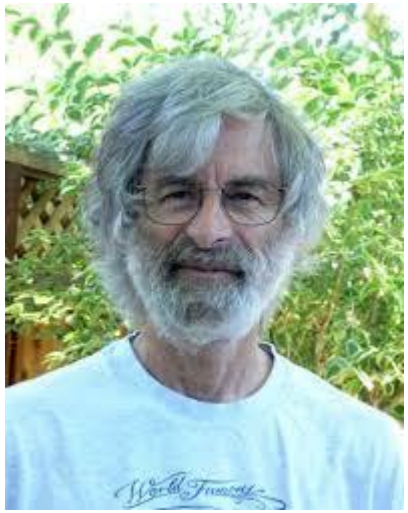
SSL, Certificates, Lamport Encryption

Presentation by:
V. Balasubramanian
SSN College of Engineering

Objectives

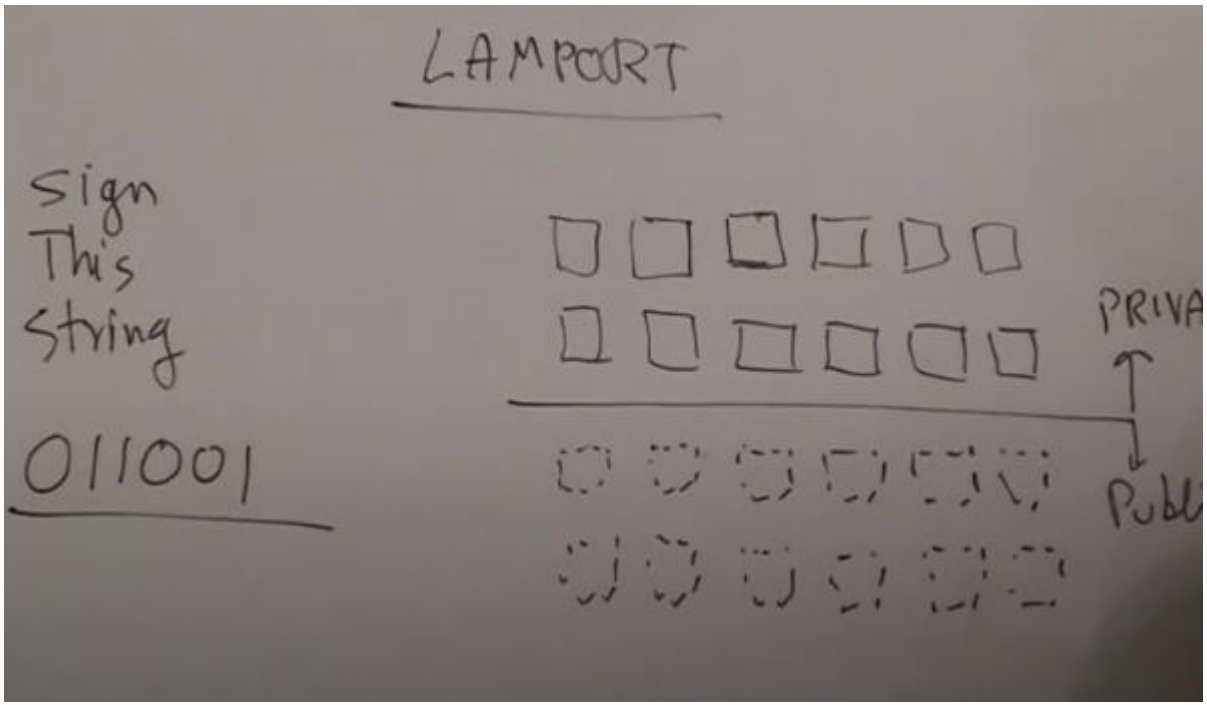
- SSL
- X.509 Certificates
- Lamport Encryption

Leslie Lamport – Turing Award 2013

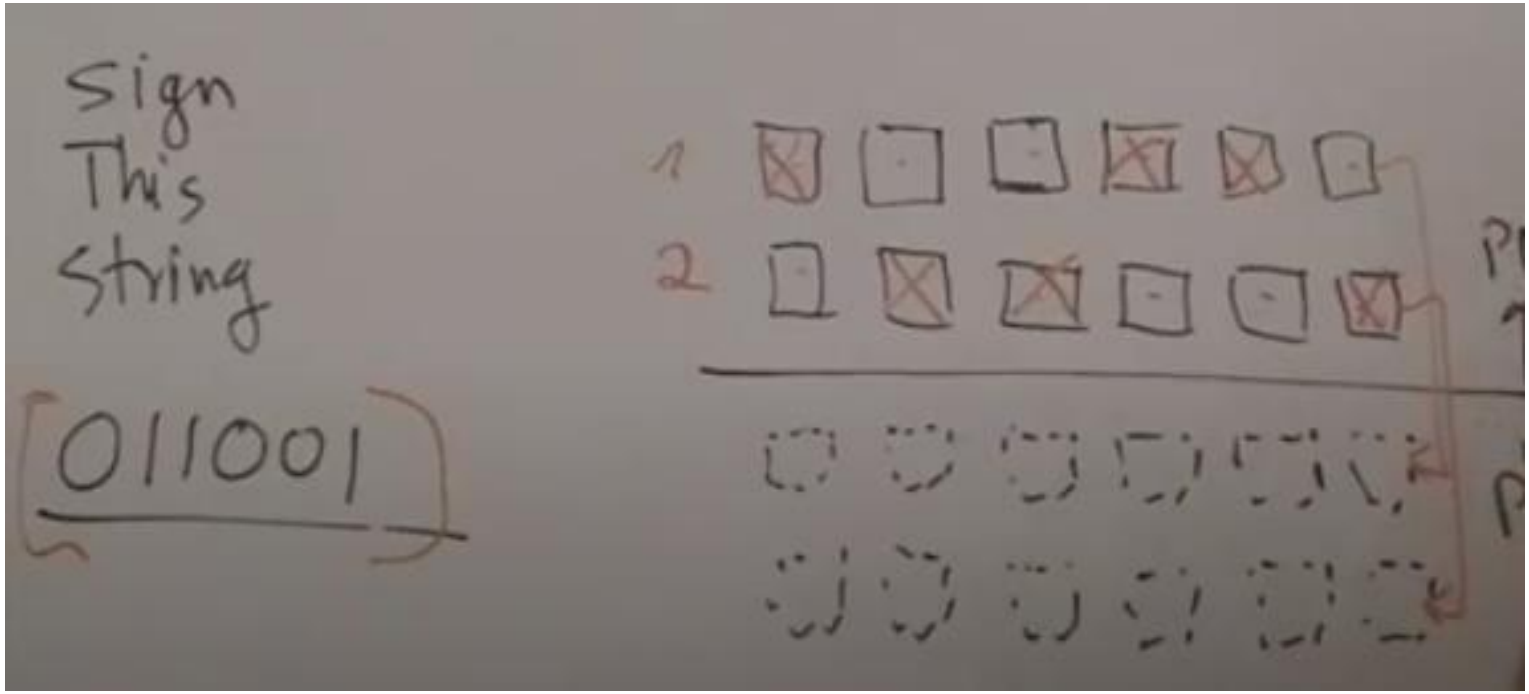


Introduction

Lamport Digital Signature



Digital Signature



$$sk = \begin{bmatrix} x_{1,0} & x_{2,0} & x_{3,0} \\ x_{1,1} & x_{2,1} & x_{3,1} \end{bmatrix}$$

$$pk = \begin{bmatrix} y_{1,0} & y_{2,0} & y_{3,0} \\ y_{1,1} & y_{2,1} & y_{3,1} \end{bmatrix}$$

$$x_{i,j} \in \{0,1\}^n \ (uniform)$$

$$y_{i,j} = f(x_{i,j})$$

f is a One-Way Function

$$sk = \begin{bmatrix} x_{1,0} & x_{2,0} & x_{3,0} \\ x_{1,1} & x_{2,1} & x_{3,1} \end{bmatrix}$$

$$pk = \begin{bmatrix} y_{1,0} & y_{2,0} & y_{3,0} \\ y_{1,1} & y_{2,1} & y_{3,1} \end{bmatrix}$$

$$Sign_{sk}(011) = (x_{1,0}, x_{2,1}, x_{3,1})$$

$$sk = \begin{bmatrix} x_{1,0} & x_{2,0} & x_{3,0} \\ x_{1,1} & x_{2,1} & x_{3,1} \end{bmatrix}$$

$$pk = \begin{bmatrix} y_{1,0} & y_{2,0} & y_{3,0} \\ y_{1,1} & y_{2,1} & y_{3,1} \end{bmatrix}$$

$$Sign_{sk}(011) = (x_{1,0}, x_{2,1}, x_{3,1})$$

$$\mathrm{Vrfy}_{pk}(011, (x_1, x_2, x_3)) = \begin{cases} 1 & \text{if } f(x_1) = y_{1,0} \wedge f(x_2) = y_{2,1} \wedge f(x_3) = y_{3,1} \\ 0 & \text{otherwise} \end{cases}$$

Request signatures of both 0^n and 1^n.

$$sk = \begin{bmatrix} x_{1,0} & x_{2,0} & x_{3,0} \\ x_{1,1} & x_{2,1} & x_{3,1} \end{bmatrix}$$

$$Sign_{sk}(000) = (x_{1,0}, x_{2,0}, x_{3,0})$$

$$Sign_{sk}(111) = (x_{1,1}, x_{2,1}, x_{3,1})$$

Lamport

- 1979
- A message to be signed is a binary k-tuple.
- we assume that the value of k is fixed ahead of time.

We illustrate the scheme by considering one possible implementation using the exponentiation function $f(x) = \alpha^x \bmod p$, where α is a primitive element modulo p. Here $f : \{0, \ldots, p-2\} \rightarrow \mathbb{Z}_p{}^*$. We present a toy example to demonstrate the computations that take place in the scheme.

