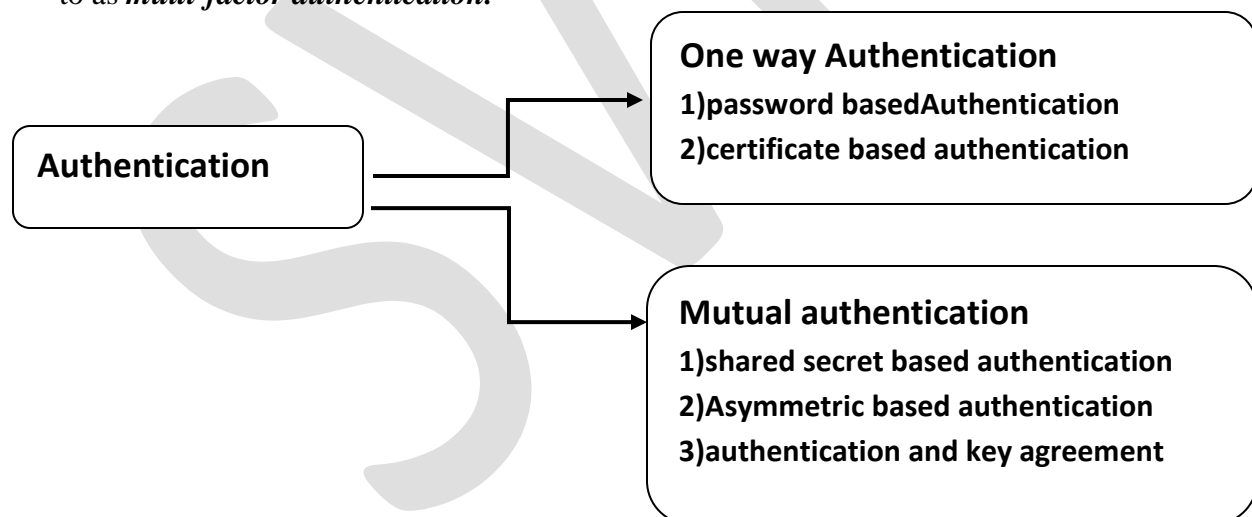
In general,

$$(k_1 u_1 + k_2 u_2) \cdot v = k_1 (u_1 \cdot v) + k_2 (u_2 \cdot v)$$

CHAPTER 2

Authentication-I

- *Authentication is a process in which a principal proves that he/she/it is the entity it claims to be.*
- The principal is referred to as the **prover**, while the party to whom proof is submitted identity verification is called the **verifier**.
- Authentication may be based on what the principal knows (e.g., a password or a passphrase) or has (an identity card or passport, for example).
- A principal is often a **human ,a computer, an application, or a robot**.
- In the case of a human principal, authentication may use physical characteristics such as **voice, a fingerprint, a retinal scan, or even a DNA sample** — this form of authentication is referred to as **biometric authentication**.
- With password-based authentication, an individual is often expected to communicate his/her password to a verifying entity. However, in many cases it may not be advisable for the individual to reveal his/her password.
- Instead, he/she may be required to perform some "one-way" cryptographic operation using his/her secret, which cannot be performed without knowledge of it.
- Finally, many authentication systems today use a combination of techniques. This is referred to as **multi-factor authentication**.



3.6 ONE-WAY AUTHENTICATION

- In client—server communications over a campus, network, for example, it is often the case that the client authenticates itself to the server.
- The server may or may not be authenticated to the client. This is referred to as **one-way authentication**.

➤ **Categorized to**

1. **password based authentication**
2. **certificate based authentication**

3.6.1 Password-based Authentication

- One of the most common mechanisms to implement authentication is the *password*.
- To login to a server, a user enters his/her *login name and password*.
- The password is the secret that is known only to the *user and server*.
- The *login name identifies a user*, while the user's knowledge of the corresponding *password constitutes proof* that he/she is the person with the given login name.
- As shown in below Fig the server uses the login name "Alka" to index into a database of (login name, password pairs),
- It Verifies that the *submitted password matches* the one stored against "Alka."
- Drawbacks/threats :
- First, the **password is sent in the clear**, so an attacker can eavesdrop on the message containing the password and later impersonate the real user.
- Second, the *passwords are stored in unencrypted form in a file on the server*.
- If an internal attacker obtains access to that file, all passwords stored on that server could get compromised.

Figure a : Communicating Password

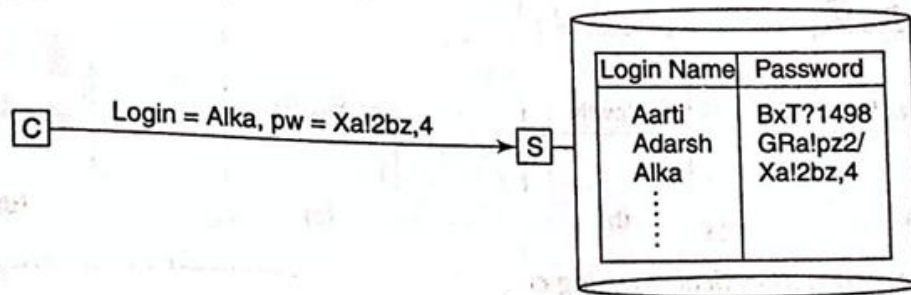


Figure b: Communicating Hash of Password

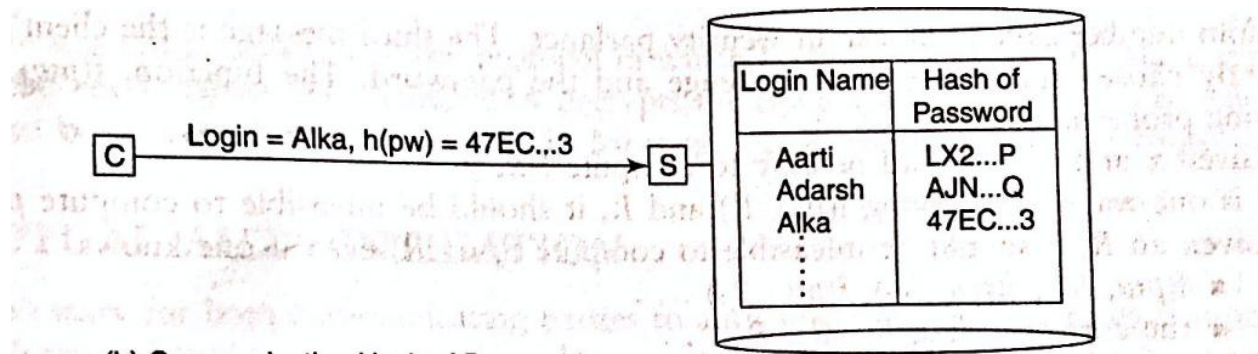


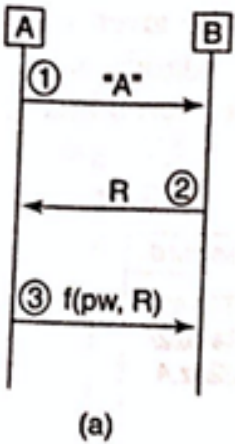
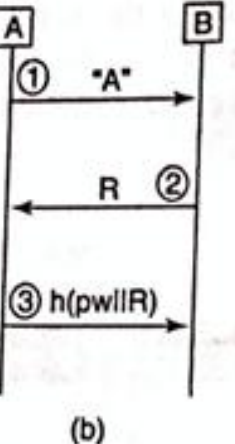
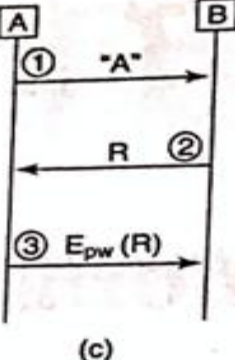
Figure: Password-based one-way authentication

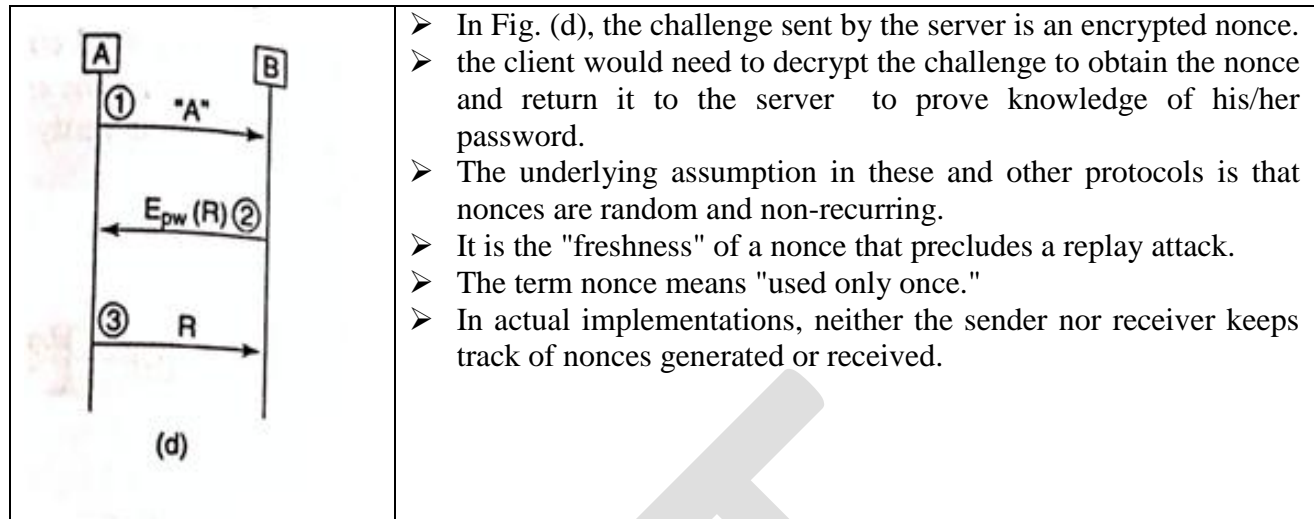
- In Fig(b), the cryptographic hash of the password is stored on the server.
- Also, the login software prompts the user for his/her password and computes its hash which is transmitted.
- The one-way property of the cryptographic hash helps prevent an attacker from deducing user passwords from information in the password file or from communications on the transmission line. However, an attacker could snoop on the communications between Alka and the server and obtain the hash of the password.
- He can, at a later point in time, replay it to the server thus impersonating Alka.
- Such an attack in which one plays back all or a part of one or more previous messages, with the intent of impersonating a legitimate user, is referred to as a **replay attack**.

Challenge response protocol

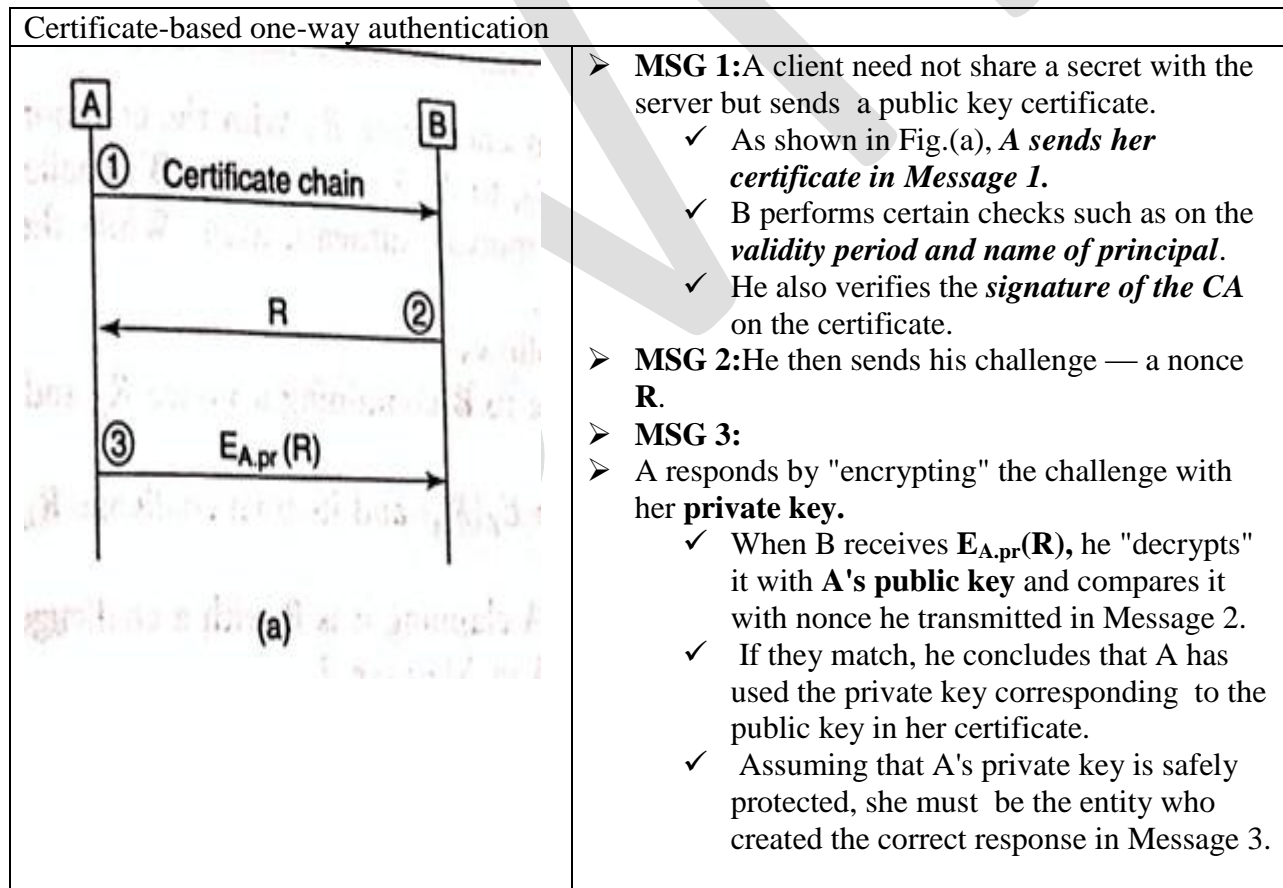
- An effective strategy to thwart a replay attack is for the verifier (in this 'case the server) to offer a fresh challenge to the prover (the client).
- In response, the client **does not communicate its password** but rather proves that it knows the password.
- The server is thus able to verify whether the client is genuine or not.
- The freshness of the challenge requires previous response to answer the current challenge. Such an authentication protocol is commonly referred to as a **Challenge—Response Protocol**.

