



95-702 Distributed Systems

An Introduction To Cryptographic Protocols



Computer Security

- Needed because of the desire to share resources.
- Security policies are enforced by security mechanisms.
- Cryptography provides the basis for most security mechanisms but is a distinct subject.
- Two great books are Schneier's "Applied Cryptography" and "The Code Book" by Singh.



Threat Categories

- Leakage is any unauthorized acquisition of information
- Tampering is unauthorized alteration of information
- Vandalism is interference with proper operation with no gain to the perpetrator



Some Attacks

- Eavesdropping
- Masquerading
- Tampering, e.g., “the man in the middle attack”
- Replaying
- Denial of service
- Today’s big assumption: “I’m OK, you’re OK, the network is the problem!”



Assumptions & Guidelines

- Interfaces are exposed.
- Networks are insecure.
- Algorithms are available to attackers.
We assume they understand RSA, DES, etc.
- Attackers may have have large resources.
- Limit the lifetime and scope of secrets.
- Minimize the trusted base.



An Interesting Example: E-Voting

Two types of E-Voting (From Wikipedia):

- (1) e-voting which is physically supervised by electoral authorities
 - (2) remote e-Voting where voting is performed within the voter's sole influence, and is not physically supervised by authorities (e.g. voting from one's personal computer, mobile phone, television via the internet (i-voting)).
- (1) is controversial due to the problem of trusting software.
(2) would be wonderful if we could do it securely.



Goals Of Secure Voting

- Only Authorized Voters Can Vote
- No one can vote more than once
- No one can determine for whom anyone else voted
- No one can duplicate anyone else's vote
- No one can change anyone else's vote without being discovered
- Every voter can make sure that his vote has been taken into account in the final tabulation.



First Attempt

- Each voter encrypts his vote with the public key of a Central Tabulating Facility (CTF)
- Each voter send his vote in to the CTF
- The CTF decrypts the votes, tabulates them, and makes the results public
- What are some problems with this protocol?



Second Attempt

- Each voter signs his vote with his private key
- Each voter encrypts his signed vote with the CTF's public key
- Each voter send his vote to the CTF
- The CTF decrypts the votes, checks the signature, tabulates the votes and makes the results public
- What are some problems with this protocol?



Cast of Characters

Alice	First participant
Bob	Second participant
Carol	Participant in three- and four-party protocols
Dave	Participant in four-party protocols
Eve	Eavesdropper
Mallory	Malicious attacker
Sara	A 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



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.



Scenario 1 (WWII)

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 .

Problems?

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 (Baby Kerberos)

Alice wishes to access files held by Bob.

Alice asks Sarah for a ticket to talk to Bob.

Sarah knows Alice's password so she can compute K_A .

Sarah sends to Alice $\{\text{Ticket}\}_{K_B}, K_{AB}\}_{K_A}$. **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 $\{\text{Ticket}\}_{K_B}$ and K_{AB} . How?

Alice sends a read request to Bob. She sends

$\{\text{Ticket}\}_{K_B}, \text{Alice}, \text{Read}$. 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.

Problems?

Old tickets may be replayed by Mallory. Suppose she has an old session key.

Does not scale well. Sarah must know $K_A, K_B \dots$



Scenario 3 (Non-repudiation)

Alice wishes to sign a digital message M .

She computes a digest of M , $\text{Digest}(M)$.

If the Digest method is a good one, it is very difficult to find another message M' so that $\text{Digest}(M) == \text{Digest}(M')$.

Alice makes the following available to the intended users:

$M, \{\text{Digest}(M)\}_{K_{\text{Apriv}}}$

Bob obtains the signed document, extracts M and computes $\text{Digest}(M)$.

Bob decrypts $\{\text{Digest}(M)\}_{K_{\text{Apriv}}}$ using K_{Apub} and compares the result with his calculated $\text{Digest}(M)$. If they match, the signature is valid.

