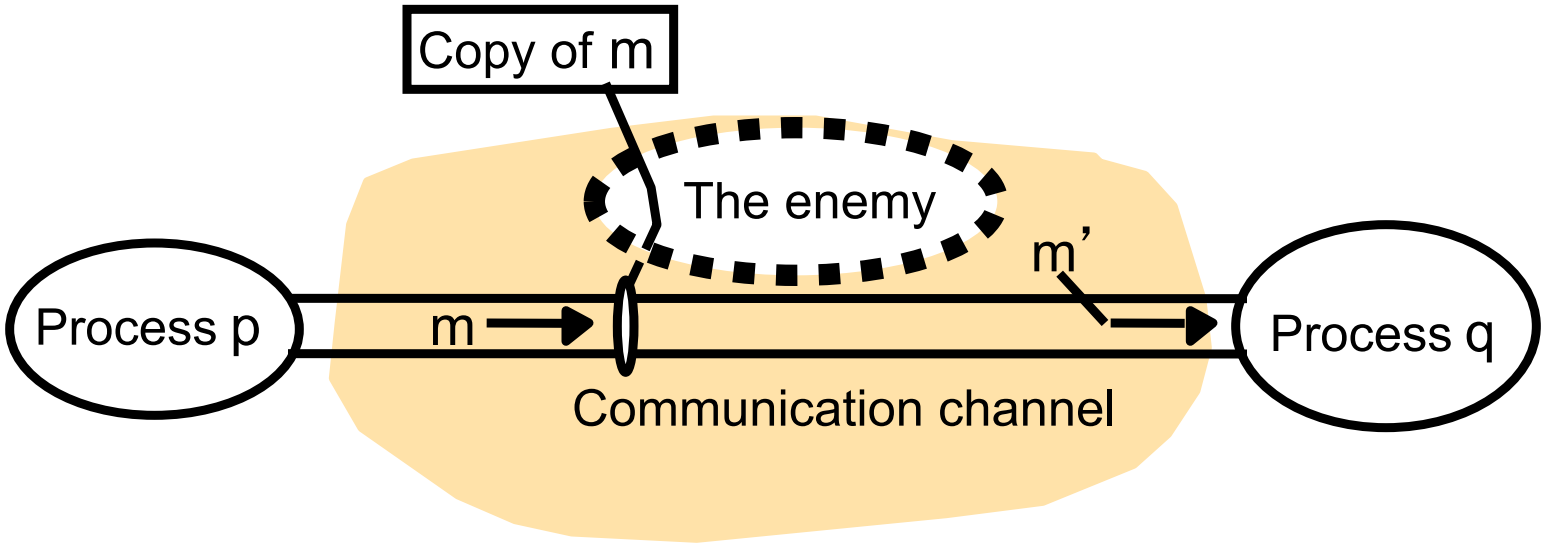
95-702 Distributed Systems

Security: Four Important Cryptographic Protocols

The enemy



Cryptographic Protocols

Definition: A protocol is a series of steps, involving two or more parties, designed to accomplish a task[1].

Examples: Interacting with a waiter at dinner, Voting, HTTP, TCP, UDP.

Definition: A cryptographic protocol is a protocol that uses cryptography[1].

Examples: Bitcoin, SSL, HTTPS, Kerberos, Project 2.

1. From "Applied Crypotography" by Schneier

Some Attacks and an Assumption

- Eavesdropping or listening in.
- Masquerading or pretending to be someone you are not.
- Tampering by a man in the middle.
- Replaying of old messages.
- Denial of service.
- Today's big assumption: "I'm OK, you're OK, the network is the problem!".

Cast of Characters

Alice	First participant
Bob	Second participant
Trent	A trusted third party
Carol	Participant in three-party protocols
Eve	Eavesdropper
Mallory	Malicious attacker
Sara	A trusted server

Cryptography Notation

K_A	Alice's key that she keeps secret.
K_B	Bob's key that he keeps secret.
K_{AB}	Secret key shared between Alice and Bob
K_{Apriv}	Alice's private key (known only to Alice in asymmetric key crypto)
K_{Apub}	Alice's public key (published by Alice for all to read)
$\{M\}_K$	Message M encrypted with key K
$[M]_K$	Message M signed with key K

Two Major Categories of Encryption Algorithms

Symmetric key encryption. Also called secret key crypto.

Alice sends $\{M\}K_{ab}$ and Bob can read it.
Bob knows K_{ab}.

Asymmetric key encryption. Also called public key crypto.

Alice sends $\{M\}K_{Bpub}$ and Bob can read it.
Bob knows K_{Bpriv}.

Public key encryption is typically 100 to 1000 times slower than secret key encryption.

Quiz

If public key crypto is so slow, why is it used?

Answer: To encrypt a symmetric key.
And to provide long term signatures.

Scenario 1 (Like WWII & TEA)

Communication with a shared secret key.

Alice and Bob share K_{AB}.
Alice computes $E(K_{AB},M_i)$ for each message i.
She sends these to Bob.
Bob uses $D(K_{AB}, \{M_i\} K_{AB})$ and reads each M_i.

How do Bob and Alice communicate the key K_{AB}?
How does Bob know that $\{M_i\} K_{AB}$ isn't a replay of an old message?