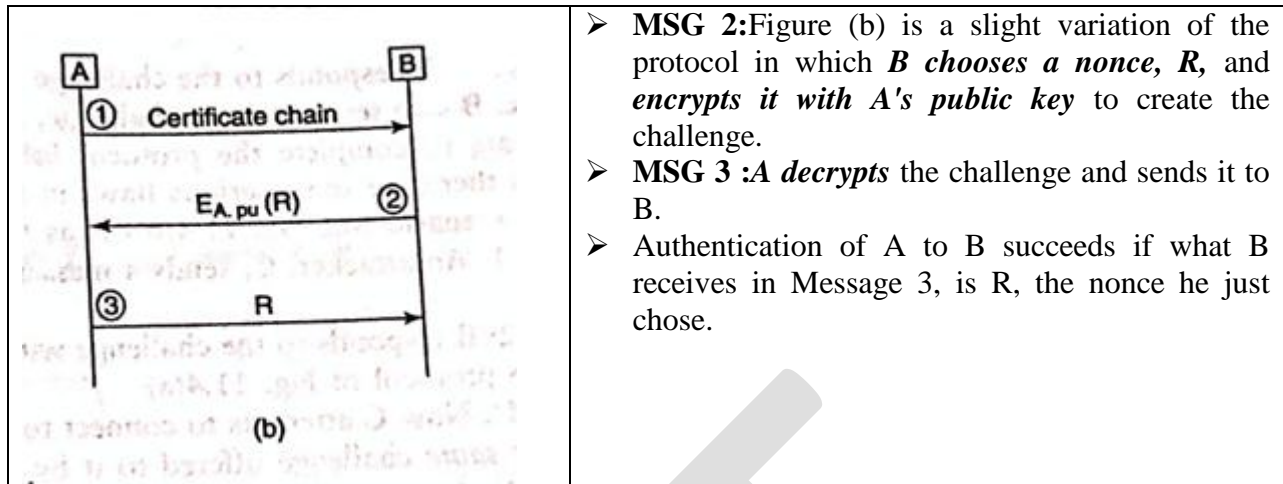
One-way authentication using challenge—response protocol

| | |
|--|--|
|  <p>(a)</p> | <ul style="list-style-type: none"> ➤ In the first message, A conveys its identity. ➤ The second message contains the challenge from the server. ➤ The challenge is a random number called a <i>nonce</i> (<i>number used only once</i>) in security parlance. ➤ The third message is the client's response - function of the challenge and the password. ➤ The function, $f(pw, R)$, has the following properties: <ul style="list-style-type: none"> ➤ Given x and y, it should be easy to compute $f(x, y)$ ➤ f is one-way; so, knowing $f(pw, R)$ and R, it should be <i>infeasible to compute pw</i> ➤ Given an R, it should be <i>infeasible to compute $f(pw, R)$</i> even if one knows ➤ $f(pw, R_1), f(pw, R_2), f(pw, R_3) \dots$ ➤ the corresponding $R_1, R_2, R_3 \dots$ |
|  <p>(b)</p> | <ul style="list-style-type: none"> ➤ An obvious choice for f is the cryptographic hash [Fig. (b)], which is applied over <i>the concatenation of the password and the nonce</i>. |
|  <p>(c)</p> | <ul style="list-style-type: none"> ➤ Another choice is a secret key encryption function with the where <i>password is used as a key</i> for encryption of random number R [Fig. (c)]. |



3.6.2 Certificate-based Authentication



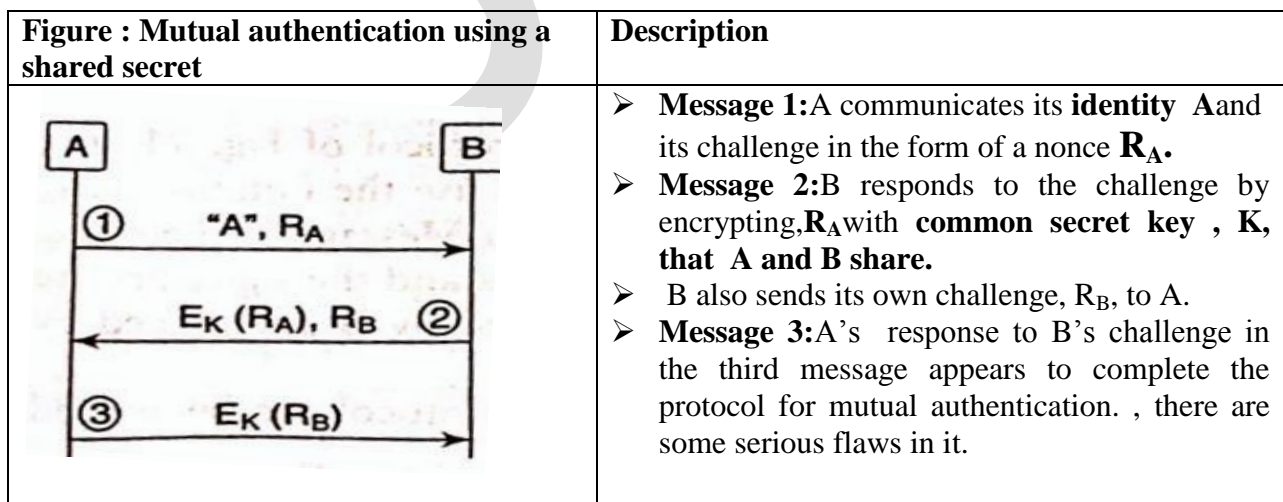


3.7 MUTUAL AUTHENTICATION

- It is often necessary for **both communicating parties to authenticate themselves to each other**.
- For example, in Internet banking, it is imperative that a customer interacts with his/her bank and not some entity posing as the bank.
- Likewise, it is important that a bank to verify the identity of the customer.

3.7.1 Shared Secret-based Authentication

- This is a mutual authentication using **a secret key shared by both parties**.



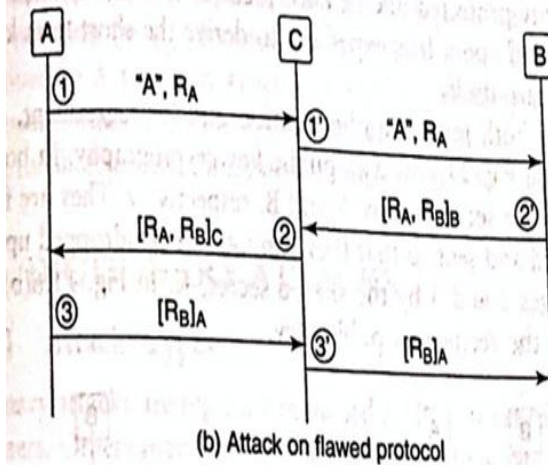
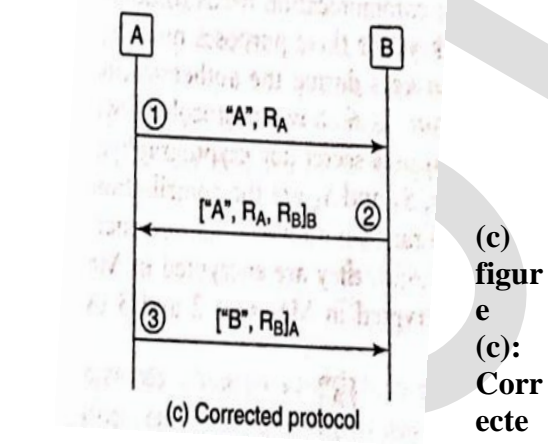
| | |
|---|--|
| <p>(a) Flawed protocol</p> | |
| | <ul style="list-style-type: none"> ➤ One attack scenario [figure (b)] is as follows: ➤ Message 1: An attacker, C, sends a message to B containing a nonce R_A and claiming to be A ➤ Message 2: B responds to the challenge with $E_K(R_A)$ and its own challenge R_B as required by the above protocol of Fig.(a). ➤ Message 1': Now C attempts to connect to A claiming it is B. with a challenge R_B. Note that this is the same challenge offered to it by B in Message 2. ➤ Message 2': A responds to the challenge with $E_K(R_B)$ and a nonce of its own. ➤ Message 3: C uses A's response $E_K(R_B)$ to complete the three-message authentication protocol with B. |
| <p>(b) Parallel session attack</p> <ul style="list-style-type: none"> ➤ What has the attacker C accomplished? ➤ C has successfully impersonated A to B. ➤ Message 3 was required to complete the authentication of C (posing as A) to B. ➤ C initiated the authentication protocol with A, presenting to A the same challenge it had received from B. ➤ A's response to the challenge in Message 2' was used by C to convince B that it was A that was trying to establish communication with him. This attack is termed a Reflection Attack since a part of the message received by an attacker is reflected back to the victim. ➤ In this case, the reflected message fragment is $E_K(R_B)$. This attack is also called a Parallel Session Attack | |

| | |
|-------------------------------|---|
| <p>(c) Corrected protocol</p> | <ul style="list-style-type: none"> ➤ Figure c: the protocol might require the responder to encrypt his challenge, while the initiator would be required to decrypt herchallenge. ➤ Encrypting both R_A and R_B |
|-------------------------------|---|

3.7.2 Asymmetric Key-based Authentication

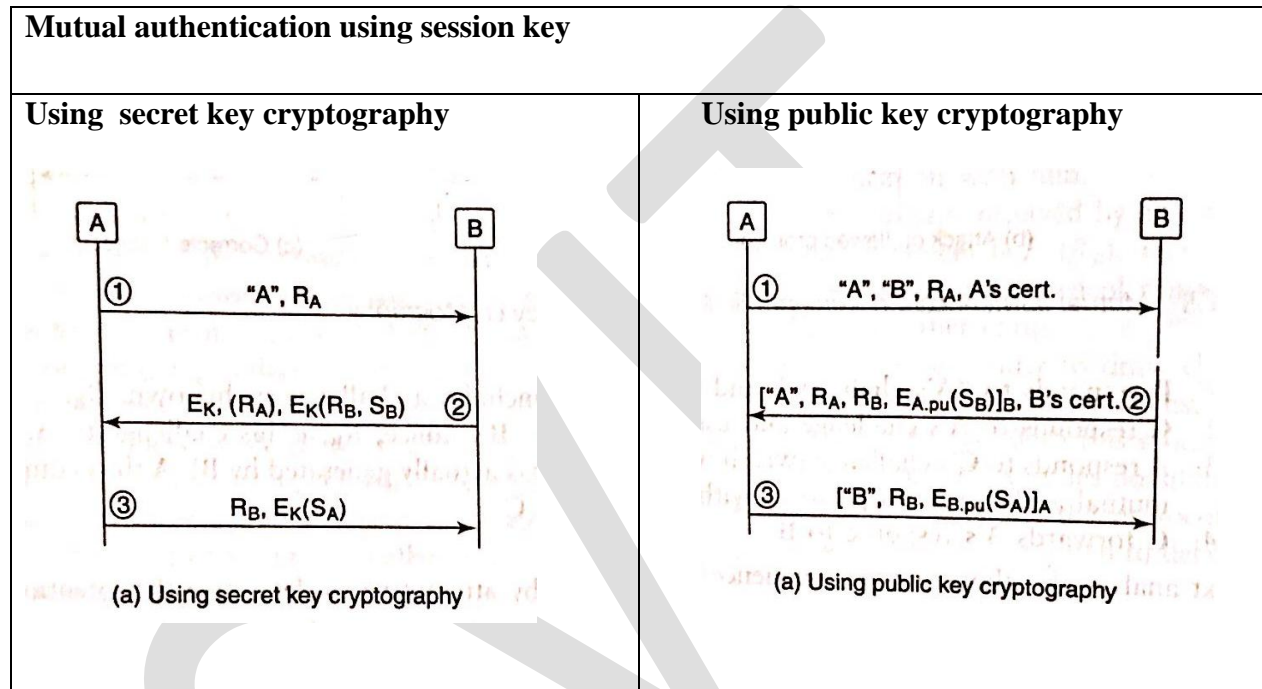
- We assume that both *A* and *B* have *public key/private key pairs*.
- The notation $[m]_A$ means a message, sent together with *A*'s signature on *m*.
- In the protocol of Fig. (a), each party transmits its own nonce and challenges the other to sign it.

| Asymmetric key based authentication /public key based authentication | Description |
|--|--|
| <p>(a)flawed protocol</p> | <ul style="list-style-type: none"> ➤ figure (a) shows Mutual authentication using public key cryptography /asymmetric based authentication ➤ MSG1: Identity of A, challenge sent by A , which is R_A, A's certificate ➤ MSG2: the string obtained by concatenating R_A , R_B signed by B, B's certificate. ➤ MSG3: R_B is thechallenge signed by A(encrypted using A's private key) |
| | <p>Figure b shows attack on flawed protocol:</p> <ul style="list-style-type: none"> ➤ MSG1: A initiates communication with C, sending the challenge R_A. ➤ MSG 1': C initiates communication with B using the same nonce R_A ➤ MSG2': B responds to "A's challenge" and includes a challenge of his own, R_B ➤ MSG 2: C responds to A's challenge and uses B's random number , R_B, as his challenge to A. ➤ MSG3: A responds to C's challenge (which was actually generated by B). A thus |

| | |
|---|--|
|  <p>(b) Attack on flawed protocol</p> | <p>completes the mutual authentication protocol with C.</p> <ul style="list-style-type: none"> ➤ MSG3': C forwards A's response to B. <p>It is clear from Fig.(b) :</p> <ul style="list-style-type: none"> ➤ That Message 1' is sent by C includes A's identity. And attempts to convince B that A intends to talk to him. ➤ B responds to what appears to be A's intention to communicate with him. ➤ Note that, in the current scenario, A may not wish to communicate with B and is not aware that C is attempting to do so on her behalf. ➤ Yet, after B receives Message 3', he feels A intends to communicate with him since Message 3' contains her signature on a nonce chosen by him. |
|  <p>(c) Corrected protocol</p> <p>(c) figure (c): Corrected protocol</p> | <ul style="list-style-type: none"> ➤ One solution to the above problem is for the entities to include the identity of the recipient in all messages signed. ➤ This is shown in Fig.(c). ➤ MSG 2:the string obtained by concatenating nonce R_A and R_B is signed by B is sent . (Means encrypted using B's private key) ➤ MSG 3: R_B is the challenge provided by B and signed by A in response .(means encrypted using A's private key) |

3.7.3 Authentication and Key Agreement

- In previous sections, authentication was performed using operations involving a long-term, shared secret or a private key.
- Since private key operations are very expensive, the communication can be integrity-protected and/or encrypted using short term keys or session keys .
- S_A and S_B are the contributions to the secret key by A and B, respectively.
- They are freshly chosen random numbers that are encrypted and sent so that they cannot be eavesdropped upon
- The key finally chosen could be a simple function of S_A and S_B , $S=S_A(xor) S_B$.