Problem: 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 (Baby SSL)

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

Alice uses a key distribution service to get 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 $\{\{K_{AB}\}K_{Bpub}\} K_{Bpriv} == K_{AB}$.

Problem:

The man in the middle attack may be used when Alice first contacts the key distribution service. Mallory may return his own public key (also signed by Trent).



Alice's Bank Account Certificate

1. <i>Certificate type</i>	Account number
2. <i>Name</i>	Alice
3. <i>Account</i>	6262626
4. <i>Certifying authority</i>	Bob's Bank
5. <i>Signature</i>	$\{Digest(field\ 2 + field\ 3)\}_{K_{Bpriv}}$

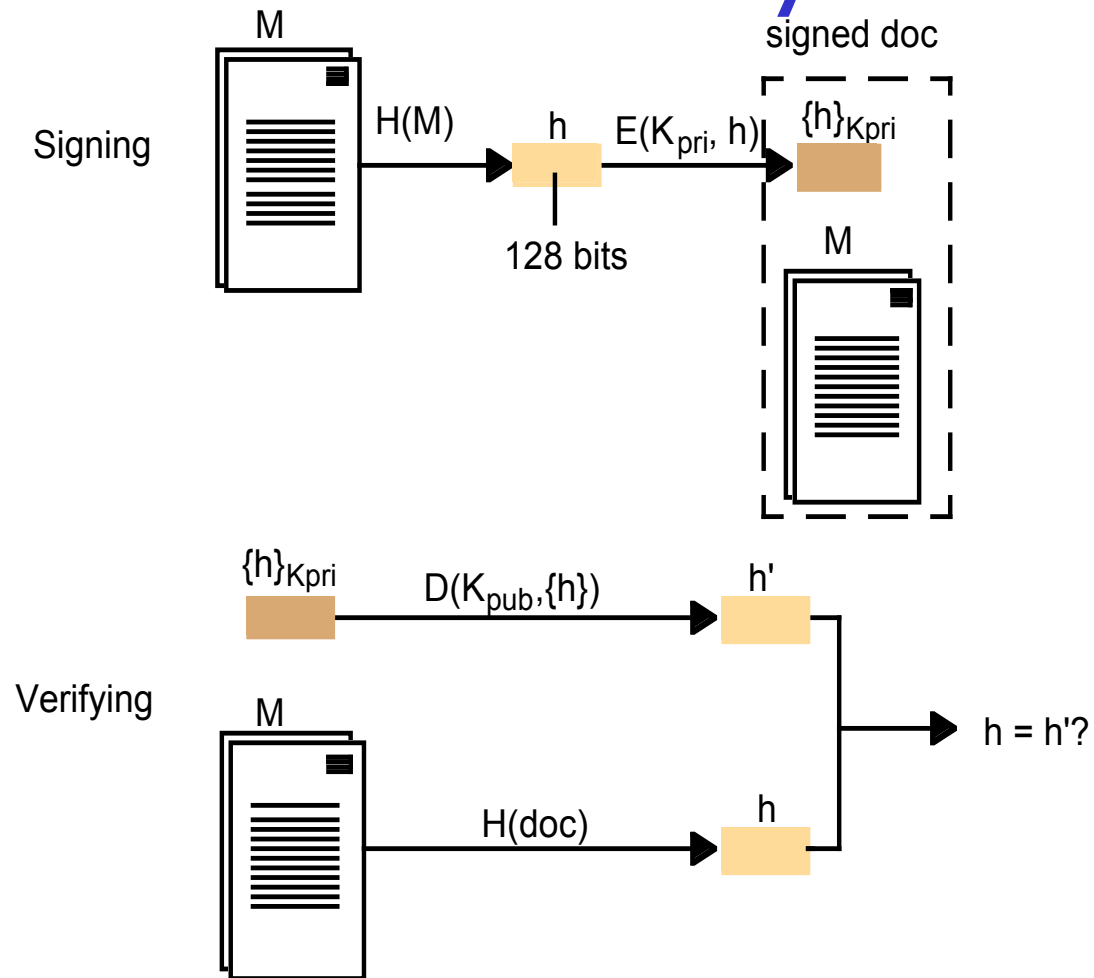


Public-Key Certificate for Bob's Bank

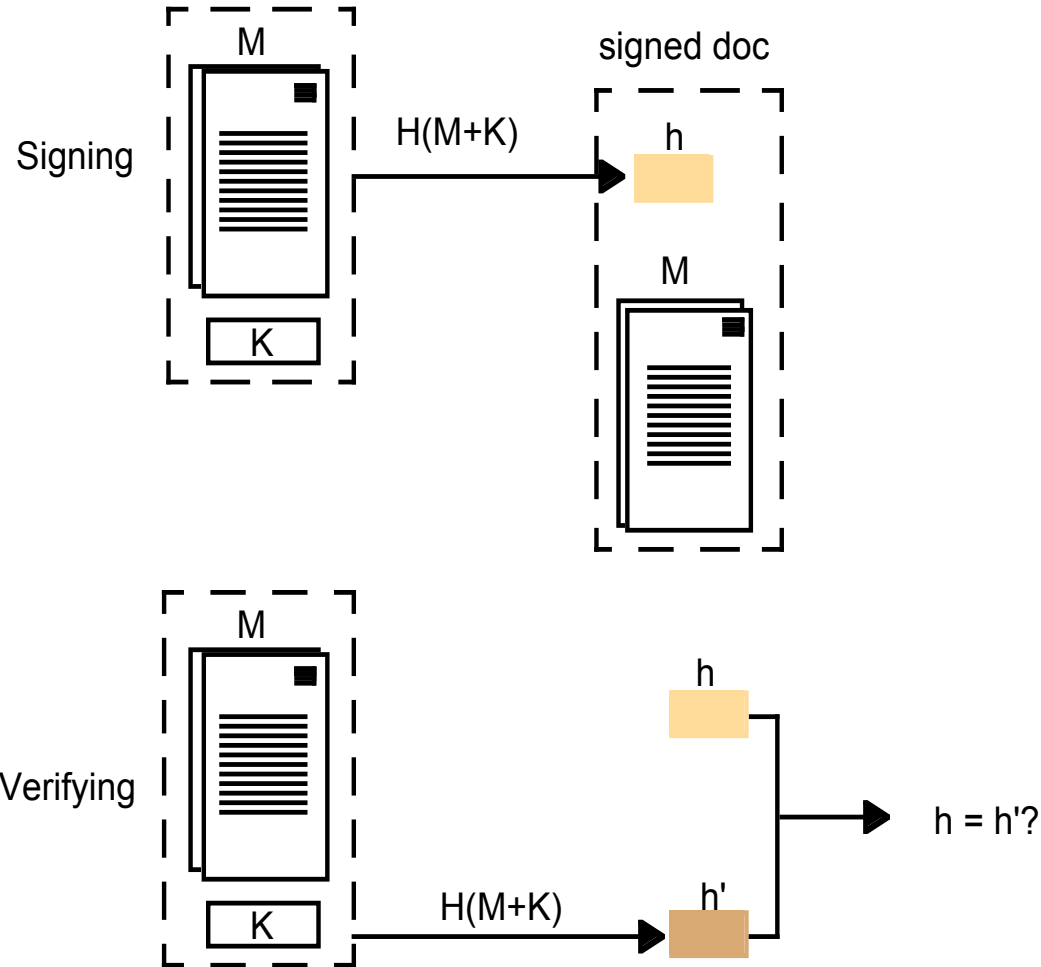
1. <i>Certificate type</i>	Public key
2. <i>Name</i>	Bob's Bank
3. <i>Public key</i>	K_{Bpub}
4. <i>Certifying authority</i>	Fred – The Bankers Federation
5. <i>Signature</i>	$\{Digest(field\ 2 + field\ 3)\} K_{Fpriv}$