Illustration

7879 is prime and 3 is a primitive element in $\mathbb{Z}_{7879}{}^*$. Define

$$f(x) = 3^x \bmod 7879.$$

Suppose $k = 3$, and Alice chooses the six (secret) random numbers

$$\begin{aligned} y_{1,0} &= 5831 \\ y_{1,1} &= 735 \\ y_{2,0} &= 803 \\ y_{2,1} &= 2467 \\ y_{3,0} &= 4285 \\ y_{3,1} &= 6449. \end{aligned}$$

Then Alice computes the images of these six y's under the function f:

$$\begin{aligned} z_{1,0} &= 2009 \\ z_{1,1} &= 3810 \\ z_{2,0} &= 4672 \\ z_{2,1} &= 4721 \\ z_{3,0} &= 268 \\ z_{3,1} &= 5731. \end{aligned}$$

These z's are published. Now, suppose Alice wants to sign the message

$$x = (1, 1, 0).$$

The signature for x is

$$(y_{1,1}, y_{2,1}, y_{3,0}) = (735, 2467, 4285).$$

To verify this signature, it suffices to compute the following:

$$\begin{aligned} 3^{735} \bmod 7879 &= 3810 \\ 3^{2467} \bmod 7879 &= 4721 \\ 3^{4285} \bmod 7879 &= 268. \end{aligned}$$

Hence, the signature is verified.

Lamport Signature

- signature schemes based on hash functions by considering the relatively weak notion of one-time-secure signature schemes.
- Informally, such schemes are “secure” as long as a given private key is used to sign only a single message.

Let $\Pi = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ be a signature scheme, and consider the following experiment for an adversary $\mathcal{A}$ and parameter n:

The one-time signature experiment $\mathsf{Sig\text{-}forge}^{\text{1-time}}_{\mathcal{A},\Pi}(n)$**:**

1. $\mathsf{Gen}(1^n)$ *is run to obtain keys* (pk, sk).
2. *Adversary* $\mathcal{A}$ *is given* pk *and asks a* single *query* m' *to its oracle* $\mathsf{Sign}_{sk}(\cdot)$. $\mathcal{A}$ *then outputs* (m, σ) *with* $m \neq m'$.
3. *The output of the experiment is defined to be* 1 *if and only if* $\mathsf{Vrfy}_{pk}(m, \sigma) = 1$.

DEFINITION 12.14 *Signature scheme* $\Pi = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ *is* existentially unforgeable under a single-message attack, *or is a* one-time-secure signature scheme, *if for all probabilistic polynomial-time adversaries* $\mathcal{A}$, *there exists a negligible function* negl *such that:*

$$\Pr\left[\mathsf{Sig\text{-}forge}^{1\text{-}\mathsf{time}}_{\mathcal{A},\Pi}(n) = 1\right] \leq \mathsf{negl}(n).$$

Algorithm

Leslie Lamport showed a construction of a one-time-secure signature scheme in 1979. We illustrate the idea for the case of signing 3-bit messages. Let H be a cryptographic hash function. A private key consists of six uniform values $x_{1,0},\ x_{1,1},\ x_{2,0},\ x_{2,1},\ x_{3,0},\ x_{3,1} \in \{0,1\}^n$, and the corresponding public key contains the results obtained by applying H to each of these elements. These keys can be visualized as two-dimensional arrays:

$$pk = \begin{pmatrix} y_{1,0} & y_{2,0} & y_{3,0} \\ y_{1,1} & y_{2,1} & y_{3,1} \end{pmatrix} \quad sk = \begin{pmatrix} x_{1,0} & x_{2,0} & x_{3,0} \\ x_{1,1} & x_{2,1} & x_{3,1} \end{pmatrix}.$$

Sign & Verify

To sign a message $m = m_1 m_2 m_3$ (where $m_i \in \{0, 1\}$), the signer releases the appropriate preimage x_{i,m_i} for each bit of the message; the signature σ consists of the three values $(x_{1,m_1}, x_{2,m_2}, x_{3,m_3})$. Verification is carried out in the natural way: presented with the candidate signature (x_1, x_2, x_3) on the message $m = m_1 m_2 m_3$, accept if and only if $H(x_i) \stackrel{?}{=} y_{i,m_i}$ for $1 \leq i \leq 3$. This is shown graphically in Figure 12.3, and the general case—for messages of any length ℓ—is described formally in Construction 12.15.

Illustration

Signing $m = 011$:

$$sk = \begin{pmatrix} \boxed{x_{1,0}} & x_{2,0} & x_{3,0} \\ x_{1,1} & \boxed{x_{2,1}} & \boxed{x_{3,1}} \end{pmatrix} \Rightarrow \sigma = (x_{1,0}, x_{2,1}, x_{3,1})$$

Verifying for $m = 011$ and $\sigma = (x_1, x_2, x_3)$:

$$pk = \left.\begin{pmatrix} \boxed{y_{1,0}} & y_{2,0} & y_{3,0} \\ y_{1,1} & \boxed{y_{2,1}} & \boxed{y_{3,1}} \end{pmatrix}\right\} \Rightarrow \begin{aligned} H(x_1) &\stackrel{?}{=} y_{1,0} \\ H(x_2) &\stackrel{?}{=} y_{2,1} \\ H(x_3) &\stackrel{?}{=} y_{3,1} \end{aligned}$$

Lamport signature scheme

CONSTRUCTION 12.15

Let $H : \{0,1\}^* \to \{0,1\}^*$ be a function. Construct a signature scheme for messages of length $\ell = \ell(n)$ as follows:

- Gen: on input 1^n, proceed as follows for $i \in \{1, \ldots, \ell\}$:

  1. Choose uniform $x_{i,0}, x_{i,1} \in \{0,1\}^n$.
  2. Compute $y_{i,0} := H(x_{i,0})$ and $y_{i,1} := H(x_{i,1})$.

  The public key pk and the private key sk are

$$pk = \begin{pmatrix} y_{1,0} & y_{2,0} & \cdots & y_{\ell,0} \\ y_{1,1} & y_{2,1} & \cdots & y_{\ell,1} \end{pmatrix} \quad sk = \begin{pmatrix} x_{1,0} & x_{2,0} & \cdots & x_{\ell,0} \\ x_{1,1} & x_{2,1} & \cdots & x_{\ell,1} \end{pmatrix}.$$

- Sign: on input a private key sk as above and a message $m \in \{0,1\}^\ell$ with $m = m_1 \cdots m_\ell$, output the signature $(x_{1,m_1}, \ldots, x_{\ell,m_\ell})$.
- Vrfy: on input a public key pk as above, a message $m \in \{0,1\}^\ell$ with $m = m_1 \cdots m_\ell$, and a signature $\sigma = (x_1, \ldots, x_\ell)$, output 1 if and only if $H(x_i) = y_{i,m_i}$ for all $1 \le i \le \ell$.

Adversary

Algorithm $\mathcal{I}$:
The algorithm is given 1^n and y as input.

1. Choose uniform $i^* \in \{1, \ldots, \ell\}$ and $b^* \in \{0,1\}$. Set $y_{i^*,b^*} := y$.
2. For all $i \in \{1, \ldots, \ell\}$ and $b \in \{0,1\}$ with $(i,b) \neq (i^*,b^*)$:
   - Choose uniform $x_{i,b} \in \{0,1\}^n$ and set $y_{i,b} := H(x_{i,b})$.
3. Run $\mathcal{A}$ on input $pk := \begin{pmatrix} y_{1,0} & y_{2,0} & \cdots & y_{\ell,0} \\ y_{1,1} & y_{2,1} & \cdots & y_{\ell,1} \end{pmatrix}$.
4. When $\mathcal{A}$ requests a signature on the message m':
   - If $m'_{i^*} = b^*$, then $\mathcal{I}$ aborts the execution.
   - Otherwise, $\mathcal{I}$ returns the signature $\sigma = (x_{1,m'_1}, \ldots, x_{\ell,m'_\ell})$.
5. When $\mathcal{A}$ outputs (m, σ) with $\sigma = (x_1, \ldots, x_\ell)$:
   - If $\mathcal{A}$ outputs a forgery at (i^*, b^*), then output x_{i^*}.

Negligible Probability

$$\Pr[\mathsf{Invert}_{\Pi,H}(n) = 1] \geq \frac{1}{2\ell} \cdot \Pr[\mathsf{Sig\text{-}forge}^{\text{1-time}}_{\mathcal{A},\Pi}(n) = 1].$$

Because H is a one-way function, there is a negligible function negl such that

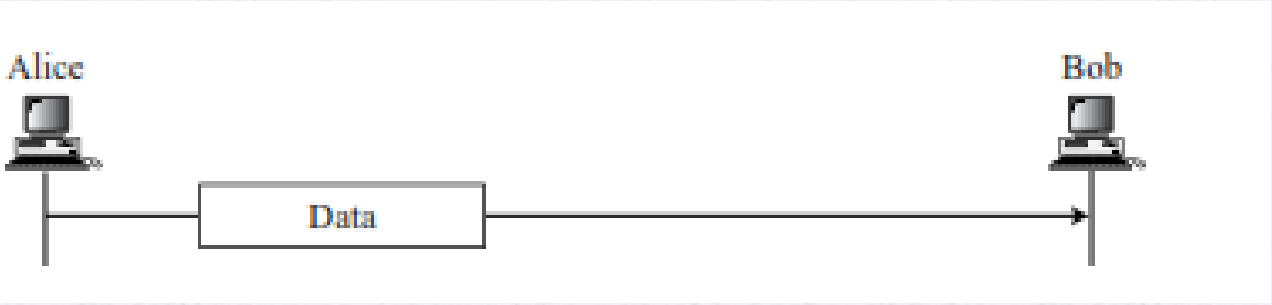
$$\mathsf{negl}(n) \geq \Pr[\mathsf{Invert}_{\Pi,H}(n) = 1].$$

Since ℓ is polynomial this implies that $\Pr[\mathsf{Sig\text{-}forge}^{\text{1-time}}_{\mathcal{A},\Pi}(n) = 1]$ is negligible, completing the proof. ■

Preety Good Privacy

- The first protocol discussed in this chapter is called Pretty Good Privacy (PGP).
- PGP was invented by Phil Zimmermann to provide e-mail with privacy, integrity, and authentication. PGP can be used to create a secure e-mail message or to store a file securely for future retrieval

