9.10 The "state matrix" just before the first round in AES is

28	A7	3B	90
34	2E	F1	89
51	71	D6	78
0C	29	9E	10

What is the value of this matrix after the "column mixing" operation of the third round?

9.11 The 8 × 8 S-Box in AES is implemented by the 16 × 16 table in the text (Fig. 9.5). The algorithm to compute the entries in the table appears in Section 9.4.2. Calculate the missing entry (row 4, column 9) in the table.

9.12 Use the S-Box of Fig. 9.5 (and *not* the Extended Euclidean algorithm) to compute the multiplicative inverse of the field element, C1 in GF(2^8). Then, obtain the product of what you have just computed and C1 and verify whether you really get 1.

9.13 AES decryption involves reversing the "Column Mix" operation. For this purpose the state matrix should be pre-multiplied by the matrix shown below. Fill in the missing entries.

$$\begin{pmatrix} 0E & \_ & \_ & \_ \\ 09 & \_ & \_ & \_ \\ \_ & \_ & \_ & \_ \\ 0B & \_ & \_ & \_ \end{pmatrix}$$

9.14 The goal of this question is to estimate the degree of non-linearity of the 8 × 8 S-Box in AES. Write a program to compute the bias of each combination of inputs and outputs to/from the S-Box (there are a total of 2^{16} such combinations). Then prepare a histogram of biases (showing the number of combinations that have a particular bias).

9.15 Determine whether linear cryptanalysis of *i*-stage AES, *i* = 1, 2, 3, ... can be performed. If so, write a program to implement it.

ANSWERS TO OBJECTIVE-TYPE QUESTIONS

9.1 (b)	9.2 (d)	9.3 (b)	9.4 (d)
9.5 (b)	9.6 (b)	9.7 (c)	9.8 (d)
9.9 (a)	9.10 (b)(d)	9.11 (a)(c)	

Chapter 10

Key Management

10.1 INTRODUCTION

Key management is related to the generation, storage, distribution, and backup of keys. The focus in this chapter is on the management of public key–private key pairs.

Recall that 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. Likewise, to verify B's signature on a message, A needs B's public key. The key issue here (pun unintended) is "How does A know B's public key?"

Possibility 1: A may frequently communicate with B in a secure fashion, so she may already have B's public key.

The above possibility makes a couple of assumptions. First, B must have securely communicated his public key to A at some point in the past. By securely, we mean that 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.

The second implicit assumption is that, if B's public key–private key pair is changed, then the new public key will be swiftly and securely communicated to A. However, this may not be always practical. For example, what if B is an e-commerce website and A is an anonymous customer to B?

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. An infrastructure to support this directory could also become a bottleneck besides being an attractive target for spoofing and denial of service attacks. Finally, there are other problems such as the *non-uniqueness of names*. For example, there could be several persons with a name like John T. Brown. So which public key would the directory return in response to a request for the public key of John T. Brown?

Possibility 3: A receives a document signed by a trusted source, C, containing B's public key.

This approach appears to obviate the need for an on-line, centralized directory site. We explore this approach in greater detail in the next section after introducing the idea of digital certificates.

10.2 DIGITAL CERTIFICATES

10.2.1 Certificate Types

A *digital certificate* is a *signed* document used to *bind* a *public key* to the *identity* of a person. An individual's identity could be his/her name, national identification number, e-mail or postal address, employer, etc. or some combination of these. The entity that issues certificates is a *trusted entity* called a *Certification Authority* (CA). CAs are often selected government agencies or banks. Certificates may be issued to individuals, to organizations, or even to servers.

There are different *types* of certificates that a CA may issue. 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." These certificates are only appropriate for applications with the least demands on security. To carry more weight, certificate issuance would require the CA to perform identity verification of the applicant. This will involve submission of credentials such as a passport or driver's license. The CA may have to obtain and verify several details of the applicant including his/her employer, e-mail address, etc. Practically speaking, this task would be delegated by the CA to a *Registration Authority* (RA). The role of an RA may be performed by a bank, for example, which has an existing relationship with the customer.

It is possible and often desirable that a CA or RA may require the applicant to demonstrate possession of the private key corresponding to the public key presented. This is achieved by, for example, getting the applicant to decrypt a random string encrypted with the applicant's stated public key. The price of such a certificate would be higher than a basic certificate but the certificate owner would now be able to use his/her certificate for more security-demanding applications.

10.2.2 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 listed next:

- Certificate Serial Number and Version
  Each certificate issued by a given CA will have a unique number.
- Issuer information
  The *distinguished name* of an entity includes his/her/its "common name," e-mail address, organization, country, etc.
- 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.

- 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). In addition, the specific use of the public key–private key pair may be included (e.g., signature only or encryption.)
- Validity period
  There are two date fields that specify the *start date* and *end date* between which the certificate is valid.
- 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.

A basic, no-frills certificate shown in Fig. 10.1 is about 1 kb in length.

10.2.3 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 which are likely to be unique. These include 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. In Section 10.3.3, we explain why this simple check-alone does not suffice but, for now, we will pretend that it does.
- 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 these cases, A knows or has access to the CA's public key. The case in which the CA or RA is unknown to A is covered in the next section.

10.3 PUBLIC KEY INFRASTRUCTURE

10.3.1 Functions of a PKI

In Section 10.2.1, we discussed the role of a CA in issuance of certificates. A Public Key Infrastructure (PKI) includes the CAs, the physical infrastructure (encryption technologies, hardware, etc.), and the formulation and enforcement of policies/procedures. It also includes the entire gamut of services required in supporting the use of digital certificates. A sample of these is as follows:

- Certificate creation, issuance, storage and archival
- Key generation and key escrow (if necessary)
- Certificate/Key updation (if necessary)
- Certificate revocation

```
Certificate:
   Data:
       Version: 1
       Serial Number: 3174
       Signature Algorithm: md5WithRSAEncryption

       Issuer: C=IN, ST=Maharashtra, L=Mumbai, O=Security Centre,
               OU=Enterprise Security,
               CN=Security Centre CA/emailAddress=agni@SecCent.com.in
       Validity
           Not Before: Jan  1 10:00:00 2010 GMT
           Not After : Dec  31 10:00:00 2010 GMT

       Subject: C=IN, ST=Goa, L=Madgaon, O=Ocean Enterprises,
                OU=IT Consulting,
                CN=Oceant emailAddress=zuari@oceant.com.in
       Subject Public Key Info:
           Public Key Algorithm: rsaEncryption
           RSA Public Key: (1024 bit)
               Modulus (1024 bit):
                   00:3b:77:a0:9b:91:28:06:34:9d:54:f2:72:3e:83:
                   fd:a7:26:51:c3:b1:03:cf:bb:27:1e:52:0e:14:2c:
                   16:6a:2υ:fe:16:12:44:ba:75:eb:e8:1c:9c:5b:66:
                   cb:1e:2d:81:3f:d0:29:c3:ee:f8:17:49:91:b1:3b:
                   a9:0b:95:27:cb:f5:8d:73:10:9c:d8:26:3b:1e:55:
                   3b:12:74:bb:4c:51:b4:ef:92:4a:28:1c:40:da:38:
                   d3:8c:41:88:ba:9d:62:f1:8d:d2:96:b8:e2:b4:22:
                   48:7c:d1:33:90:46:b7:e3:02:bf:1f:77:0a:7b:19:
                   1b:d3:82:0a:1f:f3:80:36:71:
               Exponent: 65537 (0x10001)
           Signature Algorithm: md5WithRSAEncryption
       03:b8:99:fc:2b:71:49:0e:2b:a5:93:50:1d:0f:e2:68:a1:b3:
       9c:88:34:d0:a1;e6:f8:39:27:67:bb:29:0c:7e:47:92:a3:dd:
       f3:7f:1b:02:59:c3:f7:c1:ad:49:c4:47:a9:87:e8:f2:73:44:
       9c:5d:b2:38:74:59:0c:84:5e:1f:9d:4c:26:86:25:ad:28:74:
       13:7d:3b:82:fb:01:ac:8a:3e:45:9a:2b:98:65:ab:3c:72:91:
       27:80:37:b6:cb:a3:44:73:26:8c:b0:8e:f8:91:25:7a:33:53:
       7c:91:33:fc:0f:65:c9:25:11:3b:79:c0:a4:20:c9:42:8a:69:
       32:0b
```

Figure 10.1 *A digital certificate*

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 looses 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 *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 fresh signing key and receive a new certificate carrying the corresponding public key. Indeed, given the sanctity of the private key used for signing, an individual should not have his/her signing key backed up by anyone else.

An important function of the PKI is to provide a safe archival facility for all issued certificates. Consider the situation where the signature on a will needs to be verified several years after the will was made. It is possible that the signer has misplaced his/her certificate or that the signer is no more. In these cases, the archival facility of a PKI would greatly help.

In the rest of this section, we discuss several PKI architectures and study the challenging issue of certificate revocation.

10.3.2 PKI Architectures

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 Fig. 10.2(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 them all especially if the type of certificate issued is the one that requires much background checking.

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 Fig. 10.2(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*. It is expected that every node in the tree will know the root CA's public key.

Suppose U1 in Fig. 10.2(b) needs U5's public key. U5's certificate is signed by CA6. First, U1 may not have CA6's public key to verify its signature on U5's certificate. Moreover, U1 does not have a trust relationship with CA6. So, instead of a single certificate issued by CA6 vouching for 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. The public key in the first certificate can then be used to verify the signature in the second certificate and so on.

In general, the trust relationships between CAs may be more numerous than in a simple tree of Fig. 10.2(b). A more dense *web of trust* is shown in Fig. 10.2(c) and is referred to as a *mesh-based PKI*. This could include mutually trusting CAs – CA1 trusting CA2 and CA2 trusting CA1 depicted by a bidirectional arc between CA1 and CA2. Note that, unlike in the tree-based PKI, there may be multiple trust paths between two users. For example, one trust path between users U1 and U7 passes through CA1, CA3, and CA4. Another trust path involves CA1, CA2, and CA4. Multiple paths provide greater resilience in the event of one or more CAs being compromised.

trust anhcor

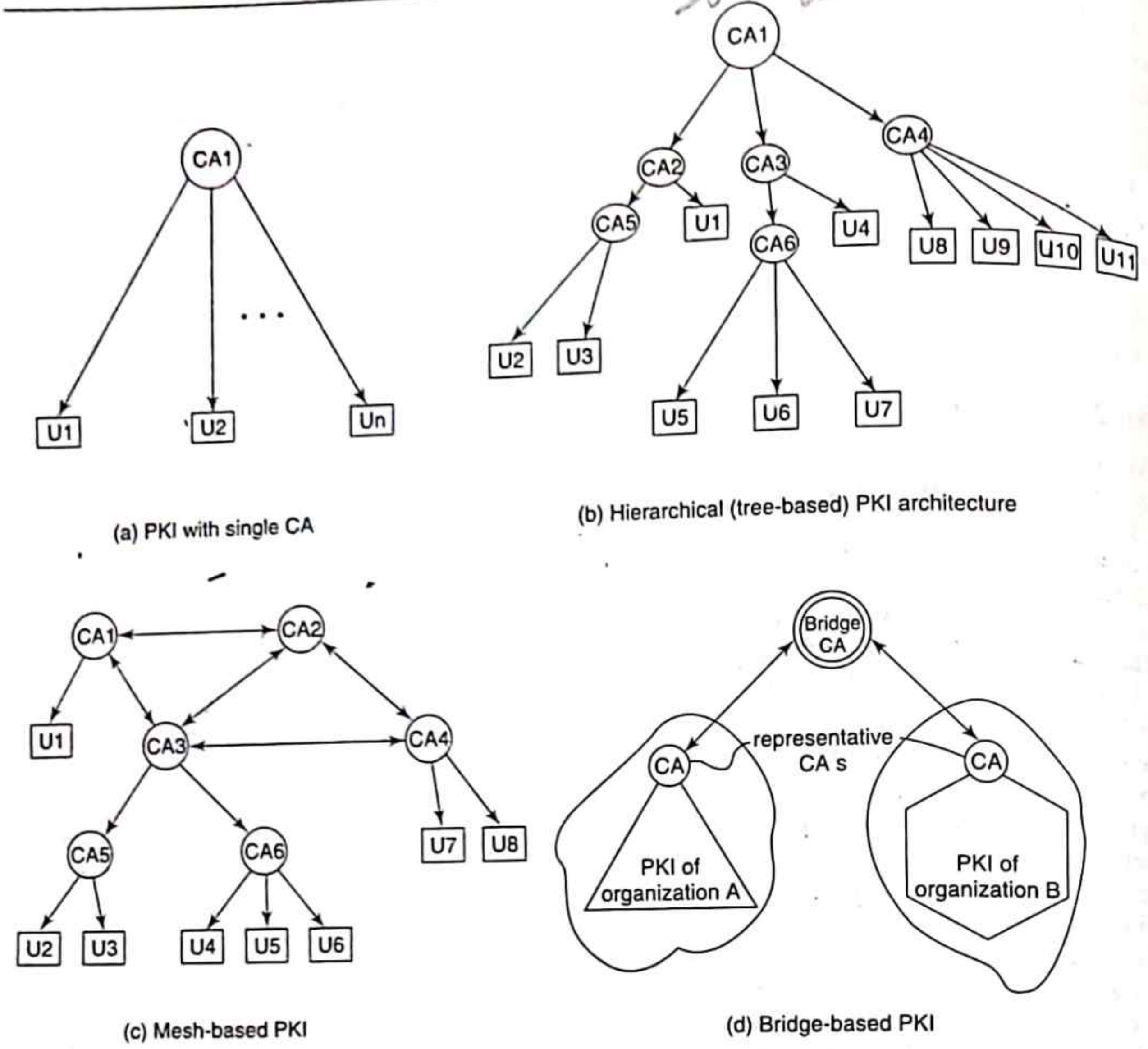


Figure 10.2 *PKI architectures*

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. In the case of an organization with a hierarchical PKI, the representative CA is the root of the tree.

Figure 10.2(d) shows a bridge CA that extends the web of trust between two existing organizational PKIs. Note that the existing PKI architectures of the two organizations have been left untouched. No inter-organizational links between CAs have been added. The only additions are the trust relationships between the representative CA of each organization and the bridge CA.

10.3.3 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's 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 to be used for key exchange/authentication and that he has made a copy of it. Note that the public key in a key exchange certificate is used by another party to encrypt a random session key. 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 erstwhile 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.

Scenario 2: Consider a single chain in a PKI (Fig. 10.3). 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. 10.3). 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.

Based on Scenario 1, we need a mechanism to revoke a certificate issued by an organization to an employee when the latter leaves or changes roles. The lesson learned from 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.

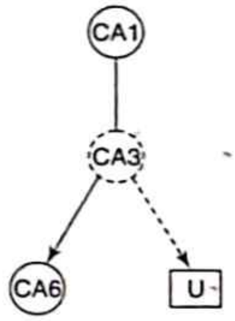


Figure 10.3 *Revocation scenario 2*

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. To an extent, this nullifies the principal advantage of digital certificates in the first place – that of obviating the need for an on-line "public key information facility."

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.

In both the above solutions, the onus is on the verifier to make sure that the certificate is currently valid. By contrast, it is possible to design a system wherein the signer requires the cooperation of a *Trusted Third Party* (TTP) in generating a signature. Such a scheme is described next.

Solution 3

Both, the signer and the TTP have a part of the private key with neither party knowing the other's part. Neither, acting on their own, can create a signature on a document. Such a scheme is proposed in [BONE04]. To sign a document, the signer would contact the TTP. Before acquiescing to cooperate in signing, 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.

Figure 10.4 summarizes the certificate revocation problem from the perspective of signature verification. The nominal validity period of a certificate is Jan 1, 2010 to Dec 31, 2010. Suppose that the private key of the certificate's subject is compromised on April 1, 2010. As a result, his certificate is revoked. Assume that the revocation information is known to the public only on April 10, 2010.

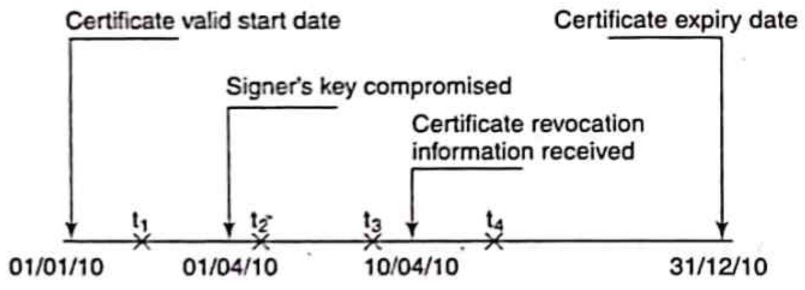


Figure 10.4 *Effect of time lag between key compromise and receipt of certificate revocation information*

We next consider four distinct cases based on when signature generation and verification occur (see Fig. 10.4 and Table 10.1).

First, if a subject has signed a document before April 1, 2010, then the verifier should accept the signature. But does he? If verification occurs at time = t_3, the verifier accepts the signature (Case 1 in Table 10.1). If, however, verification occurs at time = t_4, the verifier will not accept the signature since he has received information that the accompanying certificate has been revoked. This is Case 2 in Table 10.1.

Table 10.1 *Effect of certificate revocation lag time on signature acceptance*

Case	Signature generation time	Signature verification time	Does verifier accept signature?	Should verifier accept signature?
1	t_1	t_3	Y	Y
2	t_1	t_4	N	Y
3	t_2	t_3	Y	N
4	t_2	t_4	N	N

What happens if the signature is created after the signer's key has been compromised? We again consider two cases. Case 3 corresponds to verification at time = t_3 (before the verifier has learned of the revocation). Here the verifier accepts the signature even though he should not. On, the other hand, if verification takes place at time = t_4, the verifier rejects the signature (Case 4).

The above problems are caused due to a delay between the compromise of a signer's key and the certificate revocation information being propagated to the verifier. On the other hand, Solution 3 that involves a TTP at signing time together with a timestamp helps alleviate the problems identified here.

10.4 IDENTITY-BASED ENCRYPTION

10.4.1 Preliminaries

The digital certificate is a verifiable way of communicating the public key of a subject. Certificates are transmitted along with messages for purposes such as authentication, signature verification, and encryption.

An attractive alternative to digital certificates emerged in 1984 in the form of Identity-based Encryption (IBE). Shamir's original paper 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. While Shamir used RSA-based cryptography, this idea got a boost with work done by Boneh using *bilinear pairings* in 2001.

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. The latter are well publicized.
- 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, he 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.
- With knowledge of the PKG's public parameters and A's unique ID, anyone can compute A's public key

We next study the basics of bilinear pairings and examine how they can be used to implement IBE.

Informally, 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 of the 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, k_2 are integer constants.
A familiar example of such a mapping is the *dot product of two vectors.*

Example 10.1

Let $u = (2, 4, 1)$ and let $v = (5, 3, 2)$.
Then, $(2, 4, 1) \bullet (5, 3, 2)^T = 24$.
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,
$$\begin{aligned} (k_1 u_1 + k_2 u_2) \bullet v &= (34, 16, 23) \bullet (5, 3, 2)^T \\ &= 264 \end{aligned}$$

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

In general,
$$(k_1 u_1 + k_2 u_2) \bullet v = k_1 (u_1 \bullet v) + k_2 (u_2 \bullet v)$$

10.4.2 Use of Bilinear Pairings

The IBE scheme of Boneh and Franklin was the first practical scheme that implemented IBE. The heart of this scheme is the *bilinear pairing* defined below. It uses two cyclic groups, $\mathcal{G}$ and Γ of large prime order p. The bilinear pairing, $\beta(\mathcal{X}, \mathcal{Y})$ maps any pair of elements, $\mathcal{X}$ and $\mathcal{Y}$, from $\mathcal{G}$ to an element in Γ. $\mathcal{G}$ (an additive group) has identity $0_{\mathcal{G}}$ and Γ (a multiplicative group) has identity $1_{\mathcal{G}}$. The properties of β are as follows:

Bilinearity: For all $\mathcal{X}, \mathcal{Y}, \mathcal{Z} \in \mathcal{G}$,

$$\begin{aligned} \beta(\mathcal{X} + \mathcal{Z}, \mathcal{Y}) &= \beta(\mathcal{X}, \mathcal{Y}) * \beta(\mathcal{Z}, \mathcal{Y}) \text{ and} \\ \beta(\mathcal{X}, \mathcal{Y} + \mathcal{Z}) &= \beta(\mathcal{X}, \mathcal{Y}) * \beta(\mathcal{X}, \mathcal{Z}) \end{aligned}$$

Non-degeneracy: For a given $\mathcal{X} \in \mathcal{G}$, $\beta(\mathcal{X}, \mathcal{Y}) = 1_\Gamma$ for all $\mathcal{Y} \in \mathcal{G}$ if and only if $\mathcal{X} = 0_{\mathcal{G}}$.
Ease of computability: There exist efficient algorithms to compute $\beta(\mathcal{X}, \mathcal{Y})$ for all $\mathcal{X}, \mathcal{Y} \in \mathcal{G}$.