Digital Signatures With Public Keys



Low-Cost Signatures with a Shared Secret Key



X509 Certificate Format

<i>Subject</i>	Distinguished Name, Public Key
<i>Issuer</i>	Distinguished Name, Signature
<i>Period of validity</i>	Not Before Date, Not After Date
<i>Administrative information</i>	Version, Serial Number
<i>Extended Information</i>	

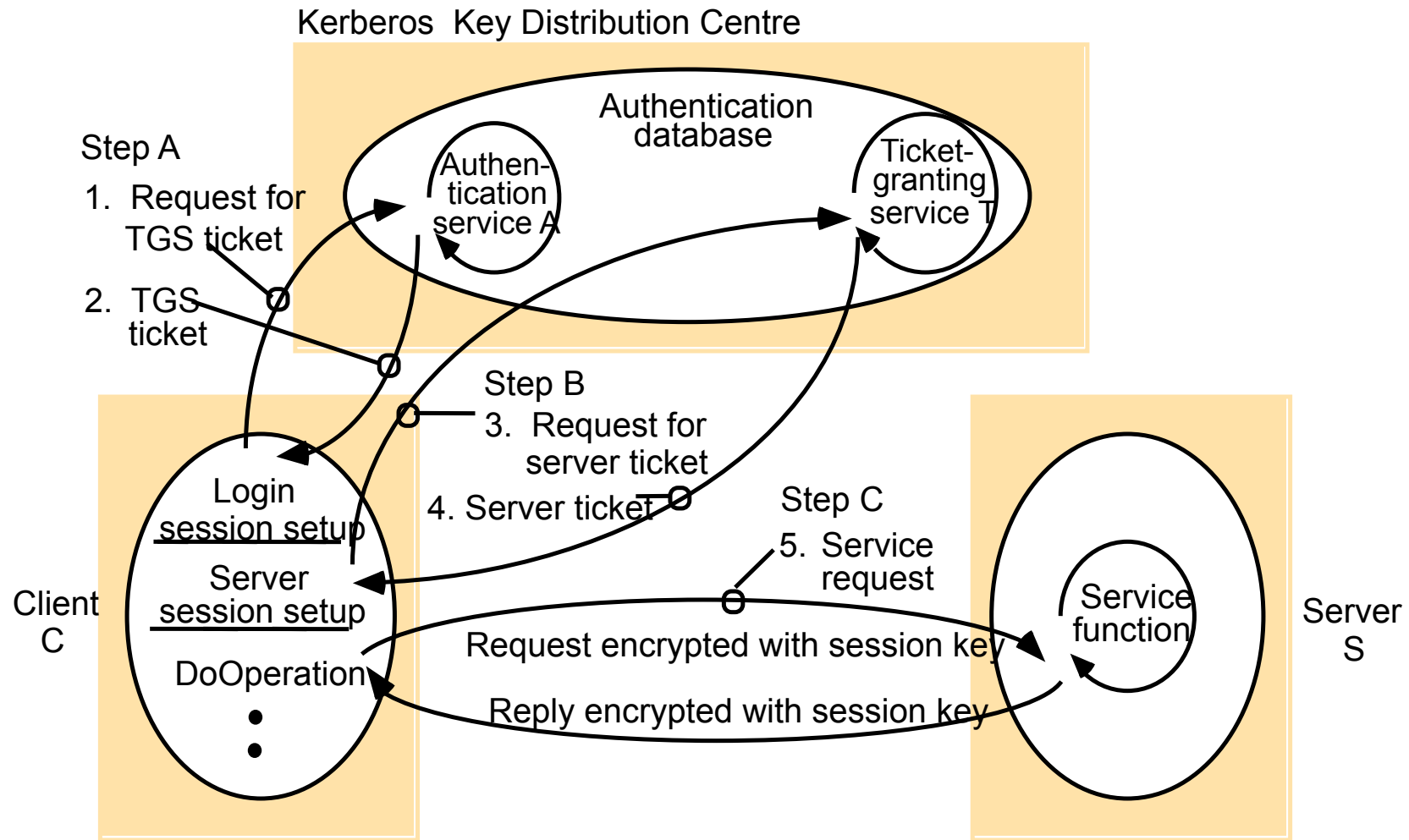


The Needham–Schroeder Secret-Key Authentication Protocol

<i>Header</i>	<i>Message</i>	<i>Notes</i>
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

