Arquitecturas Cliente/Servidor

Implantación de servidores con criptografía y código seguro

Implantación de servidores con criptografía y código seguro

- **Contenido:**
  - **5.1 Servidores con criptografía.**
  - **5.2 Clientes con criptografía.**
  - **5.3 Servidores y clientes implantando código seguro.**

Introducción. Niveles de Comunicación Cliente/Servidor Seguros

- El proceso de identificación del usuario constituye una parte fundamental en la creación de un sistema seguro.

- La forma más sencilla de seguridad es la **autenticación** que simplemente identifica al **cliente**. Constituye la primera puerta que tiene que pasar un usuario a través de un sistema seguro.

- El procedimiento de entrada o conexión sólo comprueba el nombre de usuario y contraseña.

- El siguiente nivel de seguridad, la **autorización**, que permite o deniega acceso a los servicios del sistema.

- La **certificación** ofrece mayor nivel de seguridad con respecto a la autenticación. Requiere una tercera parte de confianza que permita comprobar tanto la identidad del servidor como del cliente.

Intercambio de datos

- El intercambio de datos tiene dos problemas fundamentales: **escrutinio e intrusión.**

- Estos problemas pueden darse en un intercambio público de información, en donde cualquier computadora en la misma conexión LAN física puedan ver los mensajes intercambiados

Formas de ataque

Conocer las formas de ataque habituales, puede ayudar a proteger la comunicación cliente/servidor:

- Intervención de la línea
  - Un espía escucha los mensajes esperando recibir algunos datos importantes, p. e. Sniffers
- Corte de la línea
  - Un programa malicioso limita el acceso al servidor o cliente, representa retraso o incluso puede eliminar totalmente la comunicación, p. e. Ataque con ping y DoS (Denial-of-Service)

- Secuestro de la línea
  - Un programa podría obtener el control de la comunicación cliente/servidor alterando el contenido, la pila de algunos protocolos.

Pueden existir otras formas de intrusión de seguridad, pero todas se ajustan de alguna forma a alguna de las tres mencionadas

Cifrado de mensajes

- La idea principal de cifrado es asegurar que solo los receptores entenderán el mensaje.
- En la actualidad los algoritmos de encriptación son tan habituales como las computadoras
- Existe dos tipos principales de cifrado que utilizan las computadoras interconectadas:
  - Cifrado de clave pública
  - Cifrado de clave simétrica

Tipos de cifrados

- El cifrado de clave pública o **asimétrico** utilizan dos claves, una de cifrado (pública) y otro de descifrado(privada). A menudo un servidor utiliza este algoritmo para el proceso de cifrado o simplemente para realizar transacciones.

- El cifrado de clave **simétrica** utiliza la misma clave para cifrar y descifrar. Los dos hosts deben mantener la clave en secreto.

Algoritmos más utilizados

- Clave pública o asimétrica:
  - RSA (*Rivest, Shamir y Adlemn, nombres de los autores)*
  - DSA (*Digital Signature Algorithm*)
  - Diffie-Hellman

- Clave simétrica:
  - DES (*Data Encryption Standard)*
  - 3DES
  - AES *(Advanced Encryption Standard)*
  - RC2/RC4 *(Rivest Ciphers)*

Seguridad a nivel de Sockets (SSL)

- SSL (Socket Secure Layer) Es un protocolo que nos proporciona una canal (socket) seguro entre un cliente y un servidor.

- Tiene básicamente tres características:

  - Es una capa transparente a la aplicación
  - Permite comprobar la autenticidad del servidor (y opcionalmente del cliente).
  - Todos los datos circulan encriptados.

