Clients and Servers

- The *client* is the actor that requests to talk.
- The *server* is the actor that accepts to talk.

The client can create a socket to start a conversation to a server app anytime.
The server must be repared in aadvance to accept an incoming conversation.

Sockets

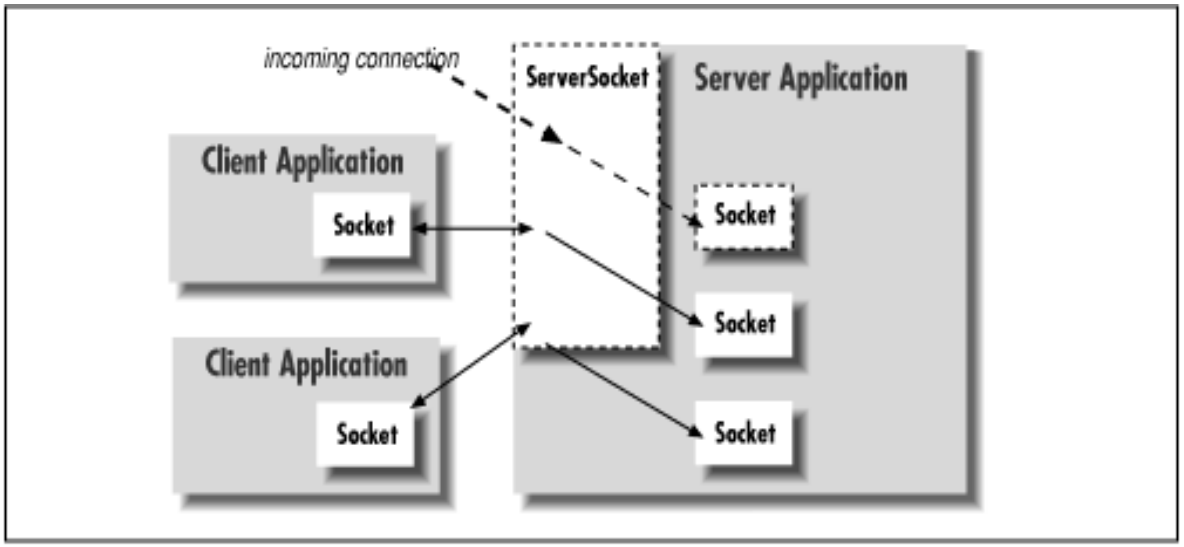
The java.net.Socket class represents a side of connection (regardless if client o or server).

The server uses the java.net.ServerSocket class to wait for incoming conversations. It creates a ServerSocket object and waits, blocked on a accept() call until a connection comes. Then it creates a Socket object to be used to communicate with the client.

Sockets

A server can maintain many conversations simoultaneously.

There is only one ServerSocket, but one Socket for every client.



Server port

The client needs two pieces of info to establish a connection: a hostname (to get the server's address) and a port number (to identify a process on the server machine).

A server app listens on a predefined port while waiting for a connection.

Port numbers are coded in the RFC (Es. Telnet 23, FTP 21, ecc.), but they can be freely chosen for custom services.

Client port

The client's port number is generally assigned by the OS, and in general you do not care about it.

When the server responds it opens a new socket whose number is assigned by the OS. It then continues listening on the original port, and serves the particular cliens on the new socket.

Sockets

The first choice is which protocol to use:

connection-oriented (TCP)

or

connectionless (UDP).

The Java Socket class uses TCP

java.net.Socket

This class implements a socket for interprocess communication over the network.

The constructor methods create the socket and connect it to the specified host on the specified port.

java.net.Socket - main methods

The constructor methods create the socket and connect it to specified host and port.

Once the socket is created, getInputStream() e getOutputStream() return InputStream e OutputStream objects (usable as I/O channels).

getInetAddress() e getPort() return address and port to which the socket is connected.

getLocalPort() returns the local port used by the socket .

close() closes la socket.

java.net.ServerSocket

During creation you specify on which port to listen

The accept() starts listening and blocks until there is an incoming call.

At that point, accept() accepts the connection, creates and returns a Socket that the server can use to talk to the client.

java.net.ServerSocket – main methods

getInetAddress() returns the local address

getLocalPort() returns the local port .

close() closes the socket.

