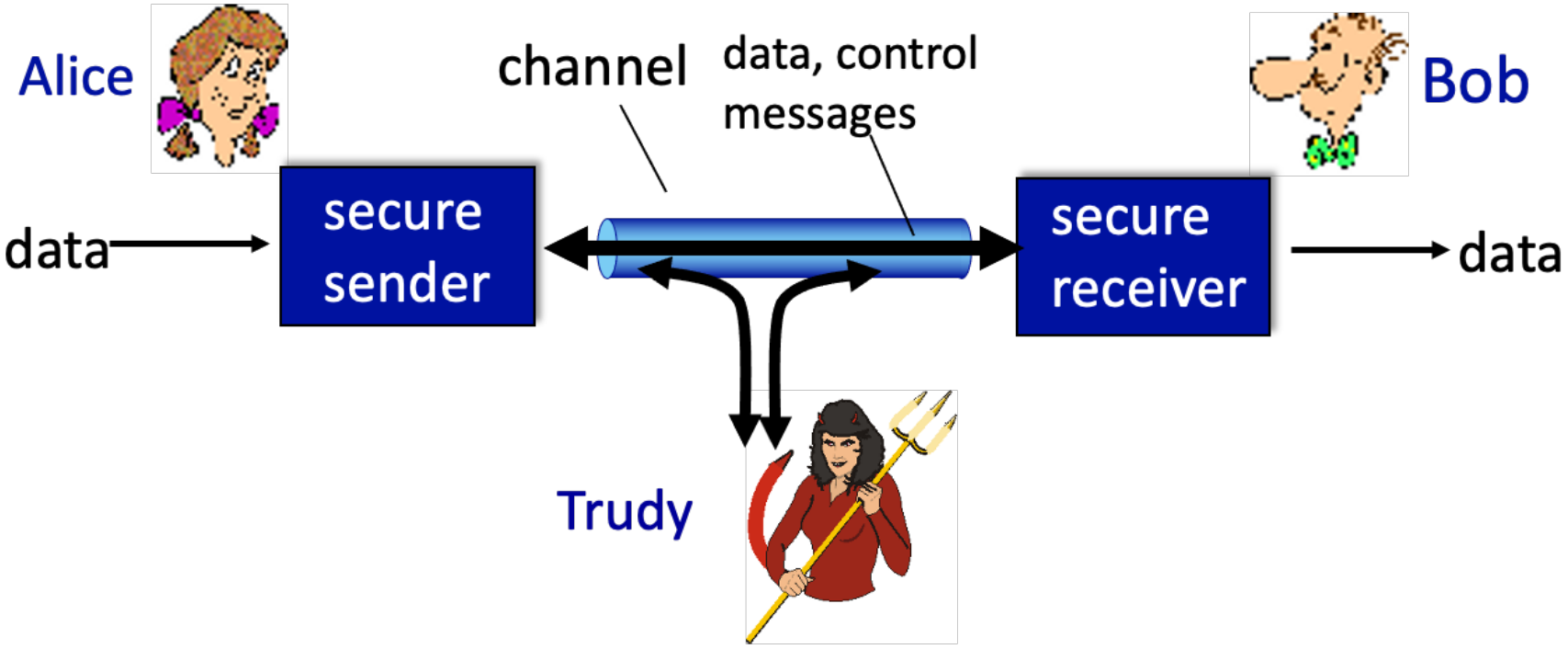
Secure Sockets

Hyang-Won Lee
Department of Computer Science and Engineering
Konkuk University
leehw@konkuk.ac.kr
https://sites.google.com/view/leehwko/

Friends and Enemies

- Bob and Alice (lovers!) want to communicate securely
- Trudy (intruder) may intercept, delete, add messages



Friends and Enemies

- Who might Bob and Alice be?
  - Real-life Bobs and Alices
  - Web browser/server for electronic transactions (e.g., online purchases)
  - Online banking client/server
  - DNS servers
  - BGP routers exchanging routing table updates

A Bad Guy Can…

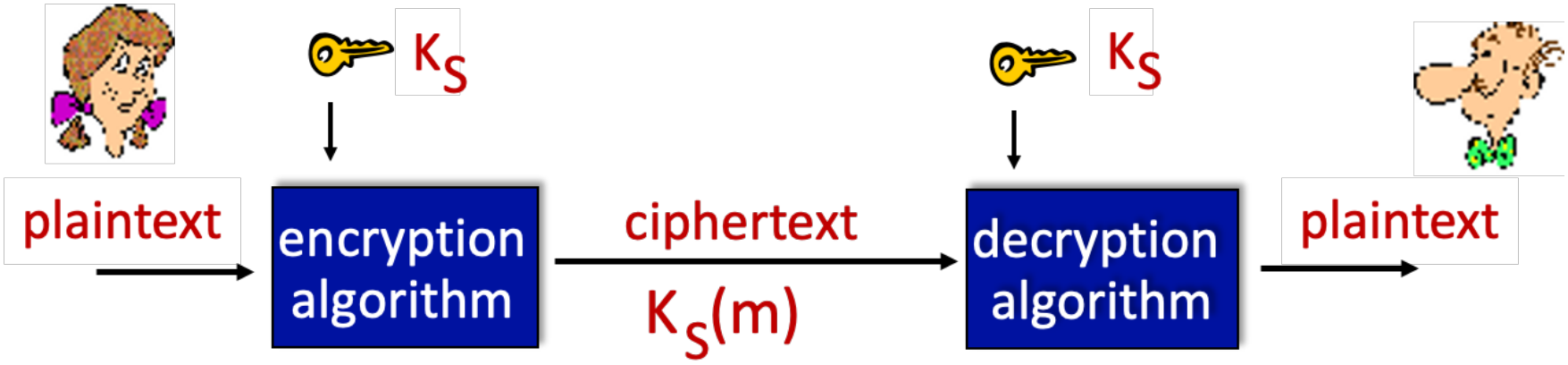
- Eavesdrop (intercept messages)
- Actively insert messages into connection
- Impersonate
  - Fake (spoof) source address in packet (or any field in packet)
- Hijack
  - Take over ongoing connection by removing sender or receiver, inserting himself in place
- Flood
  - Denial of service: prevent service from being used by others (e.g., by overloading resources)

Network Security

- Confidentiality
  - Only sender, intended receiver should understand message contents
    - Sender and receiver both encrypt messages
- Authentication
  - Sender, receiver want to confirm identity of each other
- Message integrity
  - Sender, receiver want to ensure message not altered without detection
- Access and availability
  - Services must be accessible and available to users

Symmetric Key Cryptography

- Shared key



- Example: mono-alphabetic substitution cipher

```
plaintext:   abcdefghijklmnopqrstuvwxyz

ciphertext:  mnbvcxzasdfghjklpoiuytrewq
```

e.g.:

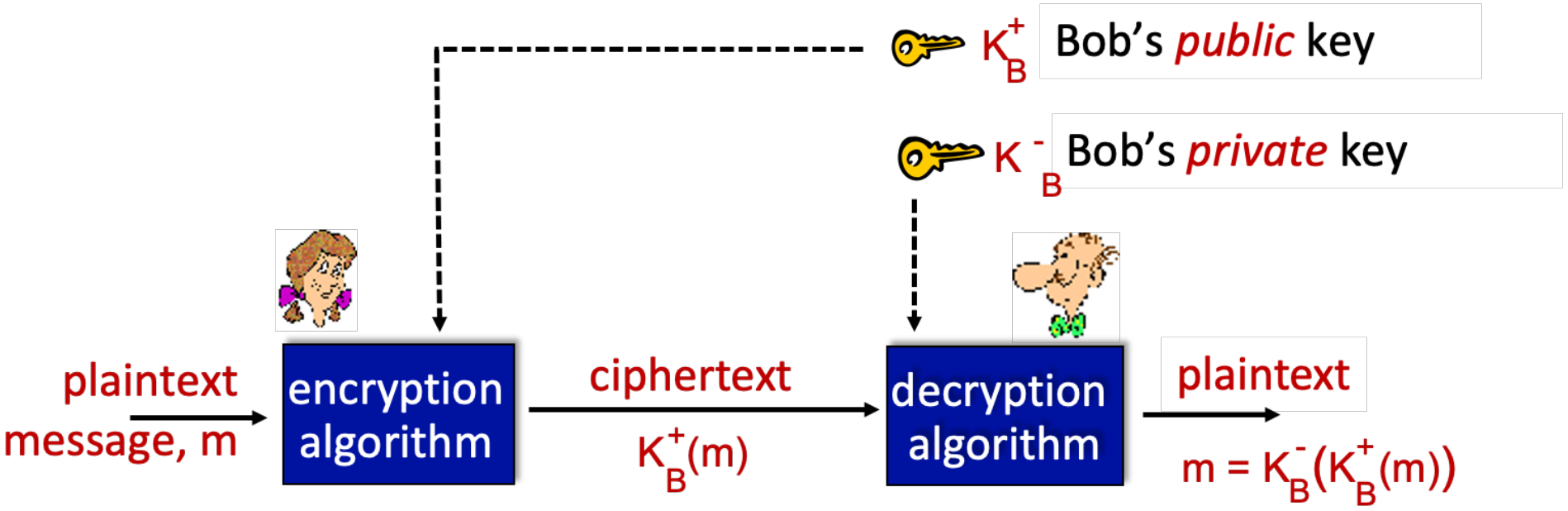
```
Plaintext: bob. i love you. alice
ciphertext: nkn. s gktc wky. mgsbc
```