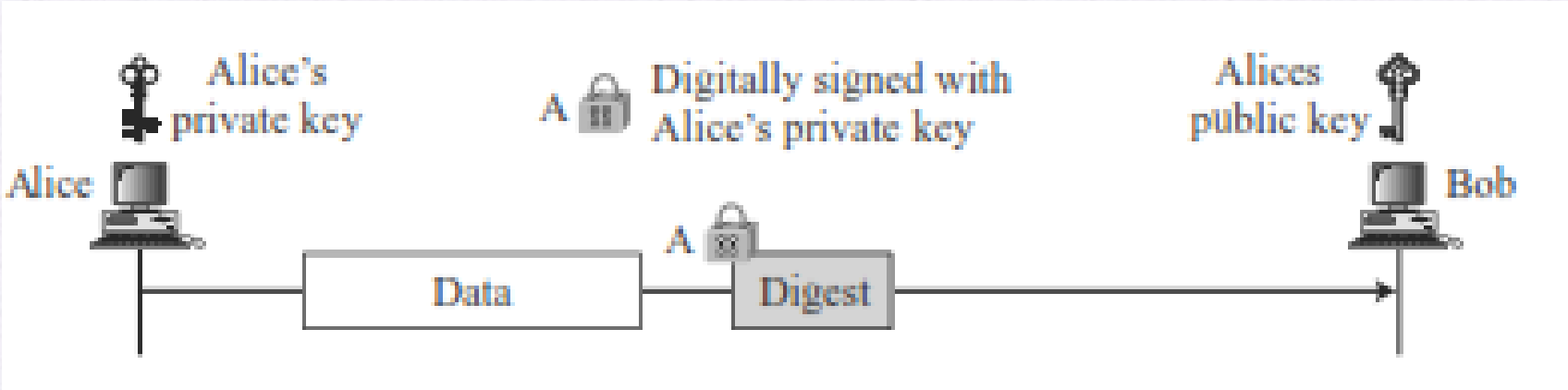
Plain Text



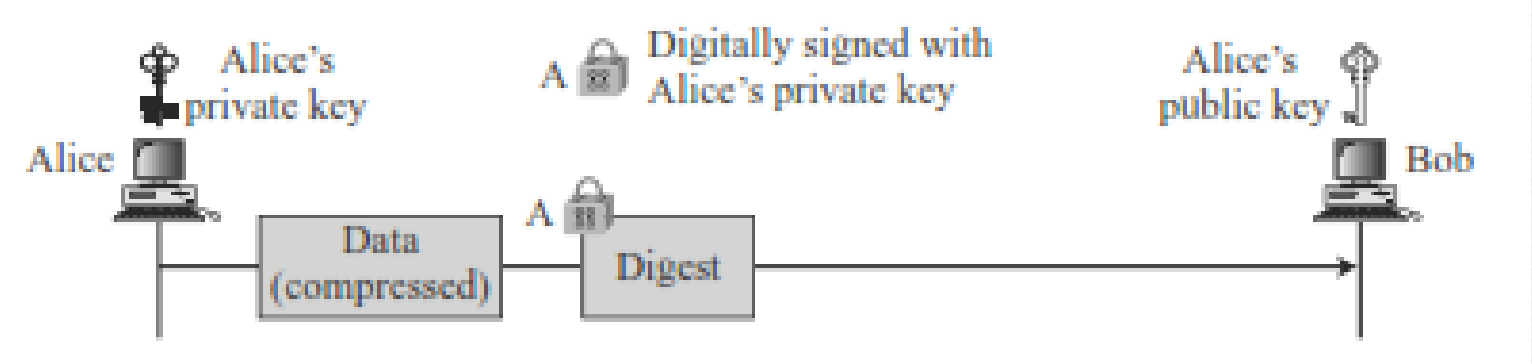
Plaintext

The simplest scenario is to send the e-mail message (or store the file) in plaintext as shown in Figure 16.2. There is no message integrity or confidentiality in this scenario. Alice, the sender, composes a message and sends it to Bob, the receiver. The message is stored in Bob's mailbox until it is retrieved by him.

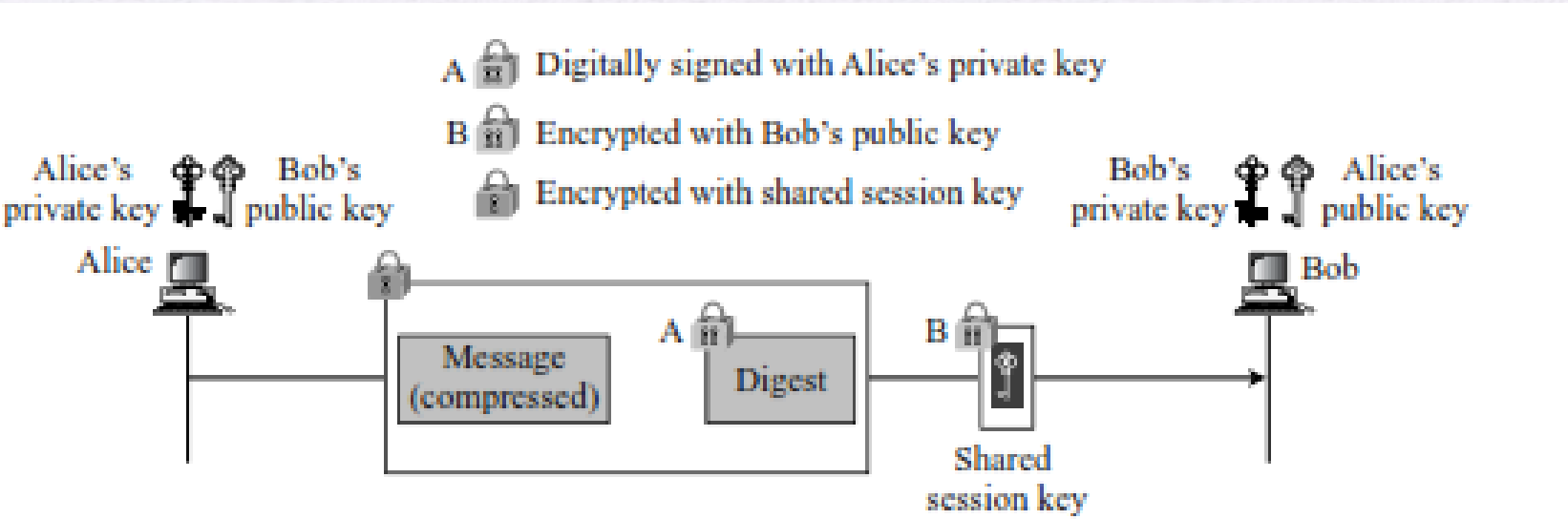
Authenticated Message



Compression (Eases Traffic)



Confidential Message



Code Conversion

Another service provided by PGP is code conversion. Most e-mail systems allow the message to consist of only ASCII characters. To translate other characters not in the ASCII set, PGP uses Radix-64 conversion. Each character to be sent (after encryption) is converted to Radix-64 code, which is discussed later in the chapter.

Segmentation

PGP allows segmentation of the message after it has been converted to Radix-64 to make each transmitted unit the uniform size as allowed by the underlying e-mail protocol.

Working

1. Alice needs to send a message to another person in the community.
   a. She uses her private key to sign the digest.
   b. She uses the receiver's public key to encrypt a newly created session key.
   c. She encrypts the message and signed digest with the session key created.

Alice receives a message from another person in the community.

a. She uses her private key to decrypt the session key.
b. She uses the session key to decrypt the message and digest.
c. She uses her public key to verify the digest.

PGP algorithms

Public-Key Algorithms The public-key algorithms that are used for signing the digests or encrypting the messages are listed in Table 16.1.

Table 16.1 *Public-key algorithms*

ID	*Description*
1	RSA (encryption or signing)
2	RSA (for encryption only)
3	RSA (for signing only)
16	ElGamal (encryption only)
17	DSS
18	Reserved for elliptic curve
19	Reserved for ECDSA
20	ElGamal (for encryption or signing)
21	Reserved for Diffie-Hellman
100–110	Private algorithms

Symmetric Key algorithms

Table 16.2 *Symmetric-key algorithms*

ID	*Description*
0	No Encryption
1	IDEA
2	Triple DES
3	CAST-128
4	Blowfish
5	SAFER-SK128
6	Reserved for DES/SK
7	Reserved for AES-128
8	Reserved for AES-192
9	Reserved for AES-256
100–110	Private algorithms

Hash algorithms

Hash Algorithms The hash algorithms that are used for creating hashes in PGP are shown in Table 16.3.

Table 16.3 *Hash Algorithms*

ID	*Description*
1	MD5
2	SHA-1
3	RIPE-MD/160
4	Reserved for double-width SHA
5	MD2
6	TIGER/192
7	Reserved for HAVAL
100–110	Private algorithms

Compression Algorithms The compression algorithms that are used for compressing text are shown in Table 16.4.

Table 16.4 *Compression methods*

ID	*Description*
0	Uncompressed
1	ZIP
2	ZLIP
100–110	Private methods

- confidentiality
  - protection from disclosure
- authentication
  - of sender of message
- message integrity
  - protection from modification
- non-repudiation of origin
  - protection from denial by sender

X.509

- **List four general categories of schemes for the distribution of public keys.**
- Public announcement. Publicly available directory. Public-key authority,Public-key certificates

- What is a public-key certificate?
- A public-key certificate contains a public key and other information, is created by a certificate authority, and is given to the participant with the matching private key. A participant conveys its key information to another by transmitting its certificate. Other participants can verify that the certificate was created by the authority.

- **What are the requirements for the use of a public-key certificate scheme?**

- **1. Any participant can read a certificate to determine the name and** public key of the certificate's owner. **2. Any participant can verify that** the certificate originated from the certificate authority and is not counterfeit. **3. Only the certificate authority can create and update** certificates. **4. Any participant can verify the currency of the** certificate.

- **What is the purpose of the X.509 standard?**

- X.509 defines a framework for the provision of authentication services by the X.500 directory to its users. The directory may serve as a repository of public-key certificates. Each certificate contains the public key of a user and is signed with the private key of a trusted certification authority.

- What is a chain of certificates?
- A chain of certificates consists of a sequence of certificates created by different certification authorities (CAs) in which each successive certificate is a certificate by one CA that certifies the public key of the next CA in the chain.

- How is an X.509 certificate revoked?
- The owner of a public-key can issue a certificate revocation list that revokes one or more certificates.