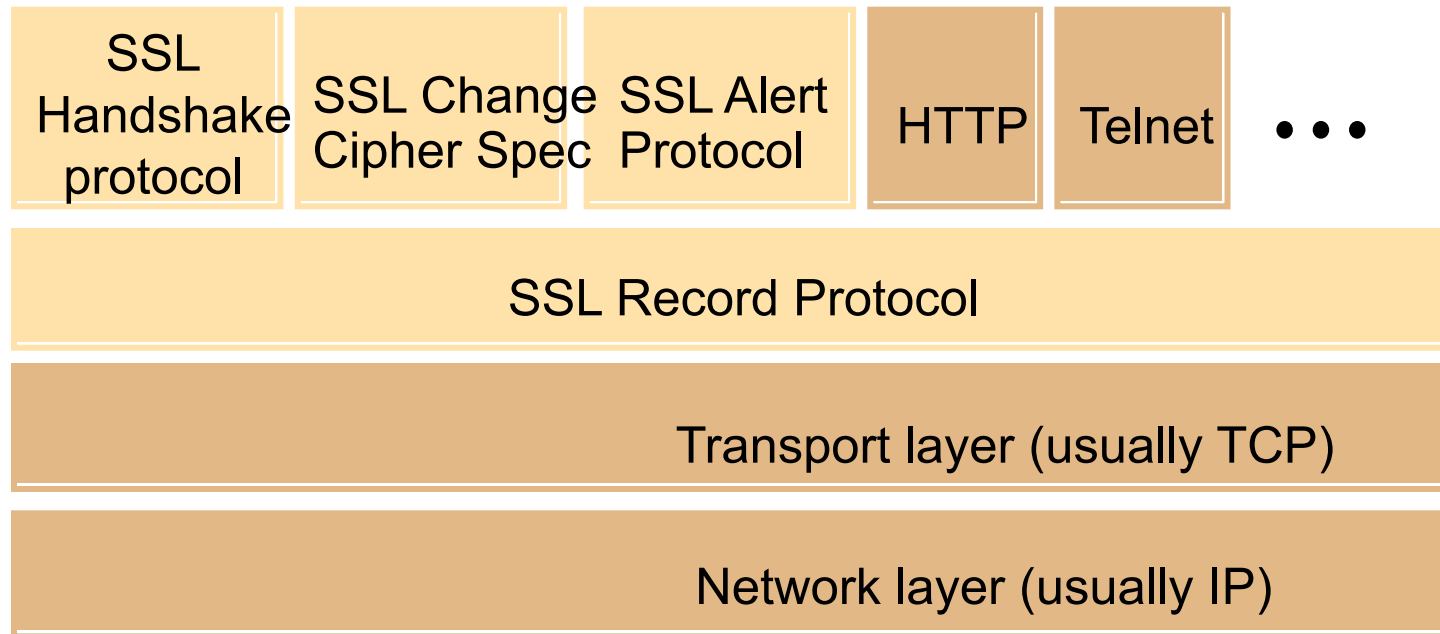


SSL Overview

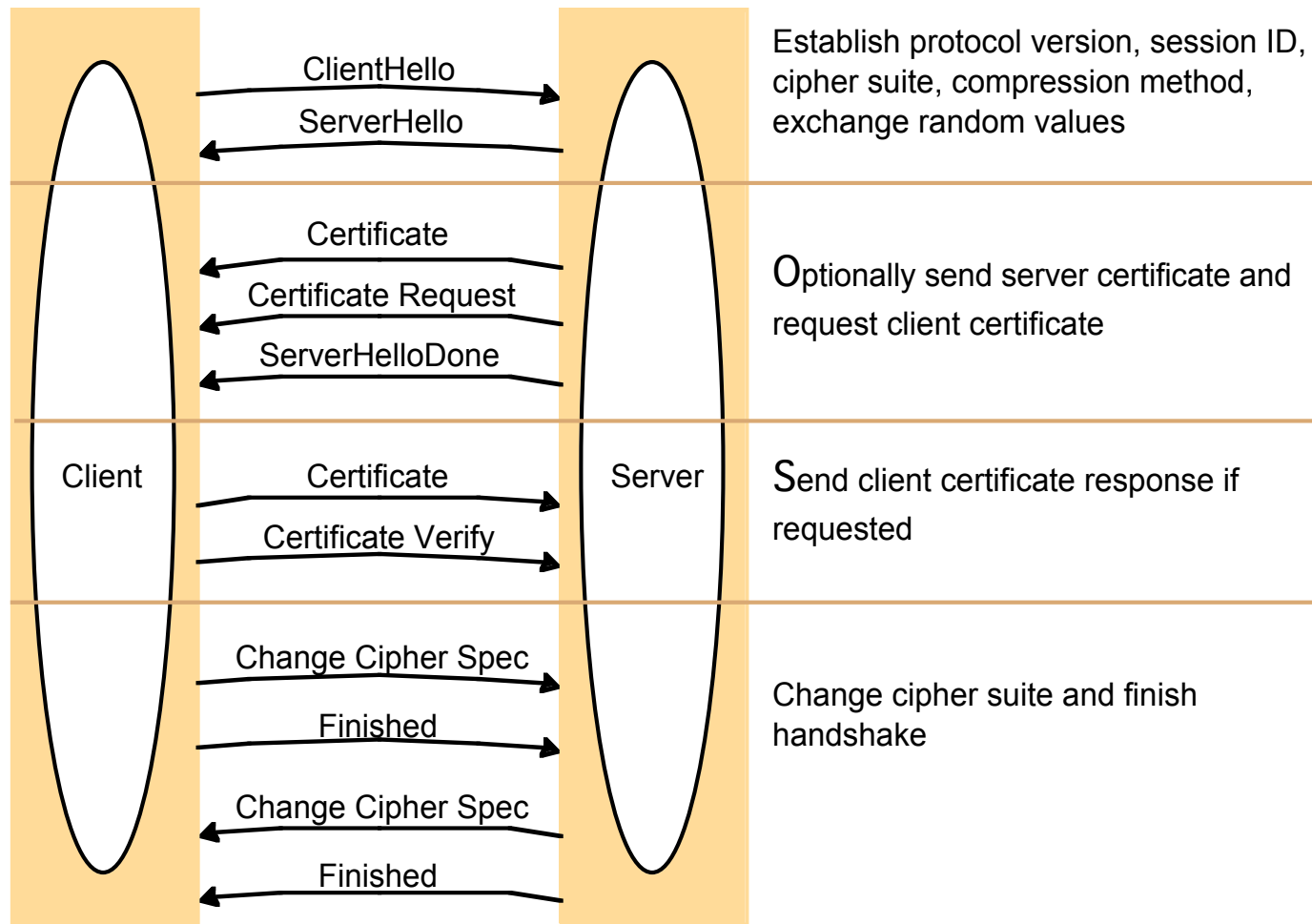
- Developed by Netscape Communications
- Authenticates servers (and optionally clients)
- Performs secret key exchange like Diffie-Hellman
- Data is encrypted with the exchanged key
- Clients do not need to provide a certificate but may be required to by the server
- Client authentication is typically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



SSL Protocol Stack



TLS Handshake Protocol



TLS Handshake Configuration Options

<i>Component</i>	<i>Description</i>	<i>Example</i>
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



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



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
95-702 Distributed Systems
Master of Information System
Management

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");

// Initialize 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.