Public Key Cryptography

- Caveat of symmetric key crypto
  - Sender and receiver need to agree on key. How on the Internet?

- Public key crypto
  - Radically different approach [Diffie-Hellman76, RSA78]
  - Sender and receiver do not share secret key
  - Public encryption key known to all
  - Private decryption key known only to receiver

Public Key Cryptography



- Requirements
  - Need $K_B^+(\cdot)$ and $K_B^-(\cdot)$ such that $K_B^-(K_B^+(m)) = m$
  - Given public key $K_B^+(\cdot)$, it should be impossible to compute private key $K_B^-(\cdot)$

Rivest-Shamir-Adleman (RSA) algorithm

- RSA algorithm has become synonymous with public key crypto
- Interchangeability (in addition to the two requirements)
  - $K_B^-(K_B^+(m)) = m$ and $K_B^+(K_B^-(m)) = m$
  - Used for authentication
- RSA is too time-consuming to encrypt and decrypt a large amount of data
  - Often used with symmetric key crypto
    - RSA for symmetric key change
    - Symmetric key cipher for data transfer

Authentication

- Use nonce and public key crypto

"I am Alice"

R

$K_A^-(R)$

Send me your public key

K_A^+

Bob computes

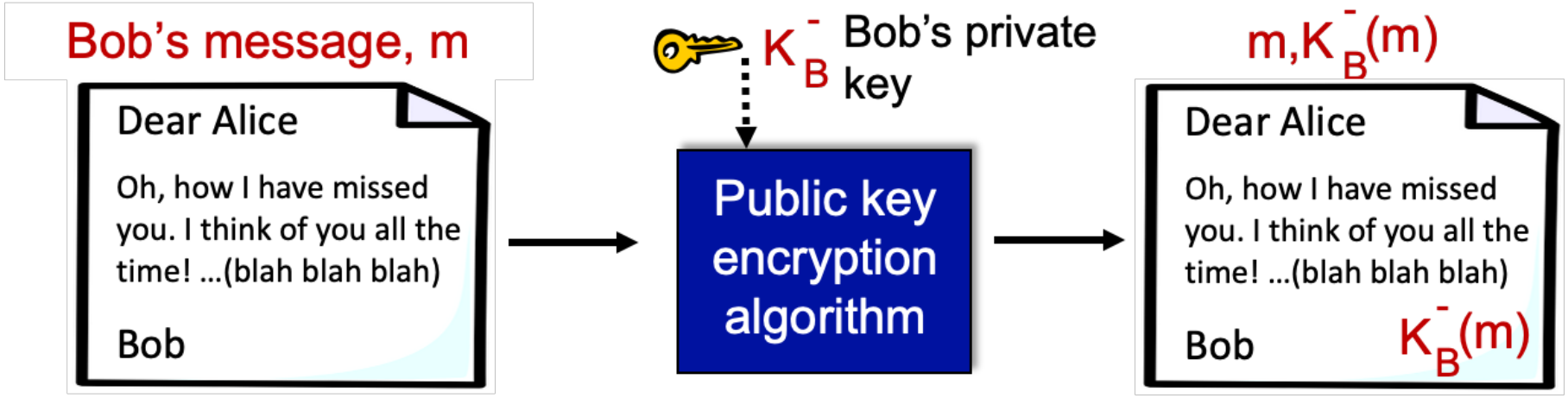
$K_A^+(K_A^-(R)) = R$

and knows only Alice could have the private key, that encrypted R such that

$K_A^+(K_A^-(R)) = R$

Digital Signatures

- Cryptographic technique analogous to hand-written signatures
  - Sender (Bob) digitally signs document
  - Recipient (Alice) can prove to someone that Bob and no one else (including Alice), must have signed document
  - Simple digital signature for message m



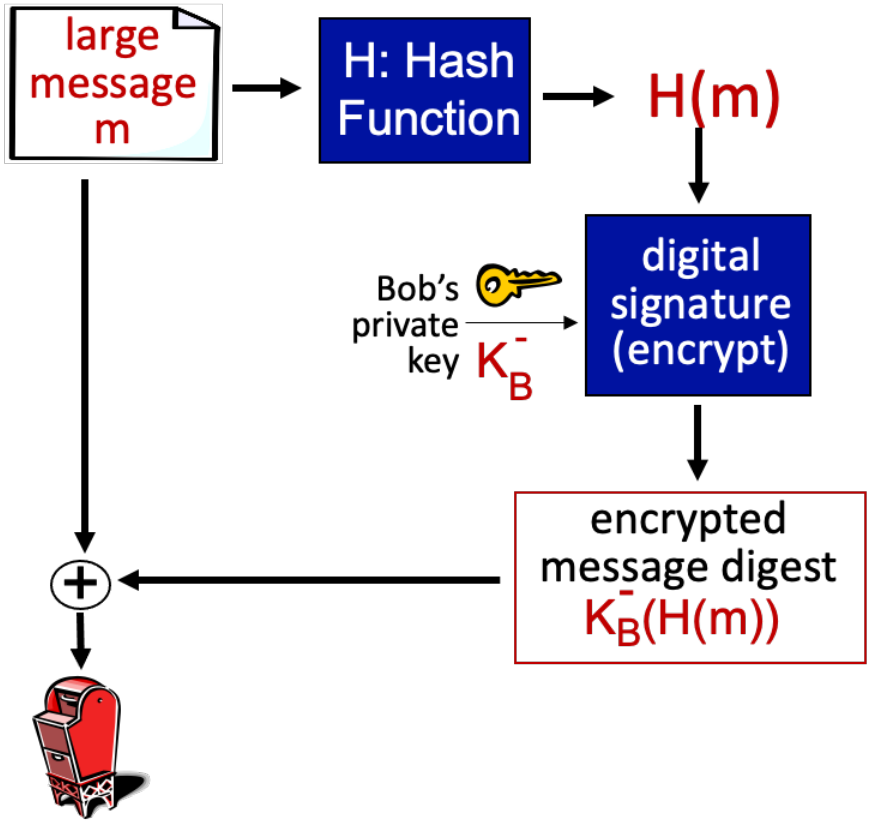
Message Digests

- Public key encryption is computationally expensive for long messages
- Apply hash function H to m and get fixed-size message digest H(m)