- SSL se puede aplicar a cualquiera de los protocolos de aplicación utilizando alguno de los siguientes enfoques:
  - Puertos separados
    - Se eligen dos puertos separados, uno para las conexiones seguras y el otro para las no seguras, p.e. HTTP en el 80 y HTTPS en el 443
  - Negociación de la conexión
    - Esta opción consiste en que un servidor instalado en el mismo puerto pueda negociar entre utilizar el protocolo tradicional sin encriptación, o encriptarlo con SSL, p.e. SMTP y POP3

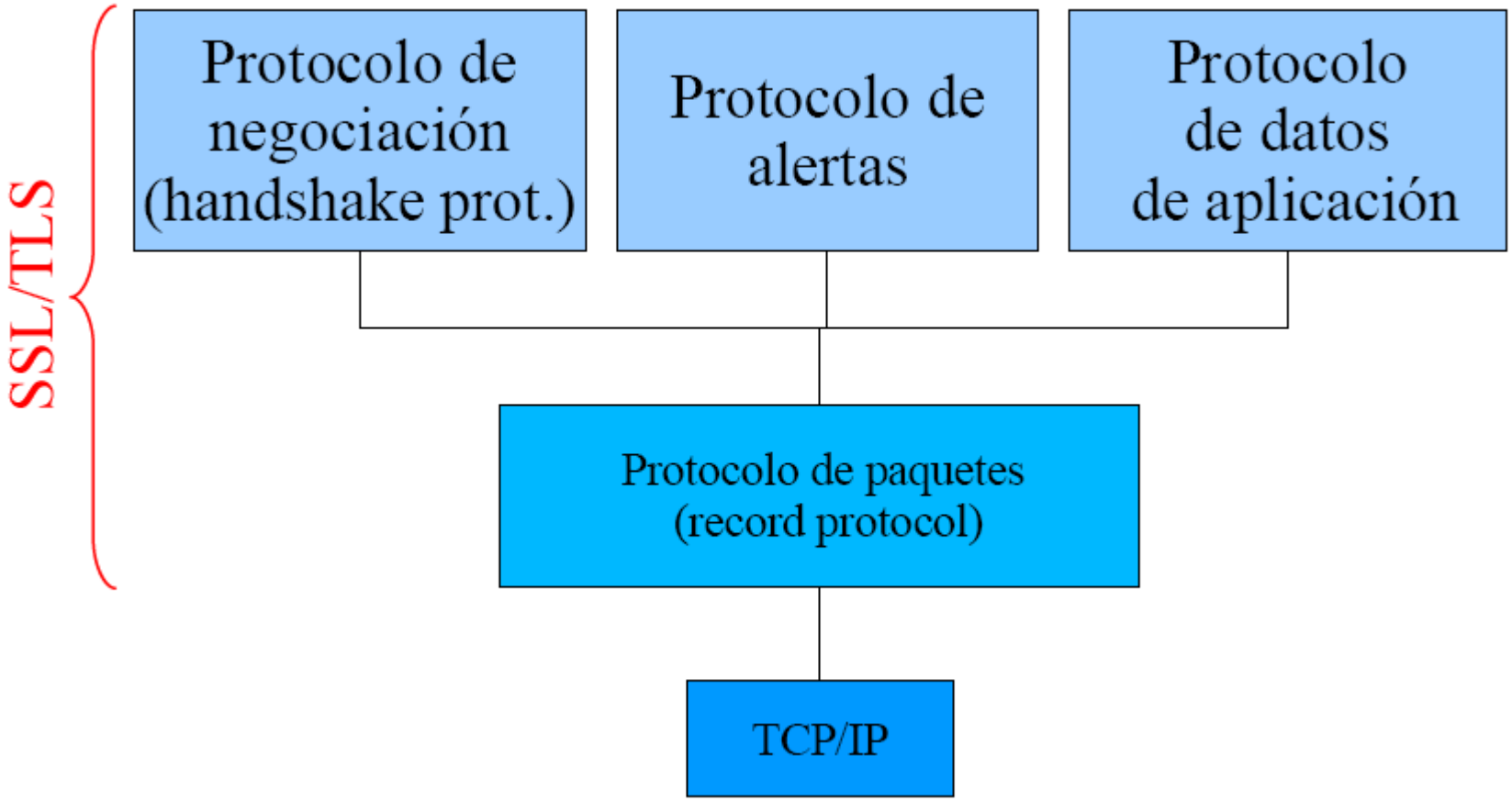
Origen SSL/TLS

- SSL desarrollado por Netscape (v2, v3)
- Estandarizado por IETF como TLS (Transport Layer Secure)
- TLS 1.0 sería SSL 3.1 (parecido a SSL 3.0 pero incompatible)
- PCT y STPL propuestas de Microsoft no aceptadas.
- Versión actual TLS 1.1 (RFC 4346)
- Comunicación segura a través de Internet
  - Confidencialidad
  - Autenticación (del servidor)
  - Integridad
- Corre debajo de los protocolos usuales
  - HTTP, FTP, POP...
-