Public Announcement

Figure 14.10 Uncontrolled Public Key Distribution

Publicly Available Directory

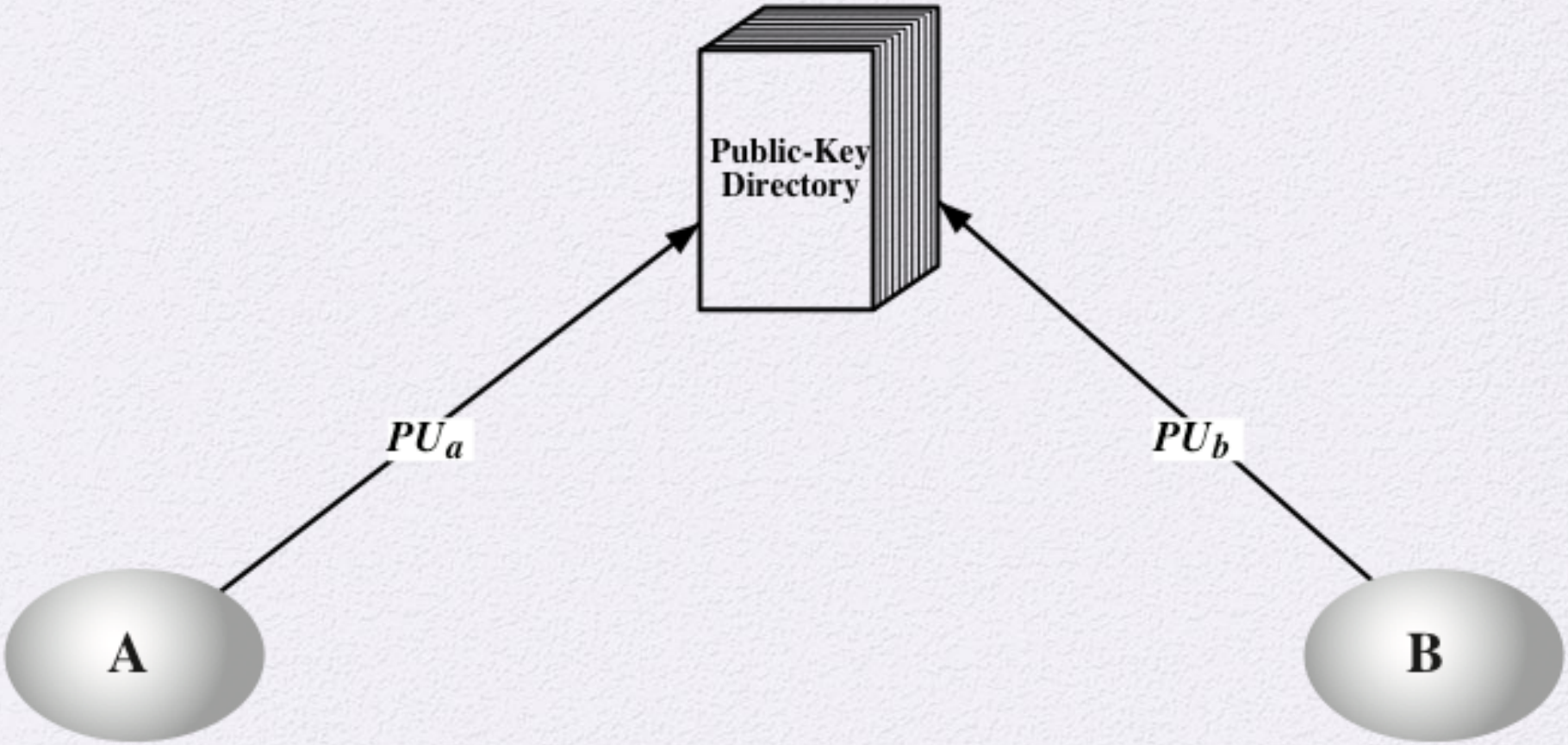


Figure 14.11 Public Key Publication

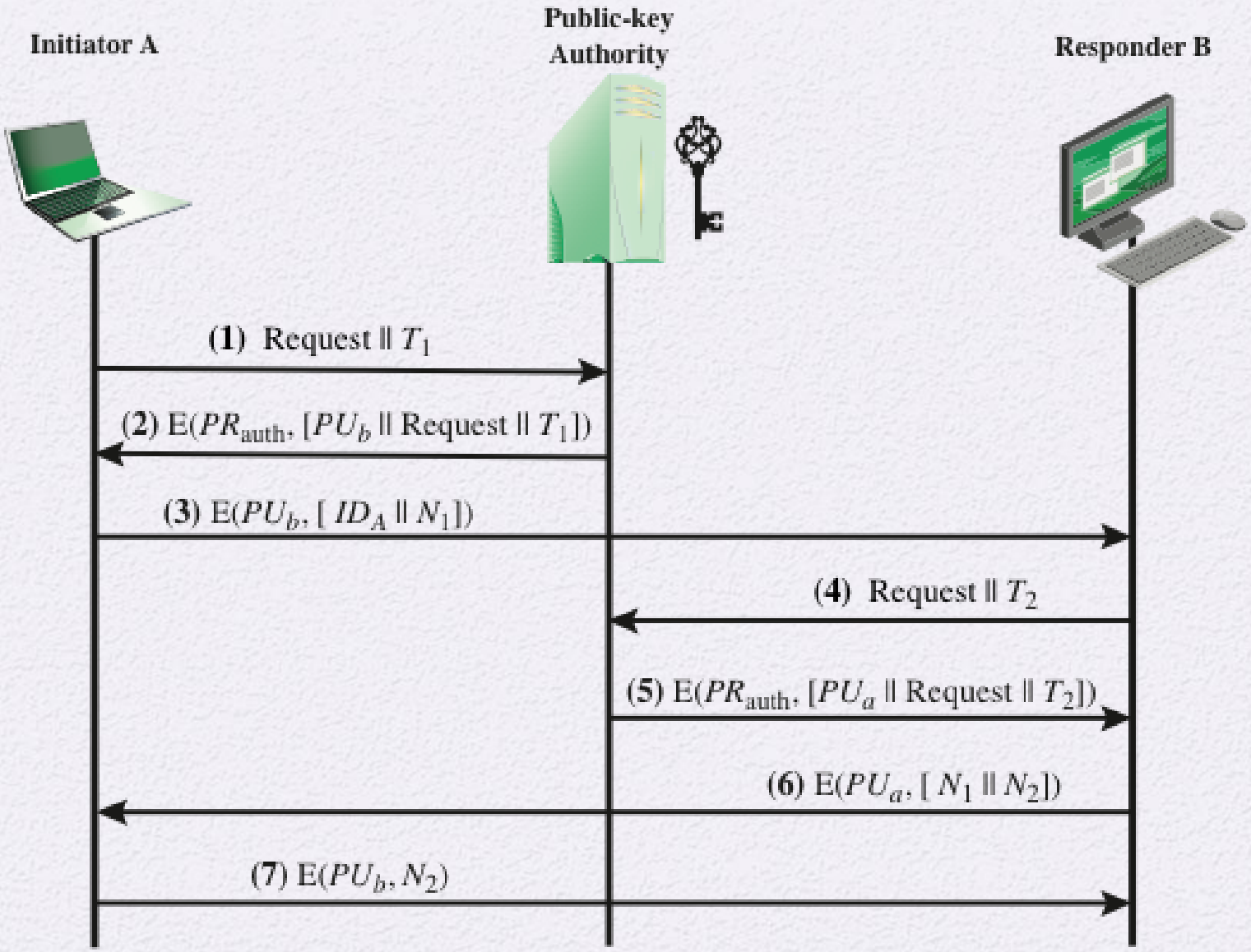
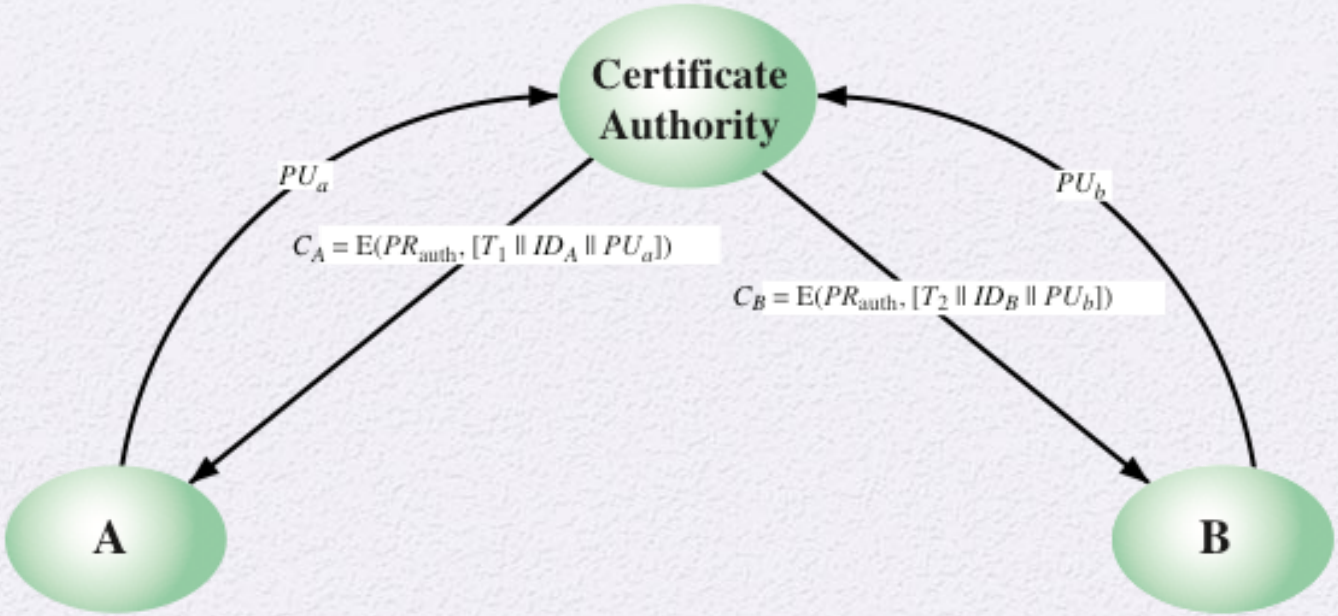
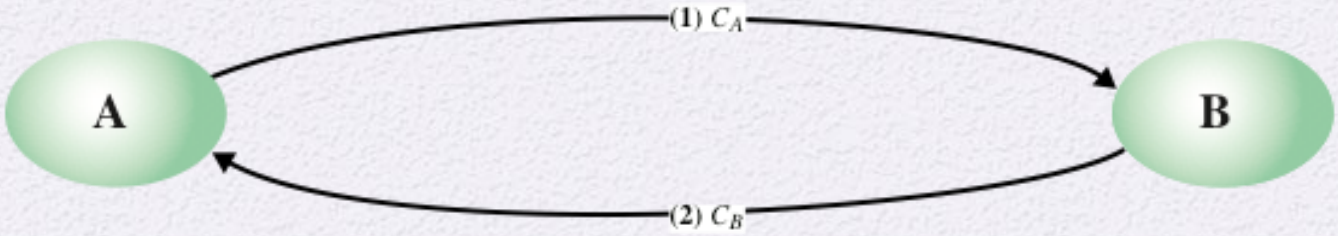