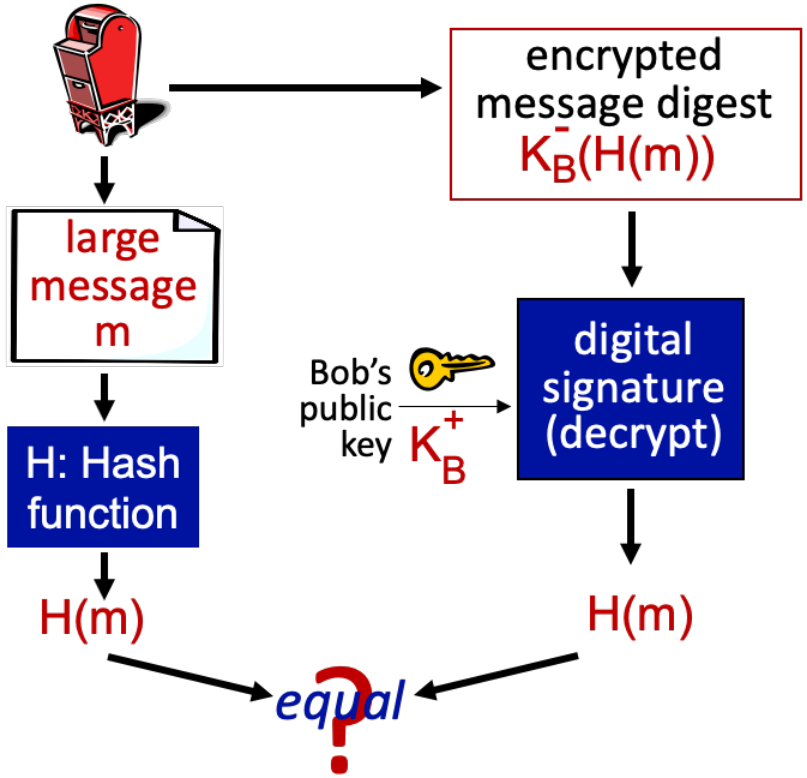
- Hash function properties
  - Many-to-1
  - Produce fixed-size message digest (fingerprint)
  - Computationally infeasible to find any two different messages m and m' such that H(m)=H(m')

Digital Signature = Signed Message Digest

Bob sends digitally signed message:



Alice verifies signature, integrity of digitally signed message:

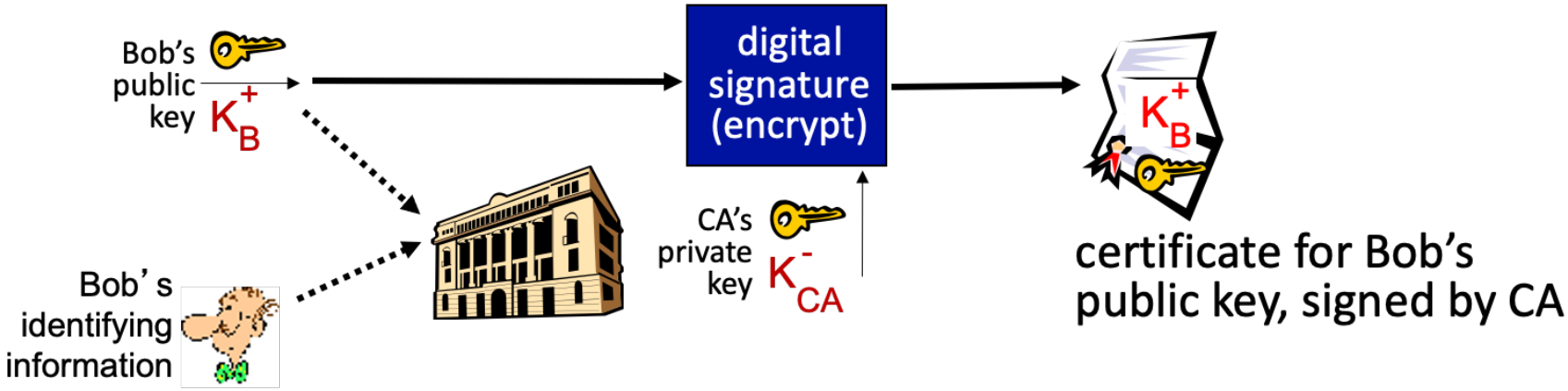


Need for Certified Public Keys

- Trudy plays pizza prank on Bob
  - Trudy creates e-mail order
    - Dear Pizza Store, Please deliver to me four pepperoni pizzas. Thank you, Bob
  - Trudy signs order with her private key
  - Trudy sends order to Pizza Store
  - Trudy sends to Pizza Store her public key, but says it's Bob's public key
  - Pizza Store verifies signatures; then delivers four pepperoni pizzas to Bob
  - Bob doesn't even like pepperoni

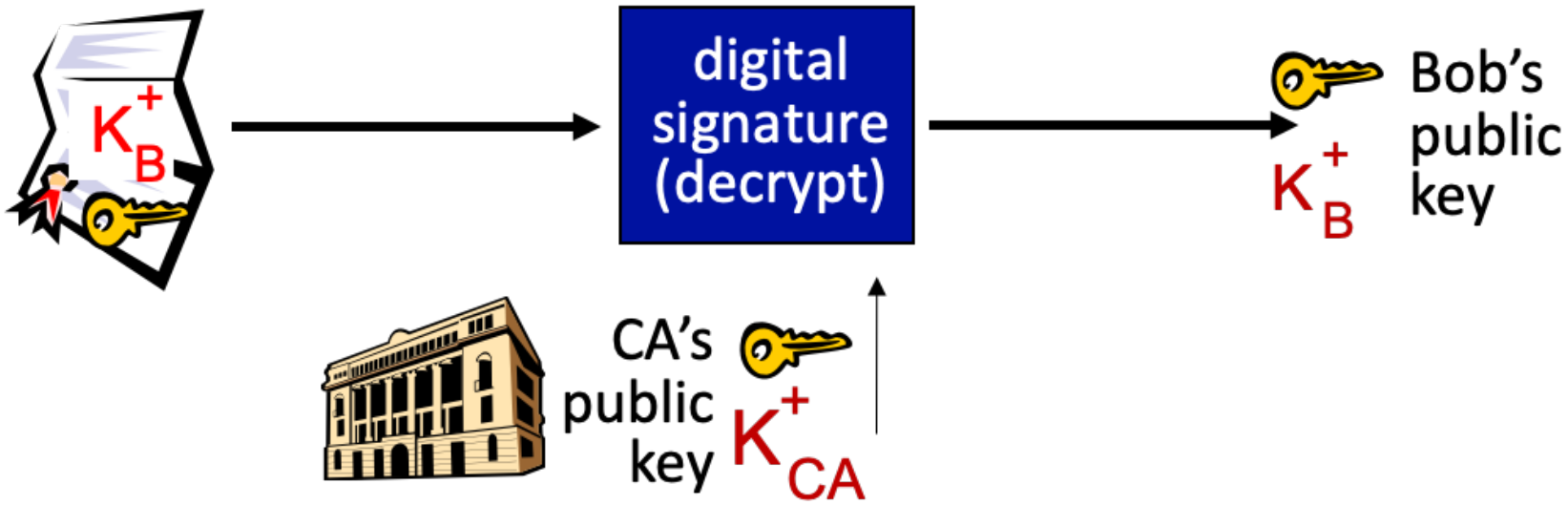
Public Key Certification Authorities

- Certification authority (CA) binds public key to particular entity E
- Entity (person, website, router) registers its public key with CA
  - CA creates certificate binding identity E to E's public key
  - Certificate containing E's public key digitally signed by CA: CA says "this is E's public key"



Public Key Certification Authorities

- When Alice wants Bob's public key
  - Get Bob's certificate (Bob or elsewhere)
  - Apply CA's public key to Bob's certificate, and get Bob's public key



Transport-Layer Security (TLS)

