

## Pràctica 6: Seguretat en HTTP

Pere Joan Martorell  
Bernat Farrero

### Introducció

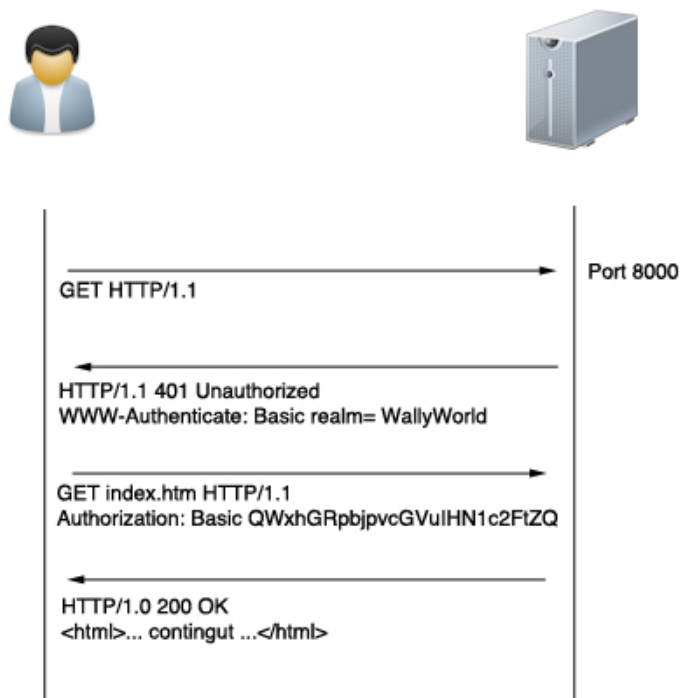
En aquesta pràctica s'analitzaran mètodes segurs d'autenticació de l'usuari. Així com la comunicació segura via HTTP.

Veurem autenticació bàsica mitjançant usuari i contrasenya codificades en base64, també com funciona la certificació de servidor, i finalment també la identificació d'usuari mitjançant certificats personals.

Per realitzar la pràctica partim del codi descarregat de la web de l'assignatura que conté l'implementació d'un servidor simple i la seva extensió a servidor segur (que usa SSL), així com un navegador simple i segur.

### Autenticació bàsica d'usuari

Vegem el procediment estàndard de comunicació que es produirà amb el nostre sistema.



L'usuari intenta accedir a un recurs, però el servidor li barra el pas responent que no està autoritzat per entrar a aquell "regne" (realm). Quan ho rep, el navegador mostra per pantalla una finestra d'identificació (amb nom d'usuari i contrasenya). L'usuari introdueix correctament les dades, el servidor l'identifica i li mostra el contingut.

Per fer això cal modificar el servidor que ens donen i afegir la comprovació que l'usuari s'hagi identificat. Modifiquem la funció `processGetRequest()`.

```

void processGetRequest(HttpServletRequest request,BufferedOutputStream outputStream) throws IOException
{
    String fileName = request.getFileName();
    File file = new File(fileName);

    // Give them the requested file
    if(file.exists())
    {
        if(request.esAutoritzat()) sendFile(outputStream,file);
        else
        {
            outputStream.write("HTTP/1.1 401 Unauthorized \r\n".getBytes());
            outputStream.write("WWW-Authenticate: Basic realm= WallyWorld \r\n".getBytes());
            outputStream.flush();
            outputStream.close();
        }
    }
    else System.out.println("File "+file.getCanonicalPath()+" does not exist.");
}

```

Només en el cas que el fitxer existeixi i la petició sigui autoritzada es cridarà la funció `sendFile` que es dedica a llegir el fitxer i mostrar-lo per pantalla juntament amb la resposta "HTTP/1.0 200 OK" a la capçalera.