Use of Timestamps

- The use of nonces was introduced to prevent replay attacks.
- Basically, each party generates a nonce which is used as a fresh challenge to the other party.
- The recipient is often expected to sign or encrypt the challenge using a secret known to only the recipient (and the sender).
- The key idea here is the freshness of the nonce — if nonces were re-used, the response to the challenge could be replayed from a previous session.
- An alternative to nonces are timestamps.
- Ideally, **by securely "stamping" a message with the current time**, you convince the receiving party of its freshness.
- Below Figureshows the use of timestamps in conjunction with public key cryptography for authentication.

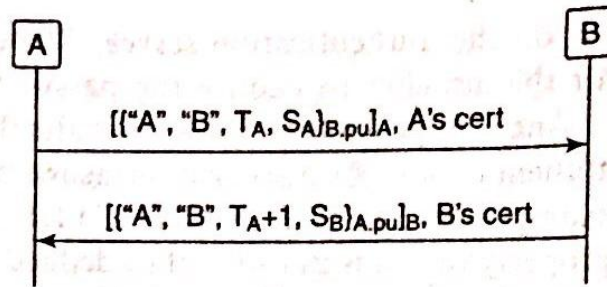


Figure :Mutual authentication with timestamp

- In Message 1, A inserts a timestamp, T_A , in her message and signs it.
- B, on receiving the message, checks whether the timestamp is sufficiently recent and then verifies the with timestamps signature.
- He increments the received timestamp, inserts it into his response message to A, and signs the message.
- The notation $\{m\}_{x,pu}$, denotes a message, encrypted using the public key of X
- If the clocks maintained by A and B are synchronized, the timestamp in Message 1 signed by A convinces B that the message was freshly created by A.
- The timestamp implicitly serves as A's challenge to B.
- By signing the incremented timestamp, B hopes to satisfy A that he is indeed responding to her message.

3.8 DICTIONARY ATTACKS

3.8.1 Attack Types

- Dictionary attacks are typically launched in the context of passwords.
- Some passwords have too few characters.
- Others may be common celebrity names, place names, etc.
- Some individuals use permutations of characters in the names of their near relatives or friends so that they are easily memorizable.
- Based on such clues, an attacker can build a dictionary of strings which are potential passwords of his/her victim.

| Password | Reason for Weakness |
|--------------|--|
| 123 or abcd | Common default passwords |
| Sm!t | Anything less than 8 characters is too short |
| nahkhkurhahs | Celebrity name — Shahrukh Khan spelt backwards |

| | |
|----------|--|
| 23-05-86 | Birthdays/anniversaries are convenient but would almost always be part of the attacker's password dictionary |
| ashyea | Permutation of letters in mother's or spouse's name, (Ayesha name in this example) is a poor choice especially if the attacker has personal information about his victim |
| Kolkata | Place names are often part of password dictionaries |

➤ There are two types of dictionary attacks —

1. **on-line**
2. **off-line.**

1. **on-line attack:**

- ✓ In on-line attacks, an intruder attempts to login to the victim's account by using the victim's login name and a guessed password.
- ✓ There is usually a system-imposed limit on the number of failed login attempts. So, unless the attacker is particularly insightful or lucky.
- ✓ an on-line attack has a limited chance of success.

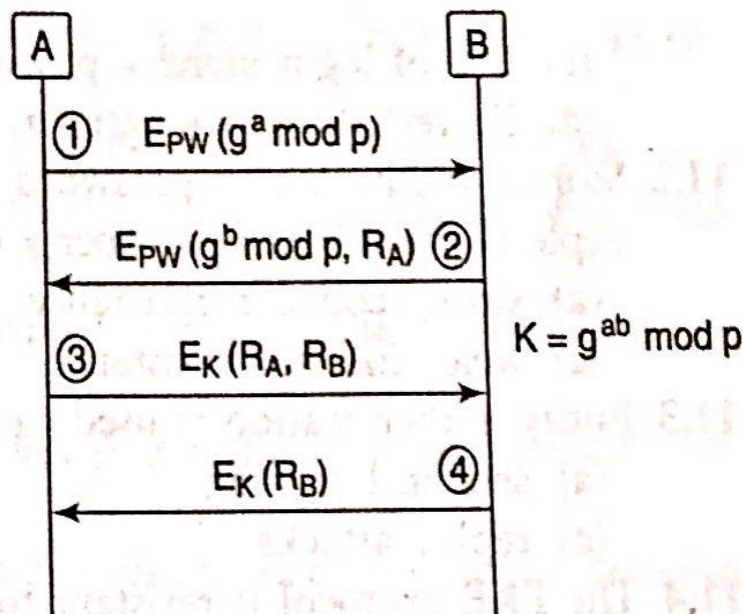
1. **off-line attack:**

- ✓ Unlike an on-line attack, an off-line dictionary attack leaves few fingerprints.
- ✓ One possibility is for the attacker to get a hold of the password file.
- ✓ Passwords are typically transformed in some way (by, for example, performing a cryptographic hash on them) before being stored on the authentication server.
- ✓ The cryptographic hash is a one-way function, so it is not easy for the attacker to deduce the password given its cryptographic hash.
- ✓ Another possibility is for the attacker to **eavesdrop on the communication link during authentication.**
- ✓ The attacker could use his/her dictionary of passwords to implement the following attack.

```
// Let D be an array containing the dictionary
// Let F denote f(pw, R) where pw is the client's password
// Let f be the number of permissible guesses (size of D)
found = false
i = 0
while ( ~found && i < n)
{
    x = f(D[i], R)
    if (x = F)
    {
        print ("CORRECT PASSWORD ")
        found = true
    }
}
```

3.8 .2 Defeating Dictionary Attacks

- One approach to frustrating a dictionary attack is to increase the cost of performing such an attack. The cost is the time to successfully complete the attack.
- The most time-consuming operation in each iteration of the dictionary attack program is $f(D[i], R)$. Hence, to decrease the attacker's chance of success, the function $f(D[i], R)$ could be made more computationally expensive.
- Suppose, for example, instead of the function f being a simple cryptographic hash, it was the cryptographic hash, h , applied successively a hundred times, that is,
- **$h (... h (h (D[i], R))$)**
- If the above function were used in the loop of the program, we would expect the program to run about 100 times slower.
- A protocol that virtually eliminates off-line dictionary attacks is the Encrypted Key Exchange (EKE) protocol.
- This is a password-based protocol that combines Diffie—Hellman key exchange with mutual authentication based on a shared secret.
- the Diffie—Hellman protocol is vulnerable to a man-in-the-middle attack which is due to the unauthenticated exchange of "partial secrets", **$g_a \bmod p$ and $g_b \bmod p$** .
- To mitigate this attack, EKE uses a novel idea — each side transmits its partial secret after encrypting it.
- The encryption key, **PW, is the hash of the password.**
- Below Figure shows the four messages that are exchanged in EKE.



- After MSG 2, both sides should be able to compute the **new session key $k = g^{ab} \bmod p$** denoted by K in the figure.

- Mutual authentication is accomplished using the familiar challenge—response protocol in which each side selects a random nonce and challenges the other side to encrypt it with the newly computed session key.
- It is assumed that the victim's password is "weak," that is, it can be guessed using moderate effort. That being the case, basic password-based mutual authentication protocols could yield to an off-line dictionary attack.
- Assume that an attacker has access to $E_{pw}(g^a \bmod p)$ and $E_{pw}(g^b \bmod p)$.
- The attacker would attempt to guess the victim's password and hence PW.
- If the attacker guessed correctly, he/she would be able to obtain the true values of
- $g^a \bmod p$ and $g^b \bmod p$. But even so, he/she would not be able to obtain the session key, $g^{ab} \bmod p$.
- This is so, since the computational Diffie—Hellman problem is infeasible in large groups that are carefully chosen,
- Thus, EKE is not susceptible to an off-line dictionary attack.
- Another property of EKE is that it provides perfect forward secrecy
- A protocol is said to have perfect forward secrecy if it is not possible for an attacker to decrypt a session between A and B even if he/she records the entire encrypted session and then at a later point in time (say a week later) obtains or steals all relevant long term secrets of A and B.

Chapter 3

Authentication-II

- **Key Distribution Centre (KDC) – a trusted third party that shares long-term keys-with clients and servers alike.**
- Two protocols that make use of a KDC—the *Needham-Schroeder protocol and Kerberos*.
- We then look at the biometric authentication as a complement to and, in some cases, as a substitute for cryptographic authentication.

3.9 CENTRALISED AUTHENTICATION

- There are a number of advantages of secret key cryptography over public key cryptography in authentication protocols.
- First, digital certificates and a public key infrastructure (PKI) are needed in support of public key cryptography.
- There is a substantial cost to set up and maintain a PKI.
- Also, public key/private key operations are relatively slow compared to secret key operations.
- In secret key cryptography, If the entity communicates with a large number of other entities over time, it must share a secret with each of those parties.

- Managing and securely storing a large number of keys is a nontrivial task.

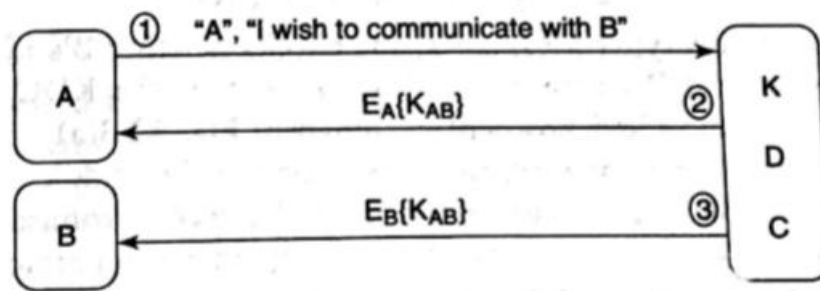
Note: Throughout this chapter,

$E_A\{m\}$ denotes a message encrypted using A's long, term secret shared with the KDC (derived using A's password).

$E_{AB}\{m\}$ denotes a message encrypted using the session key

- One approach to alleviating the risk is to employ a *trusted thirdparty* which, in this case, functions as a *key distribution centre (KDC)*.
- Each user registers with a KDC and chooses a password.
- A *long-term secret*, which is a function of the password, is to be exclusively shared by that user and the KDC.
- The main function of the KDC is to securely communicate a fresh, common session key to the two parties who wish to communicate with each other.

- In Fig below,



- **Message 1:** A informs the KDC that it intends to communicate with B
- The KDC generates a random secret, K_{AB} , and dispatches this to A and B through two encrypted messages.
- **Message 2** is encrypted using the long-term secret, K_A , that A shares with KDC.
- **Message 3** is encrypted with K_B , the secret shared between B and the KDC.
- Both A and B decrypt their messages and obtain the *short-term session key*.
- A and B then all subsequent messages during the session using K_{AB} .
- The above Figure was meant to convey the general idea in using a KDC but the protocol is susceptible to numerous types of replay and man-in-the-middle attacks.

3.9 THE NEEDHAM-SCHROEDER PROTOCOL

- Below Figure 3.10(a) enhances the protocol of Fig 3.9 to provide **mutual authentication** by including, **challenge—response phase**.
- Here, both sides proceed to challenge the other to prove knowledge of the session key, K_{AB} .
- The challenge is a **nonce**.
- The response involves decrementing the nonce and encrypting the nonce with the **session key, K_{AB}** .
- There are four versions
 1. Preliminary version 1
 2. Preliminary version 2
 3. Preliminary version 3
 4. Final version

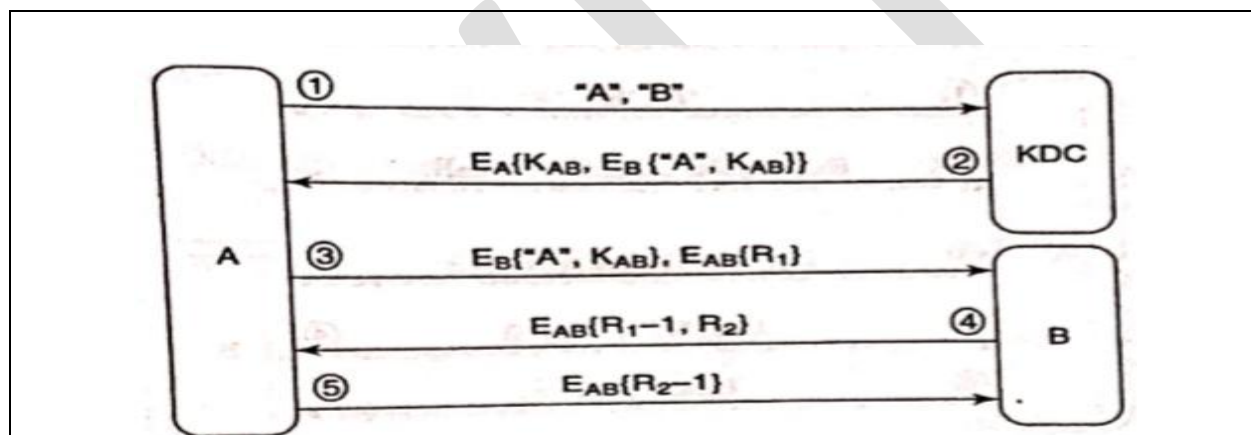


Figure 3.10 a

Preliminary Version 1

- **MSG 1:** Identity of A and B
- **MSG 2:** In response KDC encloses the ticket to B, i.e. $E_B\{A, K_{AB}\}$
- **MSG 3:** A then forwards the ticket along with the challenge to B (R_1)
- **MSG 4:** R_1 is decremented and B challenges A with R_2 .
- **MSG 5:** R_2 is decremented by A and forwarded to B.