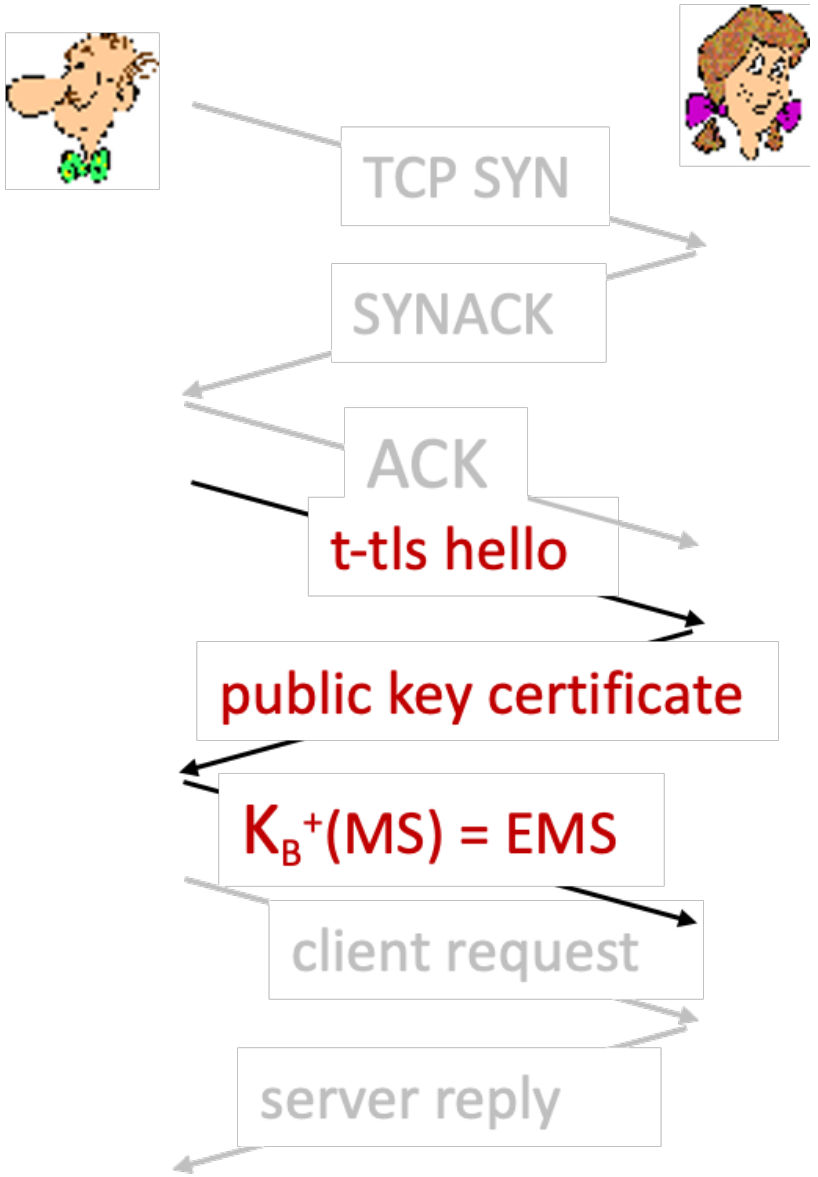
- Widely deployed security protocol above the transport layer
  - Supported by almost all browsers, web servers: https (port 443)
- Provide
  - Confidentiality via symmetric encryption
  - Integrity via cryptographic hasing
  - Authentication via public key cryptography
- History
  - Early research, implementation: secure network programming, secure sockets
  - Secure socket layer (SSL) deprecated [2015]
  - TLS 1.3: RFC 8846 [2018]

TLS: Required Pieces

- Handshake
  - Alice, Bob use their certificates and private keys to authenticate each other, and exchange or create shared secret
- Key derivation
  - Alice and Bob use shared secret to derive set of keys
- Data transfer
  - Transfer data as a series of records
- Connection closure
  - Special messages to securely close connection

Concept: Initial Handshake

- Bob establishes TCP connection with Alice
- Bob verifies that Alice is really Alice
- Bob sends Alice a master secret key (MS), used to generate all other keys for TLS session
- Potential issues
  - 3 RTT before client can start receiving data (including TCP handshake)



Cryptographic Keys

- Better to use different keys for message authentication code (MAC) and encryption
- Four keys
  - K_c: encryption key for data sent from client to server
  - M_c: MAC key for data sent from client to server
  - K_s: encryption key for data sent from server to client
  - M_s: MAC key for data sent from server to client
- Keys derived from key derivation function (KDF)
  - Takes master secret and (possibly) some additional random data to create new keys

Encrypting Data

- TCP bytestream is broken into a series of "records"
  - Each client-to-server record carries a MAC, created using M_c
  - Receiver can act on each record as it arrives

- Record encrypted using symmetric key K_c passed to TCP

Encrypting Data

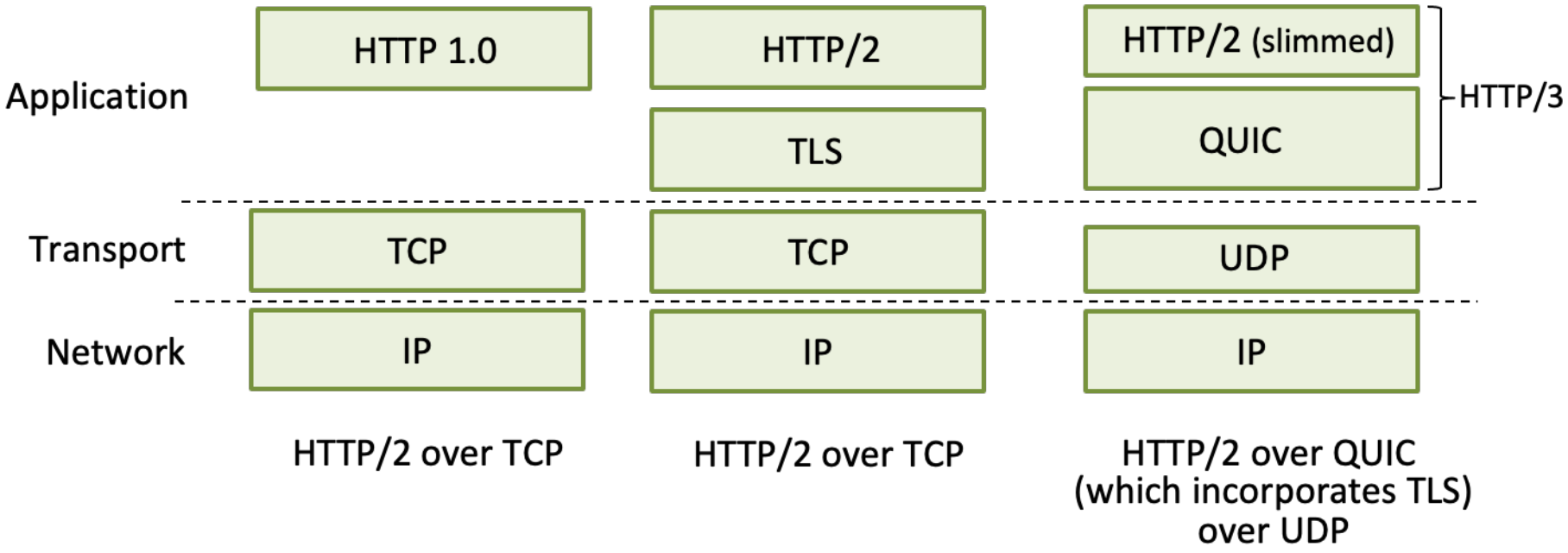
- Possible attacks on data stream
  - Reordering: main-in-the-middle intercepts TCP segments and reorders (manipulating sequence #s in unencrypted TCP header)
  - Replay, remove
- Solution
  - Use TLS sequence numbers (data, TLS-seq-# incorporated into MAC)

Connection Close

- Truncation attack
  - Attacker forges TCP connection close segment
  - One or both sides thinks there is less data than there actually is
- Solution
  - Record types, with one type for closure
  - Type 0 for data; type 1 for close
- MAC now computed using data, type, sequence #

TLS as API

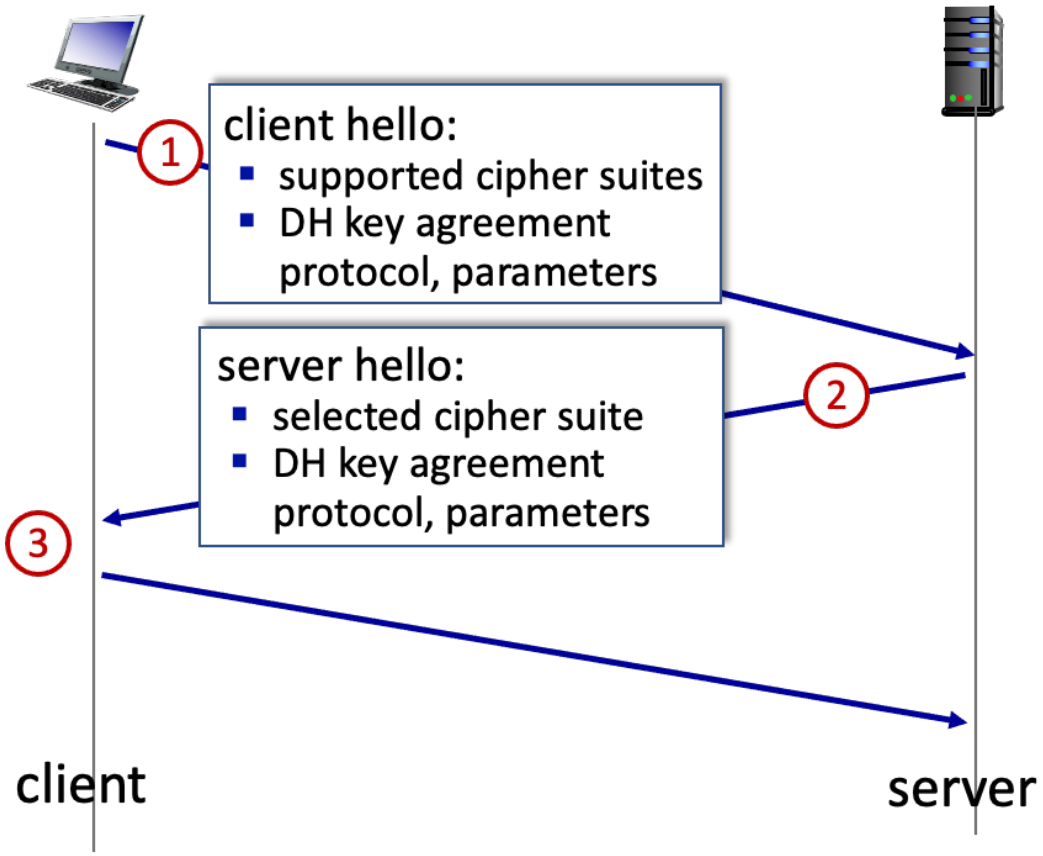
- TLS provides an API that any application can use
- An HTTP view of TLS