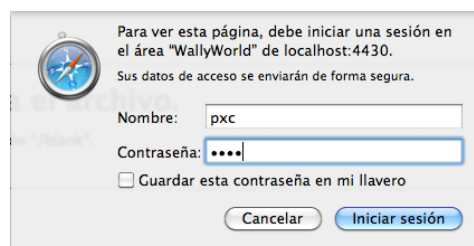
Prèviament hem creat el mètode `esAutoritzat()` de l'objecte `request`:

```

public boolean esAutoritzat()
{
    System.out.println("Resposta del client:");
    for (int i=0; i<lines.size(); i++)
    {
        System.out.println((String) lines.elementAt(i));
        String line = (String) lines.elementAt(i);
        String auth[] = line.split(" ");
        if (auth[0].equals("Authorization:"))
        {
            String id = Base64.decodeToString(auth[2]);
            String userpass[] = id.split(":");
            if (userpass[0].equals(usr) && userpass[1].equals(pwd)) return true;
        }
    }
    return false;
}

```

Com es pot veure, parsegem la informació d'autorització, ho decodifiquem de base64, llibreria de les quals també se'ns donen en el paquet de fitxers, i n'extraïem l'usuari i el password. Si corresponen amb aquells que nosaltres hem declarat com a variables globals en l'objecte `request` (petició), llavors retorna cert. En cas contrari retorna fals. El nom d'usuari i password per la nostra pràctica són respectivament `pxc` i `1234`.



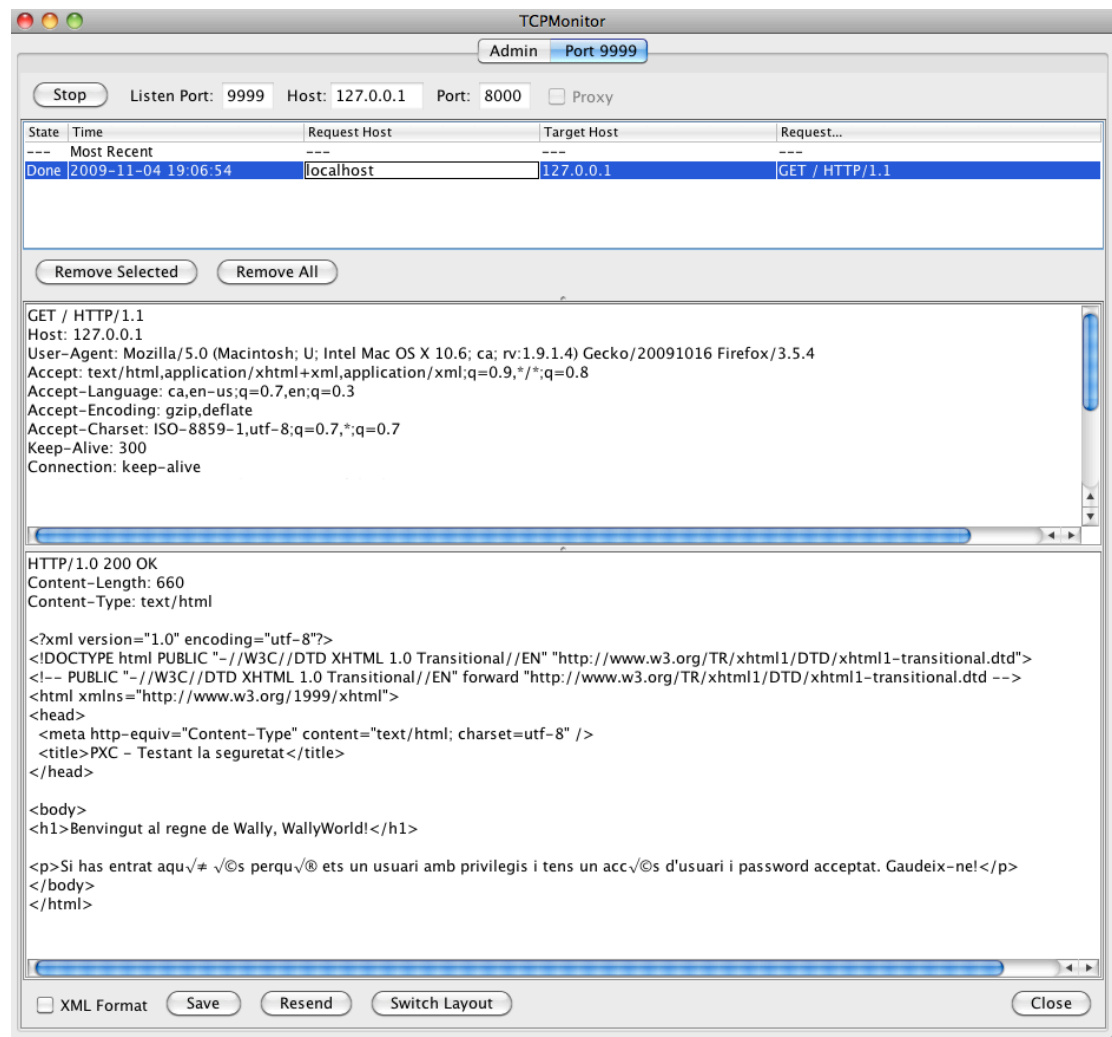
Para ver esta página, debe iniciar una sesión en el área "WallyWorld" de localhost:4430.  
 Sus datos de acceso se enviarán de forma segura.