Scenario 2 (Like Kerberos)

Alice wishes to access a service provided by Bob.

Alice asks Sarah for a ticket to talk to Bob.
Sarah knows Alice's password so she can compute K_A.
Sarah send to Alice $\{\{Ticket\}K_B,K_{AB}\}K_A$. A challenge!
Alice knows her password and is able to compute K_A.
Note that the password is never placed on the network.
Alice is able to compute $\{Ticket\}K_B$ and K_{AB}. How?
Alice sends a read request to Bob. She sends $\{Ticket\}K_B$,Alice,Read. Another challenge!
Bob uses K_B to read the content of the Ticket.
The Ticket is K_{AB},Alice. Bob and Alice then use this session key to communicate.
Old tickets may be replayed by Mallory. Suppose Mallory has captured an old session keyK_{AB} .
Does not scale well : Sarah must know K_A, K_B ….
Sarah is a single point of failure.

Scenario 3 (Authentication)

Alice wishes to convince Bob that she sent the message M.

She computes a digest of M, Digest(M).
If the Digest method is a good one, it is very difficult to find another message M' so that Digest(M) == Digest(M').
Alice makes the following available to the intended users:
M,{Digest(M)}K_{Apriv}.
Bob obtains the signed document, extracts M and computes Digest(M).
Bob decrypts {Digest(M)}K_{Apriv} using K_{Apub} and compares the result with his calculated Digest(M). If they match, the signature is valid.

Can Alice claim that she did not sign the message? What if she claims she released her K_{Apriv} ? Still useful if Bob and Alice trust each other.

Scenario 4 (Like SSL)

Bob and Alice wish to establish a shared secret K_{AB}.

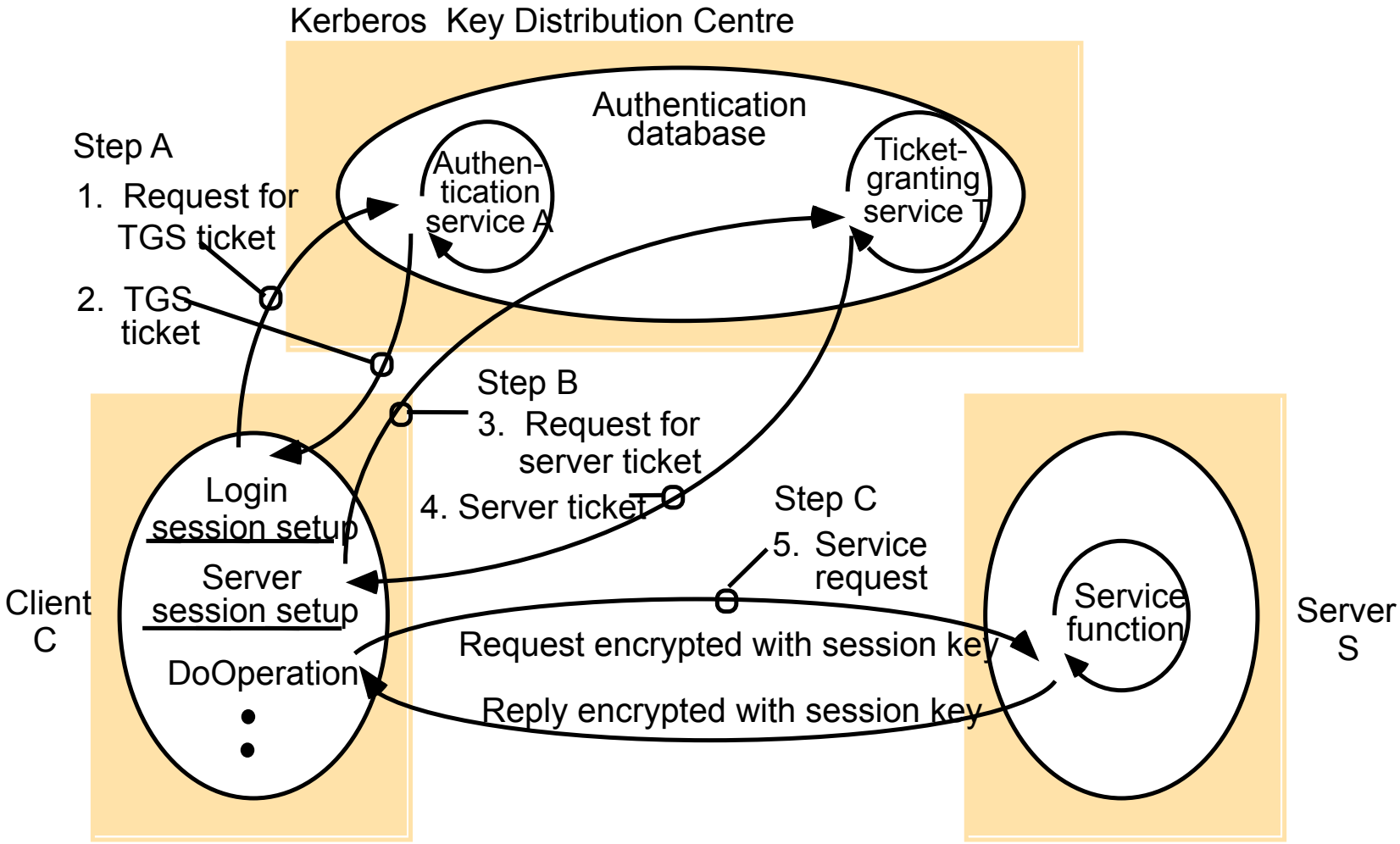
Alice retrieves Bob's public key.
This key comes in a certificate. So, Bob's public key has been signed by a trusted third party, Trent.
Alice verifies that Trent signed the public key K_{Bpub}.
Alice generates K_{AB} and encrypts it with K_{Bpub}.
Bob has many public keys and so Alice sends a key name along as well.
Alice sends key name, $\{K_{AB}\}K_{Bpub}$.
Bob uses the key name to select the correct private key and computes $\{\{K_{AB}\}K_{Bpub}\}\ K_{Bpriv} == K_{AB}$.