Cipher Suites

- Cipher suite
  - Algorithms that can be used for key generation, encryption, MAC, digital signature
- TLS 1.3(2018) more limited cipher suite choice than TLS 1.2(2008)
  - Only 5 choices, rather than 37 choices
  - Require Diffie-Hellman (DH) for key exchange, rather than DH or RSA
  - Combined encryption and authentication algorithm ("authenticated encryption") for data rather than serial encryption, authentication
    - 4 based on AES
  - HMAC uses SHA (256 or 284) cryptographic hash function

TLS 1.3 Handshake



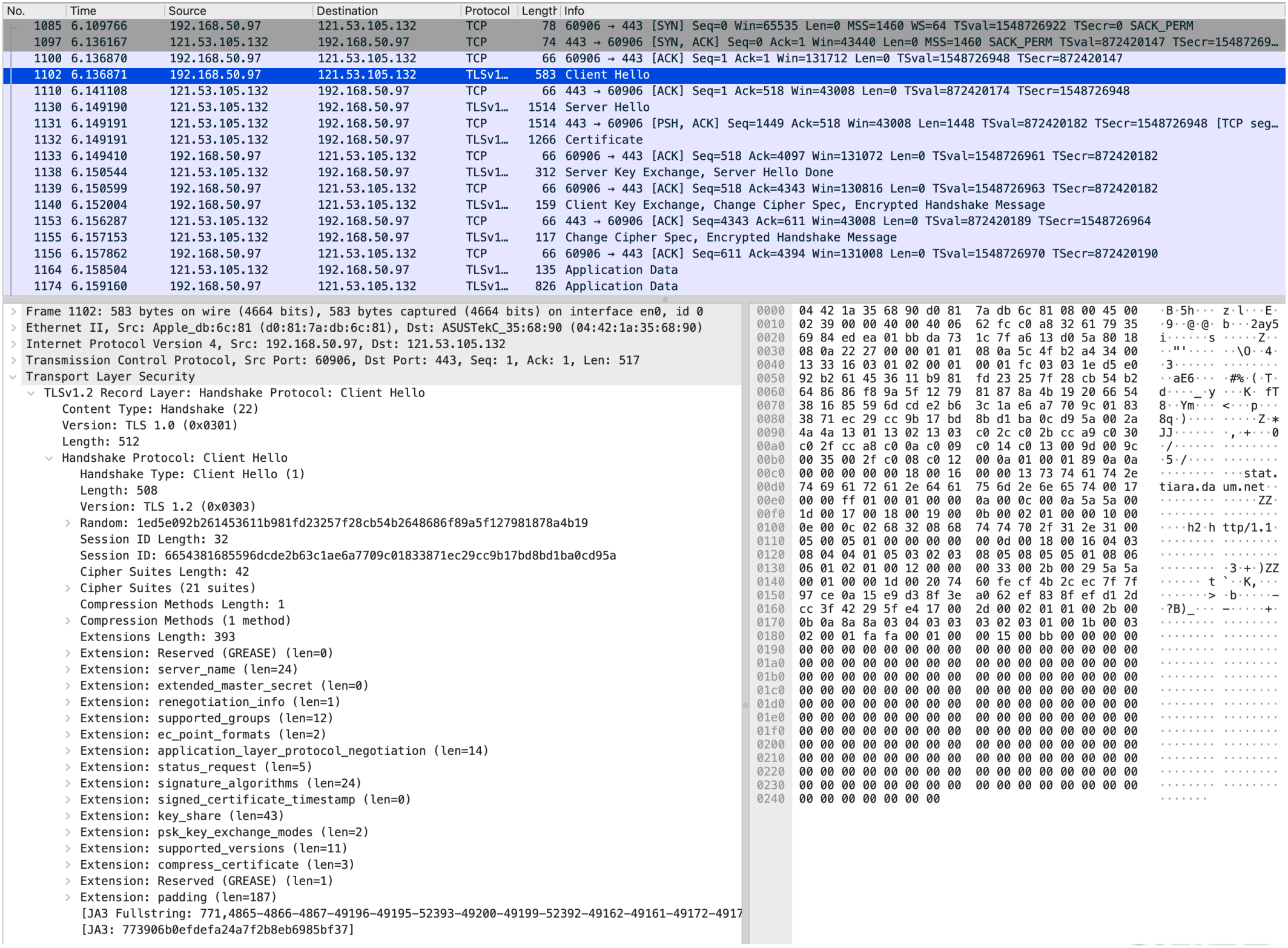
1. client TLS hello msg:
   - *guesses* key agreement protocol, parameters
   - indicates cipher suites it supports
2. server TLS hello msg chooses
   - key agreement protocol, parameters
   - cipher suite
   - server-signed certificate
3. client:
   - checks server certificate
   - generates key
   - can now make application request (e.g.., HTTPS GET)

Wireshark Example

- Chapter10 TLS-capture-example-www.daum.net-ip.addr == 121.53.105.132.pcap
- TCP connection setup before hello message
- Client hello
  - Record
    - Content type: handshake
    - Version: TLS 1.0
    - Length: 512
    - Cipher suites (21 suites)
- Server hello
  - Cipher suite
  - Extended master secret

Wireshark Example

- Chapter10 TLS-capture-example-www.daum.net-ip.addr == 121.53.105.132.pcap
- TCP connection setup before hello message
- Client hello
  - Record
    - Content type: handshake
    - Version: TLS 1.0
    - Length: 512
    - Cipher suites (21 suites)
- Server hello
  - Cipher suite
  - Extended master secret
- Application data
  - HTTP (header hidden)