Nombre:

Contraseña:

☐ Guardar esta contraseña en mi llavero

Usant el tcpmon, programa usat en pràctiques anteriors per escoltar les connexions, interceptem aquest intercanvi d'informació:



## Generació de certificat per el servidor

El sistema d'autenticació bàsica que hem vist no es considera un mètode segur d'autenticació a menys que no s'estigui usant conjuntament amb un sistema de xifrat de dades com SSL. La codificació en base64 es considera molt insegura pel fet que és totalment reversible.

En aquest apartat afegirem seguretat SSL al servidor. Quan el client iniciï una sessió SSL amb el servidor aquest últim li oferirà el seu certificat per garantir que la connexió es fa de forma segura.

Crearem un certificat pel servidor que emmagatzemarem en un *Keystore*. Els *keystores* actuen com una base de dades de parells de claus públiques i privades i de certificats d'autenticació. Per crear el certificat hem fet servir la comanda *keytool* introduït la informació necessària pel certificat:

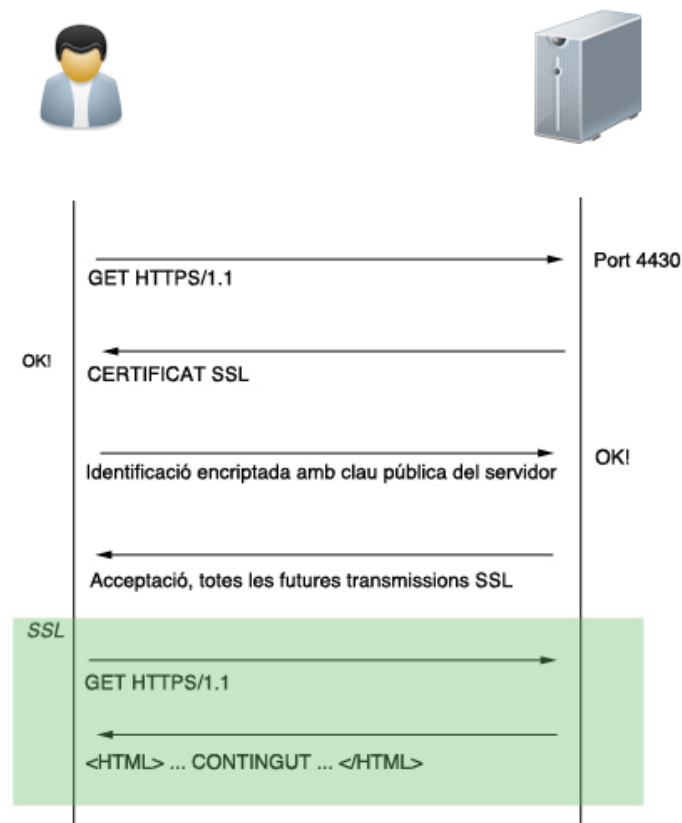
La comanda genera un certificat que queda emmagatzemat al *keystore* "certs"

amb la contrasenya “serverkspw” que ens dóna pas tant al keystore com a la clau privada de l’alias SecureServer.

```
keytool -genkey -alias SecureServer -keyalg RSA -keypass serverkspw -storepass serverkspw -keystore certs
```