Sockets

Clients

```
try {
        Socket sock = new Socket("www.pippo.it", 80);
        //Socket sock = new Socket("128.252.120.1", 80);
} catch ( UnknownHostException e ) {
        System.out.println("Can't find host.");
} catch ( IOException e ) {
        System.out.println("Error connecting to host.");
}
```

Connection-oriented protocol

Server

- Create a ServerSocket object.
- After accepting the connection, create a Socket che object.
- Create InputStream and OutputStream to read/write bytes from/to the connection.
- Optionally create a new thread for every connection, so that the serer can listen for new requests while serving arrived clients.

Reading & Writing raw bytes – Client side

```
try {
        Socket server = new Socket("foo.bar.com", 1234);
        InputStream in = server.getInputStream();
        OutputStream out = server.getOutputStream();
        // Write a byte
        out.write(42);
        // Read a byte
        Byte back = in.read();
        server.close();
} catch (IOException e ) { }
```

Reading & Writing raw bytes – Server side

```
try {
        ServerSocket listener = new ServerSocket( 1234 );
        while ( !finished ) {
                Socket aClient = listener.accept();
                // wait for connection
                InputStream in = aClient.getInputStream();
                OutputStream out = aClient.getOutputStream();
                // Read a byte
                Byte importantByte = in.read();
                // Write a byte
                out.write(43);
                aClient.close();
        }
        listener.close();
} catch (IOException e ) { }
```

Reading & Writing newline delimited strings – Client

Incapsulating InputStream and OutputStream it is possible to access streams in an easier way.

```
try {
        Socket server = new Socket("foo.bar.com", 1234);
        InputStream in = server.getInputStream();
        DataInputStream din = new DataInputStream( in );

        OutputStream out = server.getOutputStream();
        PrintStream pout = new PrintStream( out );

        // Say "Hello" (send newline delimited string)
        pout.println("Hello!");
        // Read a newline delimited string
        String response = din.readLine();
        server.close();
} catch (IOException e ) { }
```

Reading & Writing newline delimited strings – Server

```
try {
        ServerSocket listener = new ServerSocket( 1234 );
        while ( !finished ) {
                Socket aClient = listener.accept();
                // wait for connection
                InputStream in = aClient.getInputStream();
                DataInputStream din = new DataInputStream( in );
                OutputStream out = aClient.getOutputStream();
                PrintStream pout = new PrintStream( out );
                // Read a string
                String request = din.readLine();
                // Say "Goodbye"
                pout.println("Goodbye!");
                aClient.close();
        }
        listener.close();
} catch (IOException e ) { }
```

A concurrent HTTP mini-server - Introduction

TinyHttpd listens on a specified port and services simple HTTP "get file" requests. They look something like this:

```
GET /path/filename [optional stuff]
```

Your Web browser sends one or more as lines for each document it retrieves. Upon reading the request, the server tries to open the specified file and send its contents. If that document contains references to images or other items to be displayed inline, the browser continues with additional GET requests. For best performance (especially in a time-slicing environment), TinyHttpd services each request in its own thread. Therefore, TinyHttpd can service several requests concurrently.

A concurrent HTTP mini-server

```
package tinyhttpd;

import java.net.*;
import java.io.*;


public class TinyHttpd {
   public static void main( String argv[] )
    throws IOException {
       int port = 8000;
        if (argv.length>0) port=Integer.parseInt(argv[0]);
       ServerSocket ss = new ServerSocket(port);
        System.out.println("Server is ready");
       while ( true )
              new TinyHttpdConnection(ss.accept() );
   }
}
```

A concurrent HTTP mini-server

```
class TinyHttpdConnection extends Thread {

    Socket sock;

    TinyHttpdConnection(Socket s) {
        sock = s;
        setPriority(NORM_PRIORITY - 1);
        start();
    }

    public void run() {
        System.out.println("=========");
        OutputStream out = null;
        try {
            out = sock.getOutputStream();
            BufferedReader d =
              new BufferedReader(new InputStreamReader(
                    sock.getInputStream()));
            String req = d.readLine();;
            System.out.println("Request: " + req);
            StringTokenizer st = new StringTokenizer(req);
```

A concurrent HTTP mini-server - Note

. By lowering its priority to NORM_PRIORITY-1 (just below the default priority), we ensure that the threads servicing established connections won't block TinyHttpd's main thread from accepting new requests.