No. Time Source Destination Protocol Length Info
1085 6.109766 192.168.50.97 121.53.105.132 TCP 78 60906 → 443 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=64 TSval=1548726922 TSecr=0 SACK_PERM
1097 6.136167 121.53.105.132 192.168.50.97 TCP 74 443 → 60906 [SYN, ACK] Seq=0 Ack=1 Win=43440 Len=0 MSS=1460 SACK_PERM TSval=872420147 TSecr=15487269…
1100 6.136870 192.168.50.97 121.53.105.132 TCP 66 60906 → 443 [ACK] Seq=1 Ack=1 Win=131712 Len=0 TSval=1548726948 TSecr=872420147
1102 6.136871 192.168.50.97 121.53.105.132 TLSv1… 583 Client Hello
1110 6.141108 121.53.105.132 192.168.50.97 TCP 66 443 → 60906 [ACK] Seq=1 Ack=518 Win=43008 Len=0 TSval=872420174 TSecr=1548726948
1130 6.149190 121.53.105.132 192.168.50.97 TLSv1… 1514 Server Hello
1131 6.149191 121.53.105.132 192.168.50.97 TCP 1514 443 → 60906 [PSH, ACK] Seq=1449 Ack=518 Win=43008 Len=1448 TSval=872420182 TSecr=1548726948 [TCP seg…
1132 6.149191 121.53.105.132 192.168.50.97 TLSv1… 1266 Certificate
1133 6.149410 192.168.50.97 121.53.105.132 TCP 66 60906 → 443 [ACK] Seq=518 Ack=4097 Win=131072 Len=0 TSval=1548726961 TSecr=872420182
1138 6.150544 121.53.105.132 192.168.50.97 TLSv1… 312 Server Key Exchange, Server Hello Done
1139 6.150599 192.168.50.97 121.53.105.132 TCP 66 60906 → 443 [ACK] Seq=518 Ack=4343 Win=130816 Len=0 TSval=1548726963 TSecr=872420182
1140 6.152004 192.168.50.97 121.53.105.132 TLSv1… 159 Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
1153 6.156287 121.53.105.132 192.168.50.97 TCP 66 443 → 60906 [ACK] Seq=4343 Ack=611 Win=43008 Len=0 TSval=872420189 TSecr=1548726964
1155 6.157153 121.53.105.132 192.168.50.97 TLSv1… 117 Change Cipher Spec, Encrypted Handshake Message
1156 6.157862 192.168.50.97 121.53.105.132 TCP 66 60906 → 443 [ACK] Seq=611 Ack=4394 Win=131008 Len=0 TSval=1548726970 TSecr=872420190
1164 6.158504 121.53.105.132 192.168.50.97 TLSv1… 135 Application Data
1174 6.159160 192.168.50.97 121.53.105.132 TLSv1… 826 Application Data
Frame 1102: 583 bytes on wire (4664 bits), 583 bytes captured (4664 bits) on interface en0, id 0
Ethernet II, Src: Apple_db:6c:81 (d0:81:7a:db:6c:81), Dst: ASUSTekC_35:68:90 (04:42:1a:35:68:90)
Internet Protocol Version 4, Src: 192.168.50.97, Dst: 121.53.105.132
Transmission Control Protocol, Src Port: 60906, Dst Port: 443, Seq: 1, Ack: 1, Len: 517
Transport Layer Security
TLSv1.2 Record Layer: Handshake Protocol: Client Hello
Content Type: Handshake (22)
Version: TLS 1.0 (0x0301)
Length: 512
Handshake Protocol: Client Hello
Handshake Type: Client Hello (1)
Length: 508
Version: TLS 1.2 (0x0303)
Random: 1ed5e092b261453611b981fd23257f28cb54b2648686f89a5f127981878a4b19
Session ID Length: 32
Session ID: 6654381685596dcde2b63c1ae6a7709c01833871ec29cc9b17bd8bd1ba0cd95a
Cipher Suites Length: 42
Cipher Suites (21 suites)
Compression Methods Length: 1
Compression Methods (1 method)
Extensions Length: 393
Extension: Reserved (GREASE) (len=0)
Extension: server_name (len=24)
Extension: extended_master_secret (len=0)
Extension: renegotiation_info (len=1)
Extension: supported_groups (len=12)
Extension: ec_point_formats (len=2)
Extension: application_layer_protocol_negotiation (len=14)
Extension: status_request (len=5)
Extension: signature_algorithms (len=24)
Extension: signed_certificate_timestamp (len=0)
Extension: key_share (len=43)
Extension: psk_key_exchange_modes (len=2)
Extension: supported_versions (len=11)
Extension: compress_certificate (len=3)
Extension: Reserved (GREASE) (len=1)
Extension: padding (len=187)
[JA3 Fullstring: 771,4865-4866-4867-49196-49195-52393-49200-49199-52392-49162-49161-49172-4917
[JA3: 773906b0efdefa24a7f2b8eb6985bf37]
0000 04 42 1a 35 68 90 d0 81 7a db 6c 81 08 00 45 00 ·B·5h··· z·l···E·
0010 02 39 00 00 40 00 40 06 62 fc c0 a8 32 61 79 35 ·9··@·@· b···2ay5
0020 69 84 ed ea 01 bb da 73 1c 7f a6 13 d0 5a 80 18 i·····s ·····Z··
0030 08 0a 22 27 00 00 01 01 08 0a 5c 4f b2 a4 34 00 ··"'···· ··\O··4·
0040 13 33 16 03 01 02 00 01 00 01 fc 03 03 1e d5 e0 ·3····· ·····
0050 92 b2 61 45 36 11 b9 81 fd 23 25 7f 28 cb 54 b2 ··aE6··· ·#%·(·T·
0060 64 86 86 f8 9a 5f 12 79 81 87 8a 4b 19 20 66 54 d····_·y ···K· fT
0070 38 16 85 59 6d cd e2 b6 3c 1a e6 a7 70 9c 01 83 8··Ym··· <···p···
0080 38 71 ec 29 cc 9b 17 bd 8b d1 ba 0c d9 5a 00 2a 8q·)···· ·····Z·*
0090 4a 4a 13 01 13 02 13 03 c0 2c c0 2b cc a9 c0 30 JJ····· ·,·+···0
00a0 c0 2f cc a8 c0 0a c0 09 c0 14 c0 13 00 9d 00 9c ·/····· ·····
00b0 00 35 00 2f c0 08 c0 12 00 0a 01 00 01 89 0a 0a ·5·/···· ·····
00c0 00 00 00 00 00 18 00 16 00 00 13 73 74 61 74 2e ····· ···stat.
00d0 74 69 61 72 61 2e 64 61 75 6d 2e 6e 65 74 00 17 tiara.da um.net··
00e0 00 00 ff 01 00 01 00 00 0a 00 0c 00 0a 5a 5a 00 ····· ····ZZ·
00f0 1d 00 17 00 18 00 19 00 0b 00 02 01 00 00 10 00 ····· ·····
0100 0e 00 0c 02 68 32 08 68 74 74 70 2f 31 2e 31 00 ····h2·h ttp/1.1·
0110 05 00 05 01 00 00 00 00 00 0d 00 18 00 16 04 03 ····· ·····
0120 08 04 04 01 05 03 02 03 08 05 08 05 05 01 08 06 ····· ·····
0130 06 01 02 01 00 12 00 00 00 33 00 2b 00 29 5a 5a ····· ·3·+·)ZZ
0140 00 01 00 00 1d 00 20 74 60 fe cf 4b 2c ec 7f 7f ····· t`··K,···
0150 97 ce 0a 15 e9 d3 8f 3e a0 62 ef 83 8f ef d1 2d ·····> ·b·····-
0160 cc 3f 42 29 5f e4 17 00 2d 00 02 01 01 00 2b 00 ·?B)_··· -·····+·
0170 0b 0a 8a 8a 03 04 03 03 03 02 03 01 00 1b 00 03 ····· ·····
0180 02 00 01 fa fa 00 01 00 00 15 00 bb 00 00 00 00 ····· ·····
0190 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ····· ·····
01a0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ····· ·····
01b0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ····· ·····
01c0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ····· ·····
01d0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ····· ·····
01e0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ····· ·····
01f0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ····· ·····
0200 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ····· ·····
0210 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ····· ·····
0220 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ····· ·····
0230 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ····· ·····
0240 00 00 00 00 00 00 00 ·····

SSL in Java

Java Secure Socket Extension

- java.net.ssl
  - Abstract classes that define Java's API for secure network communication
- javax.net
  - Abstract socket factory classes used instead of constructors to create secure sockets
- java.security.cert
  - Classes for handling public-key certificates needed for SSL
- com.sun.net.ssl
  - Concrete classes that implement encryption algorithms and protocols in Sun's reference implementation of JSSE

Creating Secure Client Sockets

- Use Java.net.ssl.SSLSocketFactory (instead of java.net.Socket)
  - SSLSocketFactory is an abstract class
- Getting an instance of SSLSocketFactory

```
SocketFactory factory = SSLSocketFactory.getDefault();
Socket socket = factory.createSocket("login.ibiblio.org", 7000);
```

  - InstantiationException is thrown if no concrete subclass can be found

createSocket() Methods

remote

```
public abstract Socket createSocket(String host, int port)
    throws IOException, UnknownHostException
public abstract Socket createSocket(InetAddress host, int port)
    throws IOException
public abstract Socket createSocket(String host, int port,
    InetAddress interface, int localPort) local
    throws IOException, UnknownHostException
public abstract Socket createSocket(InetAddress host, int port,
    InetAddress interface, int localPort)
    throws IOException, UnknownHostException
public abstract Socket createSocket(Socket proxy, String host, int port,
    boolean autoClose) throws IOException
```