Once the groups, $\mathcal{G}$ and Γ and the bilinear pairing, β, are decided, the PKG proceeds to set up its public parameters. Thereafter, clients may request the PKG to generate private keys on their behalf.

PKG Parameter Set-up
The PKG chooses a *generator*, $\mathcal{P}$ of the group, $\mathcal{G}$. It also chooses its private key – a random integer, $k \in Z_p$. It then computes the corresponding public key, $\mathcal{K} = k\mathcal{P}$.
The PKG chooses two hash functions. The first, i , maps a person's ID (arbitrary length binary string representing an e-mail address, for example) to an element in $\mathcal{G}$. The second hash function, μ, maps an element in Γ to an l-bit binary string. Here, l is the length of a message block.

The two groups, $\mathcal{G}$ and Γ together with their order p, are publicly known. In addition, the generator, $\mathcal{P}$, the PKG public key, $\mathcal{K}$, the bilinear mapping, β, and the hash functions, $\imath$ and μ are all publicly known.

User Public Key and Private Key Generation

Let the client's ID be ID_A. The client contacts the PKG and requests a private key based on her ID, ID_A. The PKG first verifies whether the requester is actually the owner of the ID, ID_A. If so, the PKG computes the public key and private key pair for the requester as follows:

Public Key: $\mathcal{A} = \imath(ID_A)$ (10.1)

Private Key: $\alpha = k\ \mathcal{A}$ (10.2)

The PKG securely communicates $\mathcal{A}$ and α to A (possibly through an off-line channel).

Encryption

Suppose B wishes to send a message to A. B requests the public parameters of the PKG if he does not already have them. We assume that B knows ID_A, the ID of A (as stated earlier, this could be the e-mail address of A). To encrypt an l-bit message, m, to A, B does the following:

- B chooses a random number, $r \in Z_p$.
- B computes the following:

$$C_1 = r\mathcal{P} \qquad (10.3)$$

$$C_2 = m \oplus \mu\,(\beta(\mathcal{A}, r\mathcal{K})) \qquad (10.4)$$

- The pair, (C_1, C_2), is the ciphertext. B communicates the ciphertext to A.

Decryption

To recover the plaintext, A uses her private key, α, to compute

$$\begin{aligned} & C_2 \oplus \mu\,(\beta\,(\alpha, C_1)) & \\ =\ & C_2 \oplus \mu\,(\beta\,(k\mathcal{A}, r\mathcal{P})) & \text{from Eqs 10.2 and 10.3} \\ =\ & C_2 \oplus \mu\,(\beta\,(\mathcal{A}, kr\mathcal{P})) & \text{from bilinearity property of } \beta \\ =\ & C_2 \oplus \mu\,(\beta\,(\mathcal{A}, r(k\mathcal{P}))) & \\ =\ & C_2 \oplus \mu\,(\beta\,(\mathcal{A}, r\mathcal{K})) & \text{from definition of PKG's public key} \\ =\ & m & \text{from Eq. 10.4} \end{aligned}$$

Thus, A is able to recover the plaintext using her private key.

Well-known bilinear maps include the Weil and Tate pairings. The group, $\mathcal{G}$, is a subset of points on an elliptic curve defined over $F(q^k)$ of large prime order, p. The group, Γ, is the multiplicative group comprising the p-th roots of unity in $F(q^k)$. Details of these pairings are beyond the scope of this text.

SELECTED REFERENCES

The structure and format of digital certificates and certificate revocation lists are covered in [HOUS02]. The Online Certificate Status Protocol is described in [MYER99]. [PERL99] provides a good overview of various PKI trust models and architectures. [POLK03] elaborate on how PKIs across organizations may be linked. [GUID04] discusses their practical experiences in deploying PKI in their organization and the advantages gained by such a deployment.

Identity-based cryptography was first proposed by Shamir [SHAM85]. His proposal involved only identity-based signatures and the math was based on RSA public key cryptography. Boneh and Franklin [BONE01] were the first to use the Weil bilinear pairing for identity-based encryption.

OBJECTIVE-TYPE QUESTIONS

10.1 A digital certificate is used to bind
   (a) A person's public key to his private key
   (b) A person's public key to his identity
   (c) A person's private key to his identity
   (d) A person's signature to his private key

10.2 Which of the following is a necessary task of a Certification Authority?
   (a) Generating the public key/private key pair of its client
   (b) Performing back-up of the private key of its client
   (c) Verification of the private key of its client
   (d) Verification of the identity of its client

10.3 Which of the following field(s) in a digital certificate is/are optional?
   (a) Certificate Validity Period
   (b) Subject's Common Name
   (c) Issuer's Public Key
   (d) Issuer's Office Address

10.4 The motivation for holding a key in escrow is
   (a) encrypted documents may be decrypted using the back-up key in the event of the key being lost
   (b) encrypted documents may be decrypted using the back-up key under court order
   (c) documents can be signed in the event of the signing key being lost
   (d) the owner's private key can be used without his/her knowledge/permission

10.5 The main advantage of mesh-based PKI versus hierarchical PKI is
   (a) the certificate revocation problem is easier to handle
   (b) the certificates are simpler and easier to use in a mesh-based scheme
   (c) there are multiple trust paths between two users
   (d) mesh-based PKI scales up easily with increased number of users

10.6 Which of the following are most likely reasons necessitating certificate revocation?
   (a) Compromise of the private key of the owner
   (b) Compromise of the private key of the CA that signed the certificate
   (c) The certificate owner misplaces his/her certificate
   (d) The certificate owner quits his/her job

10.7 Which of the following is true of Identity-based Cryptography?
   (a) A user has to communicate with the PKG each time he/she wishes to encrypt his/her message
   (b) A user has to communicate with the PKG each time he/she wishes to decrypt a received message
   (c) A user's public key is a function of his/her identity
   (d) A user's private key is a function of his/her identity

EXERCISES

10.1 Examine your browser settings to determine whether it has been configured with root certificates. If so, who is the issuer, who is the subject, and what is the validity period? Can you list the certificates and make sense of its various fields?

10.2 Most browsers come pre-configured with a number of self-signed certificates. What is a self-signed certificate and what sense does it make?

10.3 Consider a scheme that involves a Signing Authority (SA) in generating a digital signature. The SA is responsible for keeping track of the validity status of its clients' public key–private key pairs. For this purpose, assume that a trusted third party generates a signing key, i.e., a public key–private key pair for Alka. The private key is split. A part of the private key is secretly conveyed to Alka and another part is secretly conveyed to the SA. Neither side reveals their part of the private key to the other. Neither Alka nor the SA can individually reconstruct the complete private key from only knowledge of their part. Consequently neither side, acting alone, can sign a message – each side requires the active participation of the other. To sign a message, the following steps are performed:

(a) Alka sends the message to the SA.

(b) The SA first checks whether Alka's public key–private key pair is still valid. If so, it checks whether a revocation request has been made on it. If not, it performs a part of the signature operation involving its part of Alka's private key. As an option, it may also include a timestamp in the original message.

(c) The SA returns the message and its part of the signature to Alka.

(d) Alka then performs operations to generate the final signature. At least one of these operations will involve her part of the private key.

Suggest a way of splitting Alka's private key between her and the SA. What are the operations that they separately perform to generate the signature?

10.4 Design a scheme to support delegation of signing rights from a person to his deputy. In particular, consider the following scenario. An executive proceeds on a 2-week vacation. During this period, he would like his deputy to sign all official documents on his behalf. When he returns from vacation, the deputy should no longer be able to sign on his behalf. How do you propose to support this feature in an organization with a PKI in place?

10.5 State the tradeoffs between mesh-based and hierarchical PKI architectures.

Organizations A and B are to be merged next month. Each has its own PKI. Recommend a strategy to unify their PKIs in the merged organization.

10.6 Identify two approaches to handling digital certificate revocation and mention the tradeoffs involved.

10.7 A mid-sized organization wishes to deploy a public key infrastructure. Do you recommend that they use digital certificates or identity-based cryptography or a hybrid? Clearly justify your recommendations.

10.8 Consider the *bilinear map*, $\beta : \mathcal{G} \times \mathcal{G} \rightarrow \Gamma$, defined in Section 10.4.2.
Show that if $\mathcal{P}$ is a generator of $\mathcal{G}$, then $\beta(\mathcal{P}, \mathcal{P})$ is a generator of Γ.
Is the converse true? Explain.

ANSWERS TO OBJECTIVE-TYPE QUESTIONS

10.1 (b)	10.2 (d)	10.3 (c)(d)	10.4 (a)(b)
10.5 (c)	10.6 (a)(d)	10.7 (c)(d)	

Chapter 11

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 occasionally referred to as the *prover*, while the party to whom proof is submitted for 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). Note that a principal is often a human though it may also be 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. We will see many examples of *cryptographic authentication* in this chapter and address biometric authentication in the next chapter.

Finally, many authentication systems today use a combination of techniques. This is referred to as *multi-factor authentication*. Authentication using the traditional passport is based on what a person has (the physical passport) in conjunction with an embedded photograph (a common biometric). New generation passports and smart cards can be used to store an individual's fingerprint. These are studied in Chapter 23. In addition, a smart card is often activated by entering a PIN on the panel of a smart card reader. Such smart cards are a vehicle for providing three-factor authentication.

11.1 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. We first discuss password-based authentication and then cover authentication using a public key–private key pair.

11.1.1 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 Fig. 11.1(a), the server uses the login name "Alka" to index into a database of *login name, password* pairs, and verify that the submitted password matches the one stored against "Alka."

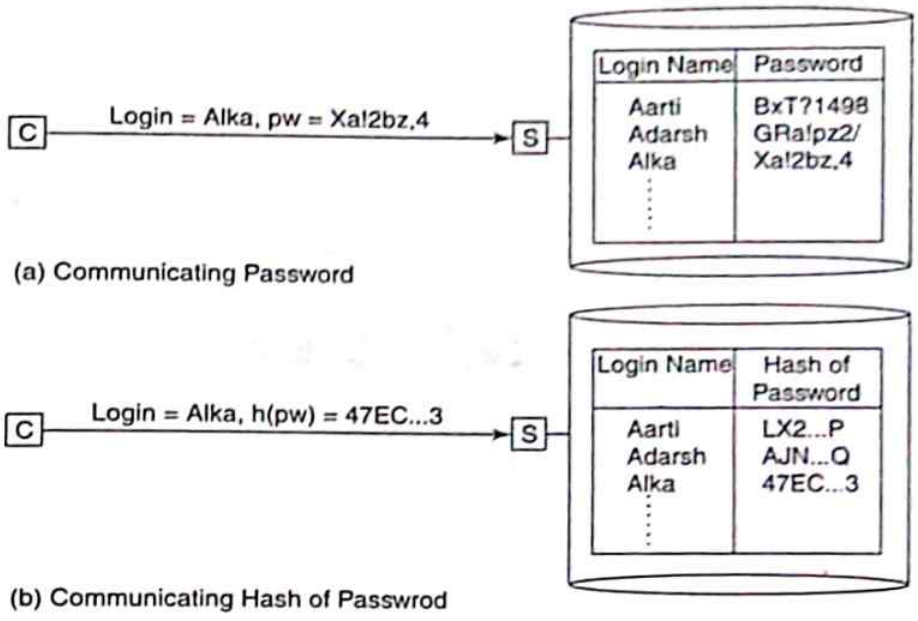


Figure 11.1 *Password-based one-way authentication*

There are two dangers associated with such an implementation. 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.

In Fig. 11.1(b), the cryptographic hash of the password rather than the password itself 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*.

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 precludes use of a previous response to answer the current challenge. Such an authentication protocol is commonly referred to as a *Challenge–Response Protocol*. CRP

Figure 11.2(a) shows a three-message one-way authentication protocol. In the first message, A conveys its identity. The second message contains the challenge from the server. The challenge is

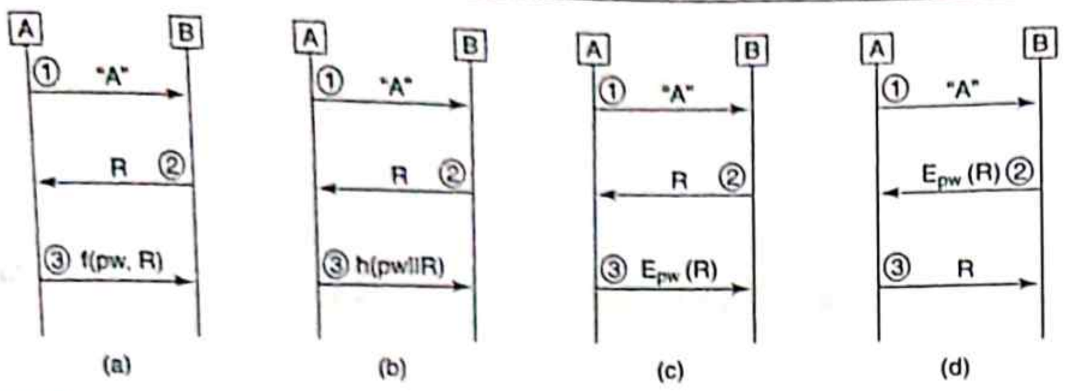


Figure 11.2 *One-way authentication using challenge–response protocol*

a random number called a *nonce* in security parlance. The third message is the client's response – a cleverly chosen function of the challenge and the password. The function, f(pw, R), has the following properties:

- 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 infeasible to compute pw
- Given an R, it should be infeasible to compute f(pw, R) even if one knows
  - f(pw, R_1), f(pw, R_2), f(pw, R_3) . . .
  - the corresponding R_1, R_2, R_3 . . .

An obvious choice for f is the cryptographic hash [Fig. 11.2(b)], which is applied over the concatenation of the password and the nonce. Another choice is a secret key encryption function with the key being the password or a function of the password [Fig. 11.2(c)]. In Fig. 11.2(d), the challenge sent by the server is an encrypted nonce. So the function f is the decryption function – the client would need to decrypt the challenge to obtain the nonce and return it to the sender to prove knowledge of his/her password.

The underlying assumption in these and other protocols are that nonces are random and non-recurring. It is the "freshness" of a nonce that precludes a replay attack. Indeed, the term nonce means "used only once." In practice, the size of a nonce is usually large, for example, 256 bits. This provides a large space from which a nonce may be selected. It should be noted that, in actual implementations, neither the sender nor receiver keeps track of nonces generated or received. The large space of nonces means that the probability of choosing the same nonce twice is infinitesimally small, assuming a "good" random number generator is employed.

We have considered protocols where the client must authenticate himself/herself to the server by proving knowledge of a secret shared between the client and server. We next study another possibility – the use of the private key–public key pair for authentication.

11.1.2 Certificate-based Authentication

A client need not share a secret with the server but may have a public key certificate. As shown in Fig. 11.3(a), A sends her certificate in Message 1. B performs certain checks such as on the validity period and name of principal. He also verifies the signature of the CA on the certificate. He then sends his challenge – a nonce R. A responds by "encrypting" the challenge with her private key. When B receives $E_{A.pr}(R)$, he "decrypts" it with A's public key and compares it with the nonce he transmitted in Message 2. If they match, he concludes that A has used the private key corresponding to the public key in her certificate. Assuming that A's private key is safely protected, she must be the entity who created the correct response in Message 3.

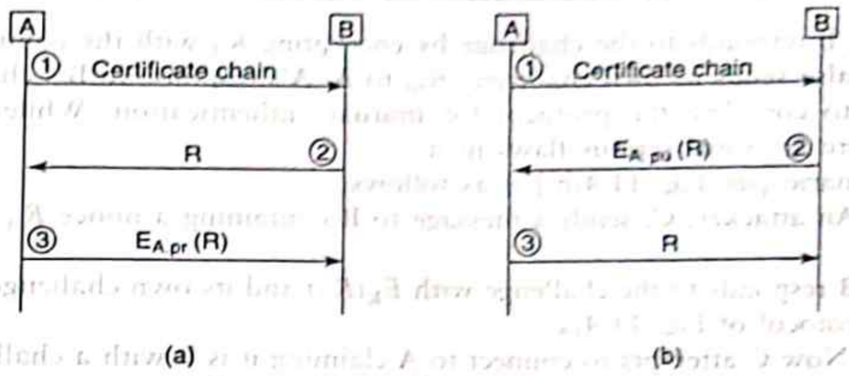


Figure 11.3 *Certificate-based one-way authentication*

Figure 11.3(b) is a slight variation of the protocol in which B chooses a nonce, R, and encrypts it with A's public key to create the challenge. A decrypts the challenge and sends it to B. Authentication of A to B succeeds if what B receives in Message 3 is R, the nonce he just chose.

11.2 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 be able to verify the identity of the customer. We next discuss mutual authentication using a secret key shared by both parties. We then use public key/private key pairs for mutual authentication. Finally, we design protocols that combine authentication and session key exchange.

11.2.1 Shared Secret-based Authentication

Figure 11.4(a) is a straightforward extension of the protocol for one-way authentication shown in Fig. 11.2(c). In Message 1, A communicates its identity and its challenge in the form of a nonce

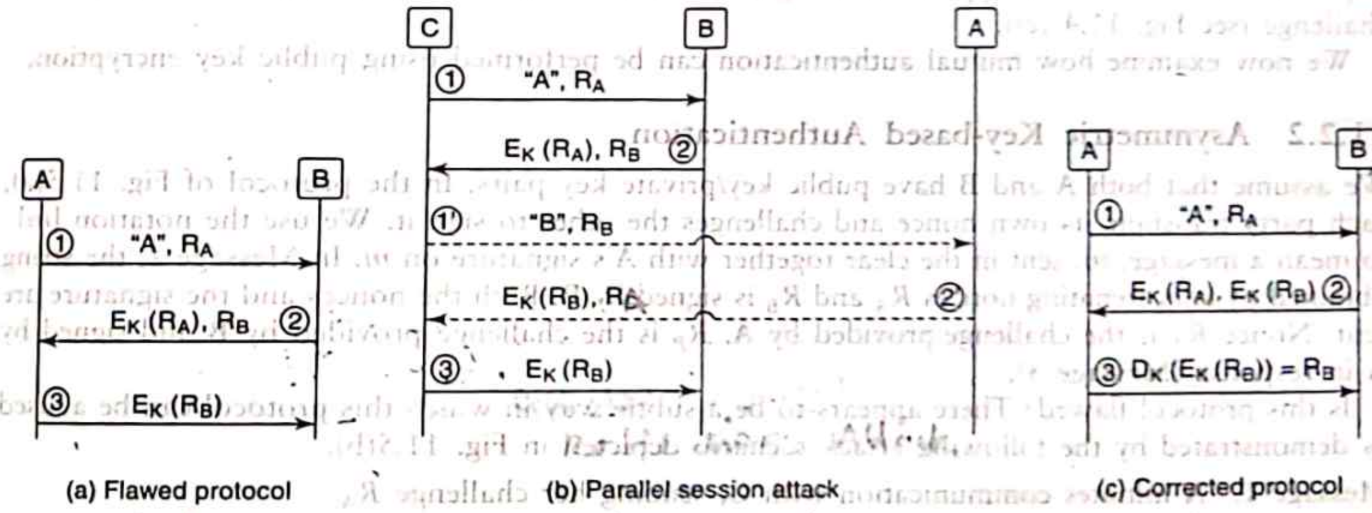


Figure 11.4 *Mutual authentication using a shared secret*

R_A. In Message 2, B responds to the challenge by encrypting R_A with the common secret, K, that A and B share. B also sends its own challenge, R_B, to A. A's response to B's challenge in the third message appears to complete the protocol for mutual authentication. While the protocol may appear sound, there are some serious flaws in it.

One attack scenario [see Fig. 11.4(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. 11.4(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.

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. However, C could not compute the response to B's challenge since that required a computation involving the secret key, K, shared between A and B. So, 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* since the attacker, in the midst of a protocol run with one entity, opens another protocol run or session with the same or another entity.

One way of thwarting reflection attacks is for the initiator and responder to draw challenges from different disjoint sets. So, in the protocol of Fig. 11.4(a), for example, A could use nonces, which are odd numbers, while B could use nonces that are even numbers. With this modification, the R_B used in Message 2 of Fig. 11.4(b) cannot be reused in Message 1'. Another possibility is to have the initiator and responder handle challenges differently. For example, the protocol might require the responder to encrypt his challenge, while the initiator would be required to decrypt her challenge (see Fig. 11.4 (c)).

We now examine how mutual authentication can be performed using public key encryption.

11.2.2 Asymmetric Key-based Authentication

We assume that both A and B have public key/private key pairs. In the protocol of Fig. 11.5(a), each party transmits its own nonce and challenges the other to sign it. We use the notation $[m]_A$ to mean a message, m, sent in the clear together with A's signature on m. In Message 2, the string obtained by concatenating nonces R_A and R_B is signed by B. Both the nonces and the signature are sent. Nonce R_A is the challenge provided by A. R_B is the challenge provided by B and signed by A in response (Message 3).

Is this protocol flawed? There appears to be a subtle way in which this protocol can be abused as demonstrated by the following attack scenario depicted in Fig. 11.5(b).

Message 1: A initiates communication with C, sending her challenge R_A.
Message 1': C initiates communication with B using the same nonce R_A supplied by A.