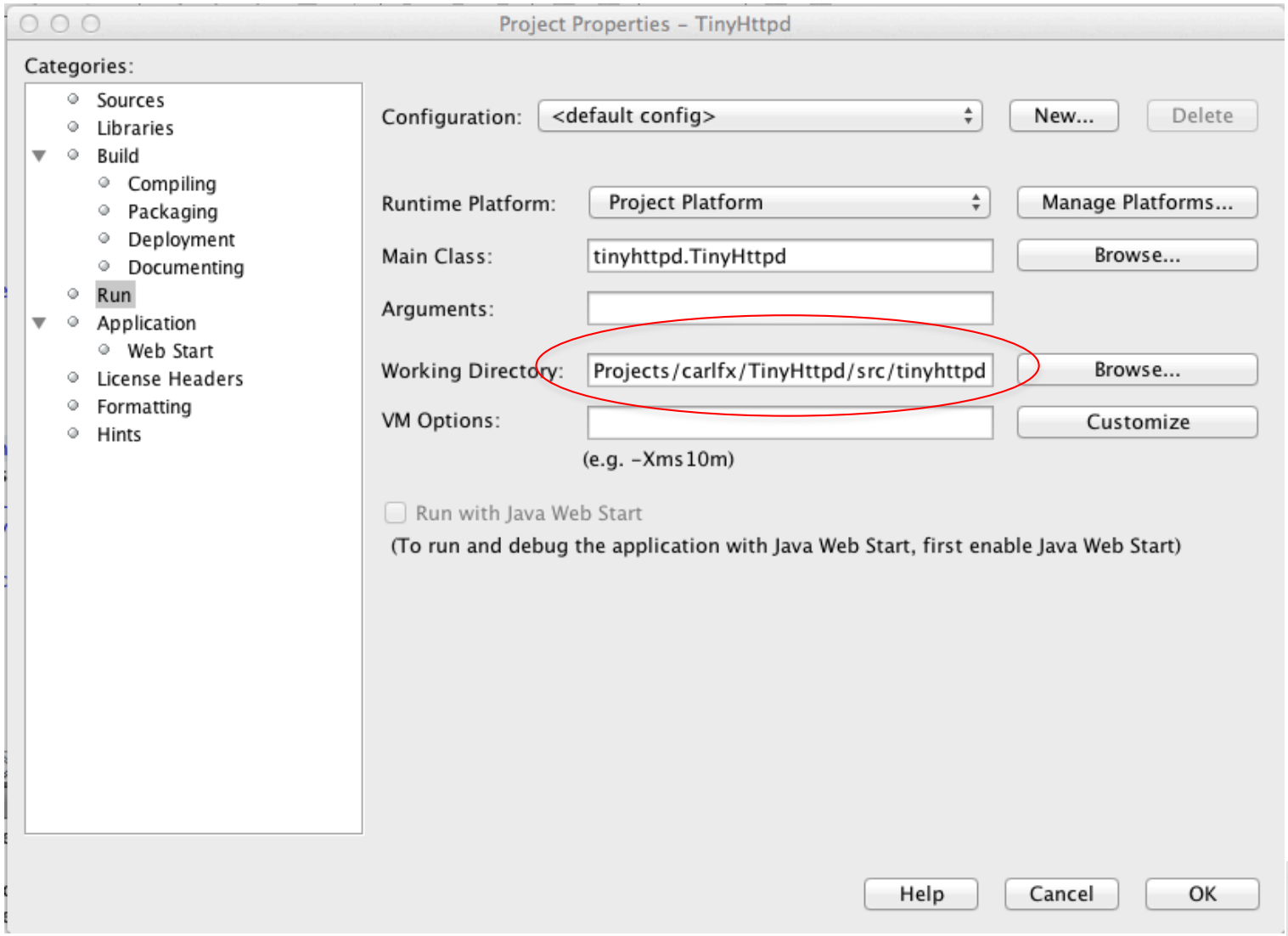
(On a time-slicing system, this is less important.)