- IOException if unable to connect

- Last method
  - proxy: existing Socket connected to a proxy server
  - Return a Socket tunneling through proxy server to the specified host and port
  - autoClose
    - True: underlying proxy socket is closed when this socket is closed

Socket Objects

- Returned Socket is a Java.net.ssl.SSLSocket which is a subclass of java.net.Socket
- Once a socket is returned, it can be used in the same way as any other socket is used
  - getInputStream(), getOutputStream(), …
- Example
  - A server accepts orders is listening on port 7000 of login.ibiblio.org
  - Each order is sent as an ASCII string using a single TCP conn.

```
Name: John Smith
Product-ID: 67X-89
Address: 1280 Deniston Blvd, NY NY 10003
Card number: 4000-1234-5678-9017
Expires: 08/05
```

Example (contd.)

```
SSLSocketFactory factory
    = (SSLSocketFactory) SSLSocketFactory.getDefault();
Socket socket = factory.createSocket("login.ibiblio.org", 7000);

Writer out = new OutputStreamWriter(socket.getOutputStream(),
    "US-ASCII");
out.write("Name: John Smith\r\n");
out.write("Product-ID: 67X-89\r\n");
out.write("Address: 1280 Deniston Blvd, NY NY 10003\r\n");
out.write("Card number: 4000-1234-5678-9017\r\n");
out.write("Expires: 08/05\r\n");
out.flush();
```

Compare with normal insecure sockets

Simple Program

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

public class HTTPSClient {

  public static void main(String[] args) {

    if (args.length == 0) {
      System.out.println("Usage: java HTTPSClient2 host");
      return;
    }

    int port = 443; // default https port
    String host = args[0];

    SSLSocketFactory factory
        = (SSLSocketFactory) SSLSocketFactory.getDefault();
    SSLSocket socket = null;
    try {
      socket = (SSLSocket) factory.createSocket(host, port);
```

```
// enable all the suites
String[] supported = socket.getSupportedCipherSuites();
socket.setEnabledCipherSuites(supported);

Writer out = new OutputStreamWriter(socket.getOutputStream(), "UTF-8");
// https requires the full URL in the GET line
out.write("GET http://" + host + "/ HTTP/1.1\r\n");
out.write("Host: " + host + "\r\n");
out.write("\r\n");
out.flush();

// read response
BufferedReader in = new BufferedReader(
    new InputStreamReader(socket.getInputStream()));

// read the header
String s;
while (!(s = in.readLine()).equals("")) {
  System.out.println(s);
}
System.out.println();
```

Header structure

```
      // read the length
      String contentLength = in.readLine();
      int length = Integer.MAX_VALUE;
      try {
        length = Integer.parseInt(contentLength.trim(), 16);
      } catch (NumberFormatException ex) {
        // This server doesn't send the content-length
        // in the first line of the response body
      }
      System.out.println(contentLength);

      int c;
      int i = 0;
      while ((c = in.read()) != -1 && i++ < length) {
        System.out.write(c);
      }

      System.out.println();
    } catch (IOException ex) {
      System.err.println(ex);
    } finally {
      try {
        if (socket != null) socket.close();
      } catch (IOException e) {}
    }
  }
}
```

Notes

- It may take a few seconds to establish a connection
  - For generating and exchanging public keys
  - => slower connection
- Only the content that needs to be private and is not latency-sensitive is better to be sent over HTTPS

Choosing the Cipher Suites

- A combination of authentication and encryption algorithms can be selected
  - e.g., ECDHE for key exchange, ECDSA for authentication, AES-128 for data encryption, SHA-256 for hash
- Getter methods
  - `public abstract String[] getSupportedCipherSuites()`
    - return available combinations of algorithms in SSLSocketFactory (some disabled)
  - `public abstract String[] getEnabledCipherSuites()`
    - return enabled cipher suites

Negotiation of Cipher Suites

- Actual suite used is negotiated between client and server
- Client and server might not agree on any suite
  - even if agreed, one or the other or both may not have keys and certificates needed to use the suite
- In this case, createSocket() method will throw an SSLException (a subclass of IOException)
- Changing the suites the client attempts to use via

```
public abstract void setEnabledCipherSuites(String[] suites)
```

  - argument: list of suites client want to use
  - each in the list should be one of those returned by getSupportedCipherSuites()
    - otherwise, IllegalArgumentException thrown

Supported Cipher Suites

- TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256
- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
- TLS_RSA_WITH_AES_128_CBC_SHA256
- TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA256
- TLS_ECDH_RSA_WITH_AES_128_CBC_SHA256
- TLS_DHE_RSA_WITH_AES_128_CBC_SHA256
- TLS_DHE_DSS_WITH_AES_128_CBC_SHA256
- TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA
- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
- TLS_RSA_WITH_AES_128_CBC_SHA
- TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA
- TLS_ECDH_RSA_WITH_AES_128_CBC_SHA
- TLS_DHE_RSA_WITH_AES_128_CBC_SHA
- TLS_DHE_DSS_WITH_AES_128_CBC_SHA
- TLS_ECDHE_ECDSA_WITH_RC4_128_SHA
- TLS_ECDHE_RSA_WITH_RC4_128_SHA
- SSL_RSA_WITH_RC4_128_SHA
- TLS_ECDH_ECDSA_WITH_RC4_128_SHA
- TLS_ECDH_RSA_WITH_RC4_128_SHA
- TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA
- TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA
- SSL_RSA_WITH_3DES_EDE_CBC_SHA
- TLS_ECDH_ECDSA_WITH_3DES_EDE_CBC_SHA
- TLS_ECDH_RSA_WITH_3DES_EDE_CBC_SHA
- SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA
- SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA
- SSL_RSA_WITH_RC4_128_MD5
- TLS_EMPTY_RENEGOTIATION_INFO_SCSV
- TLS_DH_anon_WITH_AES_128_CBC_SHA256
- TLS_ECDH_anon_WITH_AES_128_CBC_SHA
- TLS_DH_anon_WITH_AES_128_CBC_SHA
- TLS_ECDH_anon_WITH_RC4_128_SHA
- SSL_DH_anon_WITH_RC4_128_MD5
- TLS_ECDH_anon_WITH_3DES_EDE_CBC_SHA
- SSL_DH_anon_WITH_3DES_EDE_CBC_SHA

- format: protocol, key exchange algorithm, encryption algorithm, and checksum
- ECDHE for key exchange, ECDSA for authentication, AES-128 for data encryption, SHA-256 for hash

Supported Cipher Suites

- TLS_RSA_WITH_NULL_SHA256
- TLS_ECDHE_ECDSA_WITH_NULL_SHA
- TLS_ECDHE_RSA_WITH_NULL_SHA
- SSL_RSA_WITH_NULL_SHA
- TLS_ECDH_ECDSA_WITH_NULL_SHA
- TLS_ECDH_RSA_WITH_NULL_SHA
- TLS_ECDH_anon_WITH_NULL_SHA
- SSL_RSA_WITH_NULL_MD5
- SSL_RSA_WITH_DES_CBC_SHA
- SSL_DHE_RSA_WITH_DES_CBC_SHA
- SSL_DHE_DSS_WITH_DES_CBC_SHA
- SSL_DH_anon_WITH_DES_CBC_SHA
- SSL_RSA_EXPORT_WITH_RC4_40_MD5
- SSL_DH_anon_EXPORT_WITH_RC4_40_MD5
- SSL_RSA_EXPORT_WITH_DES40_CBC_SHA
- SSL_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA
- SSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA
- SSL_DH_anon_EXPORT_WITH_DES40_CBC_SHA
- TLS_KRB5_WITH_RC4_128_SHA
- TLS_KRB5_WITH_RC4_128_MD5
- TLS_KRB5_WITH_3DES_EDE_CBC_SHA
- TLS_KRB5_WITH_3DES_EDE_CBC_MD5
- TLS_KRB5_WITH_DES_CBC_SHA
- TLS_KRB5_WITH_DES_CBC_MD5
- TLS_KRB5_EXPORT_WITH_RC4_40_SHA
- TLS_KRB5_EXPORT_WITH_RC4_40_MD5
- TLS_KRB5_EXPORT_WITH_DES_CBC_40_SHA
- TLS_KRB5_EXPORT_WITH_DES_CBC_40_MD5

Notes

- DES always encrypts 64 bits
  - if 64 bits are not available, extra bits are padded
- AES can encrypt blocks of 128, 192, or 256 bits, but still has to pad the input if it does not come out an even multiple of block size
- DES/AES are fine with file transfer applications such as HTTPS and SFTP
  - all the data are available at once
- DES/AES can be problematic for user-centered protocols such as chat and Telnet
- RC4 is a stream cipher that can encrypt one byte at a time

Creating Secure Server Sockets

- *javax.net.SSLServerSocket* class