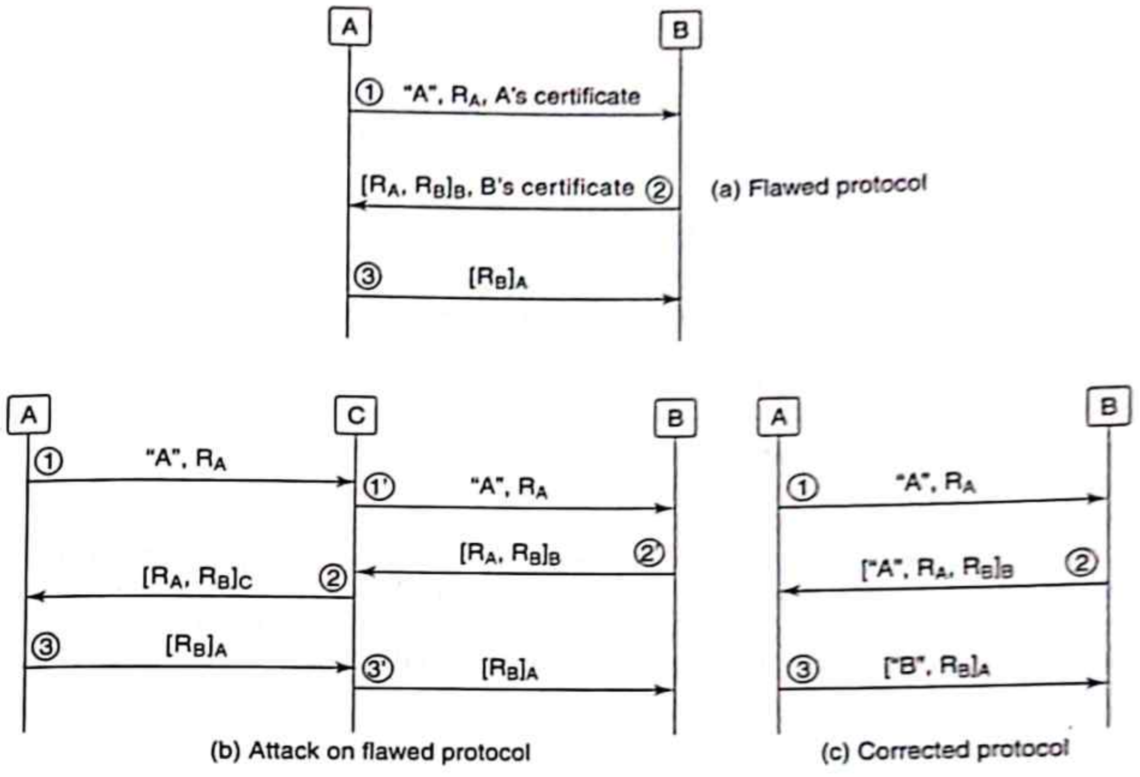


Figure 11.5 *Mutual authentication using public key cryptography*

Message 2': B responds to "A's challenge" and includes a challenge of his own, R_B.
Message 2: C responds to A's challenge and uses B's nonce, R_B, as his challenge to A.
Message 3: A responds to C's challenge (which was actually generated by B). A thus completes the mutual authentication protocol with C.
Message 4: C forwards A's response to B.

We next analyze the above protocol by attempting to determine the intentions of the three parties.

It is clear from Fig. 11.5(b) that

- A does intend to communicate with C (otherwise A would not have responded, in Message 3, to C's challenge that was transmitted in Message 2).
- B wishes to communicate with A. Otherwise, B would not have responded in Message 2' to the nonce presented in Message 1'.

Note that Message 1' is sent by C but it includes A's identity. Who is C and what sort of game is he up to? C is probably known to A. After all, A intends to talk to C. But C is also the attacker here. When A initiates communication with C, the latter seizes the opportunity (after Message 1) 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.

One solution to the above problem is for the sender to include the *identity of the recipient* in all messages signed by him. This is shown in Fig. 11.5(c). Note that with this modification, Message 3 in Fig. 11.5(b) would be ["C", R_B]$_A$. If C tries to forward this message to B, the latter will smell a rat since it is C's identity that is included in the message. So B will realize that the message was intended for C, not for him.

11.2.3 Authentication and Key Agreement

In previous sections, authentication was performed using operations involving a long-term, shared secret or a private key. It is good security practice that these keys be used sparingly to minimize the probability of compromise. Also, private key operations are notoriously expensive. If the rest of the communication needs to be integrity-protected and/or encrypted (as it often is), then short-term keys for these purposes must be agreed upon. It is expedient to derive the short-term keys or *session keys* during the authentication phase itself.

Figure 11.6 shows protocols providing both mutual authentication and key agreement. Figure 11.6(a) uses secret key cryptography, while Fig. 11.6(b) uses public key cryptography. In both the figures, 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. In Fig. 11.6(a), they are encrypted in Messages 2 and 3 by the shared secret, K. In Fig. 11.6(b), they are encrypted in Messages 2 and 3 using the recipient's public key.

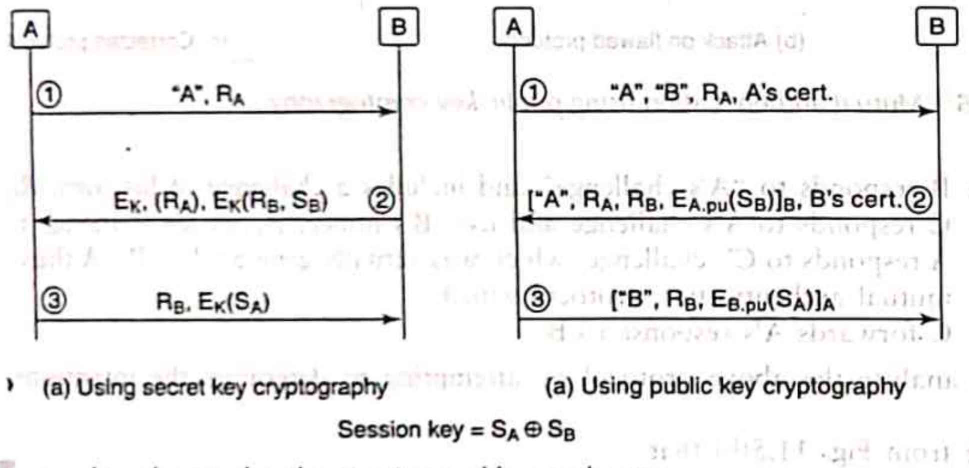

A
B
① "A", R_A
E_K. (R_A), E_K(R_B, S_B) ②
③ R_B, E_K(S_A)
(a) Using secret key cryptography
A
B
① "A", "B", R_A, A's cert.
["A", R_A, R_B, E_A.pu(S_B)]_B, B's cert. ②
③ ["B", R_B, E_B.pu(S_A)]_A
(a) Using public key cryptography
Session key = S_A ⊕ S_B

Figure 11.6 *Combined mutual authentication and key exchange*

It is possible that the session key could have been contributed exclusively by one of the communicating partners. Again however, it is good security practice that both parties contribute to the key. The key finally chosen could be a simple function of S_A and S_B, for example, $S_A \oplus S_B$.

11.2.4 Use of Timestamps

The use of nonces was introduced in Section 11.1 as a means 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. Figure 11.7 shows the use of timestamps in conjunction with public key cryptography for authentication.

- 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 signature. He increments the received timestamp, inserts it into his response message to A, and signs the message.

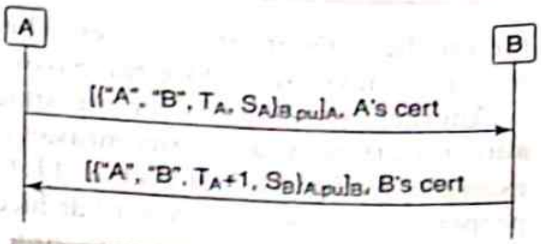

A
B
$[\{\text{"A"}, \text{"B"}, T_A, S_A\}_{B.pu}]_A$, A's cert
$[\{\text{"A"}, \text{"B"}, T_A+1, S_B\}_{A.pu}]_B$, B's cert

Figure 11.7 *Mutual authentication with timestamps*

The notation $\{m\}_{X.pu}$ denotes a message, m 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.

11.3 DICTIONARY ATTACKS

11.3.1 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
sY,u!	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
atimuhdam	Permutation of letters in mother's or spouse's name, Madhumita, is a poor choice especially if the attacker has personal information about victim
Kolkata	Place names are often part of password dictionaries

There are two types of dictionary attacks – on-line and off-line. 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 (the choice of password is particularly naive), an on-line attack has a severely limited chance of success.