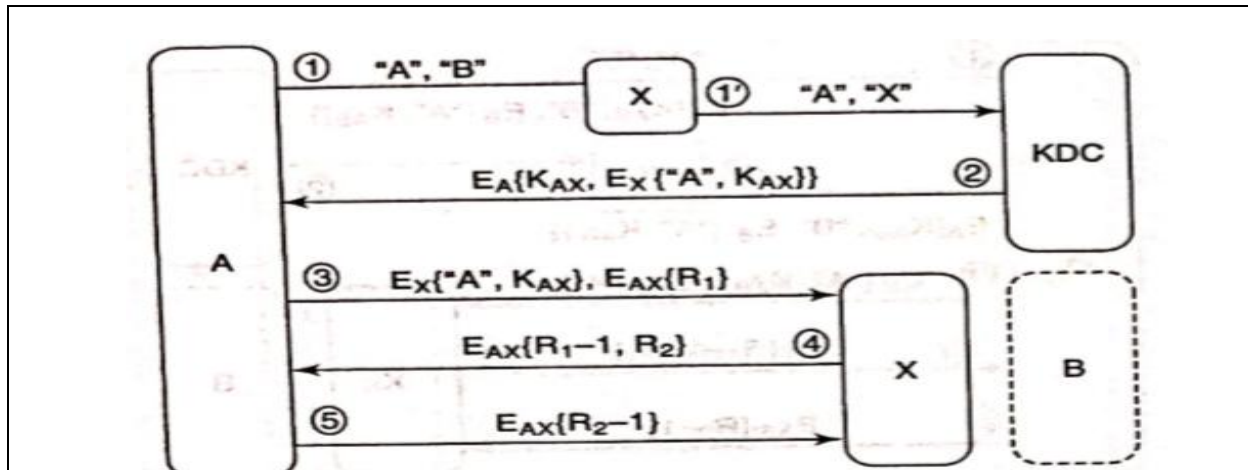


Figure 3.10(b)

Man in the middle attack on Preliminary Version 1

- The protocol in Fig. 3.10(a) is susceptible to an impersonation attack shown in Fig. 3.10(b).
- The attacker, X, is an insider who shares a long-term key with the KDC.
- The attacker, X, intercepts Message 1, substitutes "B" for "X" and sends the modified message to the KDC.
- In response, the KDC creates a ticket encrypted with X's long-term key and sends it to A in Message 2.
- Now X intercepts Message 3. He decrypts the ticket using the long-term secret he shares with the KDC. He thus obtains the session key, K_{AX} .
- Message 3 also contains A's challenge R_1 .
- X uses the session key, K_{AX} to decrypt the part of the message containing A's challenge. He successfully responds to A's challenge in Message 4.
- Thus, X successfully impersonates B to A.

Preliminary Version 2

- A simple fix to the protocol is to include B's identity in the encrypted message from the KDC to A (Message 2). The modified message is
- $E_A\{K_{AB}, \underline{\text{"B"}}, E_B\{\text{"A"}, K_{AB}\}\}$

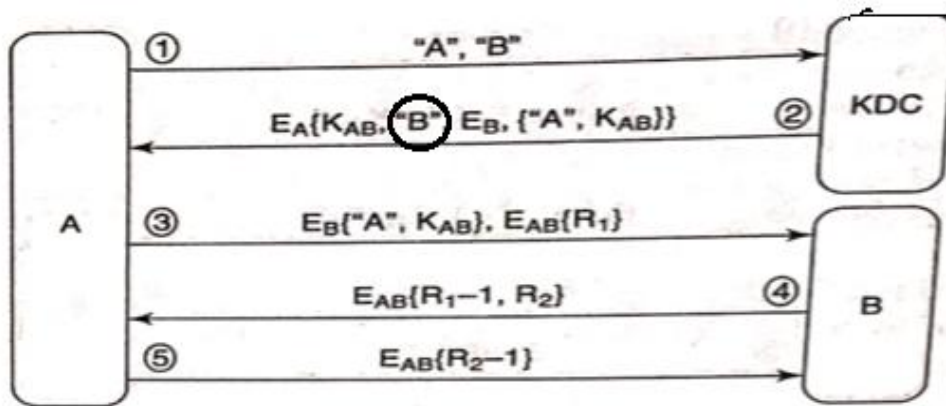


Figure 3.11 a Preliminary version 2

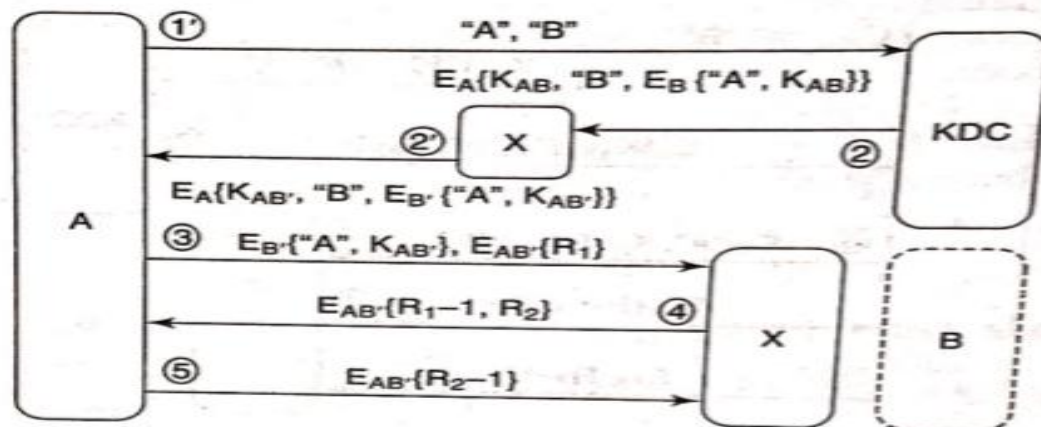


Figure 3.11 b Man in the middle attack and replay attack. (B' is the old key , B is the new key)

Preliminary version 2

- **MSG 1:** Identity of A and B
- **MSG 2:** In response KDC encloses the ticket to B. i.e $E_B\{A, K_{AB}\}$, b's identity
- Now, after A receives and decrypts Message 2, she checks whether B's identity is contained inside the message. The presence of B's identity confirms to A that the KDC knows that A wishes to communicate with B.
- **MSG 3:** A then forwards the ticket along with the challenge to B (R_1)
- **MSG 4:** R_1 is decremented and B challenges A with R_2 .
- **MSG 5:** R_2 is decremented by A and forwarded to B.

Man in the middle attack and replay attack on preliminary version 2

- The attacker, X, does the following:
 - X eavesdrops and records many of A's sessions with the KDC and with B over a period of time and steals B's password or long-term key.
 - B recognizes that his password has been stolen and immediately reports the incident to the KDC.
 - He obtains a **new long-term key, K_B** , which he uses subsequently.
 - Even then , the following scenario shows X successfully impersonates B to A.
1. A wishes to communicate with B and sends Message 1 in Fig.3.11 (b).
 2. X intercepts the KDC's response (Message 2) and instead plays a previous recording of Message 2.
 3. This message contains a ticket encrypted with **B's old key, K_B'** .
 4. X then intercepts Message 3 from A, which contains the old ticket and a fresh challenge to B. Because X has access to B's old key, he can decrypt this ticket and recover the session key, K_{AB} .
 5. Because X knows K_{AB}' , he can respond to A's challenge in Message 4. X's response is exactly what A expected to receive from B. Hence, A is convinced that she is talking to B.

Preliminary Version 3

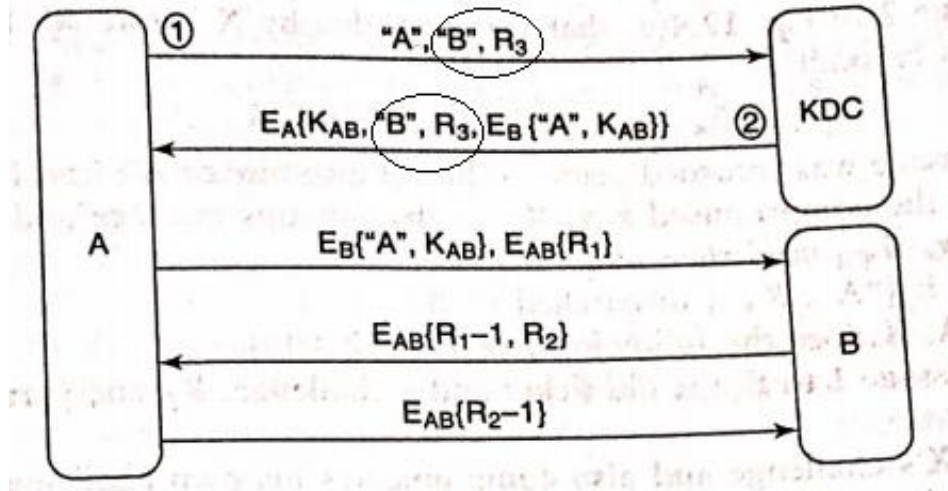


Figure 3.12 a Needham-Schroeder protocol: Preliminary version 3

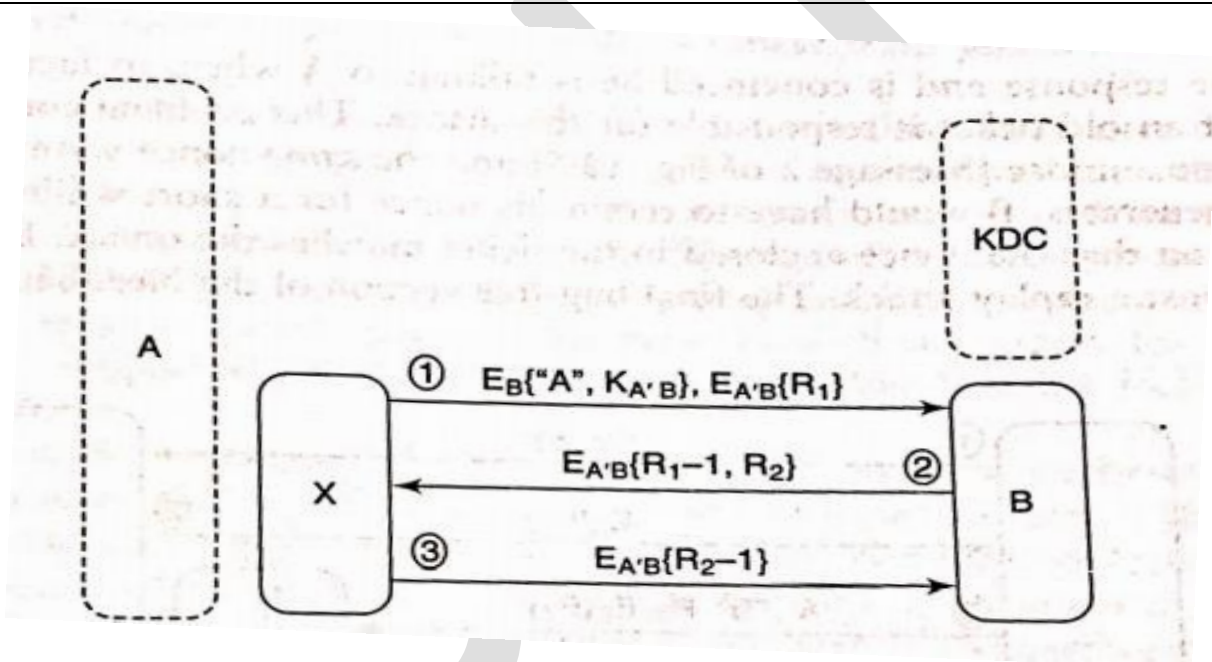


Figure 3.12 b Replay Attack on Preliminary version 3

- A' is lost password generated key
- A is new password generated key

Preliminary version 3

- We can fix this vulnerability in version 2 by ensuring the *freshness* of Message 2.
- This is accomplished by **A sending a (fresh) nonce in Message 1** [Fig. 3.12(a)] and receiving **confirmation of its receipt by the KDC in message 2.**

Replay attack on preliminary version 3

- The version 3 is still not secure despite the modifications made.
- X could still attack the protocol by recording previous messages and selectively replaying them when the right opportunity presents itself.
- Such as he attempts to steal A's password or long-term key.
- Assume again that A suspects the compromise of her password and promptly reports this to the KDC without delay.
- X then manages to steal A's long-term key that she shares with the KDC and perform an **impersonation attack.**
- A' is the old password generated key
- A is new password generated key.
- Using the compromised (old)key, X can decrypt this message and recover
- The old session key, $K_{A'B}$
- The old ticket $E_B\{A, K_{A'B}\}$
- To impersonate A, X does the following [see Fig.3.12(b)]:
 1. X sends, the old ticket and a challenge, R1, encrypted with the old session key.
 2. Message 1 to B,
 3. B responds to X's challenge and also communicates his own challenge, R2.
 4. Because X has the session key, he responds to the challenge by encrypting R2 with the session key.
- B receives the response and is convinced he is talking to A (impersonated by X).

Needham Schroeder Protocol: Final Version

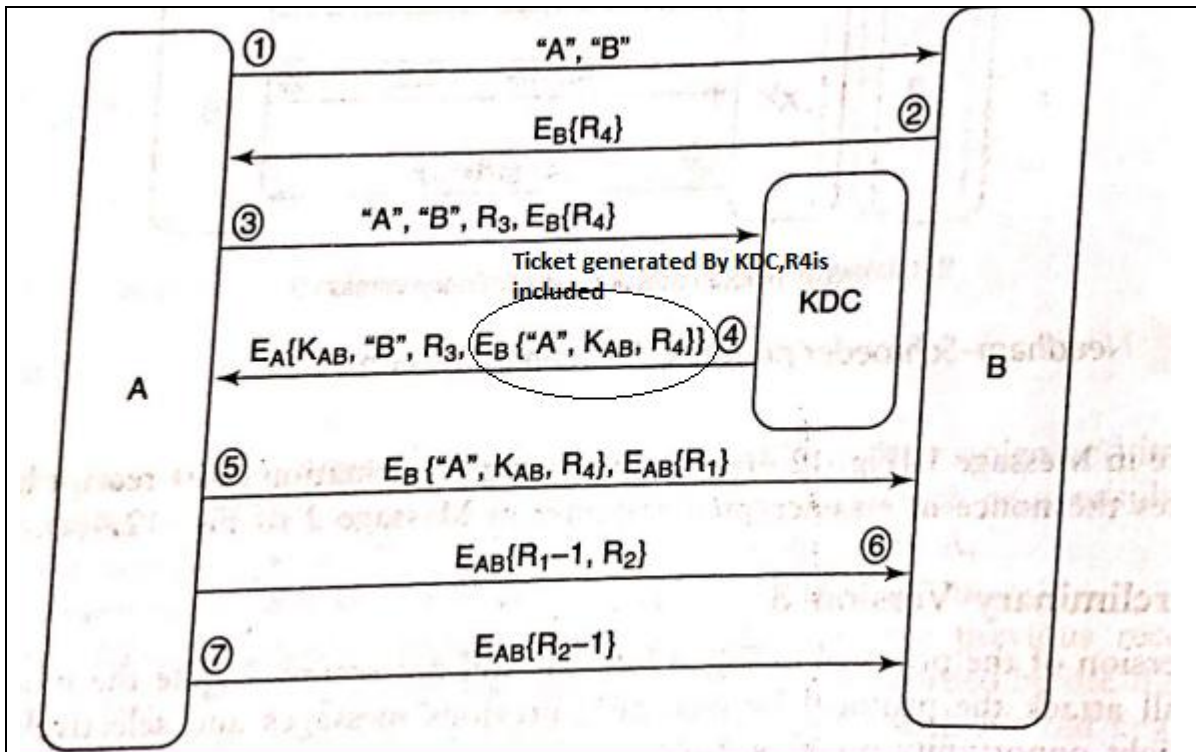


Figure 3.13

- The problem in previous versions could be fixed if B were allowed to choose a nonce(R_4) and the same nonce were enclosed by the KDC in the **ticket it generates**.
- **MSG 1:** Identity of A and B sent from A to B
- **MSG 2:** random number R_4 generated by B
- **MSG 3:** A forwards his challenge as R_3 along with R_4 .
- **MSG 4:** KDC generates ticket and includes R_4
- When B receives this ticket in msg 5, B can verify the random number/nonce generated in msg 2 is same or not.