```
public abstract class SSLServerSocket extends ServerSocket
```

  - all constructors are protected and instances are created by an abstract factory class *javax.net.SSLServerSocketFactory*

```
public abstract class SSLServerSocketFactory
    extends ServerSocketFactory
```

- *SSLServerSocketFactory.getDefault*() returns an instance of *SSLServerSocketFactory*

```
public static ServerSocketFactory getDefault()
```

-

Creating Secure Server Sockets

- *SSLServerSocketFactory* has three overloaded *createServerSocket*() methods that return instances of *SSLServerSocket*

```
public abstract ServerSocket createServerSocket(int port)
    throws IOException
public abstract ServerSocket createServerSocket(int port,
    int queueLength) throws IOException
public abstract ServerSocket createServerSocket(int port,
    int queueLength, InetAddress interface) throws IOException
```

- Caution
  - *SSLServerSocketFactory.getDefault*() returns the factory that generally only supports server authentication

Getting Encryption

- *com.sun.net.ssl.SSLContext* object is responsible for creating fully configured and initialized secure server sockets
- General procedure
  - generate public keys and certificates using key tool
  - pay money to have your certificates authenticated by a trusted third party such Comodo
  - create an *SSLContext* for the algorithm you'll use
  - create a *TrustManagerFactory* for the source of certificate material you will be using
  - create a *KayManagerFactory* for the type of key material you'll be using
  - create a *KeyStore* object for the key and certificate database (Oracle's default is JKS)
  - fill the *KeyStore* object with keys and certificates; for instance, by loading them from the filesystem using the passphrase they're encrypted with
  - initialize the *KeyManagerFactory* with the *KeyStore* and its passphrase
  - initialize the context with the necessary key managers from the *KeyManagerFactory*, trust mangers from the *TrustManagerFactory*, and a source of randomness

Example (for accepting orders and printing them on console)

```
import java.io.*;
import java.net.*;
import java.security.*;
import java.security.cert.CertificateException;
import java.util.Arrays;

import javax.net.ssl.*;

public class SecureOrderTaker {

  public final static int PORT = 7000;
  public final static String algorithm = "SSL";

  public static void main(String[] args) {
    try {
      SSLContext context = SSLContext.getInstance(algorithm);

      // The reference implementation only supports X.509 keys
      KeyManagerFactory kmf = KeyManagerFactory.getInstance("SunX509");

      // Oracle's default kind of key store
      KeyStore ks = KeyStore.getInstance("JKS");

      // For security, every key store is encrypted with a
      // passphrase that must be provided before we can load
      // it from disk. The passphrase is stored as a char[] array
      // so it can be wiped from memory quickly rather than
      // waiting for a garbage collector.
      char[] password = System.console().readPassword();
      ks.load(new FileInputStream("jnp4e.keys"), password);
      kmf.init(ks, password);
      context.init(kmf.getKeyManagers(), null, null);
```

```
Arrays.fill(password, '0');    wipe the password

SSLServerSocketFactory factory
    = context.getServerSocketFactory();

SSLServerSocket server
    = (SSLServerSocket) factory.createServerSocket(PORT);

// add anonymous (non-authenticated) cipher suites
String[] supported = server.getSupportedCipherSuites();
String[] anonCipherSuitesSupported = new String[supported.length];
int numAnonCipherSuitesSupported = 0;
for (int i = 0; i < supported.length; i++) {
  if (supported[i].indexOf("_anon_") > 0) {
    anonCipherSuitesSupported[numAnonCipherSuitesSupported++] =
                                                  supported[i];
  }
}

String[] oldEnabled = server.getEnabledCipherSuites();
String[] newEnabled = new String[oldEnabled.length
    + numAnonCipherSuitesSupported];
System.arraycopy(oldEnabled, 0, newEnabled, 0, oldEnabled.length);
System.arraycopy(anonCipherSuitesSupported, 0, newEnabled,
    oldEnabled.length, numAnonCipherSuitesSupported);
```

```
        server.setEnabledCipherSuites(newEnabled);

        // Now all the set up is complete and we can focus
        // on the actual communication.
        while (true) {
          // This socket will be secure,
          // but there's no indication of that in the code!
          try (Socket theConnection = server.accept()) {
            InputStream in = theConnection.getInputStream();
            int c;
            while ((c = in.read()) != -1) {
              System.out.write(c);
            }
          } catch (IOException ex) {
            ex.printStackTrace();
          }
        }
      } catch (IOException | KeyManagementException
          | KeyStoreException | NoSuchAlgorithmException
          | CertificateException | UnrecoverableKeyException ex) {
        ex.printStackTrace();
      }
    }
}
```

- This example loads the necessary keys and certificates from a file named jnp4e.keys in the current working directory protected with the password "2andnotafnord"
- jnp4e.keys created with *keytool* program in JDK

```
$ keytool -genkey -alias ourstore -keystore jnp4e.keys
Enter keystore password:
Re-enter new password:
What is your first and last name?
  [Unknown]:  Elliotte Harold
What is the name of your organizational unit?
  [Unknown]:  Me, Myself, and I
What is the name of your organization?
  [Unknown]:  Cafe au Lait
What is the name of your City or Locality?
  [Unknown]:  Brooklyn
What is the name of your State or Province?
  [Unknown]:  New York
What is the two-letter country code for this unit?
  [Unknown]:  NY
Is <CN=Elliotte Harold, OU="Me, Myself, and I", O=Cafe au Lait, L=Brooklyn,
ST=New York, C=NY> correct?
  [no]:  y

Enter key password for <ourstore>
        (RETURN if same as keystore password):
```

jnp4e.keys contain your public keys

Certifying Your keys

- Have your keys certified by a trusted third party such as GeoTrust or GoDaddy
  - it is not free
- use cipher suites that do not require authentication
  - SSL_DH_anon_EXPORT_WITH_DES40_CBC_SHA
  - SSL_DH_anon_EXPORT_WITH_RC4_40_MD5
  - SSL_DH_anon_WITH_3DES_EDE_CBC_SHA
  - SSL_DH_anon_WITH_DES_CBC_SHA
  - SSL_DH_anon_WITH_RC4_128_MD5
  - TLS_DH_anon_WITH_AES_128_CBC_SHA
  - TLS_DH_anon_WITH_AES_128_CBC_SHA256
  - TLS_ECDH_anon_WITH_3DES_EDE_CBC_SHA
  - TLS_ECDH_anon_WITH_AES_128_CBC_SHA
  - TLS_ECDH_anon_WITH_NULL_SHA
  - TLS_ECDH_anon_WITH_RC4_128_SHA

free but vulnerable to man-in-the-middle attack

Configuring SSLServerSockets

- SSLServerSocket provides methods to choose cipher suites, manage sessions and establish whether clients are required to authenticate themselves
- Ciphter suites

```
public abstract String[] getSupportedCipherSuites()
public abstract String[] getEnabledCipherSuites()
public abstract void     setEnabledCipherSuites(String[] suites)
```

Example

```
String[] supported = server.getSupportedCipherSuites();
String[] anonCipherSuitesSupported = new String[supported.length];
int numAnonCipherSuitesSupported = 0;
for (int i = 0; i < supported.length; i++) {
  if (supported[i].indexOf("_anon_") > 0) {
    anonCipherSuitesSupported[numAnonCipherSuitesSupported++]
        = supported[i];
  }
}

String[] oldEnabled = server.getEnabledCipherSuites();
String[] newEnabled = new String[oldEnabled.length
    + numAnonCipherSuitesSupported];
System.arraycopy(oldEnabled, 0, newEnabled, 0, oldEnabled.length);
System.arraycopy(anonCipherSuitesSupported, 0, newEnabled,
    oldEnabled.length, numAnonCipherSuitesSupported);

server.setEnabledCipherSuites(newEnabled);
```