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 in the password file 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 client authentication. As a security measure, a one-way function of the password may be transmitted. For example, in Fig. 11.2 of Section 11.1, the user transmits f(*pw*, *R*). Again, because of the one-way property of f, it is non-trivial to deduce the password, *pw* knowing *R* and f(*pw*, *R*). However, armed with the password file or with f(*pw*, *R*), the attacker could use his/her dictionary of potential 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 n 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 is D[i]")
                found = true
        }
}
```

Harnessing the power of a supercomputer or a distributed set of desk-tops greatly increases the attacker's chance of success.

11.3.2 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,

$$h(\ldots h(h(D[i], R))\ldots)$$

If the above function were used for f(*D*[*i*], *R*) in the loop of the program, we would expect the program to run about 100 times slower. For a given amount of computational power, the attacker's chances of guessing the password would decrease commensurately.

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. Recall that 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.

Figure 11.8 shows the four messages that are exchanged in EKE. After Message 2, both sides should be able to compute the new session key = g^{ab} mod p denoted by K in the figure. Mutual authentication is accomplished using the now 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.

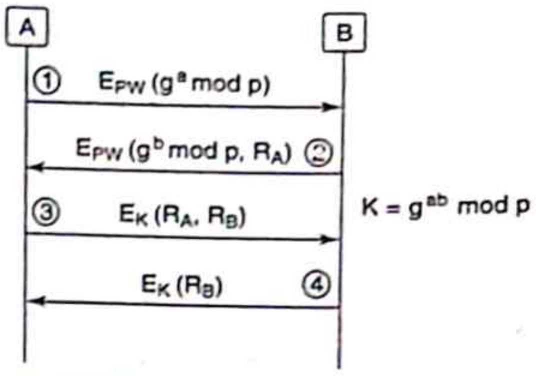


Figure 11.8 *EKE protocol*

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 such as those in Fig. 11.2 could succumb to an off-line dictionary attack. Could a similar fate await EKE?

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 mod p and g^b mod p. But even so, he/she would *not* be able to obtain the session key, g^{ab} mod p. This is so, since the computational Diffie–Hellman problem is infeasible in large groups that are carefully chosen, that is, the common secret g^{ab} mod p computed by both parties cannot be obtained from knowledge of the partial keys, g^a mod p and g^b mod p. 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. The long term secrets include their private keys (if any) and secrets shared between A and B such as a password.

Perfect forward secrecy in EKE follows from the fact that knowing the long-term secret both sides share does not help in determining the session key. Once again, the long-term secret shared by the two parties (client and server) is the client's password. The two partial secrets, g^a mod p and g^b mod p, are both, encrypted using a function of the password. If the password is stolen, the values of g^a mod p and g^b mod p can be obtained. But because of the infeasibility of the computational Diffie–Hellman problem as stated earlier, these will not enable us to deduce the session key g^{ab} mod p.

SELECTED REFERENCES

[BURR90] is a formal treatment of authentication protocols, while [ABAD96] is a more recent paper highlighting a number of principles and guidelines for designing cryptographic protocols. [KAUF02] and [MAO03] are two books in the area of network security/cryptography with extensive coverage of various cryptographic protocols.

OBJECTIVE-TYPE QUESTIONS

11.1 The use of which of the following provides two-factor authentication?
- (a) Digital image of a person's fingerprint stored on an electronic passport
- (b) PIN-enabled chip card for electronic payment (basically a smart card)

(c) Use of login name + password
(d) Driver's license + national ID card

11.2 Storing the hash of a password rather than the password itself on the authentication server exploits the following property of the cryptographic hash to enhance security
(a) strong collision resistance (b) one-way property
(c) weak collision resistance (d) easy computability of the hash

11.3 Entity authentication is used to protect against
(a) session hijacking (b) impersonation
(c) replay attacks (d) identity theft

11.4 The EKE protocol is resistant to
(a) replay attacks (b) man-in-the-middle attacks
(c) dictionary attacks (d) reflection attacks

11.5 Which of the following measures is/are effective in thwarting on-line dictionary attacks?
(a) Store the hash of the password on the authentication server
(b) Send the hash of the password over the communication link
(c) Client responds to server nonce by computing a one-way function of the nonce and the password
(d) Server disables login after four incorrect login attempts

11.6 Which of the following measures may be effective in thwarting off-line dictionary attacks?
(a) Storing encrypted passwords on the authentication server
(b) Repeatedly performing the hash of passwords prior to storage
(c) Protecting the communication link against eavesdropping
(d) None of the above

11.7 Which of the following is/are true about nonces/timestamps?
(a) A nonce may or may not be predictable
(b) Use of timestamps requires fewer number of messages compared to nonce-only authentication protocols
(c) Server keeps track of nonces it has used in the past
(d) Timestamp-based protocols are sensitive to clock skew

EXERCISES

11.1 In this chapter, we studied protocols for both one-way and mutual authentication. In the real world, can you think of applications where
(a) client-to-server authentication is mandatory but server to-client authentication is not?
(b) server-to-client authentication is mandatory but client-to-server authentication is not?
(c) both server-to-client and client-to-server authentication are mandatory?

11.2 Design a protocol that achieves mutual authentication and key agreement between two parties using the minimum number of messages. Assume that both sides share a secret. Also, assume that both sides have the ability to compute cryptographic hashes but have NO support for performing symmetric key or asymmetric key encryption/decryption.

11.3 Is it possible to design a protocol that accomplishes both authentication and (session) key exchange with only TWO messages and without timestamps? Explain. Consider each of the following cases separately:
(a) The two parties share a common long-term secret.

(b) Both communicating parties have a public key–private key pair. Each party knows the other's public key.

11.4 In previous chapters, we studied various applications of public key cryptography. These include encryption, signing, and session key exchange. In this chapter, we encountered yet another application – authentication.

Why is it necessary to have separate keys (and associated certificates) for signing and for authentication?

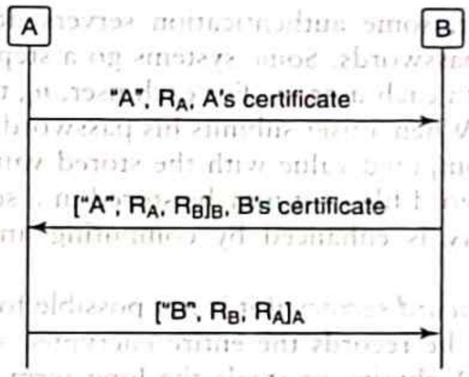
Is it necessary to have yet another public key–private key pair and associated certificate for session key exchange? Explain.

11.5 Two parties, each have Diffie–Hellman certificates. Can they use these certificates to perform mutual authentication? If yes, explain how, else explain why not.

11.6 The protocol below was designed for mutual authentication. Assume that, both, A and B have signing certificates.

(a) With the help of a neat sketch, show how an attacker may impersonate A. Clearly state the conditions under which the attack will succeed.

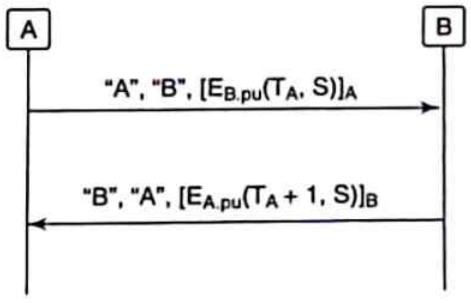
(b) Fix the protocol so that it is secure.



11.7 The protocol in the figure below was designed for mutual authentication and key agreement. Assume that both, A and B have public key–private key pairs with associated certificates for encryption and signing. Also, assume that both parties have access to the certificates of the other even before this run of the protocol.

(a) With the help of a neat sketch, show an attack scenario on the protocol. Clearly state what the attack has accomplished. In the figure, T_A and S are, respectively, the timestamp and session key.

(b) Fix the protocol so that it is secure.


A
B
"A", "B", [E_B.pu(T_A, S)]_A
"B", "A", [E_A.pu(T_A + 1, S)]_B

11.8 Consider the following password-based authentication protocol.

- A chooses a password, *pw*, and a large number, *n*, say 1000.
- The user registration software computes hash^n(*pw*). The notation hash^n(*pw*) refers to $\underbrace{\text{hash (hash (hash} \ldots pw) \ldots))}_{n \text{ hashes}}$. This value and *n* are stored on the server against A's log-in name.
- To log in, A types her name and pw. A's workstation informs the server that she wishes to log in.
- The server responds by sending *n*.
- A's station computes hash^{n-1}*(pw)* and sends it to the server.
- The server performs a further hash computation on the received value, hash^{n-1}*(pw)* and compares whether this matches with what it has stored against A's name. If they match, the server considers A authenticated. It replaces hash^n*(pw)* with hash^{n-1}*(pw)* and decrements *n*.
- When $n = 1$, A needs to get a new password.

Create an attack scenario wherein, under the right conditions, an attacker successfully impersonates A.

11.9 As mentioned in the text, some authentication servers do not store user passwords but instead the hash of user passwords. Some systems go a step further and associate a random number, called *salt*, s_i, with each user, u_i. For each user, u_i, the system stores the value, $h(pw_i \,\|\, s_i)$ in the password file. When a user submits his password, pw_i, the system computes $h(pw_i \,\|\, s_i)$ and compares the computed value with the stored value. The value, s_i for each user is stored, either in the password file or it may be stored in a separate file. In each case, explain whether and why security is enhanced by computing and storing salted hashes of user passwords.

11.10 A protocol has *perfect forward secrecy* if it is not possible for an attacker to decrypt a session between A and B even if he records the entire encrypted session and then at a later point in time (say a week later) obtains or steals the long term secrets of A and B.

Design a protocol with the property of perfect forward secrecy that provides mutual authentication and session key agreement between A and B. Your design should be economical in the number of messages.

ANSWERS TO OBJECTIVE-TYPE QUESTIONS

11.1 (a)(b)	11.2 (b)	11.3 (b)	11.4 (c)
11.5 (d)	11.6 (a)(b)(c)	11.7 (b)(d)	

Chapter 12

Authentication–II

In the first part of this chapter, we continue our discussion of authentication protocols. We introduce the idea of a 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—are studied. We then look at the biometric authentication as a complement to and, in some cases, as a substitute for cryptographic authentication.

12.1 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. With secret key cryptography, however, an entity must share a key with each party it wishes to communicate with. 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 non-trivial task.

One approach to alleviating the risk is to employ a *trusted third party* 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. 12.1, A informs the KDC that it intends to communicate with B (Message 1). The KDC generates a random secret, K_{AB}, and dispatches this to A and B through two encrypted messages

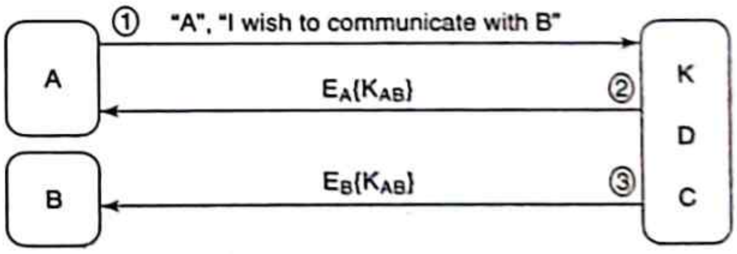


Figure 12.1 *Message confidentiality using a KDC*

(Messages 2 and 3). (Throughout this chapter, $E_A\{m\}$ denotes a message encrypted using A's long-term secret shared with the KDC. $E_{AB}\{m\}$ denotes a message encrypted using the session key shared between A and B.) Message 2 is encrypted using the long-term secret, K_A, that A shares with the KDC. Likewise, Message 3 is encrypted with K_B, the secret shared between B and the KDC.

Both sides decrypt their messages and obtain the *short-term session key*. A and B then encrypt all subsequent messages during the session using K_{AB}. We refer to the encrypted session key that B receives in Message 3 as a *ticket*.

Figure 12.1 was meant to convey the general idea in using a KDC but the protocol is susceptible to numerous types of replay or man-in-the-middle attacks. The protocol we study next is intended to thwart all such attacks.

12.2 THE NEEDHAM–SCHROEDER PROTOCOL

Figure 12.2(a) enhances the protocol of Fig. 12.1 to provide mutual authentication by including a challenge–response phase (Messages 3, 4, and 5). 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}.

In addition, the KDC encloses the ticket to B in its message to A (Message 2). A then forwards the ticket together with her challenge to B in Message 3. We next study a couple of attacks, all aimed at impersonating either A or B.

12.2.1 Preliminary Version 1

The protocol in Fig. 12.2(a) is susceptible to an impersonation attack shown in Fig. 12.2(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.

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

$$E_A\{K_{AB}, \text{“B”}, E_B\{\text{“A”}, K_{AB}\}\}$$

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. The modified protocol is shown in Fig. 12.3(a).

The previous attempt at impersonation involves a man-in-the-middle attack. Impersonation can also be a consequence of replay attacks in scenarios involving compromised passwords or long-term keys. Trivially, a compromised key opens the door to impersonation attacks. Any security protocol is susceptible to such attacks, not just the current one.

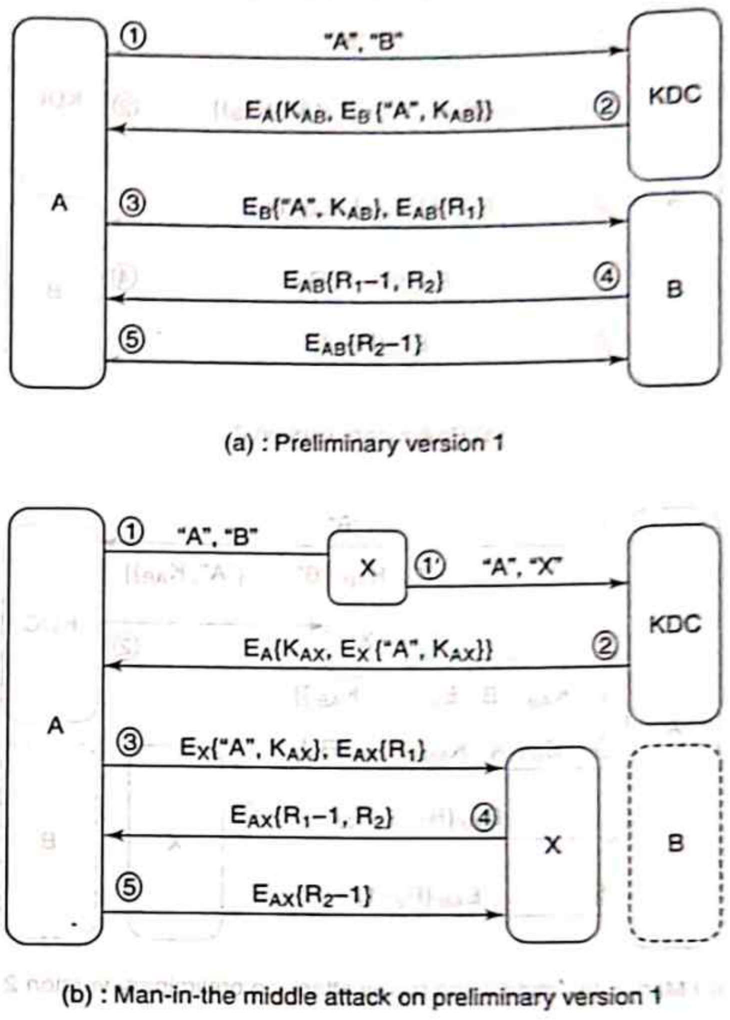

① "A", "B"
② $E_A\{K_{AB}, E_B\{"A", K_{AB}\}\}$
KDC
A
③ $E_B\{"A", K_{AB}\}, E_{AB}\{R_1\}$
④ $E_{AB}\{R_1-1, R_2\}$
⑤ $E_{AB}\{R_2-1\}$
B
(a) : Preliminary version 1
① "A", "B"
X
①' "A", "X"
KDC
② $E_A\{K_{AX}, E_X\{"A", K_{AX}\}\}$
A
③ $E_X\{"A", K_{AX}\}, E_{AX}\{R_1\}$
④ $E_{AX}\{R_1-1, R_2\}$
⑤ $E_{AX}\{R_2-1\}$
X
B
(b) : Man-in-the middle attack on preliminary version 1

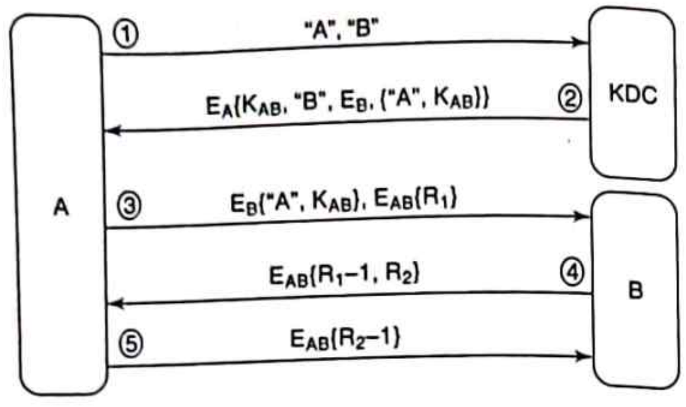
Figure 12.2 *Needham–Schroeder protocol: Preliminary version 1*

The user of a system must make every effort to protect his/her password. However, it is unreasonable to expect that every single user of this system keeps his/her password perfectly secure. What happens if a user's password gets compromised? In that case, the user should immediately intimate the KDC about the loss. He/she then chooses a new password from which is computed a new long-term key which is used thereafter. We next see that, even in the event that a user behaves in such a responsible fashion, he/she can be impersonated.

12.2.2 Preliminary Version 2

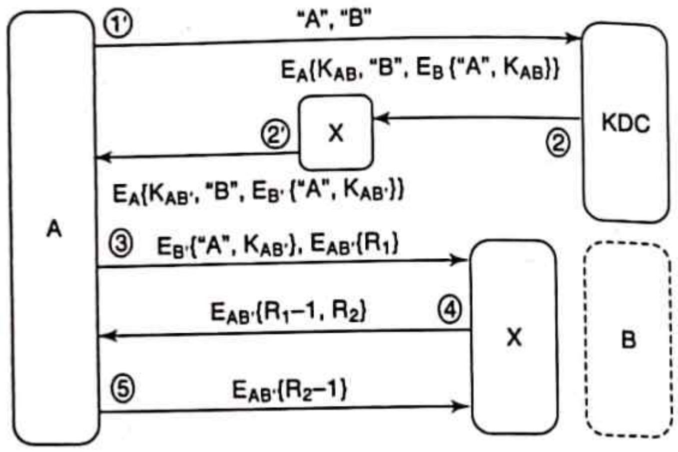
The previous attack caused B to be impersonated to A. However, despite the fix suggested above, another attack with the same effect can be launched. A determined attacker, X, does the following:

- X eavesdrops upon and meticulously records many of A's sessions with the KDC and with B over a period of time.
- He then steals B's password or long-term key.



① "A", "B"
② E_A{K_AB, "B", E_B, {"A", K_AB}}
KDC
A
③ E_B{"A", K_AB}, E_AB{R_1}
④ E_AB{R_1−1, R_2}
B
⑤ E_AB{R_2−1}

(a) Preliminary version 2



①' "A", "B"
E_A{K_AB, "B", E_B {"A", K_AB}}
X
②'
②
KDC
E_A{K_AB', "B", E_B' {"A", K_AB'}}
A
③ E_B'{"A", K_AB'}, E_AB'{R_1}
④ E_AB'{R_1−1, R_2}
X
B
⑤ E_AB'{R_2−1}

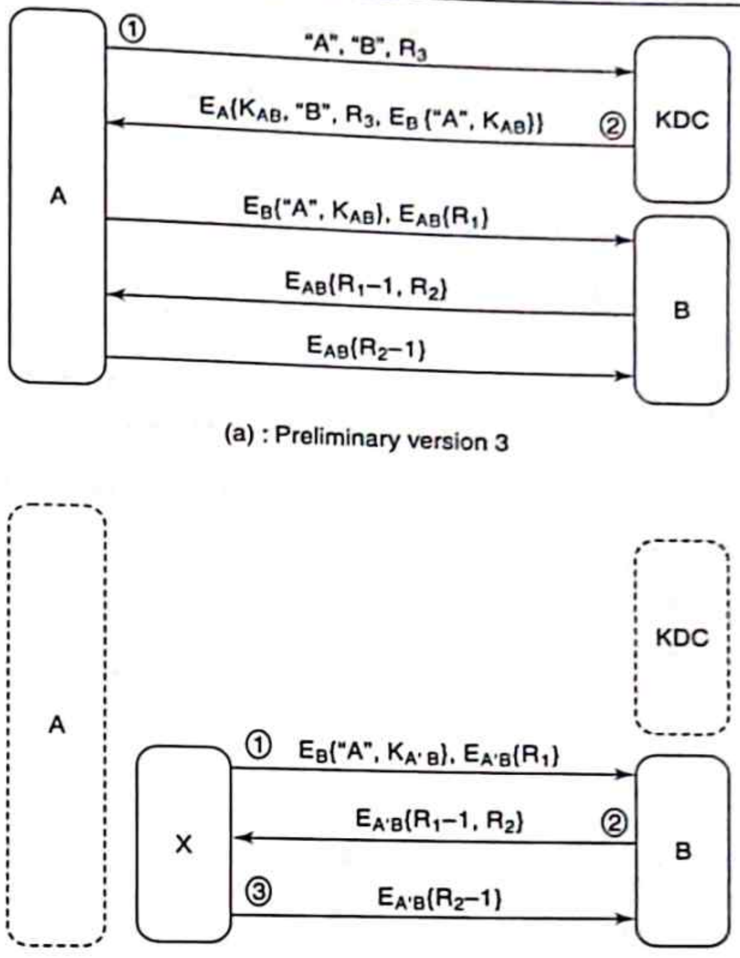
(b) Man-in-the middle and replay attack on preliminary version 2

Figure 12.3 *Needham–Schroeder protocol: Preliminary version 2*

In the most optimistic scenario, 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 so, the following scenario shows X successfully impersonating B to A.

1. A wishes to communicate with B and sends Message 1 in Fig. 12.3(b).
2. X intercepts the KDC's response (Message 2) and instead plays a previous recording of Message 2. X is careful to replay a copy of Message 2, which he recorded before B's key was compromised. Note that this message contains a ticket encrypted with B's old key, $K_{B'}$.
3. 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, $K_{AB'}$.
4. 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.

The above impersonation attack succeeds because the attacker could replay Message 2. We can fix this vulnerability by ensuring the *freshness* of Message 2. This is accomplished by A sending a



①
"A", "B", R_3
$E_A(K_{AB}$, "B", R_3, E_B("A", K_{AB}))
②
KDC
A
E_B("A", K_{AB}), $E_{AB}(R_1)$
$E_{AB}(R_1-1, R_2)$
B
$E_{AB}(R_2-1)$
(a) : Preliminary version 3
KDC
A
① E_B("A", $K_{A'B}$), $E_{A'B}(R_1)$
$E_{A'B}(R_1-1, R_2)$ ②
X
B
③ $E_{A'B}(R_2-1)$

(b) : Replay attack on preliminary version 3

Figure 12.4 *Needham–Schroeder protocol: Preliminary version 3*

(fresh) nonce in Message 1 [Fig. 12.4(a)] and receiving confirmation of its receipt by the KDC. The latter includes the nonce in an encrypted response in Message 2 of Fig. 12.4(a).

12.2.3 Preliminary Version 3

The next version of the protocol of Fig. 12.4(a) 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.

To create such an opportunity, 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. Nevertheless, an impersonation attack can occur at a later point in time.

Consider the case where the attacker X has been recording messages between A and the KDC. X then manages to steal A's long-term key that she shares with the KDC. Assume that, between the compromise of her key and her obtaining a fresh replacement, she is not a party to any communications. We next show how X can still perform an impersonation attack.

Consider Message 2 in Fig. 12.4(a) that was recorded by X before A's key was compromised. The content of this message is

$$E_{A'}\{K_{A'B}, \text{"B"}, R_3, E_B\{\text{"A"}, K_{A'B}\}\}$$

Since the above message was recorded prior to the compromise of A's key, it is encrypted with A's old key, $K_{A'}$. Using the compromised key, X can decrypt this message and recover the

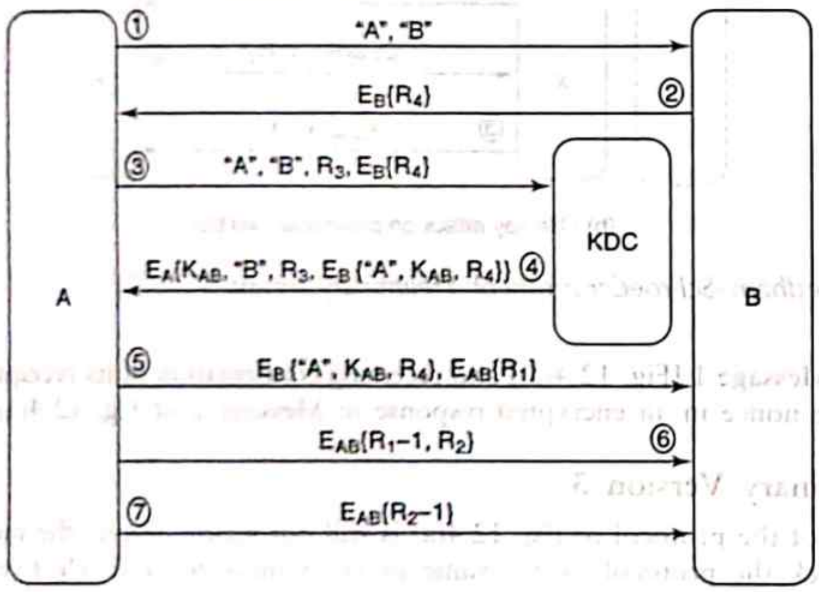
- old session key, $K_{A'B}$ used then and
- the old ticket, $E_B\{\text{"A"}, K_{A'B}\}$ dispatched to B.

To impersonate A, X does the following [see Fig. 12.4(b)]:

- X sends, in Message 1 to B, the old ticket and a challenge, R_1, encrypted with the old session key.
- B responds to X's challenge and also communicates his own challenge, R_2.
- Because X has the session key, he responds to the challenge by encrypting R_2 with the old session key.

B receives the response and is convinced he is talking to A when, in fact, he is talking to X.

The replay of an old ticket is responsible for this attack. This problem could be fixed if B were allowed to choose a nonce [Message 2 of Fig. 12.5] and the *same* nonce were enclosed by the KDC in the ticket it generates. B would have to retain his nonce for a short while until he received the ticket. By verifying that the nonce enclosed in the ticket matches the one he had just generated, he could guard against a replay attack. The final bug-free version of the Needham–Schroeder protocol is in Fig. 12.5.



A
B
KDC
① "A", "B"
② E_B{R_4}
③ "A", "B", R_3, E_B{R_4}
④ E_A{K_AB, "B", R_3, E_B{"A", K_AB, R_4}}
⑤ E_B{"A", K_AB, R_4}, E_AB{R_1}
⑥ E_AB{R_1−1, R_2}
⑦ E_AB{R_2−1}

Figure 12.5 *Needham–Schroeder protocol: Final version*

12.3 KERBEROS

Consider a scenario with multiple users and multiple servers in an organization such as a university campus. A user, once logged in, may then wish to access different resources such as an e-mail server or a file server in the course of that login session. One possibility is for the user to have multiple passwords on each of these servers. However, having humans remember and update multiple passwords is not practical. A user could use the same password for all servers but distributing and maintaining a password file across multiple servers is a security risk.

A password-based system should ensure the following:

- The password should not be *transmitted in the clear.*
- It should not be possible to launch *dictionary attacks* using the eavesdropped- upon messages containing a function of the password.
- The password itself should not be stored on the authentication server, rather it should be cryptographically transformed before being stored. Further, it should not be possible to launch dictionary attacks by obtaining a file containing cryptographically transformed versions of the password.
- 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.*
- The password should reside on a machine for only a few milliseconds after being entered by the user. Thereafter all vestiges of the password should be destroyed (an image of the password should not linger on in either volatile or non-volatile memory of the machine).

As we will soon see, the Kerberos protocol elegantly addresses each of these issues.

Developed at MIT, Kerberos has been through many revisions. The latest is Kerberos Version 5. There continue to be several enhancements to this protocol such as support for AES, etc. It is a widely used protocol supported on several Windows Operating Systems including Vista and Microsoft Server 2008.