The man in the middle attack may be used when Alice first retrieves what she thinks is Bob's public key. Mallory may supply his own public key (also signed by Trent).

The Needham–Schroeder Secret-Key Authentication Protocol (Scenario 2 with more detail)

Header	*Message*	*Notes*
1. A->S:	A, B, N_A	A requests S to supply a key for communication with B.
2. S->A:	$\{N_A, B, K_{AB}, \{K_{AB}, A\}_{K_B}\}_{K_A}$	S returns a message encrypted in A's secret key, containing a newly generated key K_{AB} and a 'ticket' encrypted in B's secret key. The nonce N_A demonstrates that the message was sent in response to the preceding one. A believes that S sent the message because only S knows A's secret key.
3. A->B:	$\{K_{AB}, A\}_{K_B}$	A sends the 'ticket' to B.
4. B->A:	$\{N_B\}_{K_{AB}}$	B decrypts the ticket and uses the new key K_{AB} to encrypt another nonce N_B.
5. A->B:	$\{N_B - 1\}_{K_{AB}}$	A demonstrates to B that it was the sender of the previous message by returning an agreed transformation of N_B.

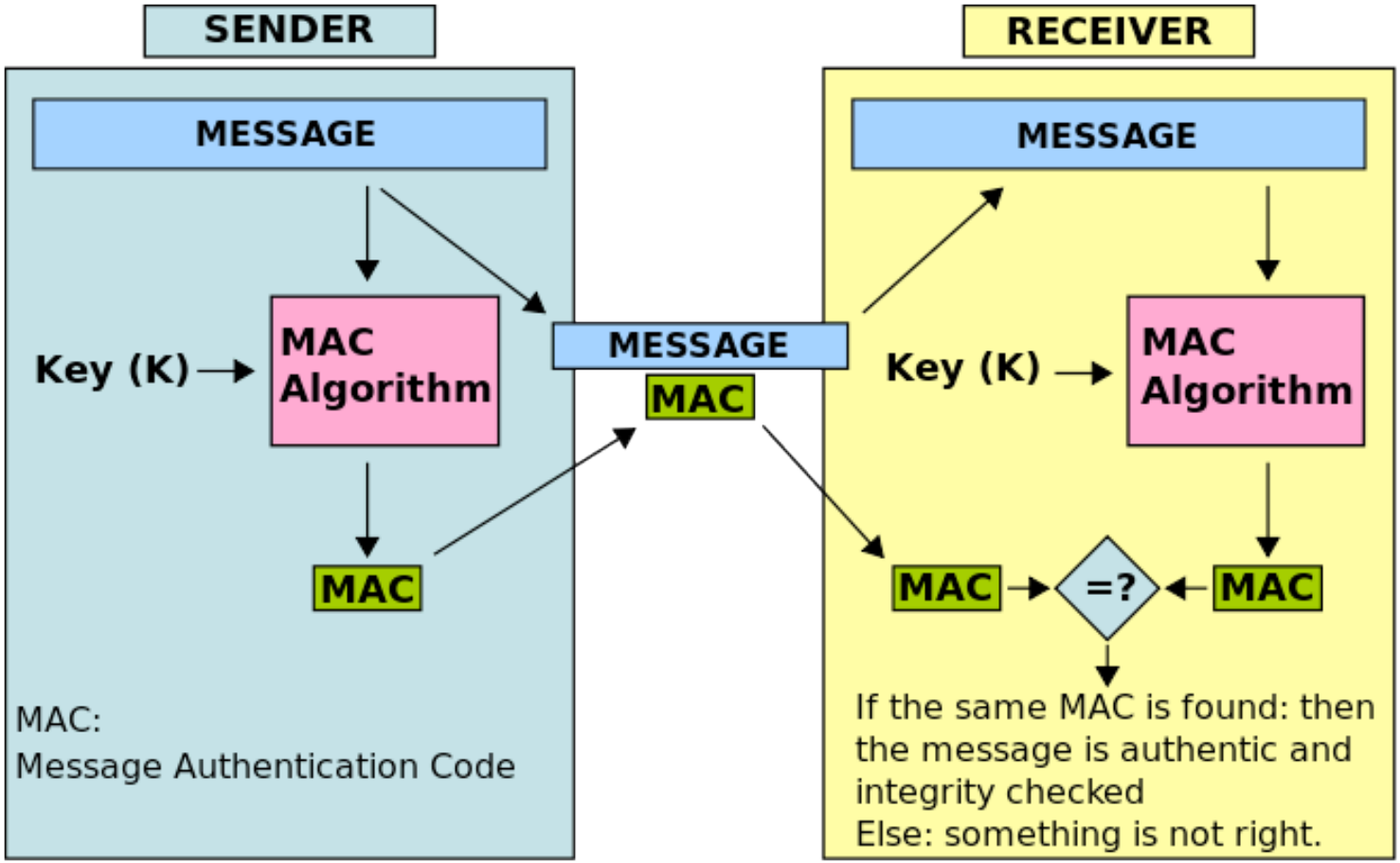
System Architecture of Kerberos



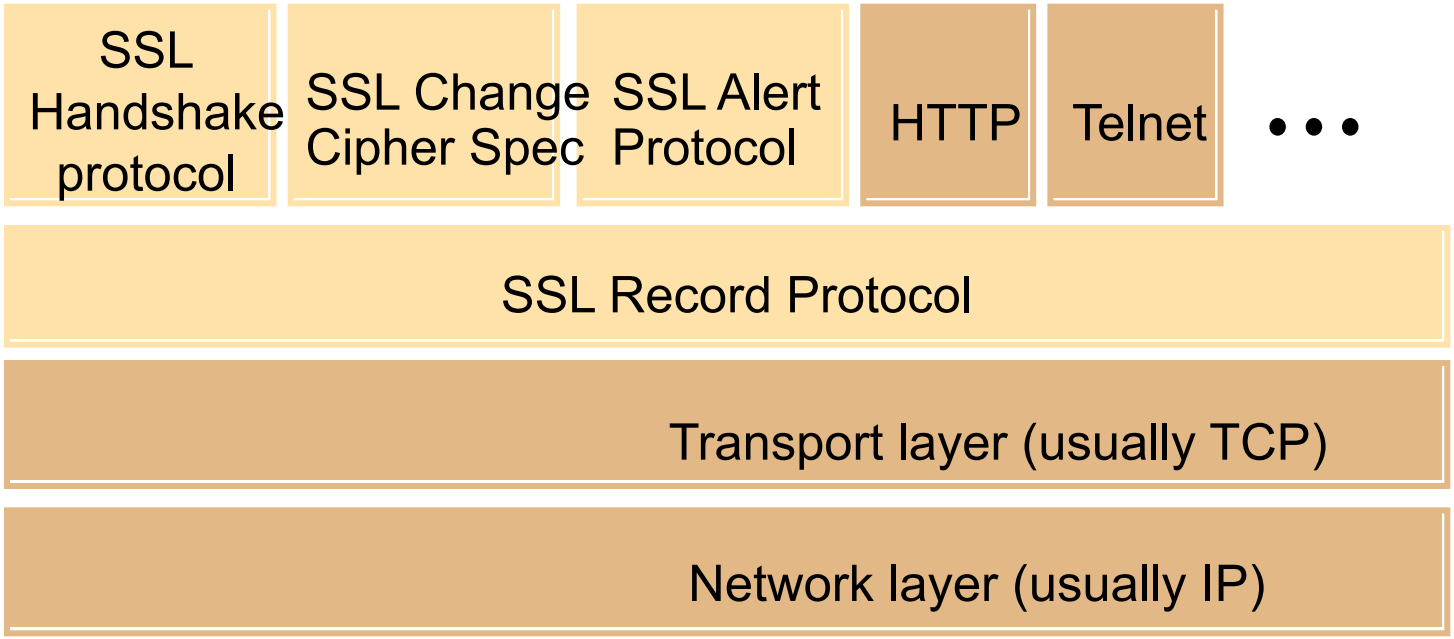
SSL Overview (Scenario 4 with more detail)

- Developed by Netscape Communications
- *Authenticates* servers (and optionally clients)
- Establishes a secret key (Diffie-Hellman or RSA)
- Data is *encrypted* with the exchanged key
- Clients do not need to provide a certificate but may be required to by the server
- Client authentication is typically done in the application layer
- Servers must provide a certificate
- Normally uses RSA
- *Data integrity* provided by Message Authentication Codes