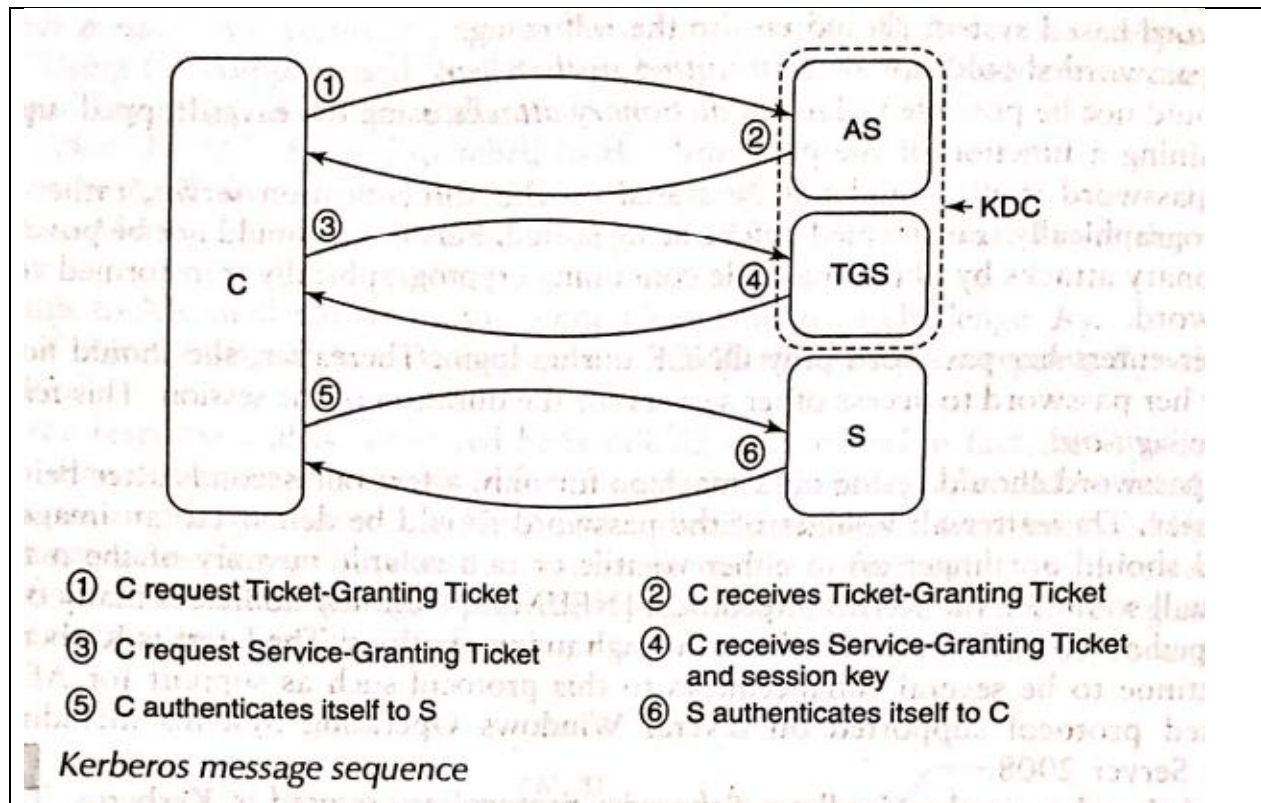
3.10 Kerberos

- A user could use the same password for all servers but distributing and maintaining a password file across multiple servers poses a security risk.
- A password-based system should ensure the following: y

1. The password should not be transmitted in the clear.
2. It should not be possible to launch dictionary attacks
3. The password itself should not be stored on the authentication server, rather it should be cryptographically transformed before being stored.
4. It should not be possible to launch dictionary attacks by obtaining a file containing cryptographically transformed versions of the password.
5. A user enters her password only ONCE during login. Thereafter, she should not have to re-enter her password to access other servers for the duration of the session. This feature is called **single sign-on**.
6. The password should reside on a machine for only a few milliseconds after being entered by the user.

The Kerberos protocol elegantly addresses many of these issues.

- Developed at MIT, Kerberos has been through many revisions.
- The latest is Kerberos Version 5.
- The KDC used in the Needham—Schroeder protocol is logically split into two entities here — the Authentication Server (AS) and the Ticket Granting Server (TGS).
- The sequence of messages exchanged between the client (C), the Kerberos servers (AS and TGS) and the requested server(S) is shown in Fig.3.14 .
- **There are three steps — each involving two messages**



Step 1: Receipt of Ticket-Granting Ticket

Message 1 C → AS: "C", "TGS", Times, R_1

Message 2 AS → C: "C", Ticket_{TGS}, $E_C \{ \text{"TGS"}, K_{C,TGS}, \text{Times}, R_1 \}$

where

$\text{Ticket}_{TGS} = E_{TGS} \{ \text{"C"}, \text{"TGS"}, K_{C,TGS}, \text{Times} \}$

Step 2: Receipt of Service-Granting Ticket

Message 3 C → TGS: "S," Times, Authenticator_C, Ticket_{TGS}, R_2

where

$\text{Authenticator}_C = E_{C,TGS} \{ \text{"C"}, TS_1 \}$

Message 4 TGS → C: "C", Ticket_S, $E_{C,TGS} \{ \text{"S"}, K_{C,S}, \text{Times}, R_2 \}$

where

$\text{Ticket}_S = E_S \{ \text{"C"}, K_{C,S}, \text{Times} \}$

Step 3: Client-Server Authentication

Message 5 C → S: Ticket_S, Authenticator_C

where

Authenticator_C = E_{C,S} {"C", TS₂}

Message 6 S → C: E_{C,S} {TS₂ + 1}

Step 1: Receipt of Ticket-Granting Ticket**Message 1**

C → AS

- In Message 1, the client informs the AS that it wishes to communicate with the TGS.
- "Times" field specifies the start time and expected duration of the login session.
- "C," is the ID of the user/client who has logged in.
- R1 is a nonce generated by C

Message 2

AS → C

- The response from the AS (Message 2) contains a session key, **K_{C,TGS}**, to be used for communication between C and the TGS.
- This key is encrypted with the long-term key, **K_C** known to C and the AS.
- This key is a function of the user's password.
- AS encrypts the nonce, that it received in Message 1.
- The nonce is used to prevent replay attacks.
- The AS also includes a TGT (**Ticket_{TGS}**) in connection with C's request.

Step 2: Receipt of Service-Granting Ticket**Message 3**

C → TGS

- In Message 3, C forwards the TGT (Ticket_{TGS}), Authenticator_C to the **TGS**
- **Using** this Ticket_{TGS}, **TGS server** extracts the session key, **K_{C,TGS}** known only to C and the TGS.
- As shown above, the **Authenticator_C** encrypts the current time (timestamp) and ID using **K_{C,TGS}**

Message 4

TGS → C

- The **TGS** generates a fresh session key, **K_{C,S}**, to be shared between C and S.
- This key is encrypted using the session key **K_{C,TGS}**, so only C can decrypt it.
- The fresh nonce, **R2**, **from C is also encrypted** by the TGS using **K_{C,TGS}**
- This convinces C that the received message is from the **TGS**
- Finally, the fresh session key **K_{C,S}** is enclosed in a *service-granting ticket* to be forwarded by C to S.

- The service-granting ticket is encrypted with the **long-term secret shared between the TGS and S.**

Step 3: Client-Server Authentication

Message 5

C → S

- C forwards to S the ticket containing the session key, $K_{c,s}$.
- C also creates and sends to S an authenticator by encrypting a timestamp with the session key $K_{c,s}$

Message 6

S → C

- S retrieves $K_{c,s}$ from the service-granting ticket.
- S verifies the authenticator from C.
- S then increments the timestamp and encrypts it with the fresh session key.
- The encrypted timestamp serves to authenticate S to C.

3.11 BIOMETRICS

3.11.1 Preliminaries

- A biometric is a **biological feature** or **characteristic of a person** that *uniquely identifies* him/her over his/her lifetime.
- Common forms of biometric identification include face recognition, voice recognition, manual signatures, and fingerprints.
- More recently, patterns in the iris of the human eye and DNA have been used.
- Behavioural traits such as keystroke dynamics and a person's walk have also been suggested for biometric identification.
- Biometric forms were first proposed as an alternative or a complement to passwords.
- Passwords are based on what a user knows.
- Commonly used ID cards, including personal smart cards, are based on what a person has.
- A biometric, on the other hand, links the identity of a person to his/her physiological or behavioural characteristics.
- The two main processes involved in a biometric system are enrolment and recognition.

1. Enrolment:

- ✓ In this phase, a subject's biometric sample is acquired.

- ✓ The essential features of the sample are extracted to create a **reference template**.
 - ✓ Sometimes multiple samples are taken and multiple templates are stored to increase the accuracy of a match in the subsequent recognition phase.
- 2. Recognition:**
- ✓ A fresh biometric sample of a person is taken and compared with the reference templates to determine the extent of a match.
- Biometrics are used in two different scenarios:
- 1. Authentication**
 - ✓ Biometric system stores (login name and biometric sample)
 - ✓ authentication involves a one-to-one match
 - 2. Identification**
 - ✓ As in authentication, a biometric sample of the subject is taken but the subject's identity is not presumed to be known beforehand.
 - ✓ It is assumed that a database of biometric samples of several users already exists.
 - ✓ The subject's biometric sample is compared with the samples in the database to determine if a match exists with any one of them.
 - ✓ identification involves a one-to-many match
 - ✓ A typical application of authentication is in access control, while identification finds widespread uses in forensics/criminology.
- The characteristics of a good biometric include the following:
- ✓ **Universality**: All humans should be able to contribute a sample of the biometric. For example, the speech-impaired may not be able to contribute towards a voice recognition system.
 - ✓ **Uniqueness**. Biological samples taken from two different humans should be sufficiently different that they can be distinguished by machine intelligence.
 - ✓ One litmus test of uniqueness is whether the biometric samples of two identical twins serves to unambiguously identify them.
 - ✓ **Permanence**. The biometric should not change over time. The samples acquired during enrolment may be several years old (even tens of years old). Still, it should be possible to detect a match between the newly acquired sample and that stored in a database of samples of thousands of individuals.
 - ✓ Permanence is not a given. For example, a person's voice may temporarily change due to a cold, the manual signature of a senior

citizen may change and fingerprints of people in certain professions may wear out over time.

Case studies

1. Fingerprints

2. Iris scan

1. Fingerprints:

- ✓ A fingerprint is an impression left by the ridges and valleys of a human finger.
- ✓ Each individual fingerprints exhibit distinctive patterns.
- ✓ During the enrolment and recognition phase ,an image of the fingertip is taken by placing it on the plane surface of a scanner.
- ✓ During the recognition phase the input template must match with the patterns stored in database.
- ✓ The simplest approach involves identification of distinctive patterns formed by ridges.these are called as singularities.
- ✓ They are:arch,loop and whorls.
- ✓ Arch : the ridge starts from one side of the finger and forms an arc and ends on other side.
- ✓ Loop: the ridge starts and ends at the same side of the finger.
- ✓ Whorls:appear as closed cycles or spirals in a fingerprints.

2. Iris scan

- ✓ The **iris is a thin opaque diaphragm of smooth muscle** situated in front of the lens in the human eye.
- ✓ Its annular shape surrounds the pupil.
- ✓ The intricate patterns on the iris appear to be unique.
- ✓ Two identical twins have iris patterns that are different as those of two unrelated individuals.
- ✓ The patterns of an iris are also stable with age.

Chapter 4

IPSec-Security at the Network Layer

4.1 SECURITY AT DIFFERENT LAYERS: PROS AND CONS

- Security may be implemented at different layers of the OSI model.
- Security is commonly implemented
 1. In the network layer or
 2. between the transport and application layers and/or
 3. within the application
- **IPsec** — the best-known protocol for providing security at the network layer.
- There are two main **IPSec protocols** and their modes of operation.