Figure 14.12 Public-Key Distribution Scenario



(a) Obtaining certificates from CA



(b) Exchanging certificates

Figure 14.13 Exchange of Public-Key Certificates

X.509 Certificates

- Part of the X.500 series of recommendations that define a directory service
  - The directory is, in effect, a server or distributed set of servers that maintains a database of information about users
- X.509 defines a framework for the provision of authentication services by the X.500 directory to its users
  - Was initially issued in 1988 with the latest revision in 2000
  - Based on the use of public-key cryptography and digital signatures
  - Does not dictate the use of a specific algorithm but recommends RSA
  - Does not dictate a specific hash algorithm
- Each certificate contains the public key of a user and is signed with the private key of a trusted certification authority
- X.509 defines alternative authentication protocols based on the use of public-key certificates

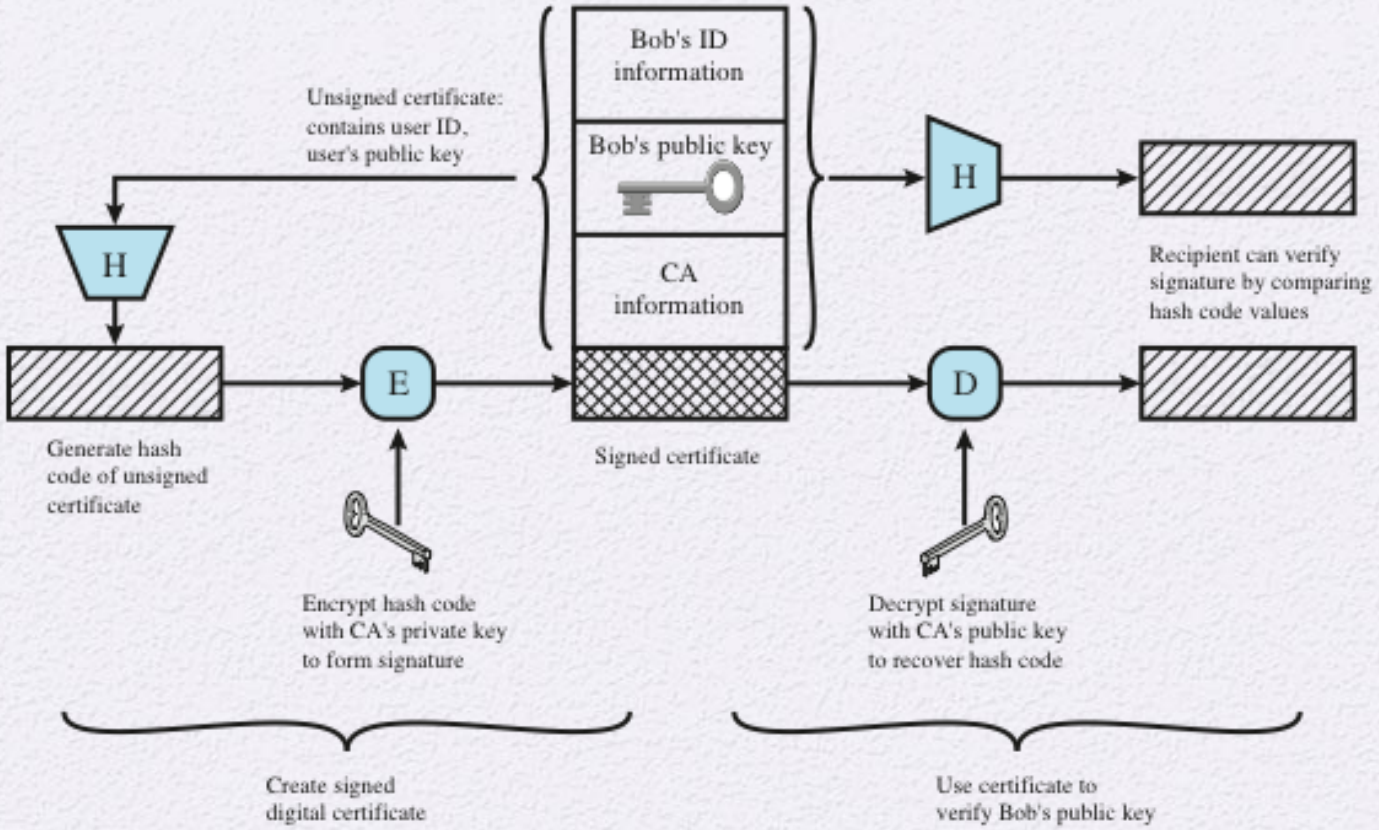


Figure 14.14 Public-Key Certificate Use

Certificates

Created by a trusted Certification Authority (CA) and have the following elements:

- Version
- Serial number
- Signature algorithm identifier
- Issuer name
- Period of validity
- Subject name
- Subject's public-key information
- Issuer unique identifier
- Subject unique identifier
- Extensions
- Signature

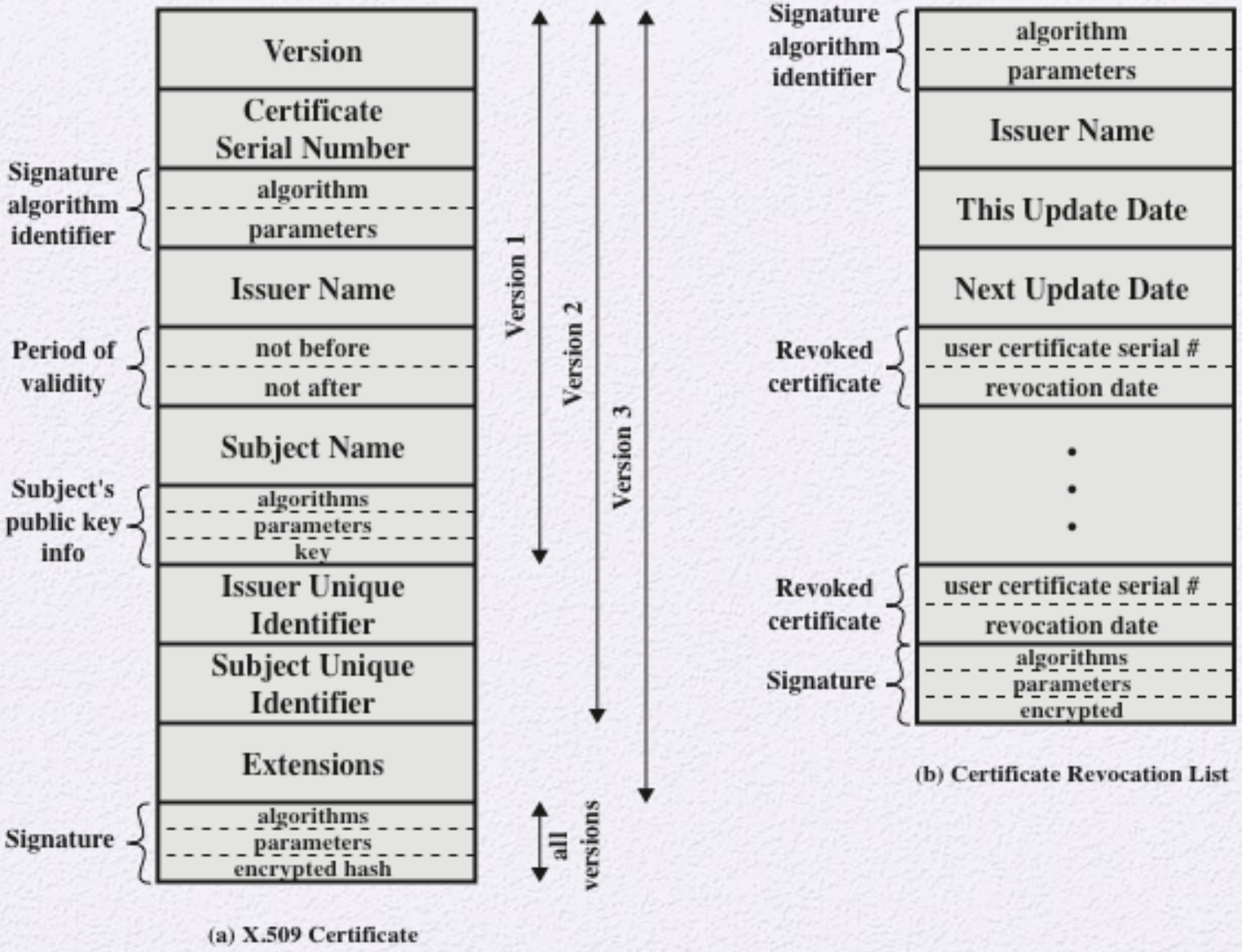


Figure 14.15 X.509 Formats

Obtaining a Certificate

User certificates generated by a CA have the following characteristics:

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

- Because certificates are unforgeable, they can be placed in a directory without the need for the directory to make special efforts to protect them
  - In addition, a user can transmit his or her certificate directly to other users
- Once B is in possession of A's certificate, B has confidence that messages it encrypts with A's public key will be secure from eavesdropping and that messages signed with A's private key are unforgeable