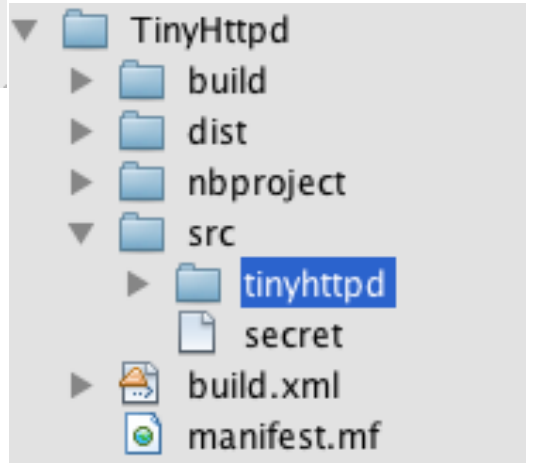
Un mini-server concorrente HTTP

```
if ((st.countTokens() >= 2) && st.nextToken().equals("GET")) {
    if ((req = st.nextToken()).startsWith("/")) {
       req = req.substring(1);
    }
    if (req.endsWith("/") || req.equals("")) {
       req = req + "index.html";
    }
    try {
         FileInputStream fis = new FileInputStream(req);
         byte[] data = new byte[fis.available()];
         fis.read(data);
         out.write(data);
    } catch (FileNotFoundException e) {
         new PrintStream(out).println("404 Not Found");
         System.out.println("404 Not Found: " + req);
    }
} else {
    new PrintStream(out).println("400 Bad Request");
    System.out.println("400 Bad Request: " + req);
    sock.close();
}
```

Un mini-server concorrente HTTP

```
} catch (IOException e) {
            System.out.println("Generic I/O error " + e);
        } finally {
            try {
                out.close();
            } catch (IOException ex) {
                System.out.println("I/O error on close" + ex);
            }
        }
    }
}
```


Project Properties - TinyHttpd
Categories:
Sources
Libraries
Build
Compiling
Packaging
Deployment
Documenting
Run
Application
Web Start
License Headers
Formatting
Hints
Configuration:
<default config>
New...
Delete
Runtime Platform:
Project Platform
Manage Platforms...
Main Class:
tinyhttpd.TinyHttpd
Browse...
Arguments:
Working Directory:
Projects/carlfx/TinyHttpd/src/tinyhttpd
Browse...
VM Options:
Customize
(e.g. -Xms10m)
Run with Java Web Start
(To run and debug the application with Java Web Start, first enable Java Web Start)
Help
Cancel
OK


TinyHttpd
build
dist
nbproject
src
tinyhttpd
secret
build.xml
manifest.mf

A concurrent HTTP mini-server - usage

Compile TinyHttpd and place it in your class path. Go to a directory with some interesting documents and start the daemon, specifying an unused port number as an argument. For example:

```
% java TinyHttpd 1234
```

You should now be able to use your Web browser to retrieve files from your host. You'll have to specify the nonstandard port number in the URL. For example, if your hostname is foo.bar.com, and you started the server as above, you could reference a file as in:

```
http://foo.bar.com:1234/welcome.html
```

A concurrent HTTP mini-server - Problems

TinyHttpd still has room for improvement. First, it consumes a lot of memory by allocating a huge array to read the entire contents of the file all at once. A more realistic implementation would use a buffer and send large amounts of data in several passes.

TinyHttpd also fails to deal with simple things like directories. It wouldn't be hard to add a few lines of code to read a directory and generate linked HTML listings like most Web servers do.

A concurrent HTTP mini-server - Problems

TinyHttpd suffers from the limitations imposed by ṭhe fickleness of filesystem access.

It's important to remember that file pathnames are still architecture dependent--as is the concept of a filesystem to begin with. TinyHttpd should work, as is, on UNIX and DOS-like systems, but may require some customizations to account for differences on other platforms. It's possible to write more elaborate code that uses the environmental information provided by Java to tailor itself to the local system.

A concurrent HTTP mini-server - Problems

The biggest problem with TinyHttpd is that there are no restrictions on the files it can access. With a little trickery, the daemon will happily send any file in your filesystem to the client.

It would be nice if we could restrict TinyHttpd to files that are in the current directory, or a subdirectory.

Assignment

Modify the simple web server so that all the urls that start with the token "process "
(e.g. http://localhost:8000/process)
launch an external process.

For instance,
http://localhost:8000/process/reverse?par1=*string*&par2=*booleanvalue*
should activate an (external) process that takes the par1 string.
If par2 is true, it returns the reversed string (e.g. ROMA -> AMOR).
If par2 is false, it checks if the string is a palindrome, and returns the answer (true or false). (e.g. ROOR -> true, ROAR –> false)

To see how to start an external process from Java, take a look at one of these:

- http://www.rgagnon.com/javadetails/java-0014.html
- https://www.mkyong.com/java/how-to-execute-shell-command-from-java/
- https://www.baeldung.com/run-shell-command-in-java

Deadline Sept. 29, 2019, 23:59

SEE WEB SITE: latemar.science.unitn.it

HTTPS Overview

https is a URI scheme which is syntactically identical to the http: scheme normally used for accessing resources using HTTP. Using an https: URL indicates that HTTP is to be used, but with a different default port (443) and an additional encryption/authentication layer between HTTP and TCP.