4.2 IPSec IN ACTION

- IPSec has been designed by committee in the late 1990's, it was intended to protect against sniffing, spoofing, hijacking, and Denial of Service (DoS) attacks.
- It provides a host of services including:
 1. **Data origin authentication and data integrity**
 2. **Protection from replay attacks**
 3. **Data confidentiality and**
 4. **Partial traffic flow confidentiality.**
- The end-points of the protocol can be **two hosts, two gateways, or a host and a gateway.**

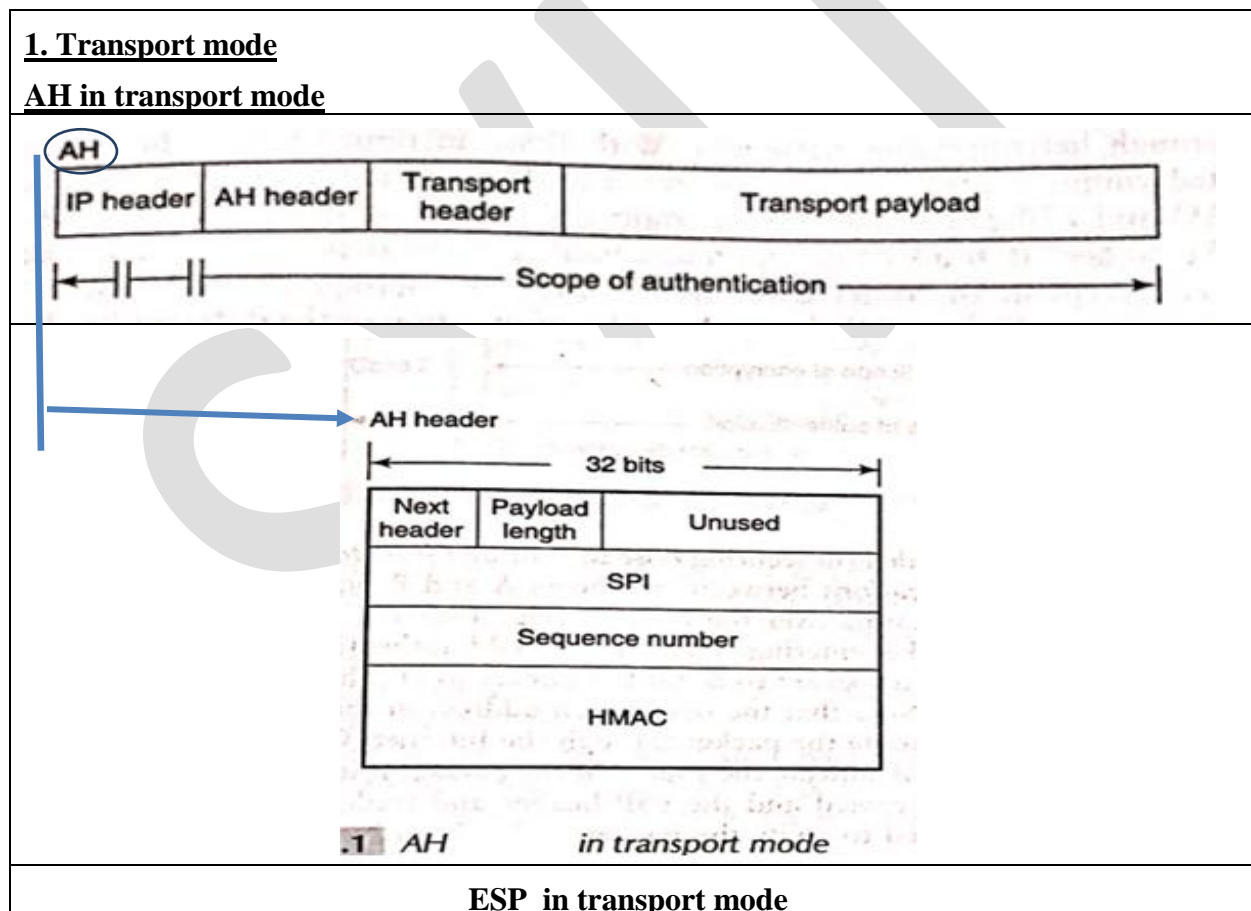
4.2.1 IPSec Security Associations

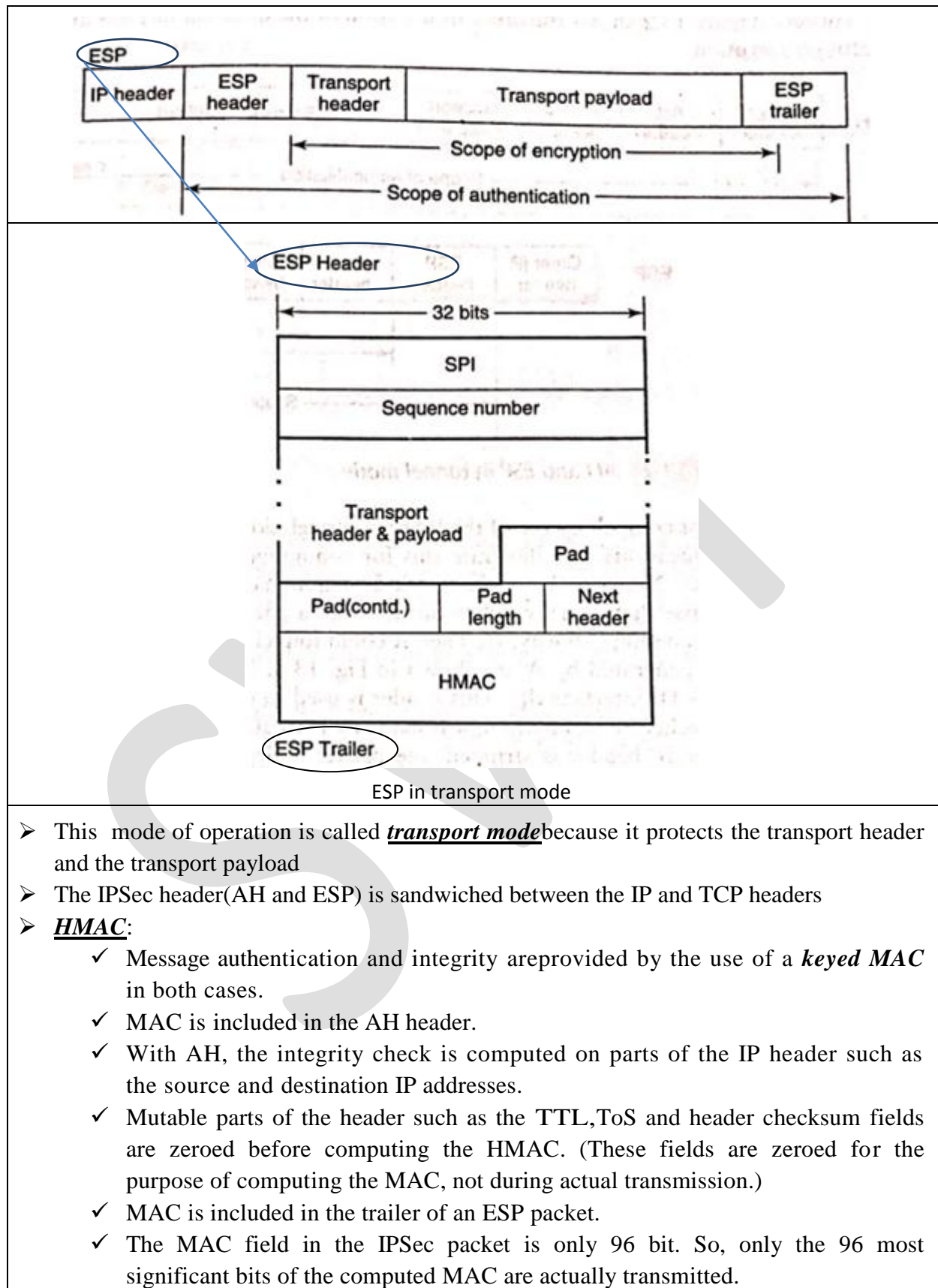
- Before two parties can communicate *securely*, they need to establish a **Security Association (SA)** with each other.
- A node (either host or gateway) may establish IPSec SAs with several nodes.
- An **SA** is uniquely identified by a combination of a **32-bit Security Parameter Index (SPI) and the IP address** of the connection endpoint.
- The information in an SA includes :
 1. **Lifetime** of the association
 2. **IPSec Mode** — transport or tunnel
 3. **Cryptographic parameters** (the algorithm used for encryption, if any, and for computing the integrity check, together with the keys)
 4. **32-bit sequence number** — the first packet protected by a newly established SA bears the
 5. **sequence number 1**, the sequence number gets incremented for each new packet sent.
 6. **anti-replay window**
- Each IPSec packet contains a value of SPI in its header. This is used by the receiving node to identify the SA to be used for processing the packet.

- Each node has a database of SAs for all connections originating from or terminating at it. This database is referred to as the SA Database (SADB).
- Finally, it should be noted that two communicating parties, A and B, establish two SAs — one for communications from A to B and another from B to A.

4.2.2 IPsec Protocols: AH and ESP

- IPsec includes two protocols —
 - ✓ AH (Authentication Header)
 - ✓ ESP (Encapsulating Security Payload).
- The main *difference* between AH and ESP is that **AH has no provision** for confidentiality.
- ESP **provides confidentiality** as an option.
- These protocols can be used in either transport or tunnel mode.
- Below Figure shows the headers introduced by AH and ESP and their coverage of authentication and encryption.





- This mode of operation is called ***transport mode*** because it protects the transport header and the transport payload
- The IPsec header (AH and ESP) is sandwiched between the IP and TCP headers
- **HMAC:**
 - ✓ Message authentication and integrity are provided by the use of a ***keyed MAC*** in both cases.
 - ✓ MAC is included in the AH header.
 - ✓ With AH, the integrity check is computed on parts of the IP header such as the source and destination IP addresses.
 - ✓ Mutable parts of the header such as the TTL, ToS and header checksum fields are zeroed before computing the HMAC. (These fields are zeroed for the purpose of computing the MAC, not during actual transmission.)
 - ✓ MAC is included in the trailer of an ESP packet.
 - ✓ The MAC field in the IPsec packet is only 96 bit. So, only the 96 most significant bits of the computed MAC are actually transmitted.

- ESP does *not* provide protection to any part of the IP header in transport mode.
- **Sequence number:**
- All IPsec headers (both AH and ESP in both modes) have 32-bit fields each for the SPI and a ***packet sequence number***, to protect against replay attacks.
- **Padding:**
- Padding is added so that the length of the encrypted payload is a multiple of block size (as required by most encryption algorithms).
- Padding also helps in hiding the actual length of the data when ESP is used with the encryption option turned on.

4.2.2 Tunnel versus Transport Mode

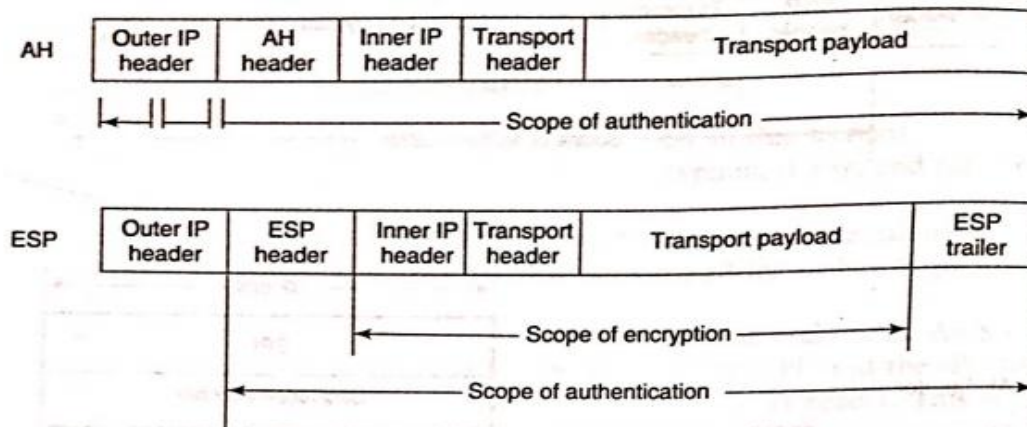
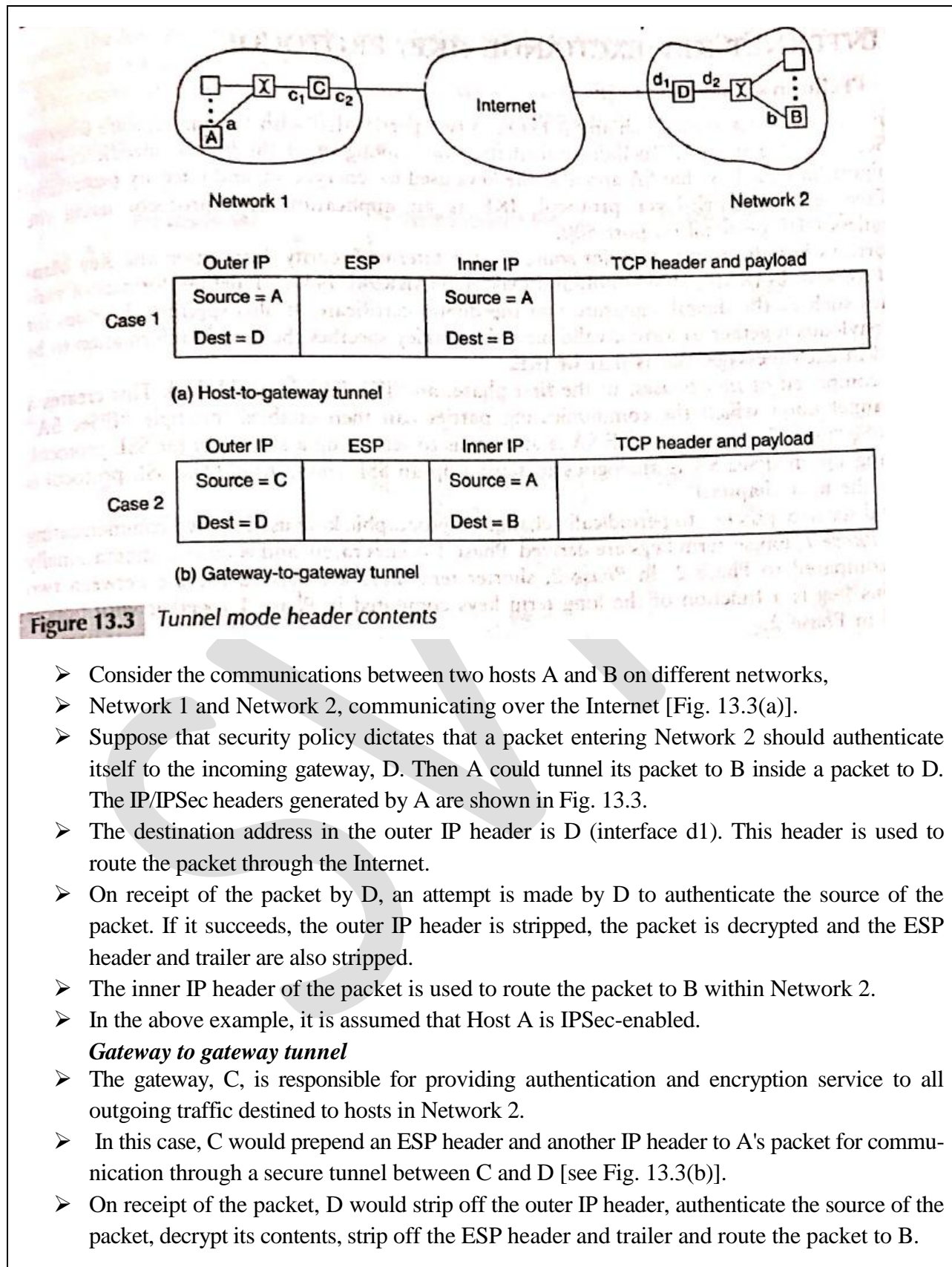


Figure 13.2 AH and ESP in tunnel mode

- To protect the entire IP header, IPsec has an option called **tunnel mode**.
- Both, AH and ESP can employ tunnel mode.
- With encryption turned on, ESP in tunnel mode encrypts the "inner" IP header thus providing limited ***traffic flow confidentiality***. Because the inner, IP header is encrypted, an "outer IP header" is used for routing.
- Figure 13.2 shows the order of insertion of the different headers and the scope of authentication/encryption.
- The most compelling use of the IPsec in tunnel mode is in securing ***host-to-host and host-to-gateway communications***.