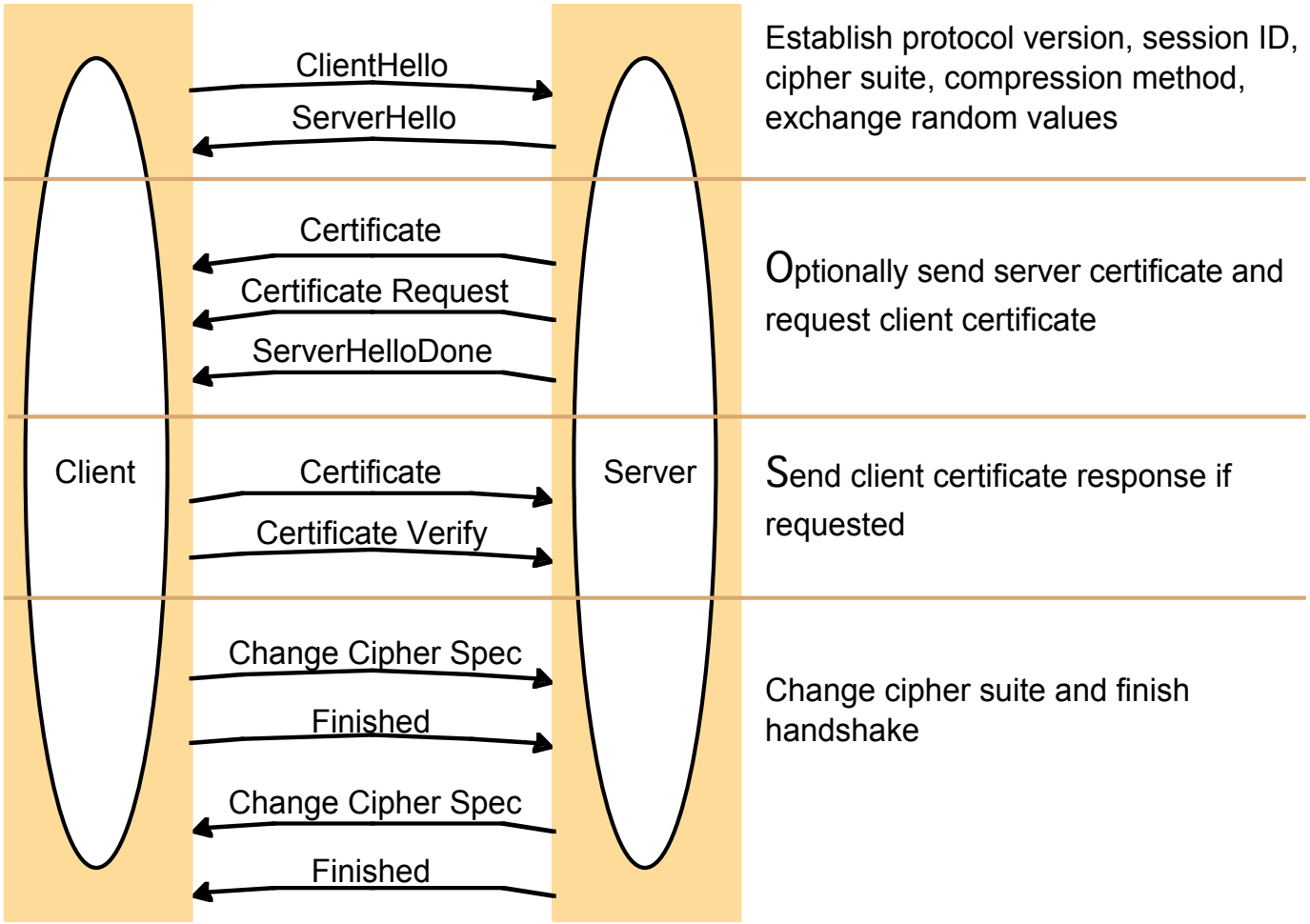
Message Authentication Codes (MACs)



SSL Protocol Stack



TLS Handshake Protocol



TLS Handshake Configuration Options

Component	*Description*	*Example*
Key exchange method	the method to be used for exchange of a session key	RSA with public-key certificates
Cipher for data transfer	the block or stream cipher to be used for data	IDEA
Message digest function	for creating message authentication codes (MACs)	SHA

Combining Protocols

Quiz:

Can we combine SSL and Kerberos?

Log on to Blackboard.

Writing a simple SSL Client

- All SSL clients must have a truststore
- If a client is to be verified by the server then the client needs a keystore as well as a truststore
- The truststore

  - holds trusted certificates (signed public keys of CA's)
  - is in the same format as a keystore
  - is an instance of Java's KeyStore class
  - is used by the client to verify the certificate sent by the server
  - may be shared with others

Creating a Truststore

(1) Use keytool –genkey to create an RSA key pair

(2) Use keytool –export to generate a self-signed RSA certificate (holding no private key)

(3) Use keytool –import to place the certificate into a truststore

(1) Use keytool - genkey to create an RSA key pair

D:\McCarthy\www\95-804\examples\keystoreexamples>
keytool -genkey -alias mjm -keyalg RSA -keystore mjmkeystore

Enter keystore password: sesame

What is your first and last name?
  [Unknown]: Michael McCarthy

What is the name of your organizational unit?
  [Unknown]: Heinz School

What is the name of your organization?
  [Unknown]: CMU

What is the name of your City or Locality?
[Unknown]: Pittsburgh

What is the name of your State or Province?
[Unknown]: PA

What is the two-letter country code for this unit?
[Unknown]: US

Is CN=Michael McCarthy, OU=Heinz School, O=CMU, L=Pittsburgh, ST=PA, C=US correct?
[no]: yes

Enter key password for <mjm>
(RETURN if same as keystore password): <RT>

```
D:\McCarthy\www\95-804\examples\keystoreexamples>dir /w
 Volume in drive D has no label.
 Volume Serial Number is 486D-D392

 Directory of D:\McCarthy\www\95-804\examples\keystoreexamples

[.]          [..]         mjmkeystore
```