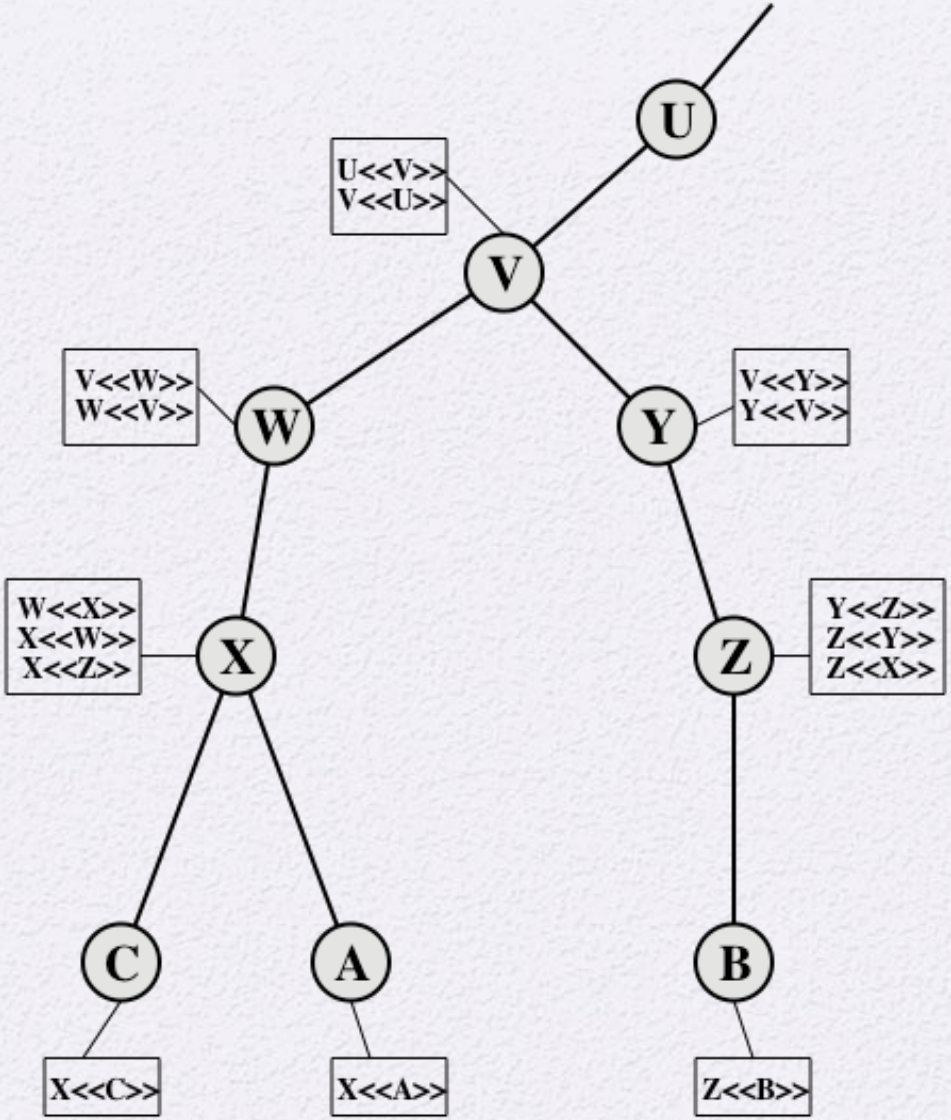


Figure 14.16 X.509 CA Hierarchy: a Hypothetical Example

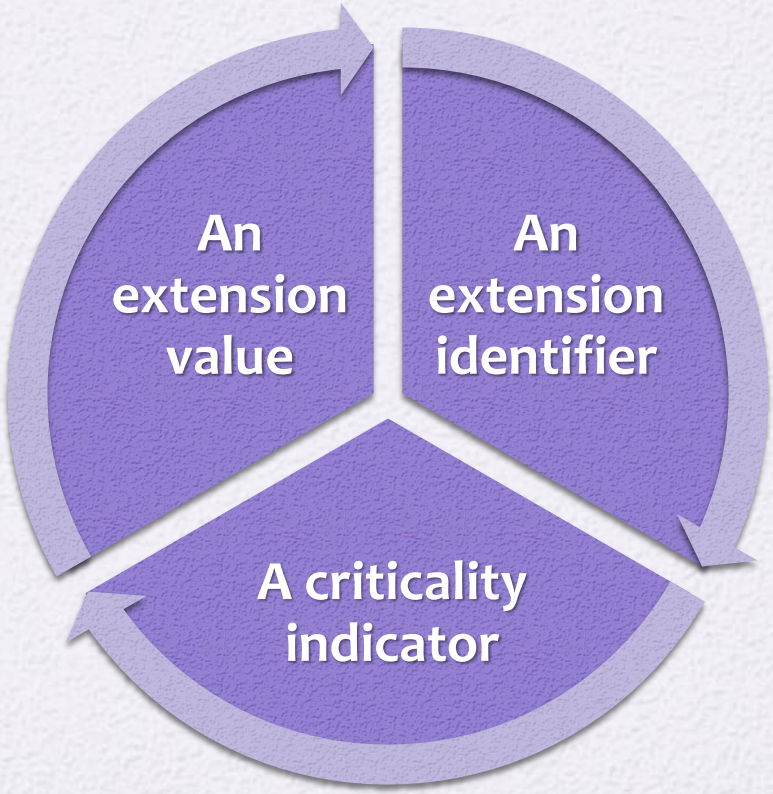
Certificate Revocation

- Each certificate includes a period of validity
  - Typically a new certificate is issued just before the expiration of the old one
- It may be desirable on occasion to revoke a certificate before it expires, for one of the following reasons:
  - The user's private key is assumed to be compromised
  - The user is no longer certified by this CA
  - The CA's certificate is assumed to be compromised
- Each CA must maintain a list consisting of all revoked but not expired certificates issued by that CA
  - These lists should be posted on the directory

X.509 Version 3

- Version 2 format does not convey all of the information that recent design and implementation experience has shown to be needed
- Rather than continue to add fields to a fixed format, standards developers felt that a more flexible approach was needed
  - Version 3 includes a number of optional extensions
- The certificate extensions fall into three main categories:
  - Key and policy information
  - Subject and issuer attributes
  - Certification path constraints

Each extension consists of:



- Certificate Signing Requests are formal requests asking a CA to sign a certificate. The CSR is signed with the private key and internally the CSR contains the details of the organization and the public key.
- 1. First you create a key pair.
- 2. Store the private key carefully
- 3. The public key must be given to the CA

- 4. Certificates also carry identity of the person / organization who owns the public - private key pair.
- 5. This identity must be verified, typically in an out-of-band way. For instance a domain is verified by running a domain query. User certificates must be identified by a personal identity card. And so on. Different CAs have different methods and requirements to verify identity. For web PKI and browser use of certificates, there are Baseline Requirements defined by the CA/Browser Forum.
- 6. Once the CA verifies the identity, they sign and issue a certificate
- 7. The certificate can also be self-signed but it's trustworthiness is dependent on the user.

- Create Key Pair
- Create a key pair and extract the public key into a separate file. Use a password if you would like to.
- ==> openssl genrsa -out certkey.key 2048
- Generating RSA private key, 2048 bit long modulus (2 primes)
- openssl rsa -pubout -out certpubkey.key -in certkey.key

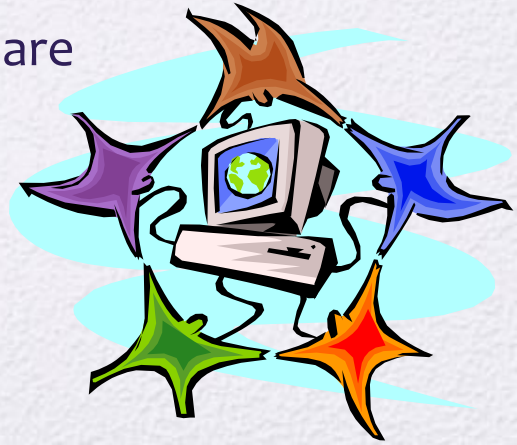
- Create CSR
- ==> openssl req -new -key certkey.key -out cert.csr
- You are about to be asked to enter information that will be incorporated
- into your certificate request.
- What you are about to enter is what is called a Distinguished Name or a DN.
- There are quite a few fields but you can leave some blank
- For some fields there will be a default value,
- If you enter '.', the field will be left blank.
- -----
- Country Name (2 letter code) [AU]:IN

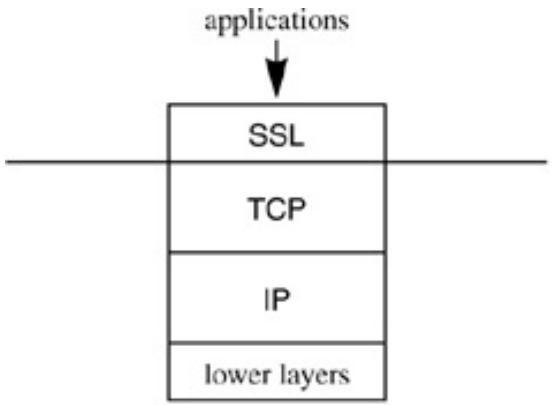
- State or Province Name (full name) [Some-State]:Tamil Nadu
- Locality Name (eg, city) []:Chennai
- Organization Name (eg, company) [Internet Widgits Pty Ltd]:My Cool Company Ltd
- Organizational Unit Name (eg, section) []:Finance
- Common Name (e.g. server FQDN or YOUR name) []:www.coolcompany.example
- Email Address []:admin@coolcompany.example
- Please enter the following 'extra' attributes
- to be sent with your certificate request
- A challenge password []:
- An optional company name []:

- ==> openssl req -in cert.csr -noout -text
- Self-sign certificate
- Openssl x509 -req -days 365 -in cert.csr -signkey certkey.key -out cert.crt