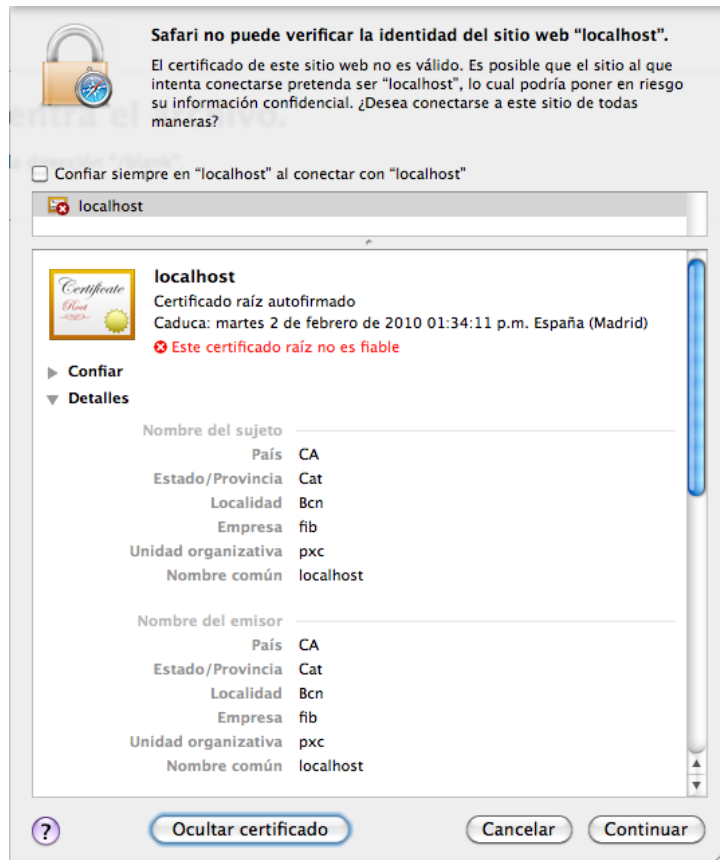
```
What is your first and last name?  
[Unknown]: localhost  
What is the name of your organizational unit?  
[Unknown]: pxc  
What is the name of your organization?  
[Unknown]: fib  
What is the name of your City or Locality?  
[Unknown]: Barcelona  
What is the name of your State or Province?  
[Unknown]: Catalunya  
What is the two-letter country code for this unit?  
[Unknown]: CA  
Is CN=localhost, OU=pxc, O=fib, L=Barcelona, ST=Catalunya, C=CA  
correct?  
[no]: yes
```

El funcionament simplificat del *handshaking* del protocol SSL és el següent:



1. El client fa una petició de connexió segura al port 4430
2. El servidor respon amb el certificat extret del seu *keystore*

3. El client rep el certificat del servidor. En el nostre cas, com que el certificat no està firmat per una *Certificate Authority* vàlida sinó que està auto-signat, el navegador ens demanarà si realment volem confiar-hi.



4. Després d'acceptar i validar el certificat, el navegador ja pot accedir al contingut HTML. En el cas que tinguem l'autenticació bàsica se'ns mostrarà la finestra emergent d'abans demanant-nos l'usuari contrasenya.

### SecureBrowser: validar el certificat del servidor

Ara tenim *SecureBrowser*, un navegador textual en Java que ens permetrà establir connexions SSL. Si provem d'executar el *SecureBrowser* contra el *SecureServer* ens adonarem que no pot validar el certificat del servidor.

Per tal que pugui validar-lo haurem de crear un *TrustStore*, és a dir, una base de dades on s'emmagatzemen els certificats en els que confia el client i que ens permetrà identificar el servidor.