This system was developed by Netscape Communications Corporation to provide authentication and encrypted communication and is widely used on the World Wide Web for security-sensitive communication, such as payment transactions.

S-HTTP Overview

Secure hypertext transfer protocol' (S-HTTP) is an alternative mechanism to the https URI scheme for encrypting web communications carried over HTTP. S-HTTP is defined in RFC 2660.

Web browsers typically use HTTP to communicate with web servers, sending and receiving information without encrypting it. For sensitive transactions, such as Internet e-commerce or online access to financial accounts, the browser and server must encrypt this information.

The https: URI scheme and S-HTTP were both defined in the mid 1990s to address this need. Netscape and Microsoft supported HTTPS rather than S-HTTP, leading to HTTPS becoming the de facto standard mechanism for securing web communications. S-HTTP is an alternative mechanism that is not widely used.

HTTPS

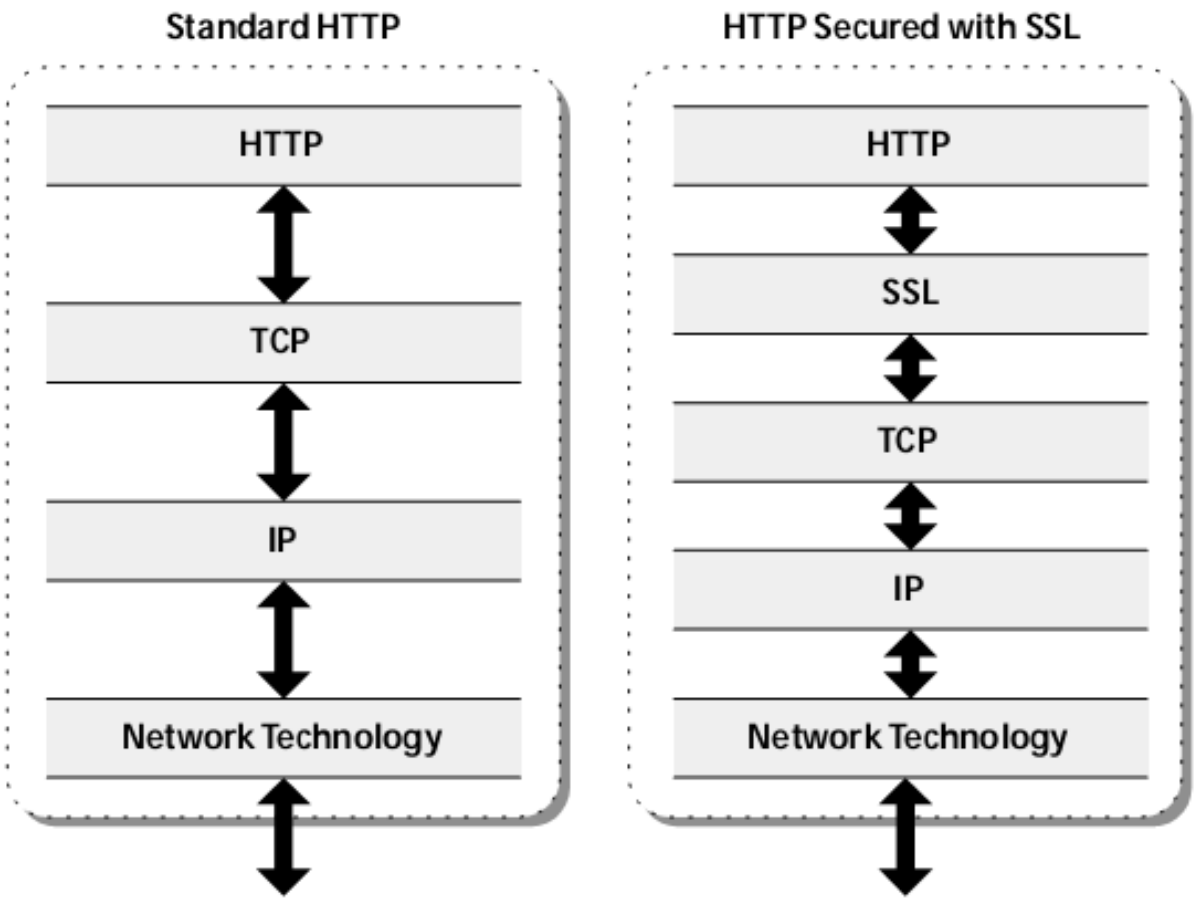
=

HTTP + SSL

Slides from HTTP vs. HTTPS by Eng. T. Aldaldooh

HTTPS

Figure 4.10 ▶
The SSL protocol inserts itself between an application like HTTP and the TCP transport layer. TCP sees SSL as just another application, and HTTP communicates with SSL much the same as it does with TCP.