Some of the ideas in the Needham–Schroeder protocol are re-used in Kerberos. 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). As in the Needham–Schroeder protocol, the ticket is the mechanism used to safely distribute session keys. Each human user, A, of the system shares a secret, K_A, with the AS. This secret is essentially the hash of the user's password. Likewise, each server, B, shares a secret K_B with the TGS. Unlike the Needham–Schroeder protocol, Kerberos makes use of *timestamps.*

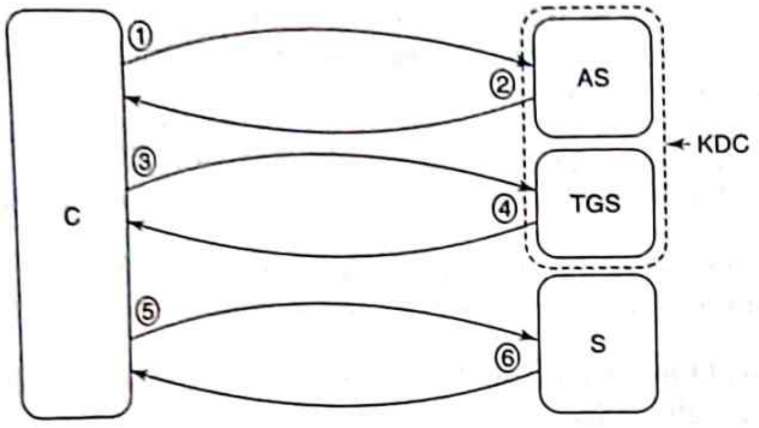
One possibility is for the *KDC* to authenticate each user and record the time of login, the maximum duration of a session and the session key. Thereafter, for each request for service from a logged-on user, the *KDC* could create tickets for the user and the requested server. Each ticket would contain the encrypted session key and related information such as the lifetime of the session key. The session key would be used for mutual authentication between the user and the requested server and for possible encryption of all subsequent communication between the two.

In Kerberos, however, *session state information* is not maintained on the KDC but rather in tickets issued by the AS. The AS is involved in authenticating a user when she attempts to log on to the system. The AS then issues a *ticket-granting ticket* (TGT) to the user.

For each different server that needs to be accessed, the client approaches the TGS with the TGT. The TGS first authenticates the user and then provides the user with a *service-granting ticket* to be handed over to the server whose service is required.

For reasons of efficiency and scalability, the AS and TGS (which have logically distinct functions) may be implemented on separate machines. Indeed, the TGS functionality could be shared by multiple machines under heavy load. The TGS could also play the role of a *policy server* – determining whether the request of a user for a particular service may be accepted or not.

The sequence of messages exchanged between the client (C), the Kerberos servers (AS and TGS) and the requested server (S) is shown in Fig. 12.6. There are three steps – each involving two messages as discussed next.



① C request Ticket-Granting Ticket
② C receives Ticket-Granting Ticket
③ C request Service-Granting Ticket
④ C receives Service-Granting Ticket and session key
⑤ C authenticates itself to S
⑥ S authenticates itself to C

Figure 12.6 *Kerberos message sequence*

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 {"TGS", $K_{C,TGS}$, *Times*, R_1}

where

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

In Message 1, the client informs the AS that it wishes to communicate with the TGS. The "*Times*" field specifies the start time and expected duration of the login session. Note that the ID of the client, denoted "C," is really the ID of the user who has logged in. (The user and client station are not differentiated in this discussion.) R_1 is a nonce generated by 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. Recall that this key is a function of the user's password. As in the previous section, the notation $E_C\{m\}$ represents a message *m* encrypted with the long-term key of C. AS encrypts and returns the nonce, R_1, that it received in Message 1. As before, the nonce is used to prevent replay attacks.

The AS also includes a TGT in connection with C's request. The ticket contains the fresh session key, $K_{C,TGS}$, and is encrypted using the long-term key shared between the AS and TGS.

Step 2: Receipt of Service-Granting Ticket

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

where

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

Message 4 TGS → C: "C", Ticket_S, $E_{C,TGS}$ {"S", $K_{C,S}$, *Times*, R_2}
where
$\text{Ticket}_S = E_S$ {"C", $K_{C,S}$, *Times*}

In Message 3, C forwards the TGT to the TGS from where the TGS extracts the session key, $K_{C,TGS}$, known only to C and the TGS. C proves knowledge of this session key by creating an *authenticator*. As shown above, the authenticator encrypts the current time (timestamp) using $K_{C,TGS}$. As before, the notation $E_{C,TGS}${...} represents a message encrypted by the fresh session key shared between C and the *TGS*.

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, R_2, from C is also encrypted by the TGS using $K_{C,TGS}$. This convinces C that the received message is from the TGS and is not a replay of a message from a previous session. 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: Ticket_S, Authenticator_C
where
$\text{Authenticator}_C = E_{C,S}$ {"C", TS_2}
Message 6 S → C: $E_{C,S}$ {TS_2 + 1}

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 $K_{C,S}$. 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 since its creation requires knowledge of $K_{C,S}$. Use of the timestamp also prevents replay attacks.

12.4 BIOMETRICS

12.4.1 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 gait 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.

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 stored to increase the accuracy of a match in the subsequent recognition phase.

Biometric templates are typically stored on an Authentication Server. They may also be securely stored in a person's smart card or e-passport (Chapter 23), thus permitting "off-line" authentication without the need to access a central server.

Recognition. In this phase, a fresh biometric sample of the person is obtained. As in the enrolment phase, a biometric template of the person is created. This is then compared with the reference templates (created during enrolment) to determine the extent of a *match*.

Biometrics is used in at least two different situations.

Authentication or Identity Verification. Authentication servers store pairs of the form (login name, password). In a similar manner, a biometric system stores (login name, biometric sample) pairs. During a login attempt, a biometric sample (such as a fingerprint scan) of the user is taken. The biometric sample is compared with the sample stored on the server. The user is authenticated only if a match between the two occurs.

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. For example, suppose that a suspected criminal has just been nabbed. His biometric (such as a photograph or fingerprint) will have to be compared against a database of biometric records of thousands of criminals to determine if a match occurs.

While authentication involves a *one-to-one* match, identification involves a *one-to-many* match and is consequently more challenging (Fig. 12.7). 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. This is not always possible with common biometrics used. 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.

Other issues in the choice of biometric are the *ease* with which a biometric sample may be collected, the *robustness* of the biometric and the *speed* and *accuracy* of a match. Also, it should not be possible to *circumvent* the system or fool it by the equivalent of a forged signature. Finally, *acceptance* by the public and their active cooperation are indispensable to make the system work.

12.4.2 ERROR MEASURES

For a given biometric, let $d(t_1, t_2)$ be a measure of the *distance* (non-match) between two templates, t_1 and t_2. The smaller the value of $d(t_1, t_2)$, the closer is the match.

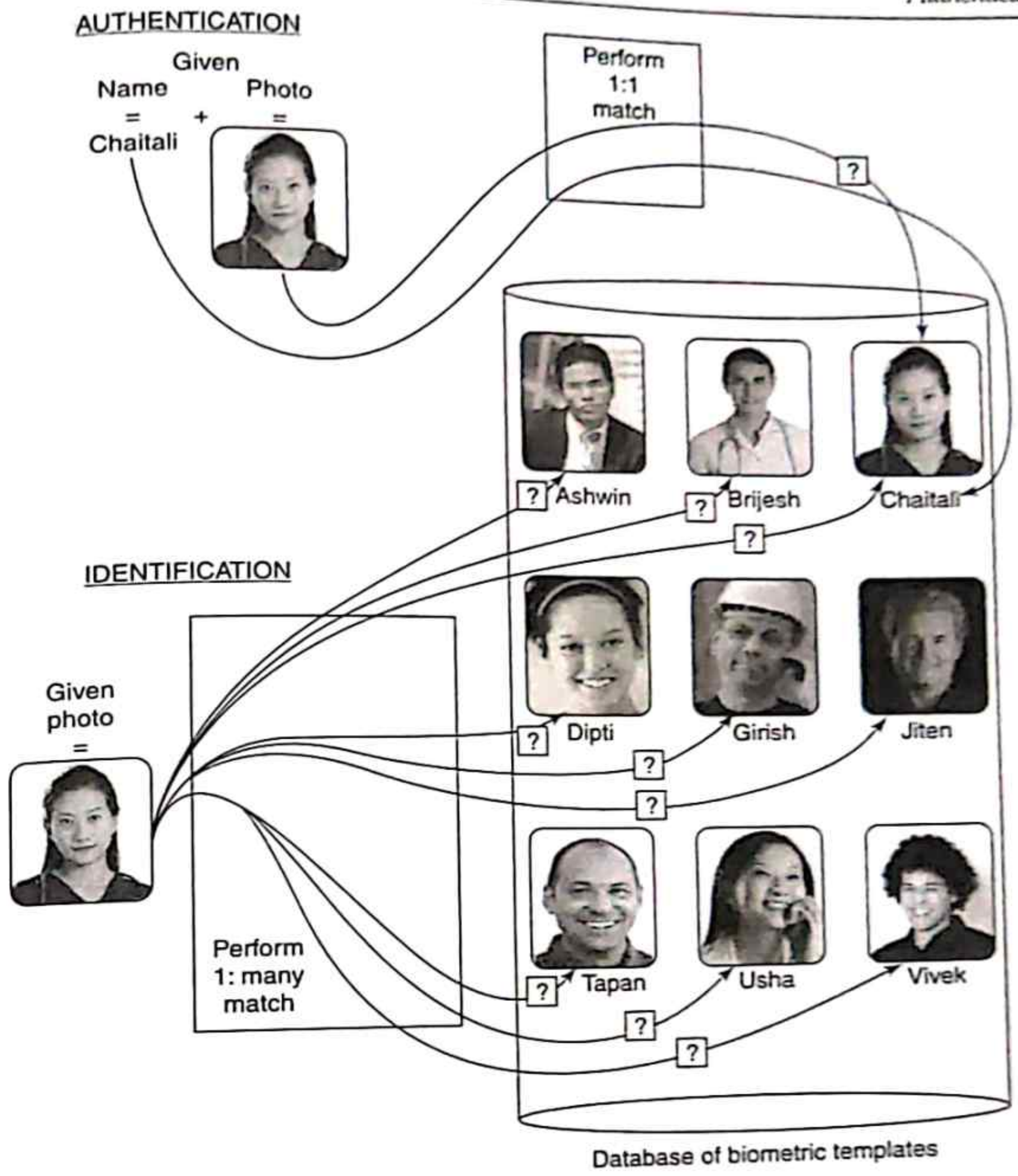


Figure 12.7 *Authentication versus identification*

Example 12.1

In the case of iris recognition, an iris scan is processed to create a 2048-bit string called the iris code. The distance between two iris codes, t_1 and t_2, is simply their *Hamming distance* (the number of positions in which t_1 and t_2 differ by).

Let d_0 be the random variable representing the distance between two different templates computed from two biometric samples of the *same individual*.

Let d_1 be the random variable representing the distance between two templates computed from the samples of two *different individuals*.

Let f_{d_0} and f_{d_1} be respectively the probability mass functions (pmf) of the random variables, d_0 and d_1, respectively. These are shown plotted in Fig. 12.8(a).

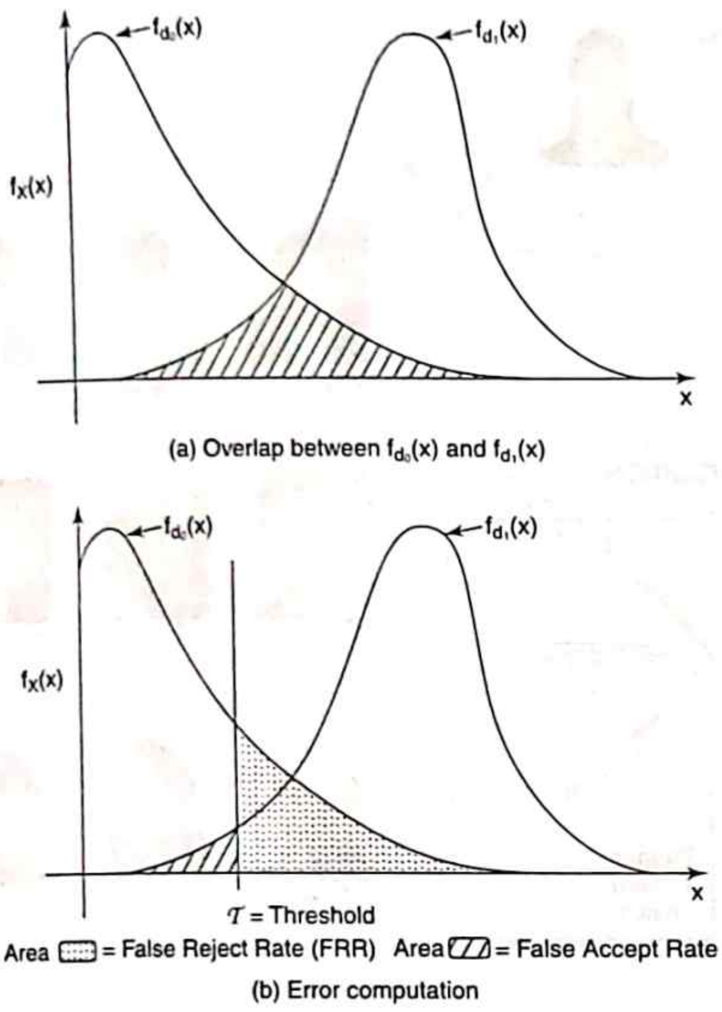


Figure 12.8 *Probability distributions of d_0 and d_1*

As mentioned earlier, identity verification of an individual **X**, involves comparing the template of a fresh biometric sample with a stored template of **X**. Let t_1 and t_2 be the fresh template and the stored template, respectively. Based on the computed value of $d(t_1, t_2)$, a determination is made whether $d(t_1, t_2)$ is more likely to be an instance of d_0 or d_1. Ideally, the distributions of d_0 and d_1 should have as little *overlap* as possible. This would be the case if

- their means are far apart and
- their distributions are sufficiently narrow (low standard deviation)

In practice, however, there would be some overlap between the two distributions as shown in Fig. 12.8(a). A *threshold*, $\mathcal{T}$, separating the two distributions is empirically set and used as follows.

Let t_1 be the template derived from a fresh biometric sample contributed by an individual **X**. Let t_2 be his stored template. The individual's claim to be **X** is settled as follows.

if $d(t_1, t_2) < \mathcal{T}$,	individual is **X**
otherwise,	individual is not **X**.

The system might reject the individual's claim to being X even though the individual is actually X. Such an outcome is said to be a *false reject*. On the other hand, the system may accept the individual's claim to being X even though the individual is not X. Such an outcome is referred to as a *false accept*. Both false rejects and false accepts are error conditions. The probability of a false reject is called the *false reject rate* (FRR) or *insult rate*. The probability of a false accept is called the *false accept rate* (FAR) or *fraud rate*.

Pictorially, the insult rate is the *area under the d_0 distribution to the right of the threshold* [Fig. 12.8(b)]. Likewise, the fraud rate is the *area under the d_1 distribution to the left of the threshold* [Fig. 12.8(b)]. We could decrease the fraud rate by decreasing the threshold but that would concomitantly increase the insult rate. Analogously, increasing the threshold has the effect of increasing the fraud rate while simultaneously decreasing the insult rate. Figure 12.8(b) captures the tradeoff between the fraud rate and the insult rate.

The relative importance of low fraud rate vis-à-vis low insult rate is application-specific. For example, *access control* to high-security installations requires a low fraud rate. On the other hand, success in *forensics/criminology* requires a low insult rate.

Fraud rates and insult rates vary for different biometrics. Even for the same biometric, fraud rates vary widely depending on the equipment used, environmental conditions, etc. Finally, there is usually wide discrepancy between error rates reported in lab tests and those reported in field tests.

12.4.3 Case Studies: Fingerprints and Iris Scans

Fingerprints

The human fingerprint is an attractive biometric since the *ridges* and *valleys* at the tip of an individual's finger exhibits distinctive patterns (Fig. 12.9). Moreover, a fingerprint is somewhat permanent in most cases though some wear and tear may occur due to age or specific occupations.

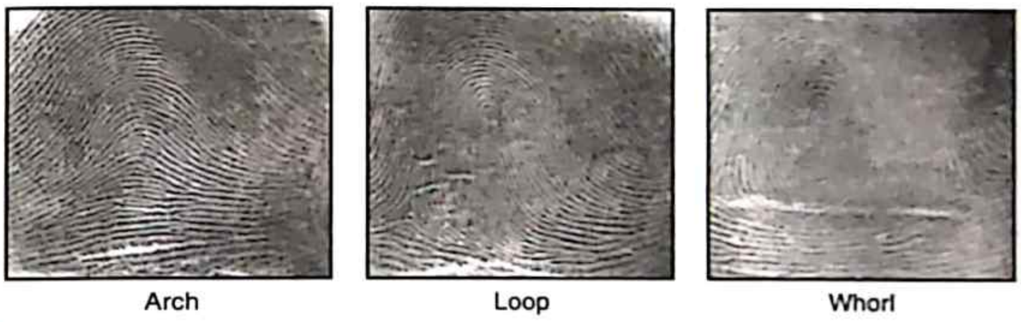


Figure 12.9 *Fingerprint singularity patterns*

In both the enrolment and recognition phases, an image of the fingertip is taken by placing it on the plane surface of a scanner. There are several types and subtypes of scanners – optical, solid state, and ultrasound. Of these, the first two are widely used.

During the recognition phase, a determination must be made whether the input template matches a given stored template. The simplest approach involves identification of a number of distinctive patterns formed by ridges. These are called *singularities*. The most basic of these are arches, loops and whorls (Fig. 12.9). In an *arch*, the ridge starts from one side of the finger, forms an arc and ends on the other side of the finger. In a *loop*, the ridge starts and ends at the same side of the finger after forming a curve. *Whorls* appear as closed cycles or spirals in a fingerprint.

A closer inspection of fingerprint images reveals that ridges are not always continuous – occasionally they end abruptly and at times they bifurcate. Minute details of ridges including ridge endings and bifurcations are called *minutiae* (see Fig. 12.10). Minutiae-based representations capture the location and orientation of the minutiae. The number of minutiae on a single fingerprint vary between 20 and about 70.

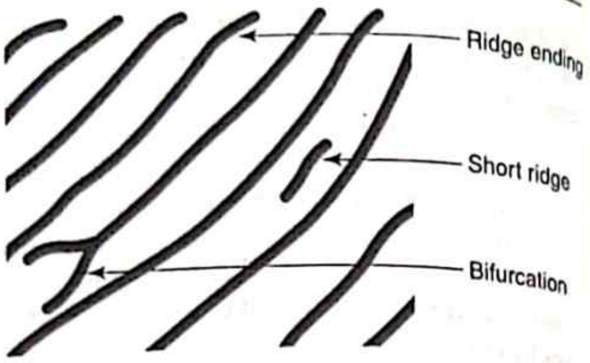


Figure 12.10 *Minutiae patterns*

One approach to fingerprint recognition is to compare the type, number, and coordinates of the singularities between the input and stored templates. This approach, used in isolation, may result in a high rate of false matches. Another approach is to superimpose the templates and estimate the extent of match between corresponding pixels in them. However, this approach is very sensitive to skin conditions and orientation of the finger.

One of the most widely used techniques involves *matching the minutiae* of the input and stored templates. Two minutiae match if their coordinates and orientations are the same to within a given tolerance. The number of matching minutiae between the input and stored templates is one measure of similarity. This approach roughly translates to a point-pattern matching problem and is computationally intensive.

Besides the algorithmic challenges of fingerprint matching, there are a number of issues related to image acquisition that affect *fingerprint quality*. The quality improves with an increase in the resolution (pixels per inch) of the scanner and an increase in the area of the fingerprint scanned. Image quality also depends upon the fingers being scanned. Dirty, wet, or extra dry fingerprints contribute to poor quality. Also, leftover fingerprint impressions from a previous subject distort the image being scanned. Finally, uneven contact of the subject's finger with the glass plate of the scanner or a slight shift in orientation can adversely affect the image.

For applications that demand very low error rates, multiple fingers may be used to enhance confidence in the match.

Iris Scans

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 (see Fig. 12.11). The muscles of the iris govern the amount of light entering the eye by controlling the size of the pupil. In the presence of bright light, the pupil constricts while it dilates in dim light.

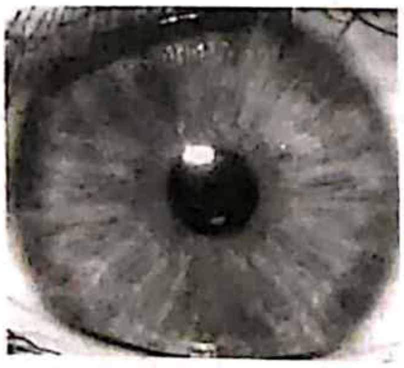


Figure 12.11 *Iris pattern*

One characteristic of the iris is its colour, which ranges between blue and dark brown depending on the amount of pigmentation. However, it is not its colour but rather the *intricate patterns* on the iris that appear to uniquely identify an individual. These patterns may include furrows, ridges, rings, and freckles as shown in Fig. 12.11. Unlike eye colour, the iris patterns are *not genetically inherited*. As a result, two identical twins have iris patterns that are as different as those of two unrelated individuals. In fact, the iris patterns of an individual's two eyes are just as unrelated!

The patterns of an iris are also *stable with age*. This property, in conjunction with the *distinctiveness* of the iris results in a theoretical false accept rate of close to zero. Unlike the human fingerprint, iris scanning is a *non-invasive* technique. It is possible to unobtrusively capture a high-quality image of the iris while a person passes, for example, through security check at an airport. The iris is captured as part of the larger image of an individual by a camera less than a metre away. Infrared illumination is typically used.

The structure of the iris as a sequence of vectors in the complex plane is extracted using two-dimensional Gabor wavelets. This process of demodulation of the rich patterns in the iris creates a *2048-bit iris signature* called the *iris code*. Given two iris templates or iris codes, it is the failure of the *test of statistical independence* that is used to conclude that the two templates are both computed by scanning the same iris.

Daugman [DAUG06] conducted a study of about 630,000 irises of different individuals from over 150 countries. He computed the pairwise Hamming distances of the iris codes corresponding to two different irises – a total of about 200 billion pairs. (The Hamming distance between two n-bit strings is the number of positions in which they differ. So, for example, the Hamming distance between 10111 and 11010 is 3 since the two strings differ in the second, third, and fifth bit positions beginning from the left.)

One important observation is that the distribution of Hamming distances between two distinct irises can be modelled by a *binomial distribution*,

$$\begin{aligned} f(m) &= \binom{n}{m} p^m (1-p)^{n-m} \\ &= \frac{n!}{m!(n-m)!} p^m (1-p)^{n-m} \qquad (12.1) \end{aligned}$$

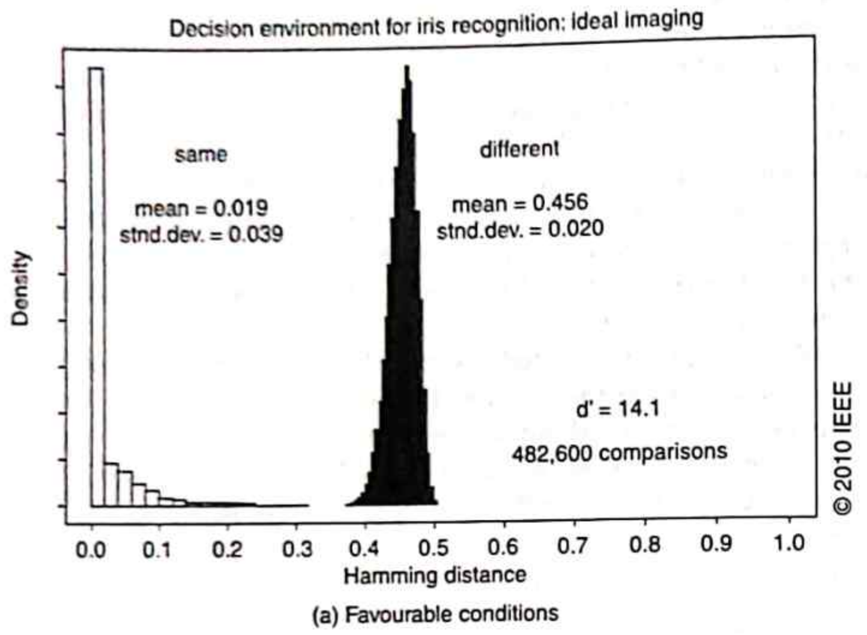
Here, $f(m)$ is the probability mass function (pmf) of the variable m = number of heads in n coin tosses. Each coin toss is a Bernouilli trial in which the probability of a head in a single coin toss is p and the probability of a tail is $(1-p)$.

In the context of iris scans, the number of tosses, n is 2048 – the length of the iris code. The outcome of a head in a coin toss corresponds to a non-match between corresponding bits in the sample iris template and the stored iris template being matched against. Likewise, the outcome of a tail corresponds to a match between corresponding bits in the sample and stored templates.

Daugman performed cross-comparisons on the nearly 630,000 irises (200 billion pairs). He then plotted the distribution of the fraction of bits that differed in *the codes of two different irises*, i.e., the random variable, $d_1 = \frac{\text{Hamming distance}}{n}$. The main results of the study were

- The corresponding bits in templates of two different irises matched with probability = 0.5 (i.e., a match was as likely as a non-match). Equivalently, the Hamming distance between two different iris codes, on average, is about $n/2 = \frac{2048}{2} = 1024$.

- The random variable, $\frac{\text{Hamming distance}}{n}$, appears to be binomially distributed. It is analogous to the random variable, $\frac{m}{n}$ in Eq. (12.1). Note that the mean number of heads in n Bernoulli tosses is $n \times p$. So, the fraction of heads in n Bernoulli tosses is p. Analogously, the mean of $\frac{\text{Hamming distance}}{n} = p = 0.5$ as noted above (see Fig. 12.12).



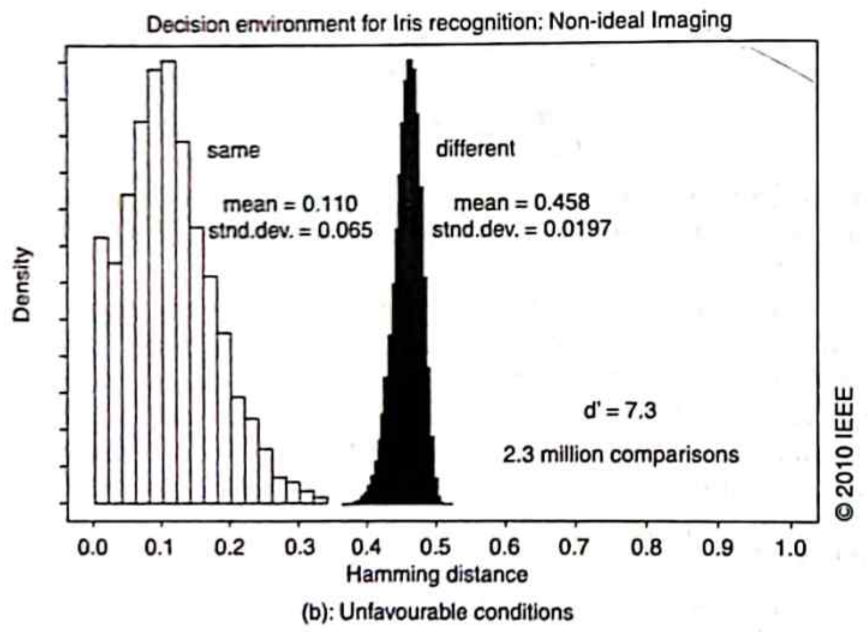


Figure 12.12 *Distribution of Hamming distances for same and different irises**

**Source:* J.G. Daugman, "How iris recognition works", *IEEE Trans. Circuits Syst. Video Technol.*, Vol. 14, No. 1, pp. 21–30, Jan 2004.

- The standard deviation of a binomially distributed variable is known to be $\sqrt{\frac{p(1-p)}{n}}$. For $p = 0.5$ and $n = 2048$, this works out to about 0.011. In reality, the standard deviation in the iris case study was somewhat higher at 0.02. This is explained by the fact that there are correlations between different bits in an iris code especially in the radial direction.
- It was found that out of the approximately 200 billion pairs involving different irises, there were very few cases wherein the codes differed by less than one-third or more than two-third of the corresponding bits. This is consistent with the fact that the binomial distribution has thin (heavily attenuated) tails.

In addition to obtaining the distribution of Hamming distances between two *different irises*, d_1, it is also very important to know the distribution of Hamming distances between two codes of the *same iris*, d_0 (see the distribution on the left of Fig. 12.12). The latter are computed from scans of an iris taken at different points in time (say, several days or years apart). The less the overlap between these two distributions – d_0 and d_1, the lower are the chances of error as measured by the FAR or the FRR.

It turns out that the Hamming distance of codes computed from the same iris is very sensitive to the conditions under which an image is taken (such as illumination level, camera zoom factor, distance from the camera, etc.). Lab tests [Fig. 12.12(a)] and field tests [Fig. 12.12(b)] yield very different results. Conditions in real-world environments are far from ideal. This creates a distribution with much higher variance as shown on the left of Fig. 12.12(b). Also, two different templates of the same iris differed in about 12% of the corresponding bits on average in field tests. However, under more ideal laboratory conditions the hamming distance reduces to less than half that value.

SELECTED REFERENCES

[KAUF02] and [MAO04] are two books in the area of network security/cryptography with extensive coverage of various cryptographic protocols. The Needham–Schroeder protocol (final version) appears in [NEED87]. RFC 4120 [NEUM05] contains a recent overview and description of Kerberos Version 5. [BELL90] addresses Kerberos security.