4.3 INTERNET KEY EXCHANGE PROTOCOL

- The main goal of IKE is to *establish an SA* between two parties that wish to communicate securely using IPsec.
- IKE is comprised of two phases.
- In the first phase, an "*IKE SA*" is established. This creates a *secure channel* upon which the communicating parties can then establish multiple "IPsec SA" instances over time.
- Setting up an IKE SA is similar to setting up a *session* in the SSL protocol, while setting up an IPsec SA is analogous to setting up an SSL *connection*.
- It is good security practice to periodically change cryptographic keys used by two communicating parties.
- In *Phase 1*, longer term keys are derived.
- Phase 1 occurs rarely and is more computationally intensive compared to Phase 2.
- In *Phase 2*, shorter term keys are derived for use between two parties. This key is a function of the long term keys computed in Phase 1 together with nonces exchanged in Phase 2.

4.3.1 IPsec Cookies

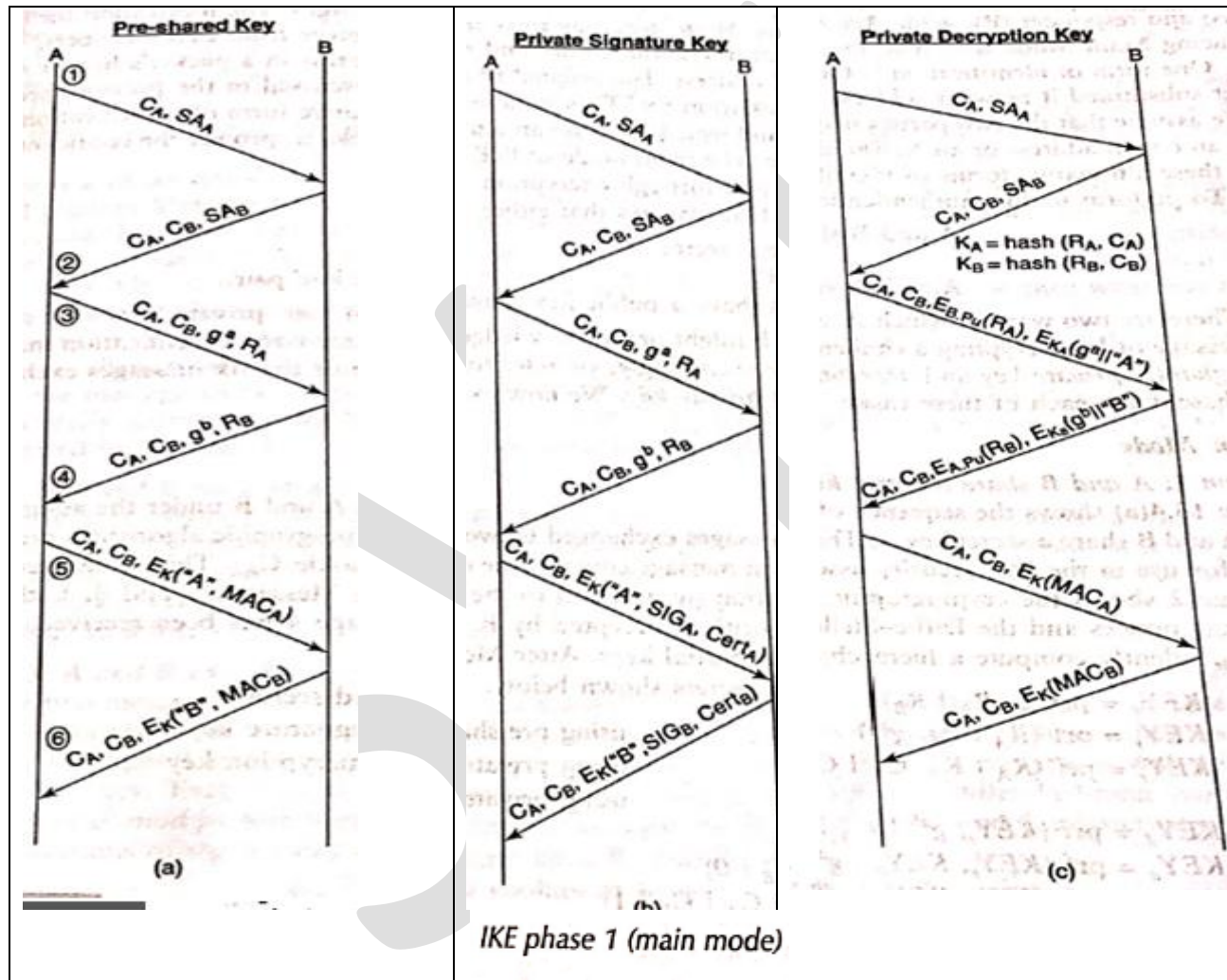
- To prevent DoS attacks, IKE makes extensive use of *cookies*.
- One cookie is created by the initiator, A, and another by the responder, B.
- *Phase 1* of IKE uses Diffie—Hellman key exchange, which involves the computationally intense modular exponentiation operation.
- IKE mandates that B should compute a 64-bit integer called a *cookie*.
- Cookie value computed by A : This is a hash function of many variables including the *IP address of A*, an *temporary secret* known only to B and possibly the *time*.
- A is required to send this cookie to B in all subsequent messages.
- On receipt of a message from A, B will check to see whether the cookie corresponds to A's IP address. If the check fails, B will abort session establishment .

4.3.3 IKE Phase I

- The following are accomplished in IKE Phase 1:
- ✓ The authentication method, encryption, and hash algorithms together with the Diffie—Hellman group to be used are negotiated.
- ✓ Both parties authenticate themselves to each other.
- ✓ Key_s , KEY_a and KEY_e , are computed.
- ✓ These keys are used for message integrity ,encryption, respectively in both, Phases 1 and 2.
- ✓ Cookies are created at the start of Phase 1 and serve the purpose of an IKE connection.
- ✓ Phase 1 uses one of two modes.
- ✓ Main Mode involves a total of six messages between *initiate (A) and responder (B)*,
- ✓ Aggressive Mode uses only three messages.

- ✓ The motivation for introducing Main Mode is to hide the identities of the sender and receiver from eavesdroppers.
- ✓ The six messages exchanged in Phase 1 for each of these cases.
- ✓ **Main Mode**
- ✓ IKE derives a hierarchy of keys — KEY_r, KEY_d, KEY_a, and KEY_e.
- ✓ The subscripts **r, d, a, and e** denote 'root', 'derived', 'authentication', and 'encryption', respectively.

IKE PHASE 1 main mode



$$\begin{aligned}
 KEY_r &= \text{prf}(s, R_A | R_B) && \text{using pre-shared secret} \\
 KEY_r &= \text{prf}(R_A | R_B, g^{ab}) && \text{using private signature key} \\
 KEY_r &= \text{prf}(R_A | R_B, C_A | C_B) && \text{using private encryption key} \\
 \\
 KEY_d &= \text{prf}(KEY_r, g^{ab} | C_A | C_B | 0) \\
 KEY_a &= \text{prf}(KEY_r, KEY_d | g^{ab} | C_A | C_B | 1) \\
 KEY_e &= \text{prf}(KEY_r, KEY_a | g^{ab} | C_A | C_B | 2) \\
 \\
 MAC_A &= \text{prf}(KEY_a, g^a | g^b | C_A | C_B | SA_A | "A") \\
 MAC_B &= \text{prf}(KEY_a, g^b | g^a | C_B | C_A | SA_A | "B")
 \end{aligned}$$

Option 1:

- ✓ A and B share a secret key Figure 13.4(a) shows the sequence of messages exchanged between A and B under the assumption that A and B share a secret key, s.
- ✓ The first message contains the cryptographic algorithms proposed by A for use in the IKE security association (in addition to the cookie CA). This is denoted SA_A.
- ✓ Message 2 shows the cryptographic algorithms accepted by B.
- ✓ In Messages 3 and 4, both sides exchange nonces and the Diffie—Hellman partial keys.
- ✓ After Message 4 has been received, A and B independently compute a hierarchy of secrets such as key_r,key_a,key_e.

Option 2 :A and B each have private signing keys

- ✓ The sequence of messages exchanged between A and B [Fig. 13.4(b) is very similar to that in the shared key case.
- ✓ The main difference is that authentication and integrity protection of messages is effected by digital signatures on MAC_A and MAC_B using their private keys.
- ✓ Also, A and B dispatch, their signing key certificates in Messages 5 and 6 so the other party can perform signature verification [Fig. 13.4(b)].

Option 3:A and B each have private decryption keys

- ✓ The first two messages used with this option [Fig. 13.4(c)] are as in the two previous cases.
- ✓ The main difference between this option and the previous ones is that both sides exchange their identities earlier in Messages 3 and 4.
- ✓ Each side generates a nonce (RA or RB) and encrypts it with the other side's public key.
- ✓ Each side encrypts its identity together with its Diffie—Hellman partial key ($g^a \text{ mod } p$ or $g^b \text{ mod } p$) with temporary keys, K_A and K_B in Messages 3 and 4, respectively.
- ✓ K_A and K_B are functions of nonces and cookies as below
- ✓ K_A = hash (RA, CA)
- ✓ K_B = hash,(RB, CB)
- ✓ In Messages 5 and 6, each side transmits a MAC.
- ✓ The MAC key is KEY_a which, in turn, is a function of the two nonces, RA and RB.
- ✓ If A or B were unable to decrypt (with their private key)the nonce generated by the other side, then they wouldnot be able to compute the correct MAC .

IKE PHASE 1 Aggressive mode

Aggressive Mode

The aggressive mode involves **only three** messages.

- As shown in Fig. 13.5, the identities of A and B are sent in the clear in messages 1 and 2.
- Another aspect of IKE Phase 1 in aggressive mode is that the Diffie—Hellman group is used and the group parameters are decided by A.
- A computes its partial key and sends it to B in Message 1.
- B has no choice but to quietly accept the group chosen by A.

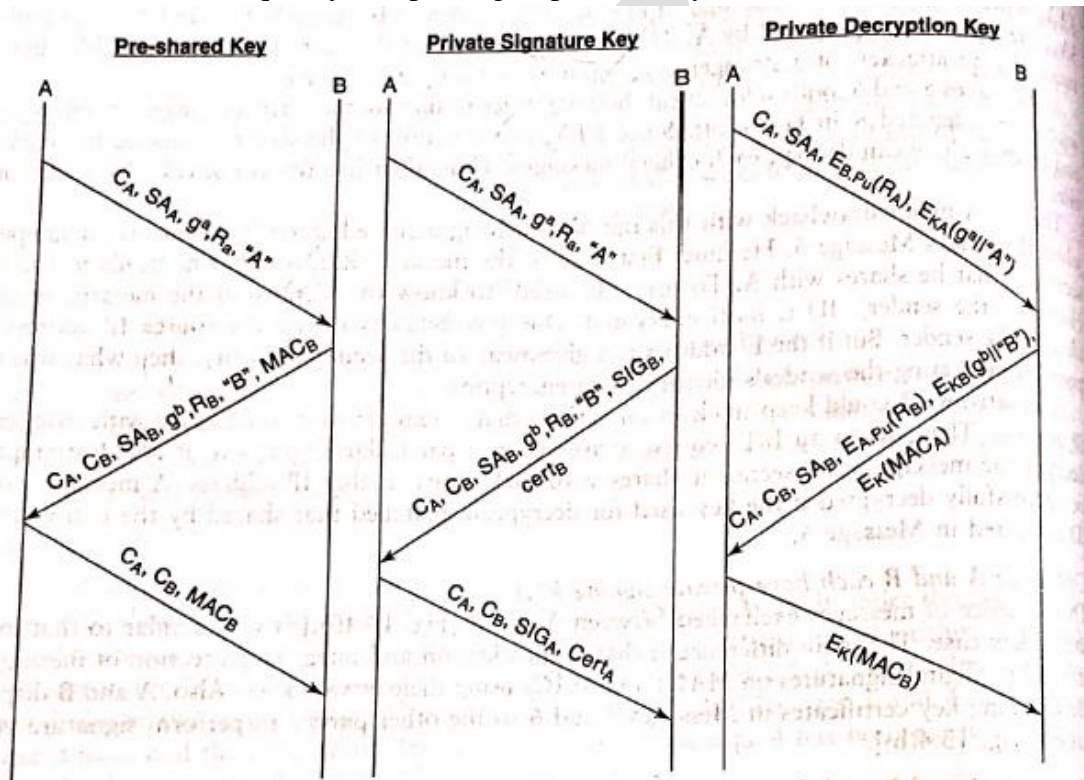
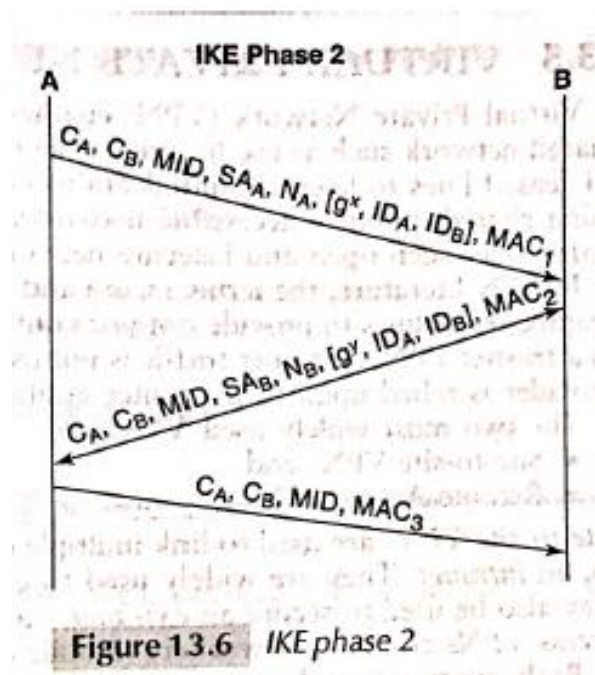


Figure 13.5 IKE phase 1 (aggressive mode)

e

IKE phase 2



IKE Phase 2

- IKE phase 2 to be used for authentication (and encryption) as part of the IPsec SA.
- Two parties participate in an IKE Phase 2 exchange in order to establish a new IPsec SA.
- Either party can initiate this phase in which the cipher suite and the keys that comprise the new IPsec SA are agreed upon.
- Above Figure shows the three messages exchanged in "Quick Mode."
- All messages, encrypted using the secret, KEY_e , computed in the previous phase.
- Message integrity and data source authentication is provided by using an HMAC.
- The key for the HMAC is KEY_a , also computed in Phase 1.
- A 32-Bit "Message ID", MID, is used to distinguish this phase 2 session from, possibly, others that may be set up concurrently within the same IKE SA.
- The MID together with the two cookies created in Phase 1, C_A and C_B , are dispatched as part of each of the three messages.
- Both sides send their proposals for a suite of cryptographic algorithms to be used in the IPsec SA.
- These are denoted SA_A and SA_B in Fig. 13.6.
- To guarantee freshness, both sides also generate and transmit nonces, N_A and N_B .
- In addition to the agreement on a cryptographic suite, the purpose of this phase is to agree on the secrets
- These secrets are computed simultaneously by both sides and are a function of KEY_d computed in Phase 1 and the nonces N_A and N_B exchanged in this phase.
- To integrity protect the messages each side computes a MAC on their messages. MACS includes integrity protection for both nonces, N_A and N_B .