(2) Use keytool –export to generate a self-signed RSA certificate (holding no private key)

```
D:\McCarthy\www\95-804\examples\keystoreexamples>
keytool -export -alias mjm -keystore mjmkeystore -file mjm.cer
Enter keystore password:  sesame
Certificate stored in file <mjm.cer>

D:\McCarthy\www\95-804\examples\keystoreexamples>dir /w
 Volume in drive D has no label.
 Volume Serial Number is 486D-D392

 Directory of D:\McCarthy\www\95-804\examples\keystoreexamples

[.]          [..]         mjm.cer      mjmkeystore
```

(3) Use keytool –import to place the certificate into a truststore

D:\McCarthy\www\95-804\examples\keystoreexamples>
keytool -import -alias mjm -keystore mjm.truststore -file mjm.cer

Enter keystore password: sesame
Owner:
CN=Michael McCarthy, OU=Heinz School, O=CMU, L=Pittsburgh, ST=PA, C=US

Issuer:
CN=Michael McCarthy, OU=Heinz School, O=CMU, L=Pittsburgh, ST=PA, C=US

Serial number: 3e60f3ce
Valid from:
Sat Mar 01 12:54:22 EST 2003 until: Fri May 30 13:54:22 EDT 2003
Certificate fingerprints:

MD5:
80:F4:73:23:4C:B4:32:4C:5F:E0:8A:B1:4D:1E:A3:0D

SHA1:
19:06:31:54:72:ED:B8:D5:B3:CF:38:07:66:B5:78:1A:34:16:56:07
Trust this certificate? [no]: yes
Certificate was added to keystore

```
D:\McCarthy\www\95-804\examples\keystoreexamples>dir /w
 Volume in drive D has no label.
 Volume Serial Number is 486D-D392

 Directory of D:\McCarthy\www\95-804\examples\keystoreexamples

[.]          [..]          mjm.cer          mjm.truststore   mjmkeystore
               5 File(s)        2,615 bytes
```

mjmkeystore will be placed in the server's directory
SSL will send the associated certificate to the client

mjm.truststore will be placed in the client's directory

File Organization

```
D:\McCarthy\www\95-804\examples\keystoreexamples>tree /f
Directory PATH listing
Volume serial number is 0012FC94 486D:D392
D:.
├───clientcode
│       mjm.truststore
│       Client.java
│
└───servercode
        mjmkeystore
        Server.java
```

Client.java

```
import java.io.*;
import javax.net.ssl.*;
import java.net.*;
import javax.net.*;

public class Client {

    public static void main(String args[]) {

        int port = 6502;
        try {
            // tell the system who we trust
            System.setProperty("javax.net.ssl.trustStore","mjm.truststore")
```

```
// get an SSLSocketFactory
SocketFactory sf = SSLSocketFactory.getDefault();

// an SSLSocket "is a" Socket
Socket s = sf.createSocket("localhost",6502);

PrintWriter out = new PrintWriter(s.getOutputStream());
BufferedReader in = new
                        BufferedReader(
                            new InputStreamReader(
                                s.getInputStream()));
out.write("Hello server\n");
out.flush();
String answer = in.readLine();
System.out.println(answer);
```

```
            out.close();
            in.close();
        }
        catch(Exception e) {
            System.out.println("Exception thrown " + e);
        }
    }
}
```

Server.java

```
// Server side SSL
import java.io.*;
import java.net.*;
import javax.net.*;
import javax.net.ssl.*;
import java.security.*;

public class Server {

    // hold the name of the keystore containing public and private keys
    static String keyStore = "mjmkeystore";

    // password of the keystore (same as the alias)
    static char keyStorePass[] = "sesame".toCharArray();
```

```
public static void main(String args[]) {

    int port = 6502;
    SSLServerSocket server;

    try {
        // get the keystore into memory
        KeyStore ks = KeyStore.getInstance("JKS");
        ks.load(new FileInputStream(keyStore), keyStorePass);

        // initialize the key manager factory with the keystore data
        KeyManagerFactory kmf =
                            KeyManagerFactory.getInstance("SunX509");
        kmf.init(ks,keyStorePass);
```

```
// initialize the SSLContext engine
// may throw NoSuchProvider or NoSuchAlgorithm exception
// TLS - Transport Layer Security most generic

SSLContext sslContext = SSLContext.getInstance("TLS");

// Inititialize context with given KeyManagers, TrustManagers,
// SecureRandom defaults taken if null

sslContext.init(kmf.getKeyManagers(), null, null);

// Get ServerSocketFactory from the context object
ServerSocketFactory ssf = sslContext.getServerSocketFactory();
```

```
//  Now like programming with normal server sockets
ServerSocket serverSocket = ssf.createServerSocket(port);

System.out.println("Accepting secure connections");

Socket client = serverSocket.accept();
System.out.println("Got connection");

BufferedWriter out = new BufferedWriter(
                                 new OutputStreamWriter(
                                        client.getOutputStream()));
BufferedReader in = new BufferedReader(
                                new InputStreamReader(
                                       client.getInputStream()));
```

```
            String msg = in.readLine();
            System.out.println("Got message " + msg);
            out.write("Hello client\n");
            out.flush();
            in.close();
            out.close();


        }
        catch(Exception e) {
            System.out.println("Exception thrown " + e);
        }
    }
}
```

On the server

```
D:\McCarthy\www\95-804\examples\keystoreexamples\servercode>
java Server
Accepting secure connections
Got connection
Got message Hello server
```

On the client

```
D:\McCarthy\www\95-804\examples\keystoreexamples\clientcode>
java Client
Hello client
```

Configuring a Web Application to Use SSL

The web server needs a certificate so that the client can identify the server.