There are several papers on biometrics and biometric systems. [JAIN04] is an excellent survey on the subject. [DAUG06] presents results on iris recognition in the real world.

OBJECTIVE-TYPE QUESTIONS

12.1 The KDC functions as a/an
(a) authentication server (b) trusted third party
(c) certification authority (d) timestamp authority

12.2 The KDC obviates the need for
(a) digital certificates
(b) message integrity verification
(c) message confidentiality
(d) having long-term secrets between every pair of entities

12.3 The Kerberos protocol protects against which of the following attacks
(a) Dictionary attack (b) Man-in-the-middle attack
(c) Replay attack (d) Denial of service attack

12.4 In Kerberos, a long-term key is shared between
(a) each client and each application server
(b) each client and the authentication server
(c) each client and the ticket-granting server
(d) the authentication server and the ticket-granting server
(e) each application server and the authentication server
(f) each application server and the ticket-granting server

12.5 Which of the following characteristics does a manual signature lack?
(a) Permanence (b) Universality
(c) Uniqueness (d) Robustness

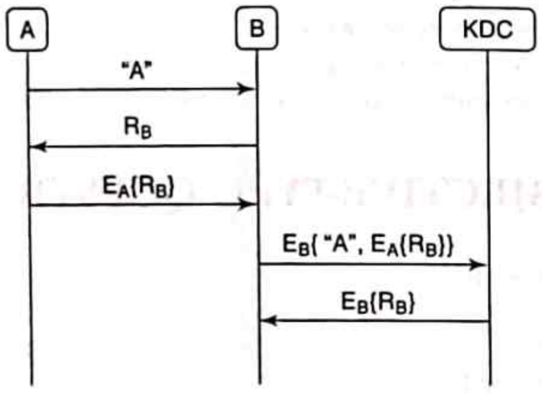
12.6 Which of the following is/are needed to compute the false reject rate in iris recognition?
(a) The distribution of Hamming distances between the iris codes for the same iris
(b) The distribution of Hamming distances between the iris codes for different irises
(c) An empirically set threshold value between the two distributions
(d) The variances of the two distributions

12.7 Which of the following characteristics is most widely used in fingerprint recognition?
(a) Number and coordinates of arch patterns
(b) Number and coordinates of loop patterns
(c) Number and coordinates of whorl patterns
(d) Coordinates and orientations of fingerprint minutiae

EXERCISES

12.1 List all possible attacks on the protocol in Fig. 12.1.

12.2 Consider the protocol below in which A authenticates herself to B using a trusted third party, i.e., the KDC. Design an attack on this protocol whereby an attacker successfully impersonates A to B.



12.3 In the text, we investigated the effect of a compromise of the long-term keys on the security of the Needham–Schroeder protocol. Suppose instead, an attacker is able to recover the short-term key used in one of the sessions.
(a) Can this lead to an attack on *preliminary version* 3 [Fig. 12.4(a)] of the Needham-Schroeder protocol? If so, how?

(b) Can this lead to an attack on the final version of the Needham–Schroeder protocol (Fig. 12.5)? If so, how?

12.4 The goal of this exercise is to reduce the number of messages in the Needham–Schroeder protocol (Final version, Fig. 12.5).

(a) Make changes in the content of some of the existing messages so that fewer messages are needed. Do not use timestamps.

(b) Using timestamps (and possibly nonces), re-design the protocol so that no more than five messages are needed.

12.5 Re-design the Needham–Schroeder protocol to use public key cryptography. (Do not use digital certificates. Instead assume that the KDC stores the public key of each principal within the organization.)

12.6 Consider the protocol below which attempts to provide mutual authentication and key exchange between A and B using the KDC. (The notation is consistent with what has been used in the text.)

(a) Clearly explain whether and why A is convinced that B is indeed live and that B is communicating with her?

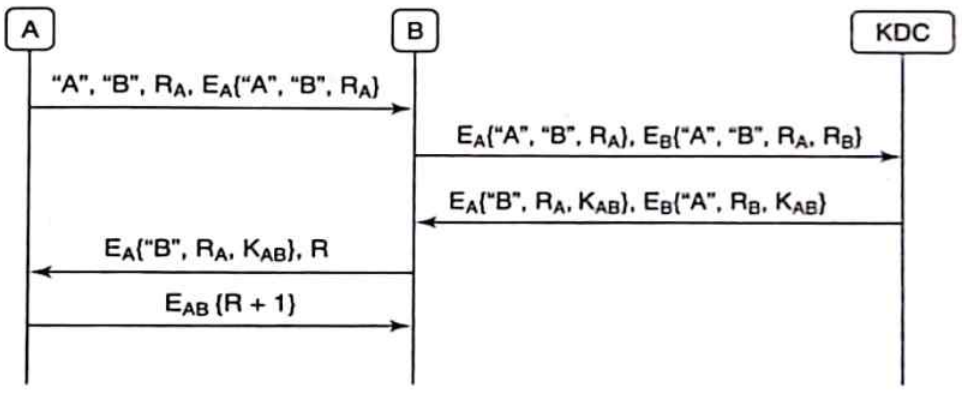
(b) Clearly explain whether and why B is convinced that A is indeed live and that A is communicating with him?

(c) Clearly explain whether and why A is convinced that the KDC is indeed live and that it is communicating with her?

(d) Clearly explain whether and why B is convinced that the KDC is indeed live and that it is communicating with him?

(e) Clearly explain whether and why the KDC is convinced that A is indeed live and that she is communicating with it?

(f) Clearly explain whether and why the KDC is convinced that B is indeed live and that he is communicating with it?


A
B
KDC
"A", "B", R_A, E_A{"A", "B", R_A}
E_A{"A", "B", R_A}, E_B{"A", "B", R_A, R_B}
E_A{"B", R_A, K_{AB}}, E_B{"A", R_B, K_{AB}}
E_A{"B", R_A, K_{AB}}, R
E_{AB} {R + 1}

12.7 Can Kerberos be re-designed to use nonces only? Explain why or why not.

12.8 What are the principal advantages of

(a) biometric authentication over cryptographic authentication?

(b) cryptographic authentication over biometric authentication?

12.9 Consider a system that performs both authentication and identification. A *false positive* occurs when, for example, the system misidentifies Rajesh as Ramesh OR incorrectly accepts Rajesh's claim to being Ramesh. Let P_1 and P_n be the false positive rates for authentication and identification, respectively. Assume that the size of the database in the identification system is n, i.e., the system stores biometric "templates" of n different individuals. Write an expression relating P_1 and P_n.

12.10 (a) Is it possible to decrease the fraud rate without simultaneously increasing the insult rate? Explain why or why not.

(b) Is it possible to decrease the insult rate without simultaneously increasing the fraud rate? Explain why or why not.

ANSWERS TO OBJECTIVE-TYPE QUESTIONS

12.1 (a)(b)	12.2 (a)(d)	12.3 (b)(c)	12.4 (b)(d)(f)
12.5 (a)(b)	12.6 (a)(c)	12.7 (d)	

Chapter 13

IPSec—Security at the Network Layer

13.1 SECURITY AT DIFFERENT LAYERS: PROS AND CONS

Security may be implemented at different layers of the OSI model. In this book, we are principally interested in security at the network layer and above. Because the Internet was not designed for security, mechanisms for security have to be retro-fitted at or between layers. More precisely, security is commonly implemented

- in the *network* layer or
- between the *transport* and application layers and/or
- within the *application*

Implementing security for each individual application is less attractive compared to implementing it at a single point which can then be availed of by all applications. Ideally, neither users nor applications need to be made aware of where and how security is provided. Providing security at the network layer has least bearing on an application. On the other hand, implementing security at the network layer will necessitate some changes in the operating system (OS) kernel. So, while the application developer may have less need to worry about security, the system administrator may have to confront issues of configuration and compatibility as he/she attempts to deploy security solutions at the network layer.

Implementing security between the transport and application layers obviates the need for any change in the OS. However, application developers should be aware of the application programming interfaces (APIs) provided by the new security layer so they can be employed in developing secure applications. A widely used example is the Open SSL suite of APIs. This contrasts with applications in which security is provided for within the application itself. Examples of the latter include many of the e-payment applications discussed in Chapter 24.

In this chapter, we study IPsec – the best-known protocol for providing security at the network layer. We first introduce the two main IPSec protocols and their modes of operation. We then explore how a cryptographic suite is negotiated between two communicating parties.

13.2 IPSec IN ACTION

The design of IPSec has been an exercise in retro-fitting security into the IP protocol. 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

- data origin authentication and data integrity
- protection from replay attacks
- data confidentiality and
- partial traffic flow confidentiality.

The end-points of the protocol can be two hosts, two gateways, or a host and a gateway.

13.2.1 IPSec Security Associations

Before two parties can communicate securely, they need to establish a *Security Association* (SA) with each other. The information in an SA includes

- *Lifetime* of the association
- IPSec *Mode* – transport or tunnel (to be explained later in this section)
- *Cryptographic parameters* (the algorithm used for encryption, if any, and for computing the integrity check, together with the keys)
- A 32-bit *sequence number* – the first packet protected by a newly established SA bears the sequence number 1, the sequence number gets incremented for each new packet sent.
- An *anti-replay window*.

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. 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.

13.2.2 IPSec Protocols: AH and ESP

IPSec includes two protocols – AH (Authentication Header) and ESP (Encapsulating Security Payload). The main difference between AH and ESP is that AH has no provision for confidentiality, while ESP provides confidentiality as an option. These protocols can be used in either transport or tunnel mode. For simplicity, we use *Transport mode* in this section and introduce tunnel mode in the next section.

Figure 13.1 shows the headers introduced by AH and ESP and their coverage of authentication and encryption. The IPSec header is sandwiched between the IP and TCP headers. Message authentication and integrity are provided by the use of a keyed MAC in both cases. While the MAC is a part of the AH header, it is included in the trailer of an ESP packet. IPSec implementations are required to support MACs based on MD-5 and SHA-1. However, 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.

There is an important difference between what is covered by the integrity check in AH and ESP. 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.) By contrast, ESP does *not* provide protection to any part of the IP header in transport mode.

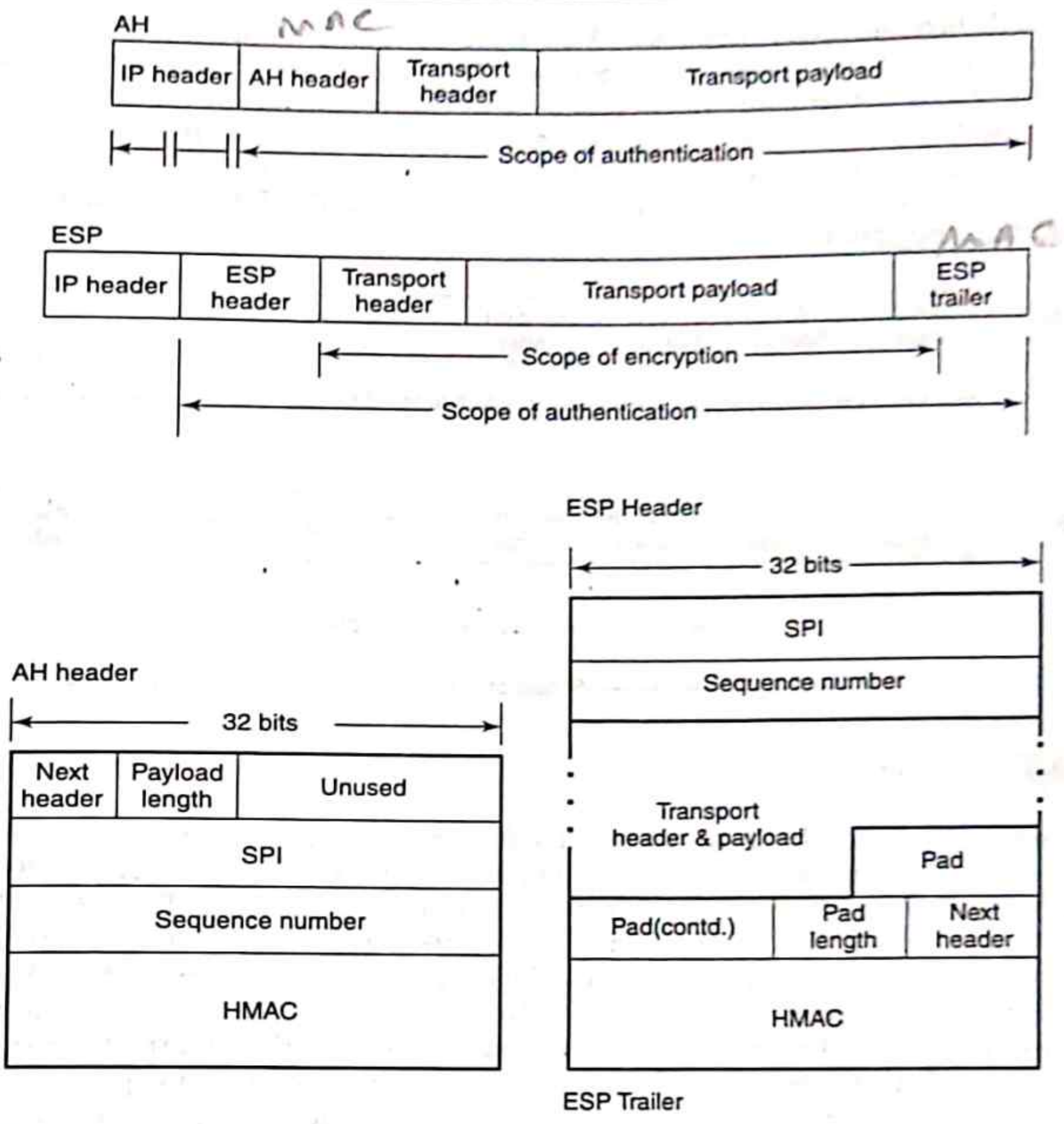


Figure 13.1 *AH and ESP in transport mode*

Figure 13.1 shows the fields in the AH and ESP headers. All IPSec headers (both AH and ESP in both modes) have 32-bit fields each for the SPI and a packet *sequence number*. The latter was intended to protect against replay attacks. 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.

13.2.3 Tunnel versus Transport Mode

The mode of operation discussed in the previous section is called *transport mode* because it protects the transport header and the transport payload. To protect the *entire* IP header, IPSec has an option called *tunnel mode*. One of the earliest applications of tunnelling in computer networks was to route packets through heterogeneous networks. With IPSec in tunnel mode, the original IP packet is encapsulated within a larger packet containing an IPSec header and an *extra IP header*.

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. The MAC covers the original IP header in its entirety. Figure 13.2 shows the order of insertion of the different headers and the scope of authentication/encryption.

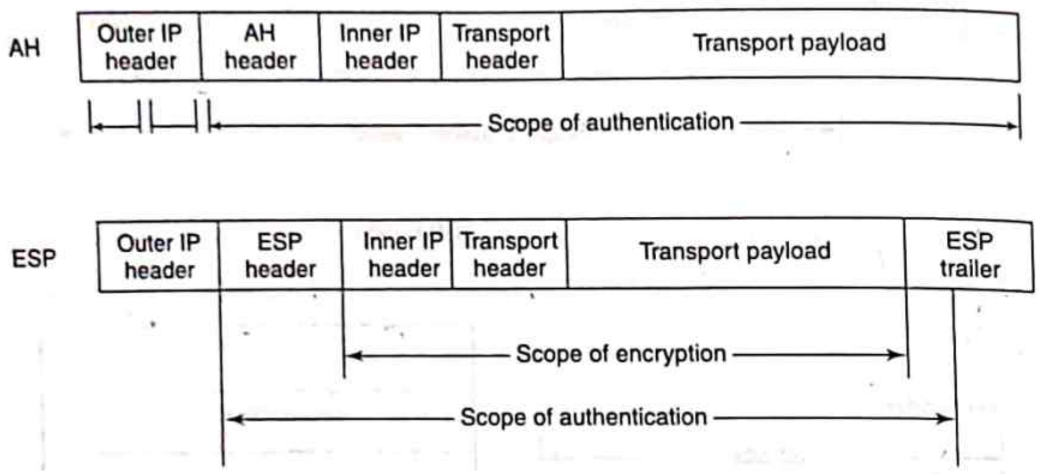


Figure 13.2 *AH and ESP in tunnel mode*

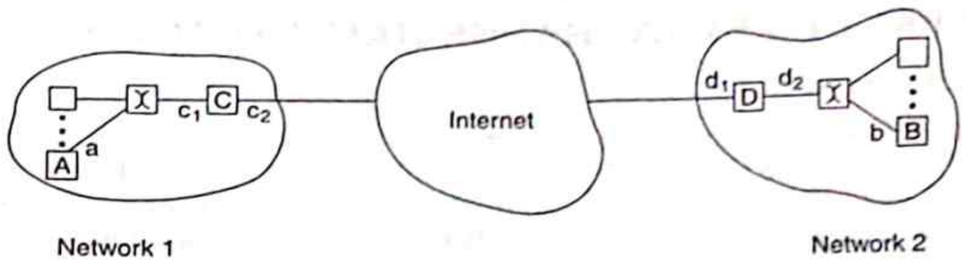
The most compelling use of the IPSec in tunnel mode is in securing *host-to-host* and *host-to-gateway* communications. We illustrate this for communications between two hosts A and B on different networks, Network 1 and Network 2, communicating over the Internet [Fig. 13.3(a)].

Suppose that security policy dictates that a packet entering Network 2 should authenticate itself to the incoming gateway, D. Then A could tunnel its packet to B inside a packet to D. The IP/IPSec headers generated by A are shown in Fig. 13.3. Note that the destination address in the outer IP header is D (interface d_1). This header is used to route the packet through the Internet. On receipt of the packet by D, an attempt is made by D to authenticate the source of the packet. If it succeeds, the outer IP header is stripped, the packet is decrypted and the ESP header and trailer are also stripped. The inner IP header of the packet is used to route the packet to B within Network 2.

In the above example, it is assumed that Host A is IPSec-enabled. Suppose that, on the other hand, IPSec was not supported on Host A. Instead, suppose that the gateway, C, is responsible for providing authentication and encryption service to all outgoing traffic destined to hosts in Network 2. In this case, C would prepend an ESP header and another IP header to A's packet for communication through a secure tunnel between C and D [see Fig. 13.3(b)]. On receipt of the packet, D would strip off the outer IP header, authenticate the source of the packet, decrypt its contents, strip off the ESP header and trailer and route the packet to B.

13.2.4 Incompatibility with NAT

Network address translation (NAT), introduced in Chapter 2, translates the source IP address of an outgoing packet. The address is initially a private one (locally valid), while the address being substituted is one that is a valid (globally routable) Internet address. NAT thus saves address space



	Outer IP	ESP	Inner IP	TCP header and payload

Case 1	Source = A
	Dest = D
			Source = A
			Dest = B	

(a) Host-to-gateway tunnel

	Outer IP	ESP	Inner IP	TCP header and payload

Case 2	Source = C
	Dest = D
			Source = A
			Dest = B	

(b) Gateway-to-gateway tunnel

Figure 13.3 *Tunnel mode header contents*

by assigning one or a small number of valid IP addresses to an organization instead of one valid address for each server. The use of NAT also enhances security by hiding the internal addressing schema. Thus, internal machines such as the database and application servers are not directly addressable from the outside world.

We next investigate whether and why NAT and IPSec may be unable to coexist. We study the cases of AH and ESP separately.

The IPSec header is prepended to a packet before it encounters a NATing device such as a firewall. With AH in transport mode, a packet has only one IP header – the source IP address in this packet is translated. With AH in tunnel mode, the IP address in the outer IP header is translated. In both cases, the *scope of the MAC* includes the source IP address. Since the source IP address gets translated *after* the MAC computation, the message integrity check at the receiver will fail.

In the case of ESP in transport mode, the IP header does not fall within the scope of the MAC. There is, however another subtle issue here. It turns out that the scope of the *checksum in the TCP header* includes the source and destination IP addresses. Because NAT modifies the source IP address, the TCP checksum is no longer valid. The TCP checksum could be re-computed by the NATing device if encryption is not used. However, in that case the message integrity test at the receiver would fail. This is because the ESP MAC covers the entire TCP header including its checksum.

The only IPSec protocol/mode that can co-exist with NAT is ESP in tunnel mode. In this case, the IP source address that gets translated is in the outer IP header. This is neither covered by the IPSec MAC nor does it fall within the scope of the TCP checksum.

13.3 INTERNET KEY EXCHANGE (IKE) PROTOCOL

13.3.1 Preliminaries

The main goal of IKE is to establish an SA between two parties that wish to communicate securely using IPSec. Recall that an SA includes information on, among other things, the specific cryptographic algorithms used by that SA and also the keys used for encryption, and integrity protection. While IPsec is a network-layer protocol, IKE is an application-layer protocol using the connectionless UDP protocol on port 500.

IKE borrows heavily from two major sources – the Internet Security Association and Key Management Protocol (ISAKMP) [MAUG98] and Oakley [HARK98]. ISAKMP defines formats of various entities such as the digital signature and the digital certificate. It also specifies the rules for stringing payloads together to form a valid message. Oakley specifies the kind of information to be exchanged in each message that is part of IKE.

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 analogous to setting up a *session* in the SSL protocol, while setting up an IPSec SA is analogous to setting up an SSL *connection*. (The SSL protocol is discussed in the next chapter.)

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.

Key agreement uses the Diffie–Hellman key-exchange protocol. However, as discussed in Chapter 8, unauthenticated key exchange is vulnerable to man-in-the-middle attacks and session hijacking. In addition, an attacker could induce its victim to compute useless modular exponentiations leading to a DoS attack. IKE is designed to withstand these attacks while at the same time offering a menu of different cryptographic algorithms and authentication methods.

13.3.2 IPSec Cookies

To thwart 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. This fact can be exploited to launch a special kind of DoS *attack*. An attacker creates many spurious messages – each one being a request to set up an IKE SA with B. A spoofed IP source address is used in each of these messages. In general, the responder would have no way of knowing that the messages are spoofed. It would respond to these messages by diligently computing the Diffie–Hellman partial key. A large number of such requests would eventually exhaust its computational resources.

To frustrate such attacks, IKE mandates that B should compute a 64-bit integer called a *cookie*. This is a hash function of many variables including the *IP address of A*, an *ephemeral secret* known only to B and possibly the *time*. A is required to send this cookie to B in all subsequent messages. The cookie will, in general, be different for different IP addresses. 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 and hence avoid performing the modular exponentiation. Short of tapping the line between A and B, the attacker will have no way to spoof the cookie created in response to a request from A.

The cookie computed by B, C_B, is sent in Message 2 of Fig. 13.4 and must be included in all messages sent by either party thereafter. In a similar manner, a cookie, C_A, is initially created by A and subsequently sent in all messages. The pair (C_A , C_B), plays the role in IKE that the SPI plays in IPSec.

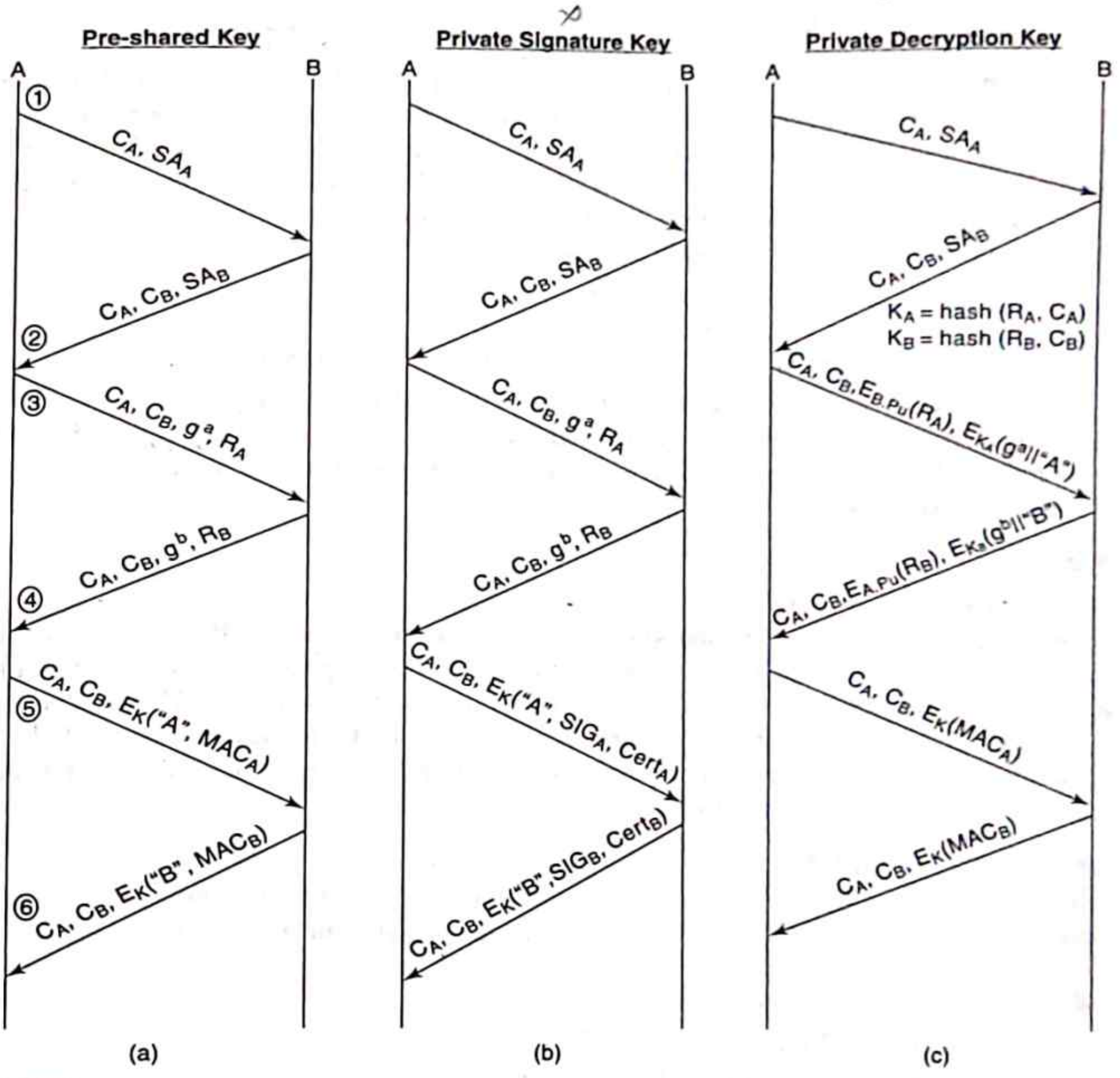

Pre-shared Key
Private Signature Key
Private Decryption Key
A
B
① C_A, SA_A
② C_A, C_B, SA_B
③ C_A, C_B, g^a, R_A
④ C_A, C_B, g^b, R_B
⑤ C_A, C_B, E_K("A", MAC_A)
⑥ C_A, C_B, E_K("B", MAC_B)
C_A, C_B, E_K("A", SIG_A, $Cert_A$)
C_A, C_B, E_K("B", SIG_B, $Cert_B$)
K_A = hash (R_A, C_A)
K_B = hash (R_B, C_B)
C_A, C_B, $E_{B.Pu}(R_A)$, $E_{K_A}(g^a||$"A")
C_A, C_B, $E_{A.Pu}(R_B)$, $E_{K_B}(g^b||$"B")
C_A, C_B, $E_K(MAC_A)$
C_A, C_B, $E_K(MAC_B)$
(a)
(b)
(c)

Figure 13.4 *IKE phase 1 (main mode)*

13.3.3 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.
- Keys, KEY_a and KEY_e, are computed. These keys are used for message integrity protection and 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 identifier.

Phase 1 uses one of two modes. *Main Mode* involves a total of six messages between initiator (A) and responder (B), while *Aggressive Mode* uses only three messages. The motivation for introducing Main Mode is to hide the identities of the sender and receiver from eavesdroppers.

One form of identification is the IP address. The original IP address in a packet's header might get substituted if network address translation (NAT) is used as discussed in the previous section. We assume that the two parties may (and usually do) use an alternative form of identification such as an e-mail address or an X.500 name. The main mode of IKE seeks to protect the confidentiality of these alternative forms of identification through encryption.

To perform mutual authentication, IKE assumes that either

A and B share a secret
OR
A and B, each, have a public key–private key pair.

There are two ways in which A and B might prove knowledge of their private keys – by signing a message or by decrypting a challenge. Accordingly, we refer to these two authentication methods as *signature private key* and *decryption private key*. We now examine the six messages exchanged in Phase 1 for each of these cases.

Main Mode

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 C_A). 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 shown below.

$KEY_r = \text{prf}(s, R_A \mid R_B)$	using pre-shared secret
$KEY_r = \text{prf}(R_A \mid R_B, g^{ab})$	using private signature key
$KEY_r = \text{prf}(R_A \mid R_B, C_A \mid C_B)$	using private encryption key

$$KEY_d = \text{prf}(KEY_r, g^{ab} \mid C_A \mid C_B \mid 0)$$
$$KEY_a = \text{prf}(KEY_r, KEY_d \mid g^{ab} \mid C_A \mid C_B \mid 1)$$
$$KEY_e = \text{prf}(KEY_r, KEY_a \mid g^{ab} \mid C_A \mid C_B \mid 2)$$

$$MAC_A = \text{prf}(KEY_a, g^a \mid g^b \mid C_A \mid C_B \mid SA_A \mid \text{“}A\text{”})$$
$$MAC_B = \text{prf}(KEY_a, g^b \mid g^a \mid C_B \mid C_A \mid SA_A \mid \text{“}B\text{”})$$

The keys are derived using a pseudo-random function, prf. Here, *s* is the pre-shared secret between A and B. g^a, g^b, and g^{ab} are each computed modulo *p*.