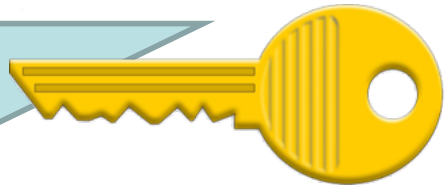
Cryptography

Important information Data, Data, Data.

Plain Text

Encryption

Encryption Algorithm = cipher



Some random String

Hh2sh!~hH==E#@ns8676%===sdf

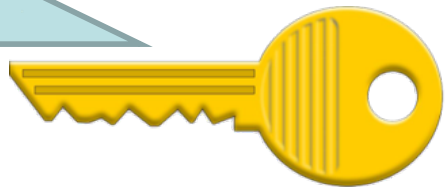
Cipher Text

Cryptography cont.

Symmetric Key

Important information Data, Data, Data.

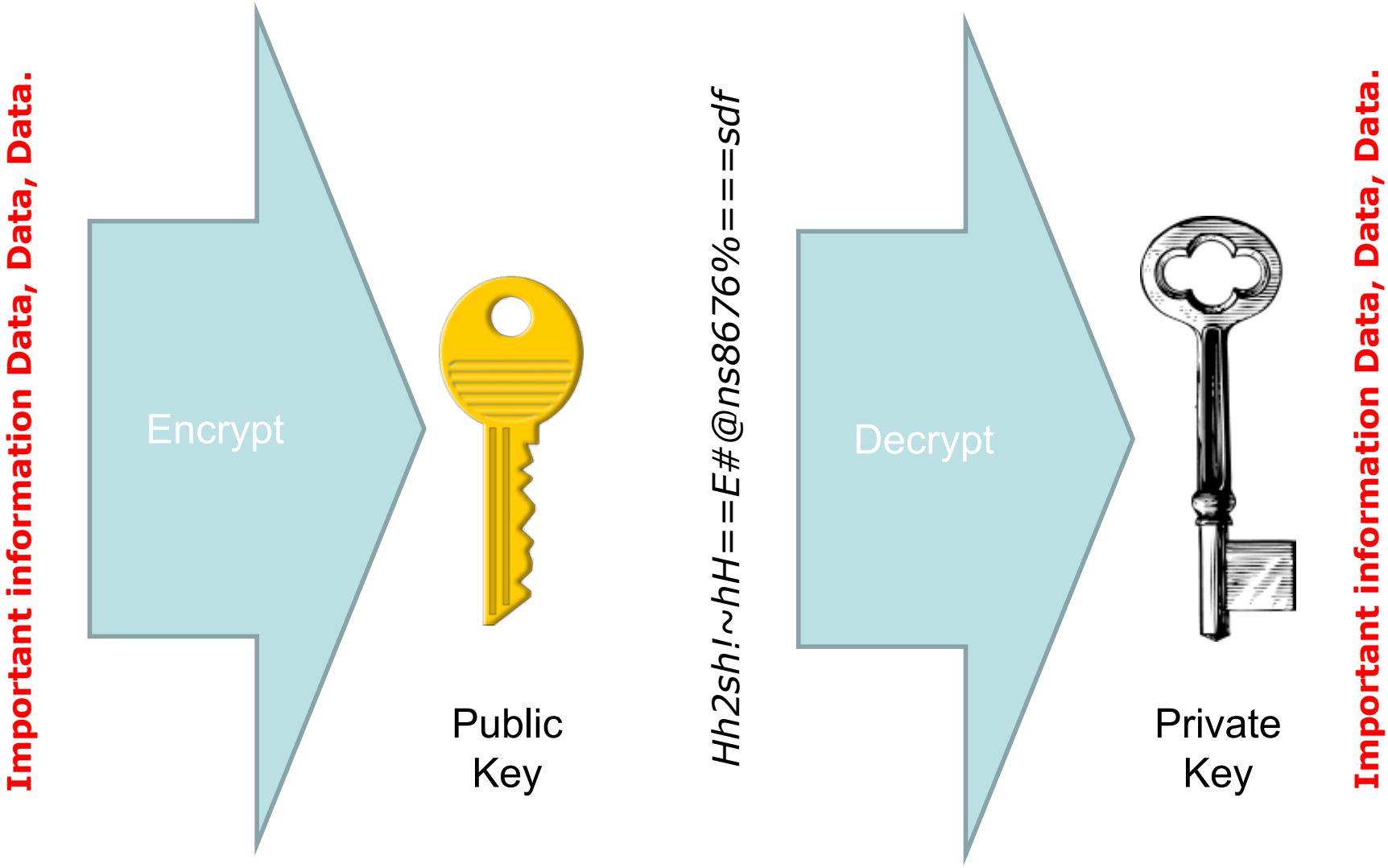
Decryption Algorithm



Some random String

Hh2sh!~hH==E#@ns8676%===sdf

Asymmetric (public-key) encryption



SSL Session

- Uses asymmetric encryption to privately share the session key
  - Asymmetric has a lot of overhead
- Uses symmetric encryption to encrypt data
  - Symmetric encryption is quicker and uses less resource

SSL Handshake Process

Client requests HTTPS session

Certificate sent back (with public key)

Client creates session key (**53**)

Session key encrypted with public key(**X$qp0**)