Primer de tot exportarem el certificat del servidor del keystore al fitxer `server.cer` amb la comanda:

```
keytool -export -alias SecureServer -storepass serverkspw -file
server.cer -keystore certs

>> Certificate stored in file <server.cer>
```

Seguidament creem el TrustStore anomenat cacerts.jks amb la comanda keytool i que ens mostra la sortida que continua:

```
keytool -import -v -trustcacerts -alias SecureServer -file
server.cer -keystore cacerts.jks -keypass serverkspw -storepass
serverkspw

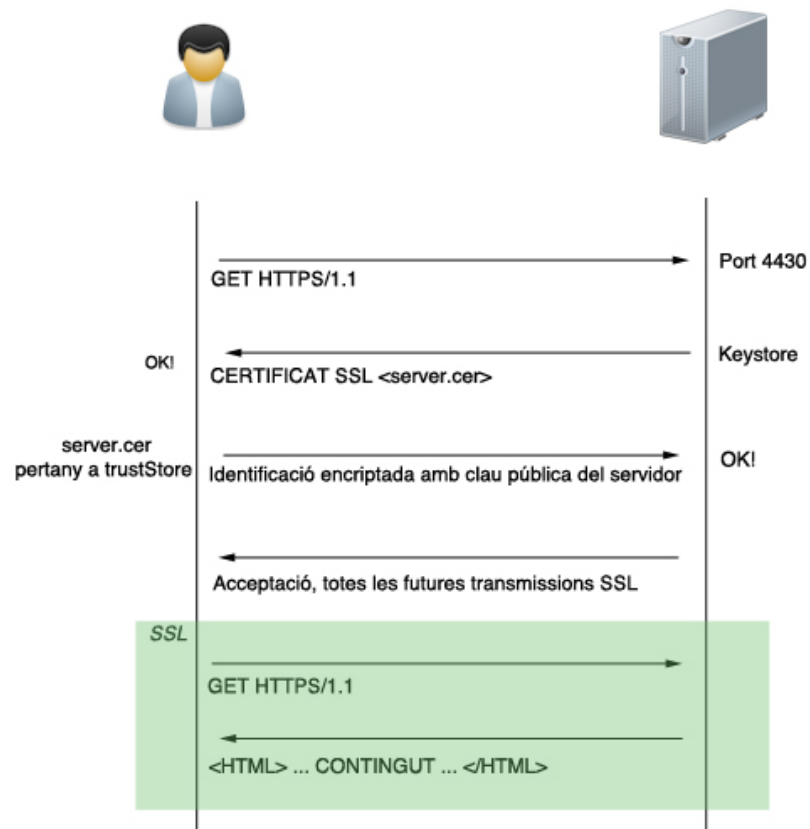
    Owner: CN=localhost, OU=pxc, O=fib, L=Barcelona,
ST=Catalunya, C=CA
    Issuer: CN=localhost, OU=pxc, O=fib, L=Barcelona,
ST=Catalunya, C=CA
    Serial number: 4af2f763
    Valid from: Thu Nov 05 17:03:47 CET 2009 until: Wed Feb 03
17:03:47 CET 2010
    Certificate fingerprints:
        MD5:  9E:07:5D:27:FD:BE:B4:5E:E5:81:54:29:0E:AD:AC:0C
        SHA1:
81:60:43:F5:A6:CF:F1:48:DC:B0:33:29:82:4A:2A:D9:12:A9:05:C5
        Signature algorithm name: SHA1withRSA
        Version: 3
    Trust this certificate? [no]: yes
    Certificate was added to keystore
    [Storing cacerts.jks]
```

Ara al SecureBrowser li haurem de definir com a truststore el fitxer cacerts.jks, a partir del qual podrà validar el certificat del servidor. Per fer això haurem d'afegir el següent codi dins del mètode constructor de SecureBrowser.java:

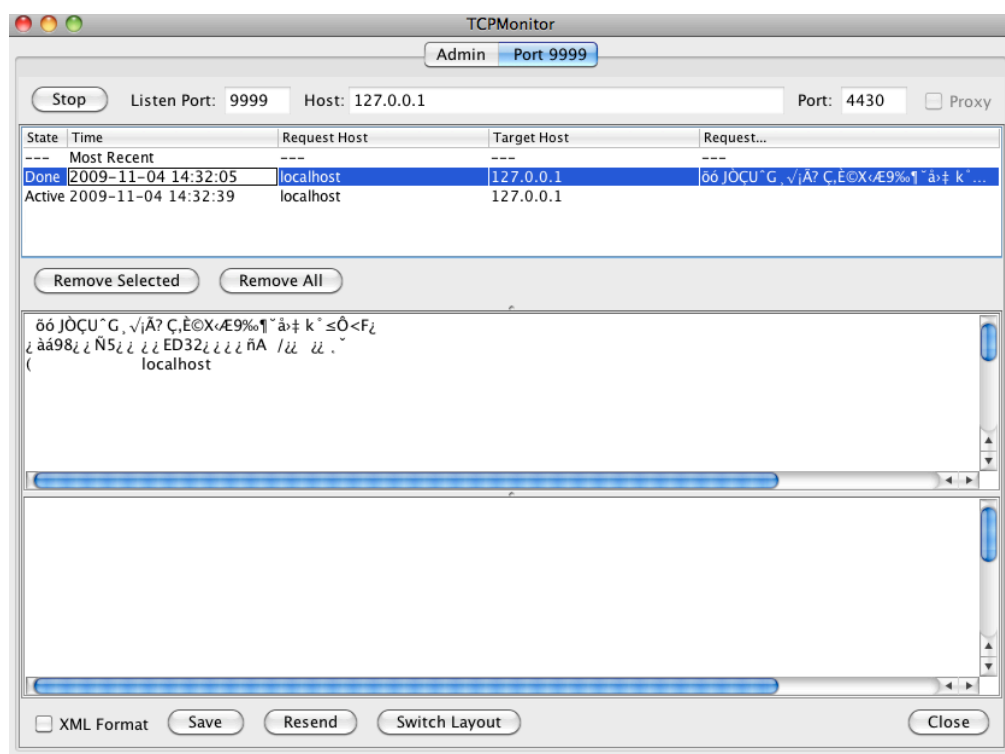
```
System.setProperty("javax.net.ssl.trustStore", "cacerts.jks");
System.setProperty("javax.net.ssl.trustStorePassword", "serverkspw");
```

Per fer-ho, usem el mètode setProperty de System que permet definir variables de Sistema.

Ara l'esquema de comunicació seria el següent:



Per tal de comprovar que la informació realment viatja encriptada hem fet servir el *tcpmon*. Es pot veure clarament que la petició del client està xifrada.



## Autenticació del client (Autenticació mútua)

En el cas anterior, sorgeix el problema que SecureServer no és capaç d'identificar la veracitat del SecureBrowser. D'aquí, la necessitat de crear un certificat també pel client que serà enviat al servidor.

Les comandes següents mostren el procés de creació del certificat del client (client.cer) i el seu *keystore* (clientcerts) i *truststore* (clientcerts.jks):

```
keytool -genkey -alias SecureBrowser -keyalg RSA -keypass
serverkspw -storepass serverkspw -keystore clientcerts
What is your first and last name?
[Unknown]: localhost
What is the name of your organizational unit?
[Unknown]: pxc
What is the name of your organization?
[Unknown]: fib
What is the name of your City or Locality?
[Unknown]: Tarragona
What is the name of your State or Province?
[Unknown]: Catalunya
What is the two-letter country code for this unit?
[Unknown]: CA
Is CN=localhost, OU=pxc, O=fib, L=Tarragona, ST=Catalunya,
C=CA correct?
[no]: yes

keytool -export -alias SecureBrowser -storepass serverkspw -file
client.cer -keystore clientcerts
Certificate stored in file <client.cer>

keytool -import -v -trustcacerts -alias SecureBrowser -file
client.cer -keystore clientcerts.jks -keypass serverkspw -
storepass serverkspw
Owner: CN=localhost, OU=pxc, O=fib, L=Tarragona,
ST=Catalunya, C=CA
Issuer: CN=localhost, OU=pxc, O=fib, L=Tarragona,
ST=Catalunya, C=CA
Serial number: 4af2ff7c
Valid from: Thu Nov 05 17:38:20 CET 2009 until: Wed Feb 03
17:38:20 CET 2010
Certificate fingerprints:
MD5: 40:3C:86:FA:74:9E:DD:5E:4F:42:8F:12:52:39:3A:F2
SHA1:
F4:59:98:1B:1B:44:58:EA:3C:84:BA:E7:77:95:FE:03:51:53:05:77
Signature algorithm name: SHA1withRSA
Version: 3
Trust this certificate? [no]: yes
Certificate was added to keystore
[Storing cacerts.jks]
```