Web Security Considerations

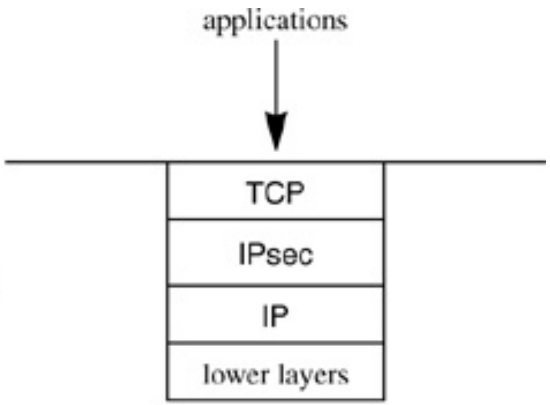
- The World Wide Web is fundamentally a client/server application running over the Internet and TCP/IP intranets
- The following characteristics of Web usage suggest the need for tailored security tools:
  - Web servers are relatively easy to configure and manage
  - Web content is increasingly easy to develop
  - The underlying software is extraordinarily complex
    - May hide many potential security flaws
  - A Web server can be exploited as a launching pad into the corporation's or agency's entire computer complex
  - Casual and untrained (in security matters) users are common clients for Web-based services
    - Such users are not necessarily aware of the security risks that exist and do not have the tools or knowledge to take effective countermeasures



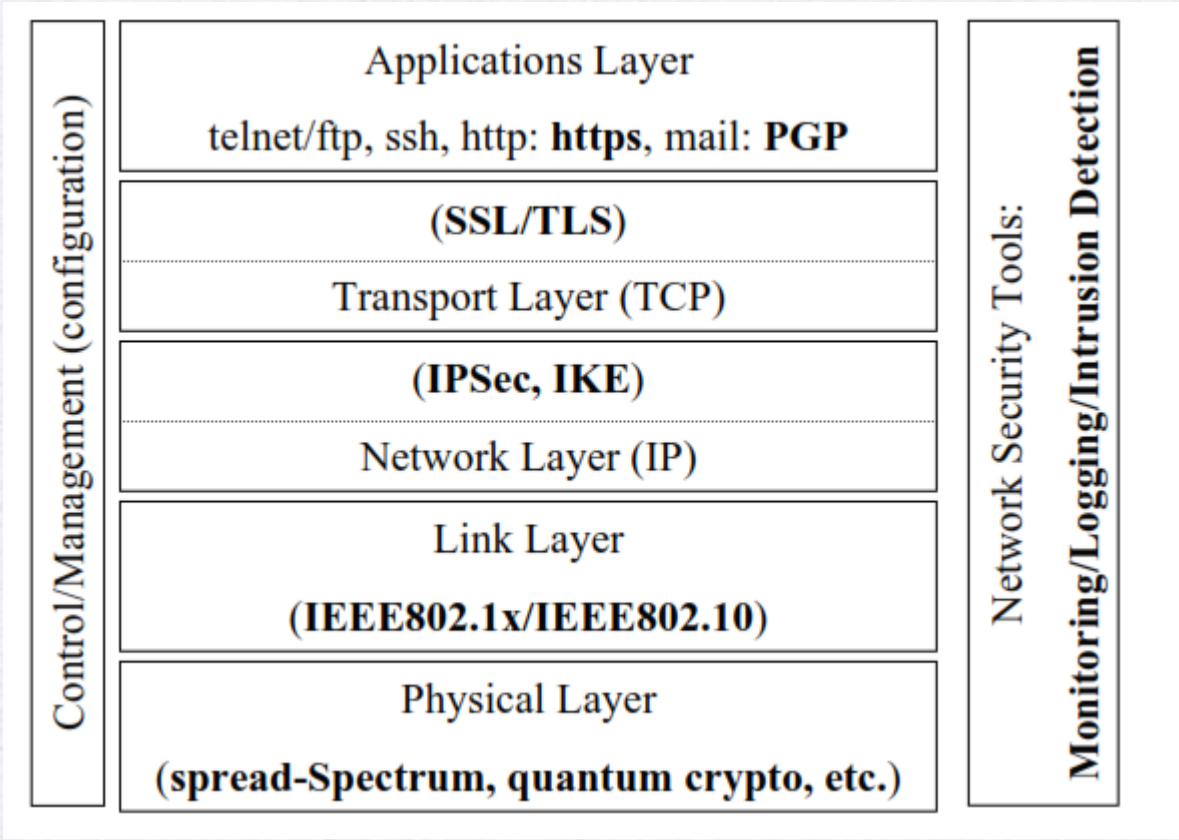


SSL/TLS or SSH operate above TCP.
OS doesn't change. Applications do.

OS



IPsec is within the OS. OS changes.
Applications and API to TCP are unchanged.


Control/Management (configuration)
Applications Layer
telnet/ftp, ssh, http: **https**, mail: **PGP**
(SSL/TLS)
Transport Layer (TCP)
(IPSec, IKE)
Network Layer (IP)
Link Layer
(IEEE802.1x/IEEE802.10)
Physical Layer
(spread-Spectrum, quantum crypto, etc.)
Network Security Tools:
Monitoring/Logging/Intrusion Detection

- SSL:
  - Avoids modifying "TCP stack" and requires minimum changes to the application
  - Mostly used to authenticate servers
- IPsec
  - Transparent to the application and requires modification of the network stack
  - Authenticates network nodes and establishes a secure channel between nodes
  - Application still needs to authenticate the users

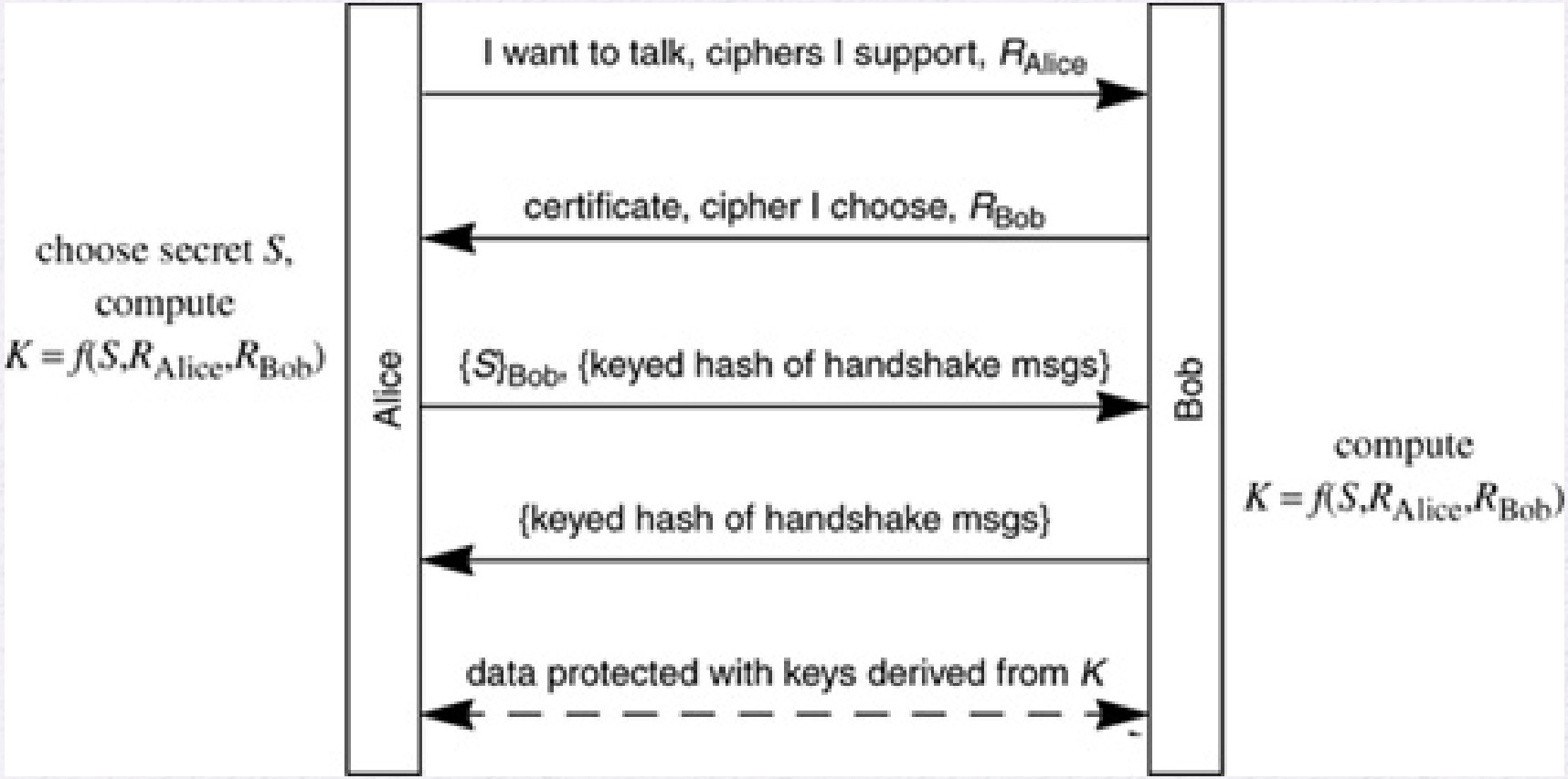
SSL/TLS

- SSL/TLS allows two parties to authenticate and establish a session key that is used to cryptographically protect the remainder of the session.
- Version 3

- SSL/TLS is designed to run in a user-level process, and runs on top of TCP.
- SSL/TLS a little simpler, because it doesn't have to worry about timing out and retransmitting lost data using TCP.
- UDP
- user-level process rather than requiring OS changes

- SSL (Secure Sockets Layer) version 2 (version 1 was never deployed) was originally deployed in Netscape Navigator in 1995.
- Microsoft improved upon SSLv2, fixing some security problems
- Netscape - version3
- SSLv3 is the most commonly deployed

SSL v3 / TLS


Alice
Bob
I want to talk, ciphers I support, R_{Alice}
certificate, cipher I choose, R_{Bob}
choose secret S,
compute
$K = f(S, R_{Alice}, R_{Bob})$
$\{S\}_{Bob}$, {keyed hash of handshake msgs}
compute
$K = f(S, R_{Alice}, R_{Bob})$
{keyed hash of handshake msgs}
data protected with keys derived from K

SSL

- Using the reliable octet stream service provided by TCP, SSL/TLS partitions this octet stream into records, with headers and cryptographic protection, to provide a reliable, encrypted, and integrity-protected stream of octets to the application.