The certificate may be signed by a Certificate Authority or it may be self-signed.

The web server needs a private key as well.

```
D:\McCarthy\www\95-804\examples\SSLAndTomcat>
keytool -genkey -keyalg RSA -alias tomcat -keystore .keystore

Enter keystore password:  sesame

What is your first and last name?
  [Unknown]:  localhost
What is the name of your organizational unit?
  [Unknown]:  Heinz School
What is the name of your organization?
  [Unknown]:  CMU
What is the name of your City or Locality?
  [Unknown]:  Pgh.
What is the name of your State or Province?
  [Unknown]:  PA
```

Generate public and private keys for Tomcat

The keystore file is called .keystore

What is the two-letter country code for this unit?
  [Unknown]: US
Is CN=localhost, OU=Heinz School, O=CMU, L=Pgh.,
ST=PA, C=US correct?
  [no]: yes

Enter key password for <tomcat>
        (RETURN if same as keystore password):<RT>

D:\McCarthy\www\95-804\examples\SSLAndTomcat>

Use admin tool to tell Tomcat about SSL

(1) Startup Tomcat
(2) Run the admin server with http://localhost:8080/admin
(3) Log in with your user name and password
(4) Select Service (Java Web Service Developer Pack)
(5) Select Create New Connector from the drop down list in the right pane
(6) In the type field enter HTTPS
(7) In the port field enter 8443
(8) Enter complete path to your .keystore file
(9) Enter keystore password
(10) Select SAVE and then Commit Changes

Tell Tomcat about .keystore

Testing

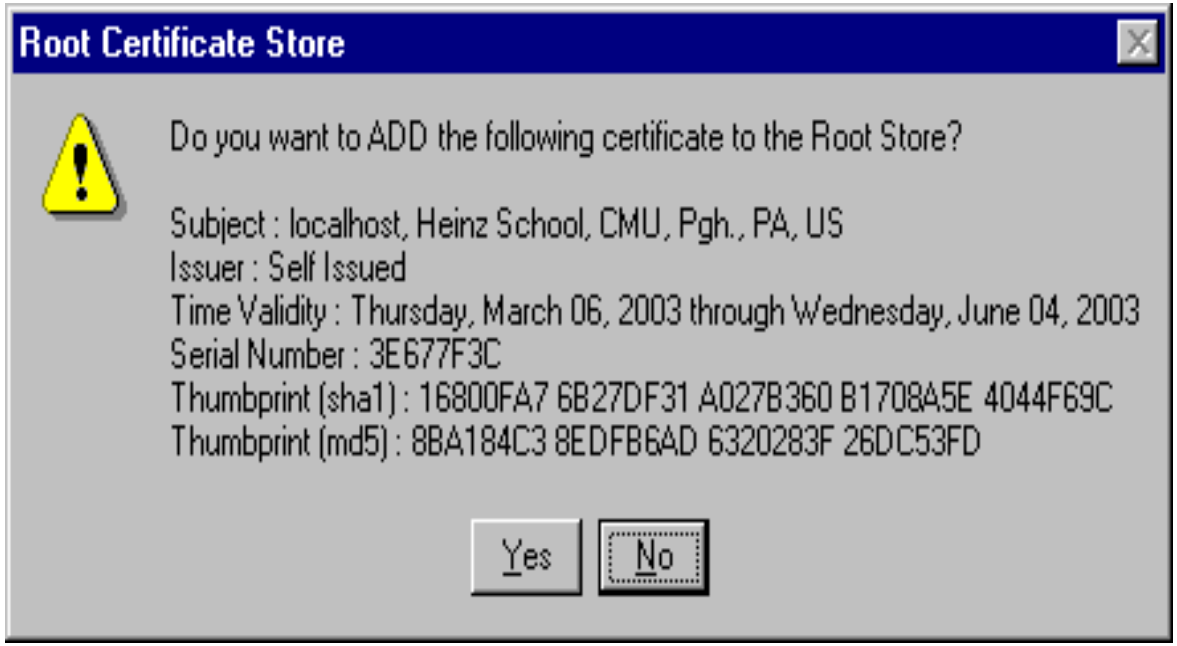
Shutdown Tomcat.