Per dir-li al servidor que identifiqui al client activem un flag al seu creador i li indiquem el seu TrustStore i el password:



```

public SecureServer()
{
    this("SecureServer", "1.0", 4430, true);
    System.setProperty("javax.net.ssl.trustStore", "clientcerts.jks");
    System.setProperty("javax.net.ssl.trustStorePassword", "serverkspw");
}

```

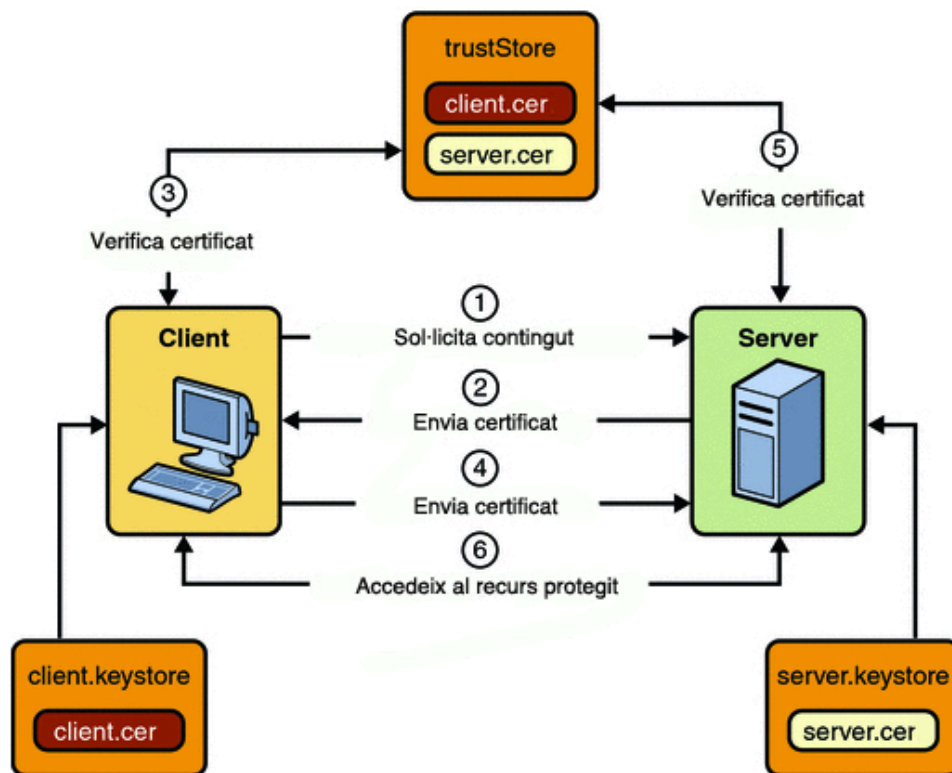
Al client, en canvi, li afegim el seu keystore (clientcerts), dins del qual hi ha el certificat del client que s'enviarà al servidor.

```

System.setProperty("javax.net.ssl.keyStore", "clientcerts");
System.setProperty("javax.net.ssl.keyStorePassword", "serverkspw");
System.setProperty("javax.net.ssl.trustStore", "cacerts.jks");
System.setProperty("javax.net.ssl.trustStorePassword", "serverkspw");

```

En aquest cas, el funcionament d'identificació mútua es farà de la forma següent:



Tal com s'esperava, després de provar d'accedir a la pàgina (index.htm) amb la comanda:

```
java SecureBrowser https://localhost:4430
```

Veiem que l'identificació mútua es produeix correctament i finalment se'ns mostra el contingut html.

## Consideracions

Per simplificar, i vist que és un exemple de proves, la contrasenya és la mateixa a tot arreu de la pràctica, *serverkspw*. Per l'exemple fet, els certificats són autosignats, cosa que fa saltar alertes als navegadors. En casos reals, el signatari és una Certification Authority (VeriSign, EnTrust, etc.) externa que garanteix unívocament la veracitat dels certificats.