At this point only client knows session key

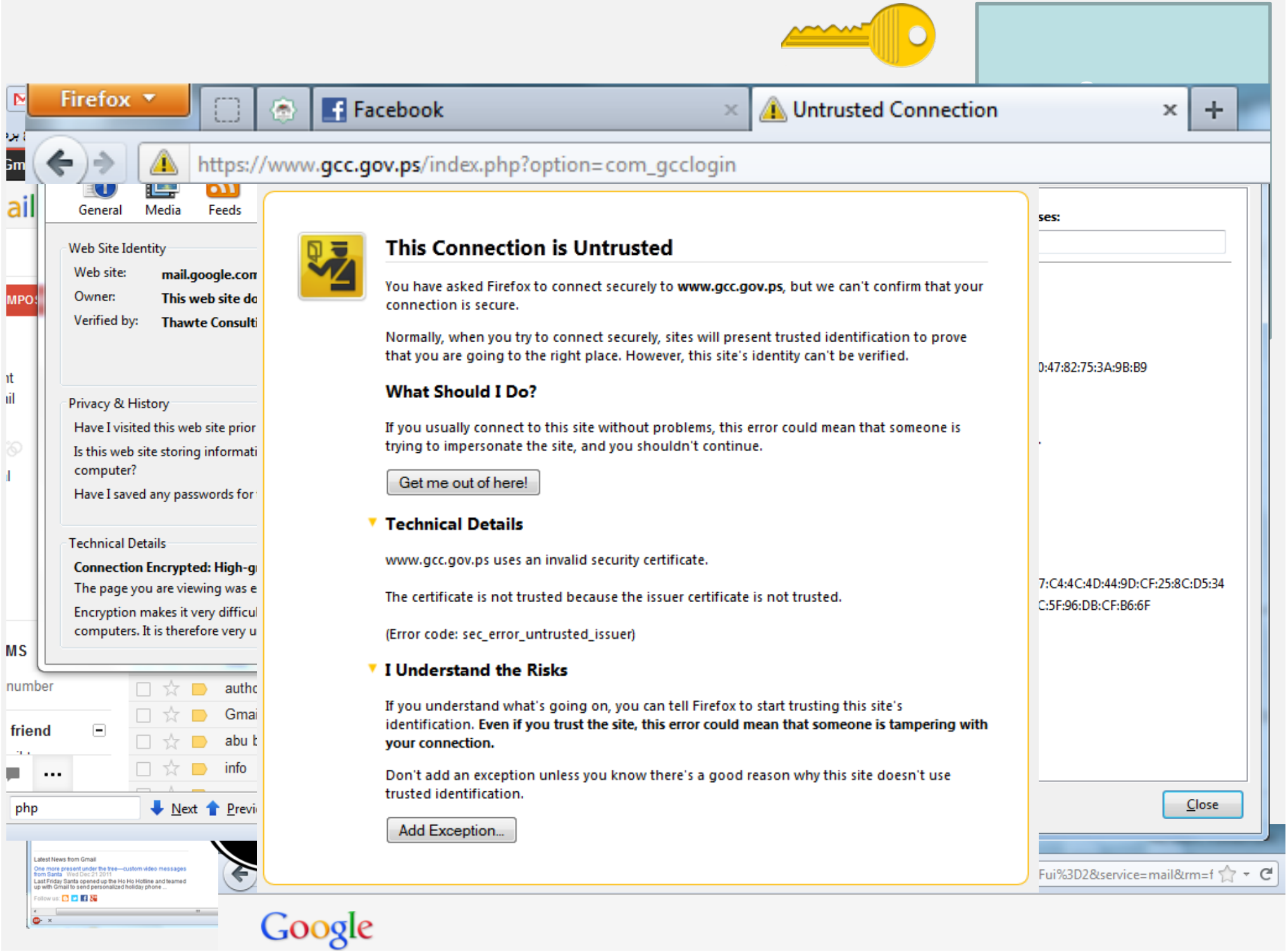
Encrypted session key sent to server

session key decrypted with private key

At this point both client and server knows session key

Session encrypted with

symmetric session key (53)


Firefox
Facebook
Untrusted Connection
https://www.gcc.gov.ps/index.php?option=com_gcclogin
General
Media
Feeds
Web Site Identity
Web site:
mail.google.com
Owner:
This web site do
Verified by:
Thawte Consulti
Privacy & History
Have I visited this web site prior
Is this web site storing informati
computer?
Have I saved any passwords for
Technical Details
Connection Encrypted: High-g
The page you are viewing was e
Encryption makes it very difficul
computers. It is therefore very u
This Connection is Untrusted
You have asked Firefox to connect securely to www.gcc.gov.ps, but we can't confirm that your connection is secure.
Normally, when you try to connect securely, sites will present trusted identification to prove that you are going to the right place. However, this site's identity can't be verified.
What Should I Do?
If you usually connect to this site without problems, this error could mean that someone is trying to impersonate the site, and you shouldn't continue.
Get me out of here!
Technical Details
www.gcc.gov.ps uses an invalid security certificate.
The certificate is not trusted because the issuer certificate is not trusted.
(Error code: sec_error_untrusted_issuer)
I Understand the Risks
If you understand what's going on, you can tell Firefox to start trusting this site's identification. Even if you trust the site, this error could mean that someone is tampering with your connection.
Don't add an exception unless you know there's a good reason why this site doesn't use trusted identification.
Add Exception...
ses:
0:47:82:75:3A:9B:B9