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. The expressions for KEY_r are different for each authentication method. However, all three methods use the same expressions for KEY_d, KEY_a, and KEY_e.

Both A and B use a MAC for message authentication and integrity. These are denoted by MAC_A in Message 5 and MAC_B in Message 6. Note that the key used in the computation of the MAC is KEY_a which is a function of the shared key, *s*, between A and B. Also, MAC_A and MAC_B involve SA_A, the cipher suite proposed by A. The inclusion of SA_A in the hash will detect possible substitution, by an attacker, of a stronger cryptographic suite for a weaker one.

In Messages 5 and 6, both sides reveal their identities to one another. The messages are encrypted with KEY_e (denoted *K* in Fig. 13.4). Since KEY_e is a function of the secret, *s*, shared between A and B, they alone can decrypt each others' messages. Thus, their identity is not revealed to anyone else.

There is a major drawback with this option involving a shared secret. Consider what happens when B receives Message 5. He must first decrypt the message. But, for that he needs to use the secret, *s*, that he shares with A. To use *s*, he needs to know the identity of the message sender. However, the sender's ID is itself encrypted. One possibility is to use the source IP address to identify the sender. But if the IP address is a give-away of the sender's identity, then what was the point in protecting the sender's identity using encryption?

Alternatively, B could keep track of all entities that it expects to communicate with from each IP address. Then, when an IKE request arrives from a particular IP address, it could attempt to decrypt the message using secrets it shares with each entity at that IP address. A message would be successfully decrypted if the key used for decryption matched that shared by the entity whose ID appeared in Message 5.

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 (R_A or R_B) and encrypts it with the other side's public key. Each side encrypts its identity together with its Diffie–Hellman partial key ($g^a \bmod p$ or $g^b \bmod p$) with temporary keys, K_A and K_B in Messages 3 and 4, respectively. K_A and K_B are functions of the nonces and cookies as below

$$K_A = \text{hash}\ (R_A, C_A)$$
$$K_B = \text{hash}\ (R_B, C_B)$$

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, R_A and R_B. If A or B were unable to decrypt (with their private key) the nonce generated by the other side, then they would not be able to compute the correct MAC. An incorrect MAC would be detected by the other party and would result in the IKE exchange being aborted.

Aggressive Mode

Unlike the main mode, the aggressive mode involves only three messages. The price to be paid for economizing on the number of messages is that the identities of the communicating parties are no longer hidden from passive eavesdroppers. As shown in Fig. 13.5, the identities of A and B are sent in the clear in messages 1 and 2.

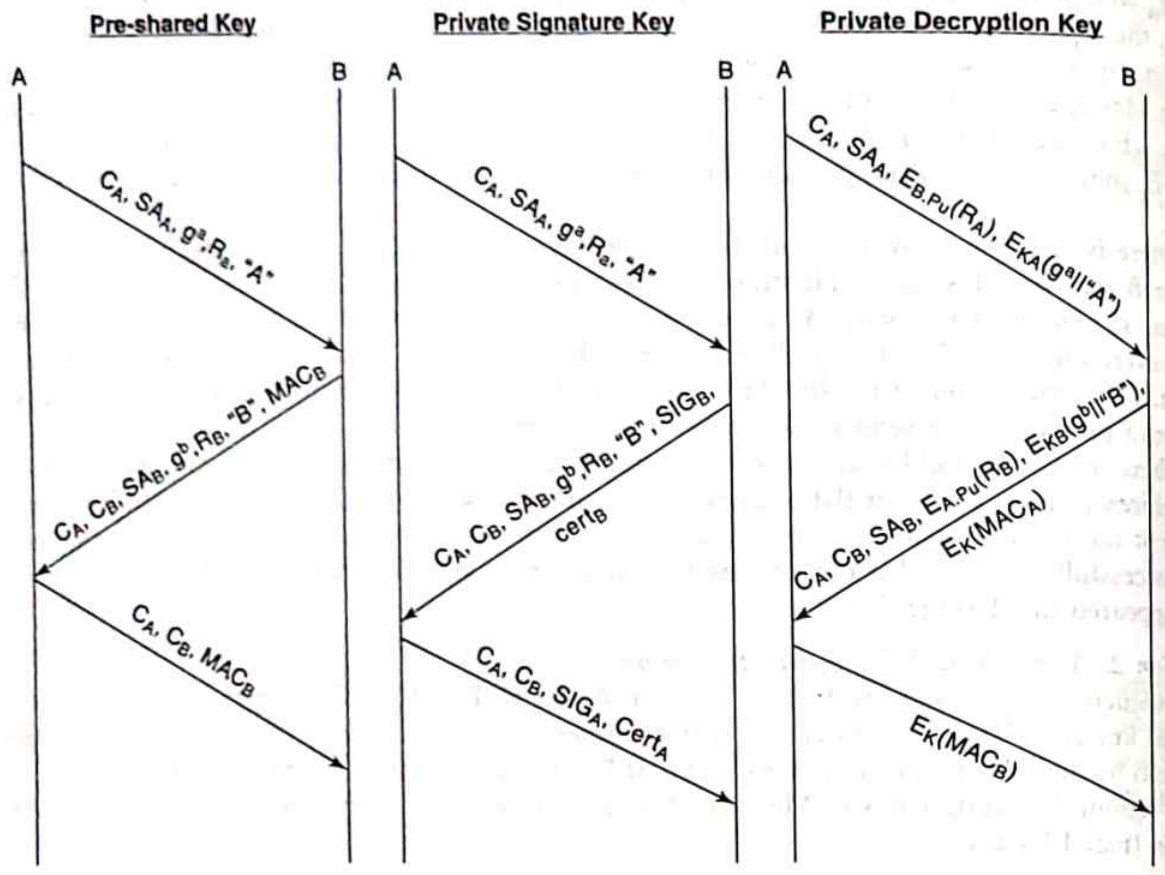

Pre-shared Key
Private Signature Key
Private Decryption Key
A
B
$C_A, SA_A, g^a, R_a, \text{"A"}$
$C_A, C_B, SA_B, g^b, R_B, \text{"B"}, MAC_B$
C_A, C_B, MAC_B
$C_A, SA_A, g^a, R_a, \text{"A"}$
$C_A, C_B, SA_B, g^b, R_B, \text{"B"}, SIG_B, cert_B$
$C_A, C_B, SIG_A, Cert_A$
$C_A, SA_A, E_{B.Pu}(R_A), E_{KA}(g^a \| \text{"A"})$
$C_A, C_B, SA_B, E_{A.Pu}(R_B), E_{KB}(g^b \| \text{"B"}), E_K(MAC_A)$
$E_K(MAC_B)$

Figure 13.5 *IKE phase 1 (aggressive mode)*

Another aspect of IKE Phase 1 in aggressive mode is that the Diffie–Hellman group used and the group parameters are decided by A. IKE offers diverse groups such as Z_p^*, various choices of elliptic curve groups, etc. However, A unilaterally chooses a group, computes its partial key and sends it to B in Message 1. B has no choice but to meekly accept the group chosen by A. The main mode, on the other hand, permits B to have a say in the choice of Diffie–Hellman group used.

13.3.4 IKE Phase 2

Under cover of an existing IKE 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. Figure 13.6 shows the three messages exchanged in "Quick Mode." All messages are encrypted using the secret, KEY_e, computed in the previous phase (not shown in Fig. 13.6). 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 to be used for authentication (and encryption) as part of the IPSec SA. 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.

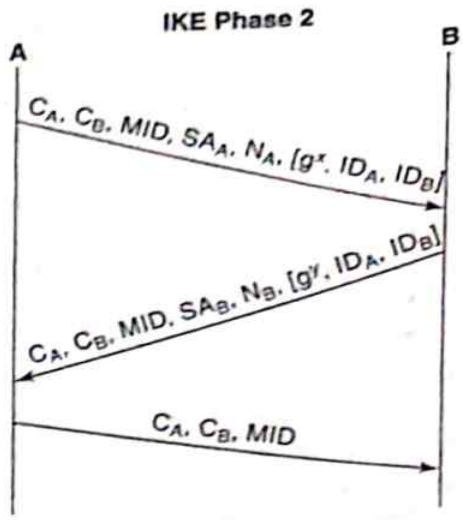

IKE Phase 2
A
B
$C_A, C_B, MID, SA_A, N_A, [g^x, ID_A, ID_B]$
$C_A, C_B, MID, SA_B, N_B, [g^y, ID_A, ID_B]$
C_A, C_B, MID

Figure 13.6 *IKE phase 2*

If, in addition, *perfect forward secrecy* (see Section 11.3 in Chapter 11) is required, then both sides also perform a Diffie–Hellman key exchange in this phase as well. The Diffie–Hellman partial secrets are abbreviated as g^x and g^y in Fig. 13.6 and represent g^x mod p and g^y mod p. The Diffie–Hellman secret, g^{xy} mod 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.

13.4 SECURITY POLICY AND IPSEC

How does the IP layer at the sender end know whether a packet handed to it by the transport layer should be protected or not? We need a mechanism to support high-level policy statements such as

"Authenticate all web traffic to IP address A.B.C.D."

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 protocol port number).

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, 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.

13.5 VIRTUAL PRIVATE NETWORKS

A Virtual Private Network (VPN) enables organizations to communicate securely over a public, shared network such as the Internet. One possibility is to use dedicated point-to-point lines such as

T1 leased lines to keep communications confidential. However, there is a clear cost advantage in using shared, publicly accessible networks instead. IPSec is just the protocol that helps secure IP traffic over such open and insecure networks.

In VPN literature, the terms *secure* and *trusted* are often mentioned. 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. Instead, the infrastructure of the service provider is relied upon to guarantee confidentiality of the traffic.

The two most widely used VPNs are

- Site-to-site VPNs and
- Remote Access VPNs

Site-to-site VPNs are used to link multiple offices of an organization in, what is commonly referred to, an *intranet*. They are widely used to connect multiple branches of a bank. A site-to-site VPN may also be used to secure an *extranet* – a network connecting multiple business partners. *Remote access VPNs* connect teleworkers (mobile users or users from home) to their offices.

Both site-to-site and remote access VPNs often use IPSec to provide secure communications. In addition, SSL-based VPNs have been deployed. One advantage of SSL-based VPNs is that they do not require any additional software beyond a browser.

SELECTED REFERENCES

The two main IPSec protocols, AH and ESP, appear in RFC 2402 [KENT98b] and RFC 2406 [KENT98c]. ISAKMP and IKE are dealt with in detail in [MAUG98] and [HARK98]. [PERL00] is an excellent critique of the many design decisions in IPSec and, specifically, the IKE protocol.

OBJECTIVE-TYPE QUESTIONS

13.1 A receiving node determines the SA an IPSec packet belongs to based on
(a) the sequence number in the IPSec header
(b) the SPI in the IPSec header
(c) source IP address
(d) a combination of source IP address and source port

13.2 An extra network-layer header is inserted by
(a) AH in transport mode (b) AH in tunnel mode
(c) ESP in transport mode (d) ESP in tunnel mode

13.3 Which of the following is NAT compatible with?
(a) AH in transport mode (b) AH in tunnel mode
(c) ESP in transport mode (d) ESP in tunnel mode

13.4 IKE borrows heavily from
(a) EKE (b) SSL
(c) Oakley (d) ISAKMP

13.5 IPSec is designed to withstand replay attacks through the use of
(a) Sequence numbers (b) Nonces
(c) Nonces + Sequence numbers (d) Timestamps

13.6 The advantage of IKE Phase 1 Main mode over IKE Phase 1 Aggressive mode is
(a) main mode uses fewer messages
(b) main mode provides greater security
(c) main mode hides the identities of the communicating entities
(d) main mode has a larger suite of options for key exchange

13.7 A VPN enables
(a) private communication between two or more parties over leased lines
(b) secure communication between business partner organizations over the Internet
(c) secure communications over a LAN
(d) remote login to internal machines within an organization

EXERCISES

13.1 IPSec has two protocols – AH and ESP. Do you feel AH is necessary or can all functionality provided by AH be provided by ESP? Explain.

13.2 What is the recipe for an IKE cookie?
The IKE cookie was intended to protect against DoS attacks. Can you think of a way in which the attacker can defeat the DoS protection provided for by the IKE cookie? If so, explain how.

13.3 The IKE protocol has two phases. Can you re-design it as a single-phase protocol? What are the pros and cons of having a two-phase IKE versus a single-phase IKE?

13.4 Can you reduce the number of messages exchanged in IKE Phase 1, Main Mode? If so, show all messages and their contents in the re-designed protocol. Consider each of the three options:
- A and B share a secret
- A and B have a private key/public key pair (private signature key)
- A and B have a private key/public key pair (private decryption key)

13.5 Security can be implemented in the network layer, at the transport layer, or in the application itself.
What are the pros and cons of providing security at the different layers?
Discuss suitable real-world applications highlighting the layer(s) in which security is provided in each case.

13.6 Company policy permits two hosts, A and B, in two different branches of the organization to communicate securely over the Internet using IPSec. Which of the four options would be most appropriate – AH in transport mode, AH in tunnel mode, ESP in transport mode, or ESP in tunnel mode? Explain your choice.
Show all the headers that-are inserted in communication and the scope of authentication, integrity checking, and encryption.
Is it necessary/possible to double-encrypt a packet between A and B? Explain.

13.7 A bank has thousands of branches across the country. In addition to the use of ATMs and Internet banking, the bank allows its customers to perform transfers or withdrawls from any branch. In particular, a customer who has an account in branch *A* may visit branch *B* and perform a withdrawal from her account in branch *A* after suitable identity verification.
"Branchless" banking is made possible through what is commonly called core banking wherein account information of all customers is stored centrally. Assuming that all branches

have access to leased line connections, propose a scalable connection architecture linking all branches with the central core.

Based on this connection architecture, propose a security solution using IPSec that protects customer transactions involving accounts in one or more branches of the bank.

ANSWERS TO OBJECTIVE-TYPE QUESTIONS

13.1 (b)(c)	13.2 (b)(d)	13.3 (d)	13.4 (c)(d)
13.5 (a)	13.6 (c)	13.7 (a)(b)(d)	

Chapter 14

Security at the Transport Layer

14.1 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). Today SSL and TLS are used interchangeably.

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 compared to their unsecured counterparts. For example, HTTP runs on port 80 but *HTTPS* runs on port 443. Similarly, FTP runs on port 21 but *FTPS* runs on port 990.

SSL is comprised of two main protocols (see Fig. 14.1)

- The *Handshake Protocol*
- The *Record Layer Protocol*

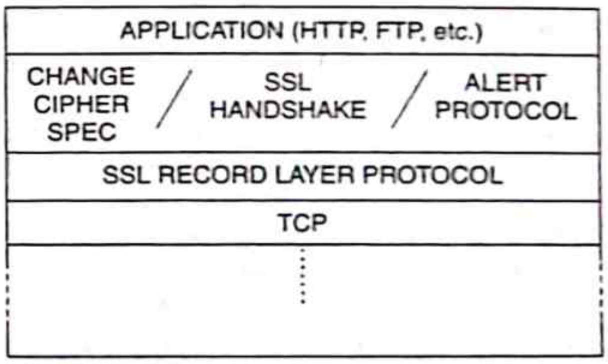


Figure 14.1 *SSL on the protocol stack*

The SSL handshake protocol is used to *negotiate* the set of algorithms to be used for securing the communication link. This includes algorithms for encryption and hash computation besides a method for key exchange. *Server authentication* in SSL is mandatory and performed as part of the handshake. The handshake 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. We next introduce the handshake protocol and then the record layer protocol.

14.2 SSL HANDSHAKE PROTOCOL

14.2.1 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.

As discussed in the next section, the overhead of establishing a new *session* far outweighs that of establishing a new *connection*. On the other hand, for long-lasting sessions, it is prudent to re-compute session keys. Starting a new connection within an existing session is a light-weight mechanism to obtain fresh keys needed for integrity protection and encryption.

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 Fig. 14.2. We next describe the steps in some detail.

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? In the former case, 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. Permissible choices include MD5 and SHA-1.
- Whether or not message confidentiality is required. If so, the encryption algorithm – DES, 3DES, AES, RC4, etc., and the key length.
- The key exchange method used for communicating the pre-master secret.

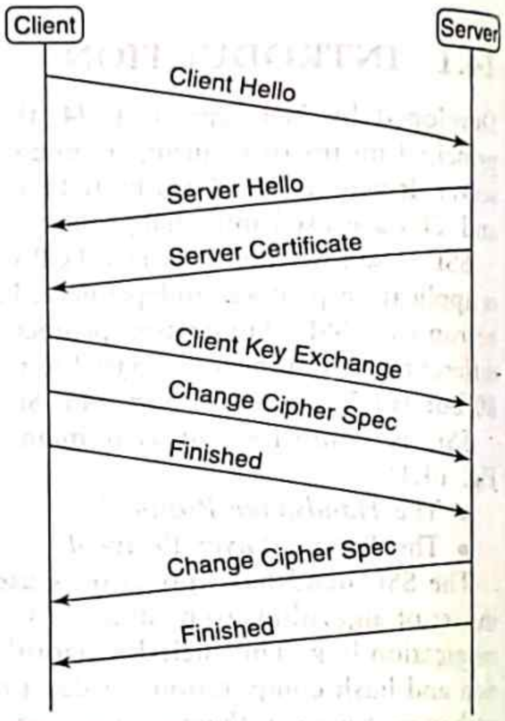


Figure 14.2 *SSL handshake*

In addition to agreeing on a cipher suite, both sides choose and exchange two 32-byte *nonces*, R_A and R_B, 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. The server's certificate is repeatedly transmitted over the internet in the clear and can be easily obtained. Anyone could impersonate the server if dispatch of the certificate was all that was necessary to establish the server's identity. 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* message.

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 MD5.

$$Master\_Secret = f(Pre\text{-}Master\_Secret,\ R_A,\ R_B,\ constants)$$

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

$$Derived\_Secret\_i = f(Master\_Secret,\ R_A,\ R_B,\ constants),\ 1 \le i \le 6$$

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 just negotiated and the keys just 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.

14.2.2 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. In lieu of RSA-based key exchange methods, Diffie–Hellman key exchange may be used. Assume the server has a certificate containing the Diffie-Hellman parameters (prime number p, generator g) and public key $g^a \bmod p$. It communicates this certificate to the client in Step 2. The client chooses a random number b, computes $g^b \bmod p$, and sends it across. Then, the server and client respectively compute $(g^b \bmod p)^a \bmod p$ and $(g^a \bmod p)^b \bmod p$. (Here, a is the private key of the server.) The two computations yield the same result, $g^{ab} \bmod p$, which is the pre-master secret. This option is referred to as *fixed Diffie–Hellman key exchange*.

Even in the absence of a certificate containing Diffie–Hellman parameters, the server can still use Diffie–Hellman key exchange. In this case, the server generates the Diffie–Hellman parameters. Assuming the server has a "signature-only certificate," it now signs p, g, and g^a mod p and sends the parameters, public key and signature to the client who first verifies the signature. The rest of the computations are as in the fixed Diffie–Hellman key exchange. Since the Diffie–Hellman parameters and public key may be chosen anew for different sessions, this variant is referred to as *ephemeral Diffie–Hellman key exchange*.

Server Authentication

The keyed hash (or MAC) computed by both parties and sent in the Finished Messages (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 the choice of 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 keyed hash computed by the server and verified by the client uses the *server MAC secret*. The latter is one of the six derived secrets computed in Step 3. 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 a 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. By far, the largest component of the overhead in creating a new session is the private key operation (decryption of the pre-master secret) at the server. This overhead is obviated by creating a new connection 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.

Apostolopoulos et al. [APOS00] performed experiments to measure the overheads incurred by the Apache 1.2.4 Web server with SSLeay 0.8. They report that SSL can slow down a web server by an order of magnitude. Luckily, session re-use mitigates the overhead. The time taken to set up a new connection (using a partial handshake) is 3 ms. But the overhead of setting up a new session on the server using 1024-bit RSA keys is over 45 ms. The server was a single IBM RS/6000 model 43P-200 running AIX 4.2.

We conclude this section by identifying what constitutes session state and what constitutes the state of a connection.

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 numbers* (one for each direction of message transfer). The latter keep track of the number and order of messages sent and received by each entity in the communication.

14.3 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 are 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 is straightforward – 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.

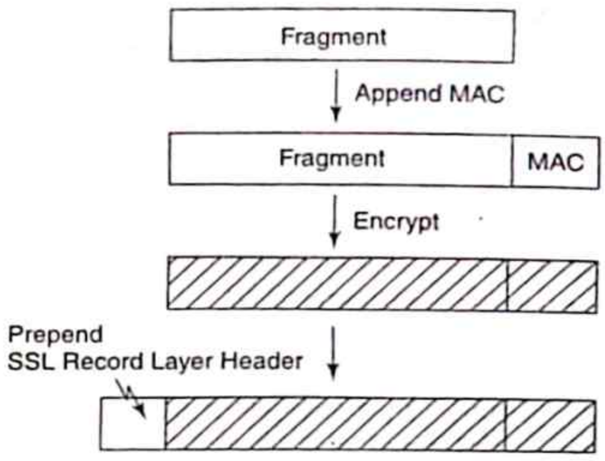


Figure 14.3 *Function of the SSL record layer protocol*

14.4 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. In addition, 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. This means that the developer can focus on the application domain and functional requirements that need to be met and rely on the OpenSSL APIs to implement the required security.

SELECTED REFERENCES

The technical details of the TLS version 1.2 protocol are found in [TLSV1.2] (RFC 5246). A critical analysis of the earlier version of SSL appears in [WAGN96]. [APOS00] and [COAR02] study the cryptographic overheads of SSL on e-commerce servers and suggest possible optimizations. Finally, [OSSL] has many useful details of the OpenSSL project.

OBJECTIVE-TYPE QUESTIONS

14.1 SSL provides security at which layer?
(a) Application (b) Transport
(c) Network (d) Data link

14.2 Which of the following is/are mandatory in SSL?
(a) Server authentication (b) Client authentication
(c) Message confidentiality (d) Non-repudiation

14.3 The SSL protocol assumes that
(a) the server has a digital certificate whose corresponding private key is to be used for signing only
(b) the server has a digital certificate whose corresponding private key is to be used for decryption only
(c) the client has a signing certificate
(d) the client and server share a secret

14.4 Which of the following may be negotiated as part of the SSL handshake?
(a) New session ID (b) Key exchange protocol
(c) Initial sequence number (d) Encryption algorithm

14.5 After a new SSL connection is established
(a) a new pre-master secret is shared between client and server
(b) a new master secret is shared between client and server
(c) an encryption algorithm is agreed upon
(d) new encryption keys are agreed upon

14.6 The SSL Record Layer Protocol handles
(a) entity authentication (b) message encryption
(c) key agreement (d) message integrity checking

14.7 SSL protects against which of the following attacks?
(a) Man-in-the-middle (b) Replay
(c) Denial of service (d) Parallel session
(e) Reflection (f) Dictionary

EXERCISES

14.1 Show the sequence of messages and their contents involved in the exchange of the pre-master secret. Assume that both sides use *ephemeral Diffie–Hellman* key exchange.

14.2 (a) Is there any provision for protecting the *confidentiality* of the SSL *handshake* messages? If so, explain why and how. If not, explain why not.
(b) Is there any provision for protecting the *integrity* of the SSL *handshake* messages? If so, explain why and how. If not, explain why not.

14.3 After which message, and following which computation in SSL, is the client certain that he/she is talking to the authentic server?

14.4 Internet banking is an example of an application that uses SSL. The client/customer authenticates himself/herself to the bank's server through a password. Is the password communicated as part of the handshake? If yes, then as part of which message? If no, then when is it transmitted?

14.5 Both, TCP and SSL use *sequence numbers*. What is the purpose of these different sequence numbers? Instead, does it make sense to have just one sequence number that combines all the necessary functionality? Is it practical to do so? Explain

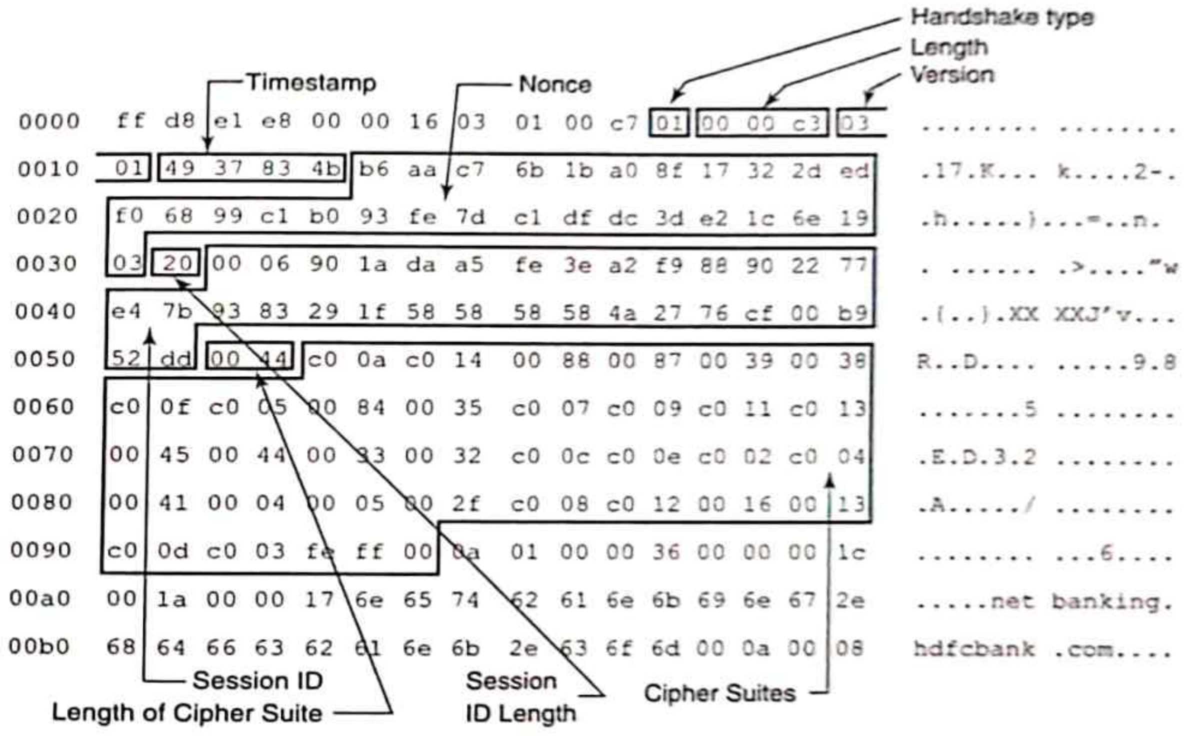
14.6 Under what conditions and in what manner can a *man-in-the-middle attack* be launched on SSL?

14.7 Many *e-commerce sites* use SSL for customer to merchant transactions over the internet. Can you think of any drawbacks in the use of SSL for this purpose and ways to counter these drawbacks?

14.8 Install Wireshark on your machine. Use this to inspect the SSL headers of packets. Specifically, consider logging in to a "secure" site – one that uses HTTPS such as

https://www.myBank.com

An example of the various fields in the SSL Handshake header for the "Client Hello" message is shown below.