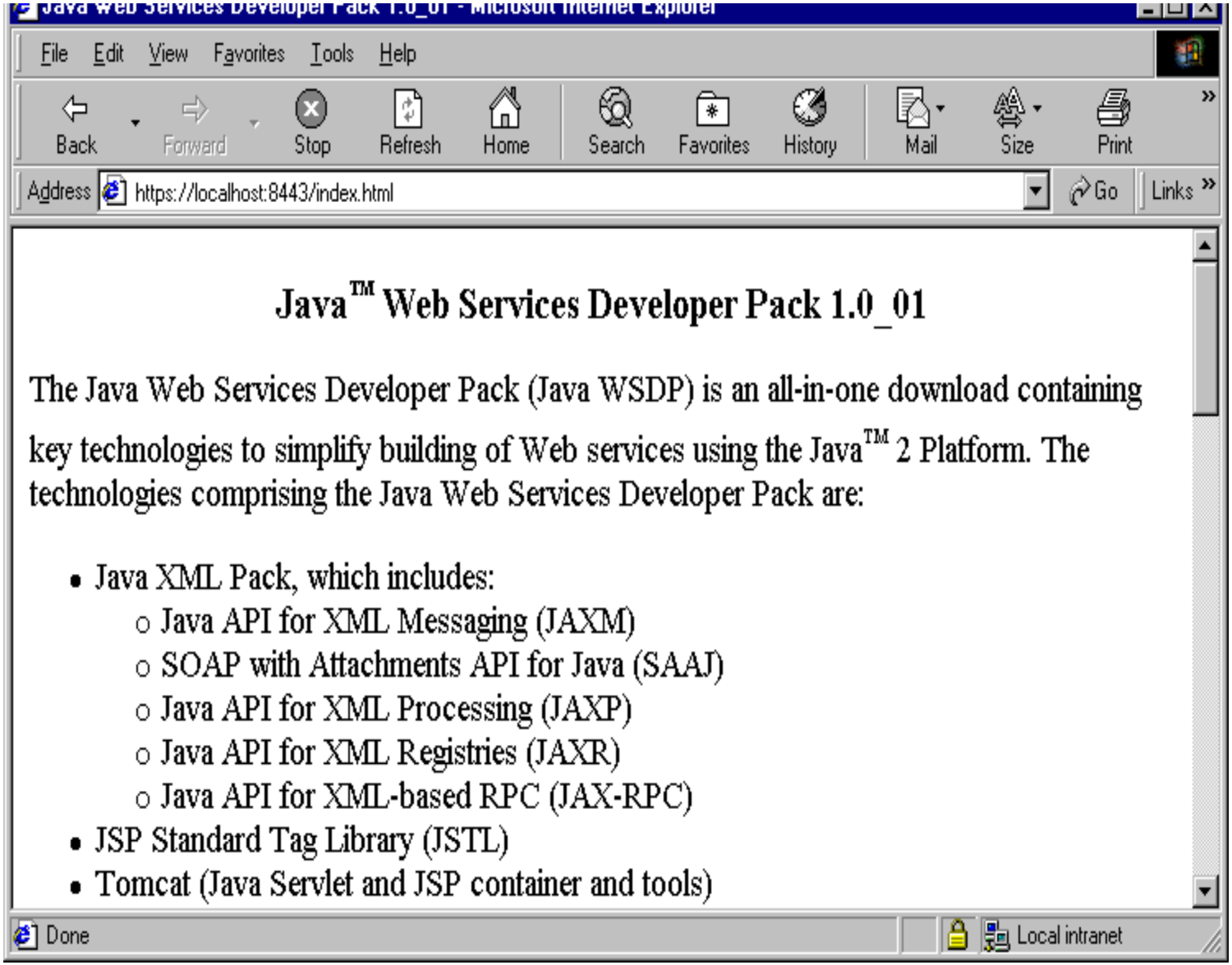
Visit Tomcat from a browser.

Use https://localhost:8443/

You can also visit your other installed web apps through https.


Security Alert
Information you exchange with this site cannot be viewed or changed by others. However, there is a problem with the site's security certificate.
The security certificate was issued by a company you have not chosen to trust. View the certificate to determine whether you want to trust the certifying authority.
The security certificate date is valid.
The security certificate matches the name of the page you are trying to view.
Do you want to proceed?
Yes
No
View Certificate


Root Certificate Store
Do you want to ADD the following certificate to the Root Store?
Subject : localhost, Heinz School, CMU, Pgh., PA, US
Issuer : Self Issued
Time Validity : Thursday, March 06, 2003 through Wednesday, June 04, 2003
Serial Number : 3E677F3C
Thumbprint (sha1) : 16800FA7 6B27DF31 A027B360 B1708A5E 4044F69C
Thumbprint (md5) : 8BA184C3 8EDFB6AD 6320283F 26DC53FD
Yes
No


File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Mail Size Print
Address https://localhost:8443/index.html
Go Links
Java™ Web Services Developer Pack 1.0_01
The Java Web Services Developer Pack (Java WSDP) is an all-in-one download containing key technologies to simplify building of Web services using the Java™ 2 Platform. The technologies comprising the Java Web Services Developer Pack are:
Java XML Pack, which includes:
Java API for XML Messaging (JAXM)
SOAP with Attachments API for Java (SAAJ)
Java API for XML Processing (JAXP)
Java API for XML Registries (JAXR)
Java API for XML-based RPC (JAX-RPC)
JSP Standard Tag Library (JSTL)
Tomcat (Java Servlet and JSP container and tools)
Done
Local intranet