7:C4:4C:4D:44:9D:CF:25:8C:D5:34
C:5F:96:DB:CF:B6:6F
Close
friend
php
Next
Google

Man-in-the-Middle (MITM) Attack Concept

- There were away to get around the encryption instead o0f trying to break it

E{a,b,c} = Ali's, Ahmed's, and Man's public keys, respectively

- Ali wants to send secure messages to Ahmed.
- Man intercepts Ali's messages.
- Man talks to Ali and pretends to be Ahmed.
- Man talks to Ahmed and pretends to be Ali.

MITM Attack Concept

- Ali uses the *public key* she thinks she received from Ahmed (Man's)
- Ahmed uses the key he thinks is Ali's (also Man's)
- As a result, Man not only gains *access* to secure information but also can *modify* it (e.g. *transfer money to a different account* etc.)

MITM and Certificates

- Digital Certificates designed to solve the problem but do they always help ?
- The MITM would have to create his own certificate with a private/public key.
- He still sit between client and server, acting as server to the client and client to the server, listening in on everything sent between the two.

The solution “chain of trust”

- To verify the authenticity and identity of the certificates themselves.
- linked back to a trustworthy source of certificates.
- Web browsers and operating systems will only trust certificates that directly or indirectly link back to one of a handful of CAs, the "root CAs.“
- Any certificate that doesn't link back to a root CA such as a self-signed certificate will generate a big scary warning in the browser.
    - **How to create a self-signed SSL Certificate ...**
    - http://www.akadia.com/services/ssh_test_certificate.html

Conclusion

- HTTPS only slightly slower than HTTP.

Reference

- HTTP Essentials Protocols for Secure, Scaleable Web Sites by Stephen Thomas .
- HTTP The Definitive Guide.
- View HTTP Request and Response Header < http://web-sniffer.net/ >