```
0000   ff d8 e1 e8 00 00 16 03  01 00 c7 01 00 00 c3 03   ........ ........
0010   01 49 37 83 4b b6 aa c7  6b 1b a0 8f 17 32 2d ed   .17.K... k....2-.
0020   f0 68 99 c1 b0 93 fe 7d  c1 df dc 3d e2 1c 6e 19   .h.....}...=..n.
0030   03 20 00 06 90 1a da a5  fe 3e a2 f9 88 90 22 77   . ...... .>...."w
0040   e4 7b 93 83 29 1f 58 58  58 58 4a 27 76 cf 00 b9   .{..).XX XXJ'v...
0050   52 dd 00 44 c0 0a c0 14  00 88 00 87 00 39 00 38   R..D.... .....9.8
0060   c0 0f c0 05 00 84 00 35  c0 07 c0 09 c0 11 c0 13   .......5 ........
0070   00 45 00 44 00 33 00 32  c0 0c c0 0e c0 02 c0 04   .E.D.3.2 ........
0080   00 41 00 04 00 05 00 2f  c0 08 c0 12 00 16 00 13   .A...../ ........
0090   c0 0d c0 03 fe ff 00 0a  01 00 00 36 00 00 00 1c   ........ ...6....
00a0   00 1a 00 00 17 6e 65 74  62 61 6e 6b 69 6e 67 2e   .....net banking.
00b0   68 64 66 63 62 61 6e 6b  2e 63 6f 6d 00 0a 00 08   hdfcbank .com....
```

Dump the headers for the other Handshake messages such as Server Hello, Client Key Exchange, etc. Attempt to identify and interpret the different fields.

14.9 The goal of this assignment is to obtain familiarity with the *OpenSSL APIs*. For this purpose you need to install the OpenSSL library on two machines – one will play the role of client and the other will play the role of server. (The OpenSSL APIs can be downloaded from http://www.openssl.org/.)

You need to write two programs – one for the client and one for the server. The client program will initiate an SSL handshake with the server to open an SSL connection. It will then request the server to send it the content of a certain file after encrypting it. Some of the APIs that you may need to use include

SSL_set_cipher_list( )
SSL_connect( )
SSL_get_peer_certificate( )
SSL_get_verify_result( )
SSL_accept( )
SSL_read( ) and
SSL_write( )

ANSWERS TO OBJECTIVE-TYPE QUESTIONS

14.1 (b)	14.2 (a)	14.3 (a) or (b)	14.4 (b)(d)
14.5 (b)(d)	14.6 (b)(d)	14.7 (b)(d)(e)(f)	