- The Diffie—Hellman partial secrets are $g^x \bmod p$ and $g^y \bmod p$
- The Diffie—Hellman secret key, $g^{xy} \bmod p$ is also used as an additional input in computing the necessary keys for the IPsec SA
- The IPsec SA set up in Phase 2 includes the mutually agreed upon cryptographic suite and secret keys for authentication and/or encryption.
- Both sides will then create an entry, for, the new SA in their database of SAs.

4.4 SECURITY POLICY AND IPSEC

- **A Security Policy Database (SPD)** is used to determine whether a packet sent or received should pass through security, bypass it, or simply be dropped. Such a decision is made based on fields in the IP and transport headers.
- These fields, called **selectors**, include the ***destination IP address, the type of transport layer protocol (whether TCP or UDP) and type of application (indicated by transport Selectors are used to index into the SPD.***
- The output of this lookup indicates whether security should be applied.
- If so, and if the packet is part of the IP traffic that already has an existing SA, protocol port number). then the SPD returns a pointer to that SA.
- If an SA does not exist or has expired, the IKE protocol is used to establish an SA between the sender and receiver.

4.5 Virtual private networks

- ✓ A virtual private network (VPN) enables organization to communicate securely over a public, shared network such as internet.
- ✓ A secure VPN uses cryptographic techniques to provide not just confidentiality but also authentication and message integrity.
- ✓ In a trusted VPN, customer traffic is not usually encrypted, service provider has to guarantee confidentiality of traffic.
- ✓ The two most widely used VPNs are:
- ✓ ***Site to site VPN***: are used to link multiple offices of an organization, known as intranet.
- ✓ Site to site VPN may also be used to secure an extranet—a network connecting multiple business partners.
- ✓ ***Remote access VPNs*** connect teleworkers to their offices.

Chapter 5

Security at the Transport Layer

3.12 INTRODUCTION

- Developed by Netscape in 1994, the **Secure Sockets Layer (SSL)** protocol has emerged as the principal means of *securing communications between an Internet client (such as a browser) and a server.*
- It was standardized by IETF in 1999 and called **Transport Layer Security (TLS).**

SSL(Secure Sockets Layer)

- SSL is sandwiched between *TCP (it only runs over TCP) and an application layer protocol.*
- It is application protocol independent.
- Protocols such as HTTP, FTP, SMTP, IMAP, and POP can all be run over SSL.
- Application protocols **secured by SSL are usually suffixed by an "S"** and run on different port numbers.
- For example, *HTTP runs on port 80 but HTTPS runs on port 443.*
- FTP runs on *port 21 but FTPS runs on port 990.*
- SSL is comprised of two main protocols (see Fig. 14.1)

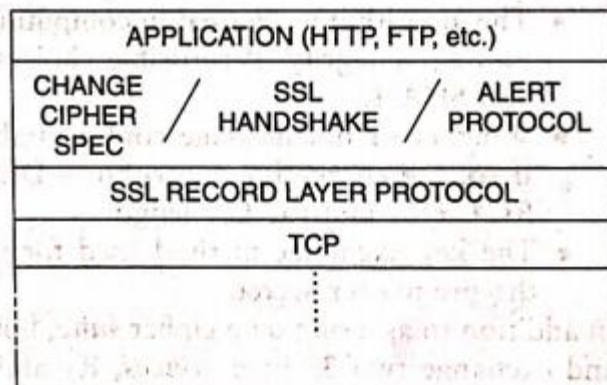


Figure 14.1 SSL on the protocol stack

1. The Handshake Protocol

2. The Record Layer Protocol

- The SSL handshake protocol is used to negotiate the set of algorithms to be used for securing the communication link.
- Server authentication in SSL is mandatory and performed as part of the **handshake.**
- The hand-shake protocol is also responsible for deriving keys , for encryption and MAC computation

- The actual job of providing *message authentication + integrity checking and encryption* is performed by the **SSL record layer protocol**.
- It sits just below the handshake protocol and protects each message exchanged by the two communicating parties.
- The record layer protocol also detects *replayed, re-ordered, and duplicate packets*.

3.13 SSL HANDSHAKE PROTOCOL

3.13.1 Steps in the Handshake

- The client initiates a handshake with the server to either
 - (a) **Start a new session or**
 - (b) **resume an existing session or**
 - (c) **Establish a new connection within an existing session.**
- The main steps in the SSL handshake for establishing a new session are as follows:
 - (1) Agreement on a *common cipher suite* to be used in the new session
 - (2) Receipt and validation of the *server certificate* by the client
 - (3) Communication of a "*pre-master secret*" and computation of derived secrets
 - (4) *Integrity verification* of handshake messages and *server authentication*
- These steps are realized by the sequence of messages shown in below figure
- The steps are:

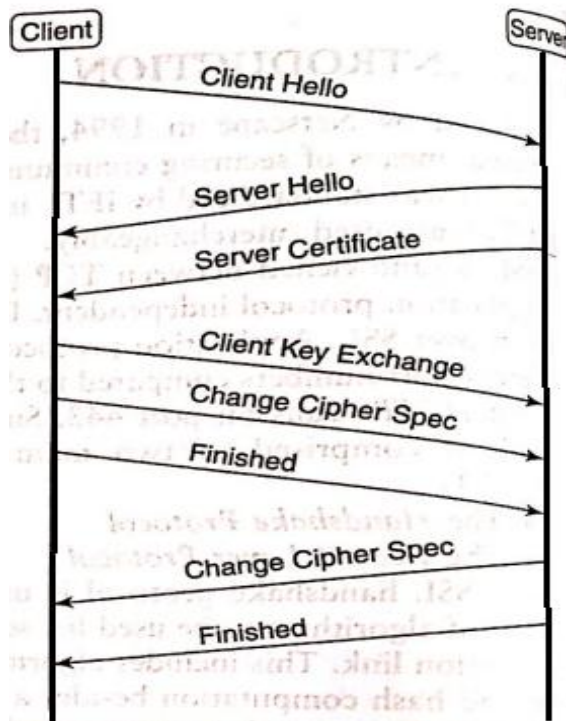


Figure 14.2 SSL handshake

- **Step 1:** Two messages are communicated in this step — *Client Hello* and *Server Hello*.

The following decisions are taken here:

- Should a new session be established or should an existing one be re-used?
- For a new session the session ID field in the Client Hello message is 0; else the field is set to the ID of the session to be re-used,
- The session ID field in the Server Hello message is the ID of the new session to be established or the ID of an existing session.
- The algorithm to be used in computing the MAC for message integrity include MD5 and SHA-1.
- The key exchange method used for communicating the pre-master secret.
- In addition to agreeing on a cipher suite, both sides choose and exchange two 32-byte *nonces*, **RA** and **RB**, in this step.
- **Step 2.** The server communicates its *certificate* to the client (see Fig. 14.2).
- On receipt of the certificate, the client checks the owner's name/URL and validity period.
- It also verifies the signature of the CA on the certificate.
- Successful verification of these fields does not guarantee the authenticity of the sender
- Authentication of the server only occurs at the end of Step 4,

Step 3.

- The client chooses a *pre-master secret* — a 48-byte random number.
- The pre-master secret is encrypted with the server's public key and sent to the server in the Client key exchange messages.
- Thereafter, both client and server compute the master secret. This is an HMAC style function, f , of the pre master secret, the two nonces exchanged in step 1 and some pre defined constants.
- The computation uses a standard cryptographic hash function such as the SHA-1 or the MDS.

$$\mathbf{Master_Secret = f(Pre-Master_Secret, R_A, R_B, constants)}$$

- Finally six secrets are derived using HMAC-style functions of the master secret, the two nonces, and different pre-defined constants

$$\mathbf{Derived_Secret_i = f(Master_Secret, R_A, R_B, constants), 1 < i < 6}$$

- The six derived secrets are:
 - ✓ Initialization vector for encrypting messages from client to server
 - ✓ Initialization vector for encrypting messages from server to client
 - ✓ Secret key for encrypting messages from client to server
 - ✓ Secret key for encrypting messages from server to client
 - ✓ Secret for computing keyed hash on messages from client to server (Client MAC Secret)
 - ✓ Secret for computing keyed hash on messages from server to client (Server MAC Secret)

Step 4: This step involves the exchange of two messages in each direction.

- The first of these is the "**Change Cipher Spec**" message (Fig. 14.2).
- The party that sends this message signals that from now on the cipher suite and the keys computed will be used.
- The second message in this step is the "Finished" message.
- This message includes a keyed hash on the concatenation of *all* the handshake messages sent in the preceding steps + a pre-defined constant.
- The keyed hash serves as an *integrity check* on the previous handshake messages.
- After the server receives the "Change_Cipher_Spec" and "Finished" messages from the client, it verifies the computation of the keyed hash.

- It then computes its own keyed hash that covers the previous handshake messages + a pre-defined constant, which is distinct from the one used by the client.
- The client receives the keyed hash and verifies it. Only at this point is the server authenticated to the client.
- On the other hand, client authentication as part of the SSL handshake is optional.

3.13.2 Key Design Ideas

Key Exchange Methods

- In Step 2, the server dispatches its certificate so the client can use the public key contained in the certificate to encrypt the pre-master secret.
- In some cases, however, the server's certificate may be a "signature-only certificate."
- This means that the public key in the certificate and the corresponding private key may only be used exclusively for signature generation/verification, not for encryption.
- In that case, SSL permits the server to create a **temporary public key/private key pair**. The public key (including modulus) are signed by the server using the private key corresponding to the public key in the **signature-only certificate**.
- The signed public key and certificate are communicated by the server to the client.
- The client verifies the signature on the public key and then uses it to encrypt the pre-master secret.
- SSL offers a rich set of options for key exchange.
- Such as RSA-based key exchange methods, Diffie—Hellman key exchange may be used.

Server Authentication

- The MAC computed by both parties and sent in step 4 is used as an **integrity check** on the previous handshake messages.
- All the handshake messages are sent in the clear (except for encryption of the pre-master secret).
- It is possible for an attacker to alter one or more of the handshake messages.
- For example, he may replace **128-bit DES by a 56-bit DES**.
- This may induce both parties to use a weaker cipher, which can be compromised by the attacker.
- The MAC detects any modification in the handshake messages.
- The hash computed by the server and verified by the client uses the *server MAC secret*,
- It is a function of the master secret which in turn is a function of the pre-master secret.
- Recall that the pre-master secret is chosen by the client and encrypted with the server's public key so that the server alone can read it. So, nobody but the server and client could compute the six secrets.
- Only after the client receives and verifies the keyed hash from the server, is it convinced that it is talking to the authentic server.

Sessions and Connections

- It is good security practice to change keys during a long-lasting session.
- SSL has provision for changing keys by creating new connections within an existing session.
- In creating a new connection, the pre-master secret which is part of the existing session state is *not* chosen anew.
- Instead, a new master secret is computed as a function of the *existing pre_ master secret* and two *fresh nonces* contributed by the client and server.
- The *session state* includes the *pre-master secret*, the negotiated *cipher suite* and, of course, the *session ID*.
- The *state of a connection* includes the two *nonces*, the *master secret*, the six *derived secrets*, and two message *sequence*(one for each direction of message transfer).

3.14 SSL RECORD LAYER PROTOCOL

- The SSL record layer protocol is used to securely transmit data using the negotiated cipher suite and the keys derived during the SSL handshake.
- Its main tasks are computation of a per-message MAC and encryption.
- If the data to be transmitted is very large, it performs fragmentation.
- Each fragment is 16 kb or less.
- When a connection is established, both sides initialize a sequence counter to zero.
- The counter is incremented for each packet sent.
- The sequence number itself is not sent. However, it is used in the computation of the MAC (at the sender) and in its verification (at the receiver).
- The MAC is computed on the concatenation of the 64-bit sequence number and the compressed fragment (if compression is used).
- The next step after computing the MAC is encryption.
- If the combined size of the data fragment and MAC is not a multiple of block size, a pad is appended.
- The data fragment, MAC, and pad (if any) are then encrypted, prepended with a header, and passed on to the TCP layer for further processing.
- The SSL record layer protocol header :there is a 1-byte Content Type field, which identifies the higher layer protocol used to process the fragment.
- Two bytes are used to specify the Version number.
- Finally, the Length field indicates the fragment size in bytes.

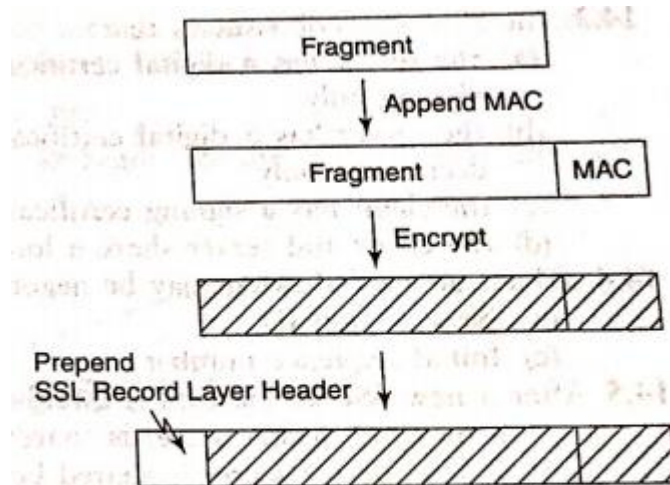


Figure 14.3 Function of the SSL record layer protocol

3.15 OpenSSL

- **OpenSSL** is open source software that implements the SSL/TLS protocol.
- It is comprised of a number of libraries that implement various cryptographic algorithms.
- It provides extensive support for communicating and validating digital certificates.
- OpenSSL is based on the **SSLeay** library developed by Eric A. Young and Tim J. Hudson.
- OpenSSL enhances the productivity of application developers by providing a rich set of APIs that handle diverse aspects of SSL-enabled communication from connection set-up and tear-down to certificate storage, management, and verification.
- The developers can rely on the OpenSSL APIs to implement the required security.