Record Types

- four types of records:
  - user data,
  - handshake messages,
  - Alerts
  - change cipher spec

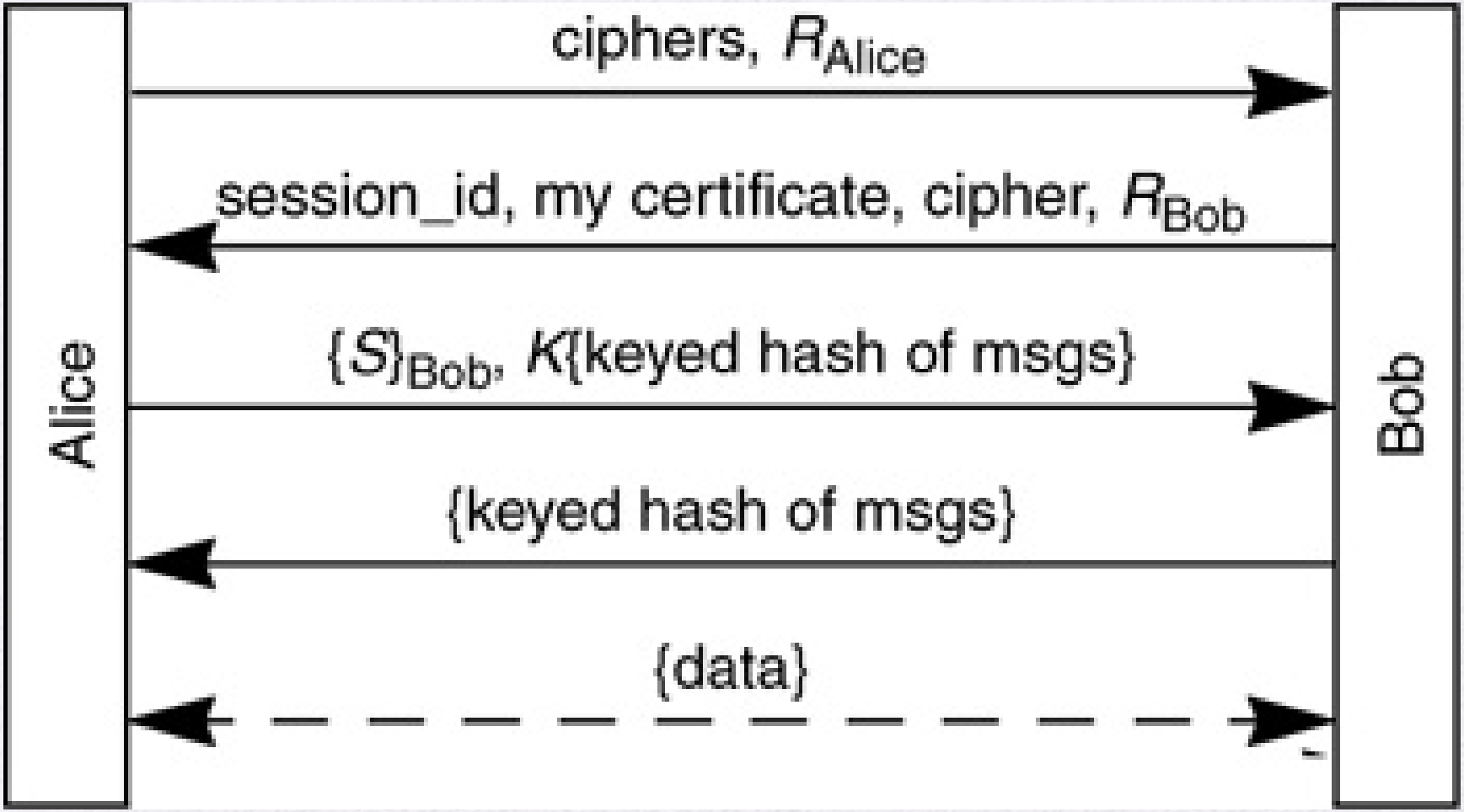
Working

- The client (Alice) initiates contact with the server (Bob).
- Then Bob sends Alice his certificate.
- Alice verifies the certificate, extracts Bob's public key, picks a random number S from which the session keys will be computed,
- and sends S to Bob, encrypted with Bob's public key. Then the remainder of the session is encrypted and integrity-protected with those session keys

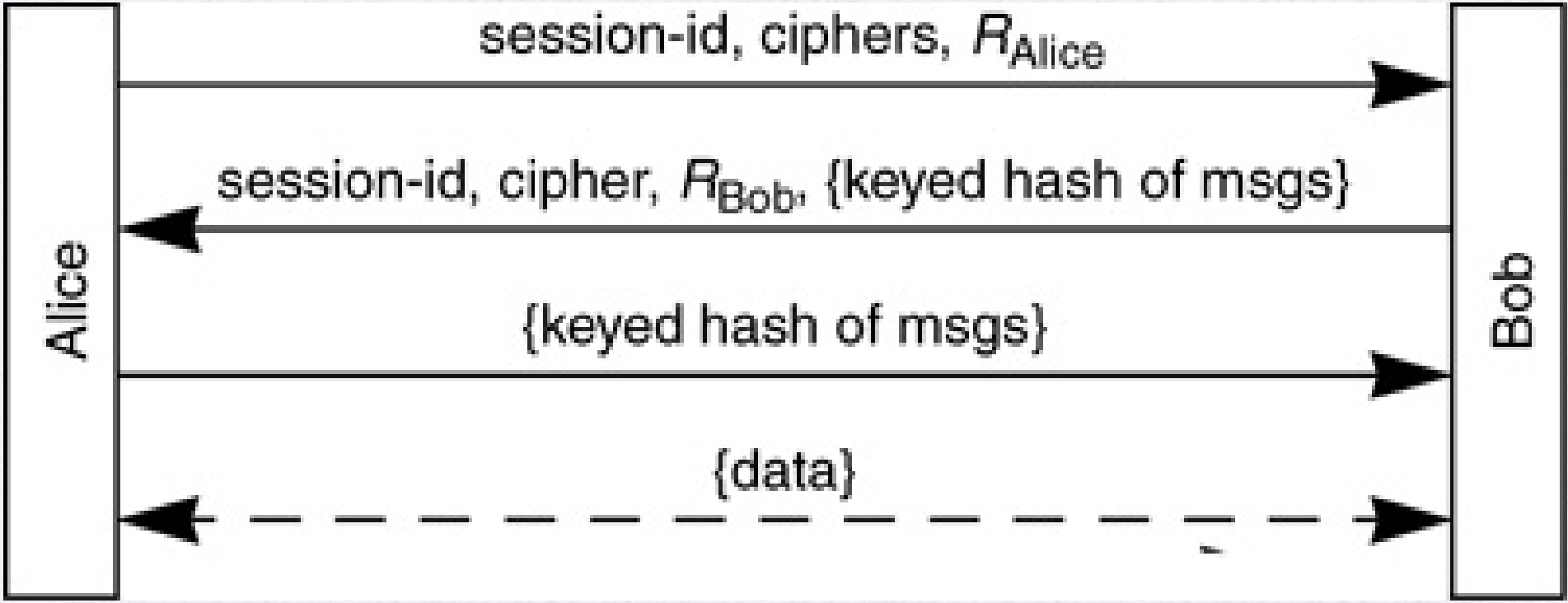
Mutual Authentication

- Alice has authenticated Bob, but Bob has no idea to whom he's talking.
- As deployed today, authentication is seldom mutual – the client authenticates the server but the server does not authenticate the client.
- It is usually done by having the user Alice send her name and password to the server Bob, cryptographically protected with the session keys.

Session initiation if no previous state



Session resumption if both sides remember session-id



Secret Key S

- The secret S sent in the first exchange is the pre-master secret.
- It is shuffled with the two Rs to produce the master secret K. In other words K $=f(S,R_{Alice},R_{Bob})$.
- The Rs are 32 octets long, and it is recommended (but not enforced) that the first 4 octets not be randomly chosen, but instead be the Unix time (seconds since January 1, 1970) when the message was generated

- with Diffie-Hellman using a fixed Diffie-Hellman number as your public key, the same two parties (Alice and Bob) will always compute the same S. If S were kept around in memory, it's possible that malicious software might steal it.

- the master secret is shuffled with the two Rs to produce the six keys used for that connection (for each side: encryption, integrity, and IV).

Keys: derived by hashing K and R_A and R_B

- 3 keys in each direction: encryption, integrity and IV
- Write keys (to send: encrypt, integrity protect)
- Read keys (to receive: decrypt, integrity check)

Client Authentication

- In SSL/TLS the server (Bob) to request that the client (Alice) authenticate herself. This is done by having Bob send a "certificate request" in message 2.
- Alice, upon seeing the request, sends her certificate, and her signature on a hash of the handshake messages, proving she knows the private key associated with the public key in the certificate.

Server Authentication

- The server sends a certificate to the client, and if it's signed by one of the CAs on the client's list, the client will accept the certificate.
- If the server presents a certificate signed by someone not on the list, certificate couldn't be verified because it was signed by an unknown authority,

Version Numbers

- it is possible to distinguish a v2 message from a v3 or TLS message (if they hadn't moved the VERSION NUMBER field it would have been easy). For instance, it happens that the first octet of the v2 client-hello will be greater than 128, and the first octet of the v3 client-hello will be something between 20 and 23

Cipher Suite

- A cipher suite is a complete package (encryption algorithm, key length, integrity checksum algorithm - specify all the crypto SSL/TLS will need).
- Cipher suites are predefined and each is assigned a numeric value. In SSLv2 the value is 3 octets long, but in SSLv3 and TLS, it's 2 octets.
- The number of suites that could potentially be defined in SSLv3/TLS to about 65000.

Choosing Cipher

- Bob returns the subset of Alice's suggested cipher suites that he is willing to support, but lets Alice make the final choice and announce it in message 3.
- Client.

- SSL means SSLv3, (the SSLv2 cipher suites have names starting with SSL2, e.g., SSL2_RC4_128_WITH_MD5),
- RSA means RSA,
- EXPORT means it's exportable (i.e., weak crypto, exportable before the rules were relaxed),
- WITH means the names weren't long enough,
- DES40 means DES with 40 bit keys,
- CBC means CBC-mode encryption, and
- SHA means HMAC-SHA is used for the MAC

- Compression is NULL =0, because of patent issues.