Componentes

- Encripción simétrica
- Claves públicas para
  - Autenticación
  - Intercambio de claves
- PKI centralizado
  - Autoridades de certificación
  - Certificados X.509
-
- Algoritmos negociados
  - Claves públicas: RSA, DSA, Diffie-Hellman
  - Cifrados simétricos: RC2, RC4, IDEA, DES, 3DES, AES
- Funciones de hash: MD5, SHA-1

Arquitectura SSL



Funcionamiento SSL

- *Handshake*
  - Se autentica el servidor, y se realiza un *key agreement* entre cliente y servidor.

- Transmisión de datos segura
  - Los datos son transmitidos de forma segura.

- Cierre de la conexión
  - Cierre seguro

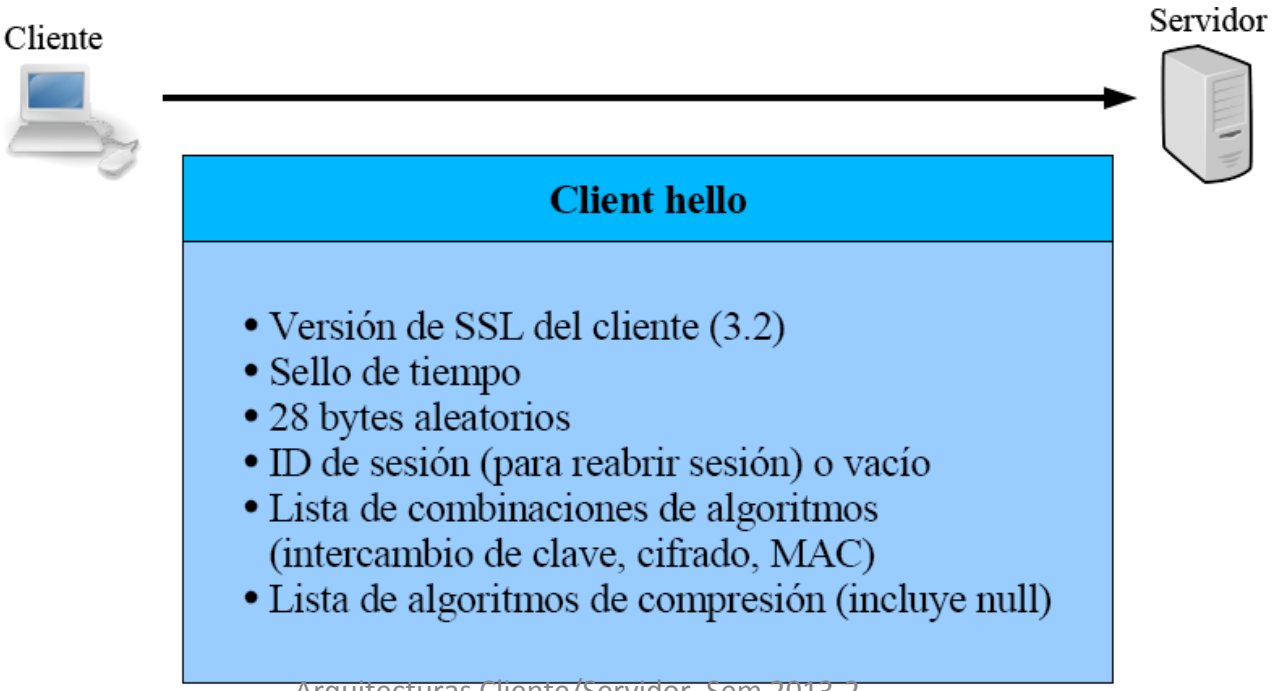
Handshake

Handshake

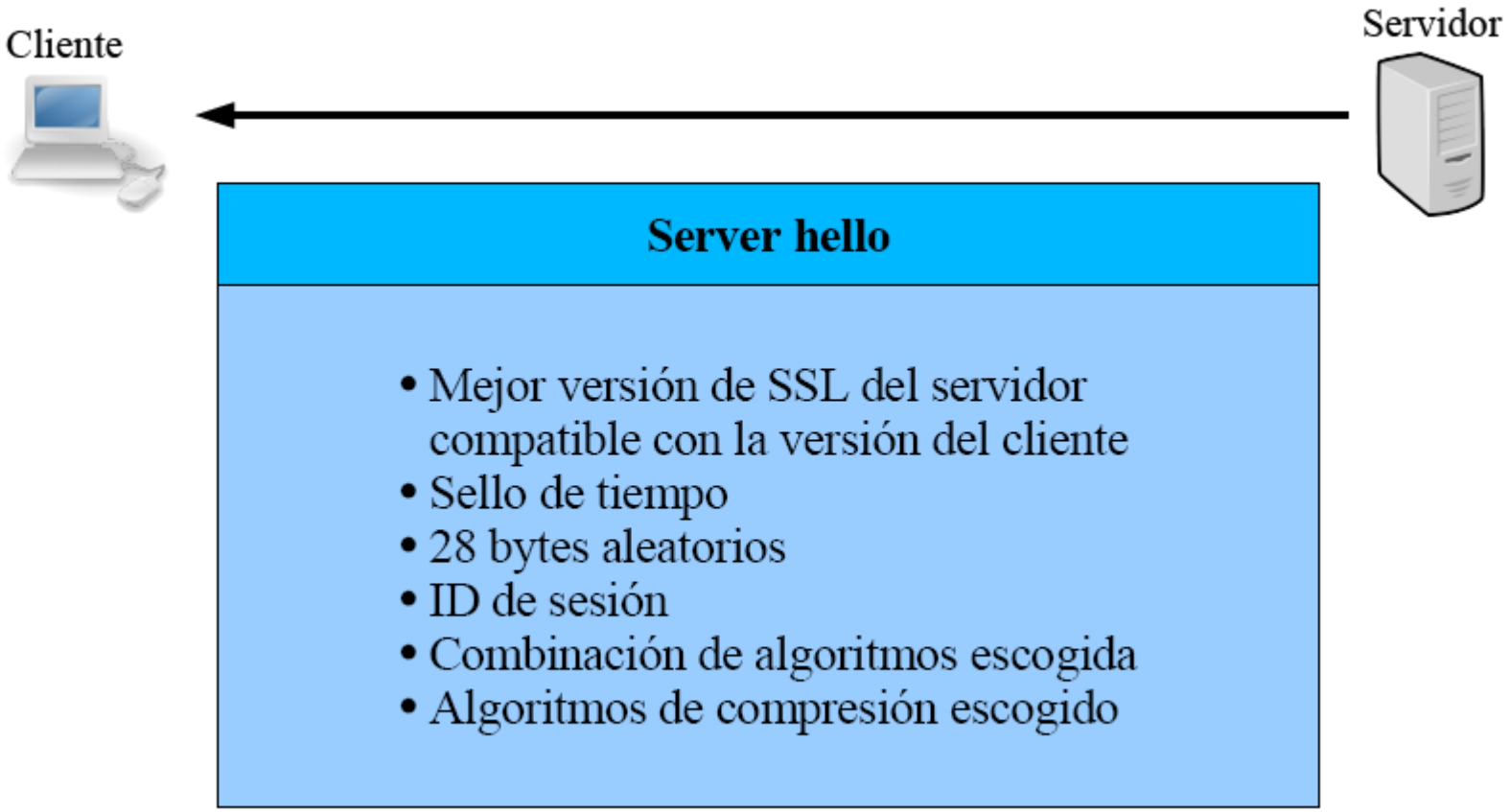
- El *handshake* tiene tres propósitos:
  - Acordar los algoritmos criptográficos a usar durante la conexión.
  - Acordar las claves criptográficas de sesión.
  - Identificar al servidor (y opcionalmente al cliente).

Protocolo de Negociación (*handshake*)

- El cliente envía al servidor una lista con los algoritmos criptográficos que soporta, junto con un número aleatorio, que usará para el *key a*



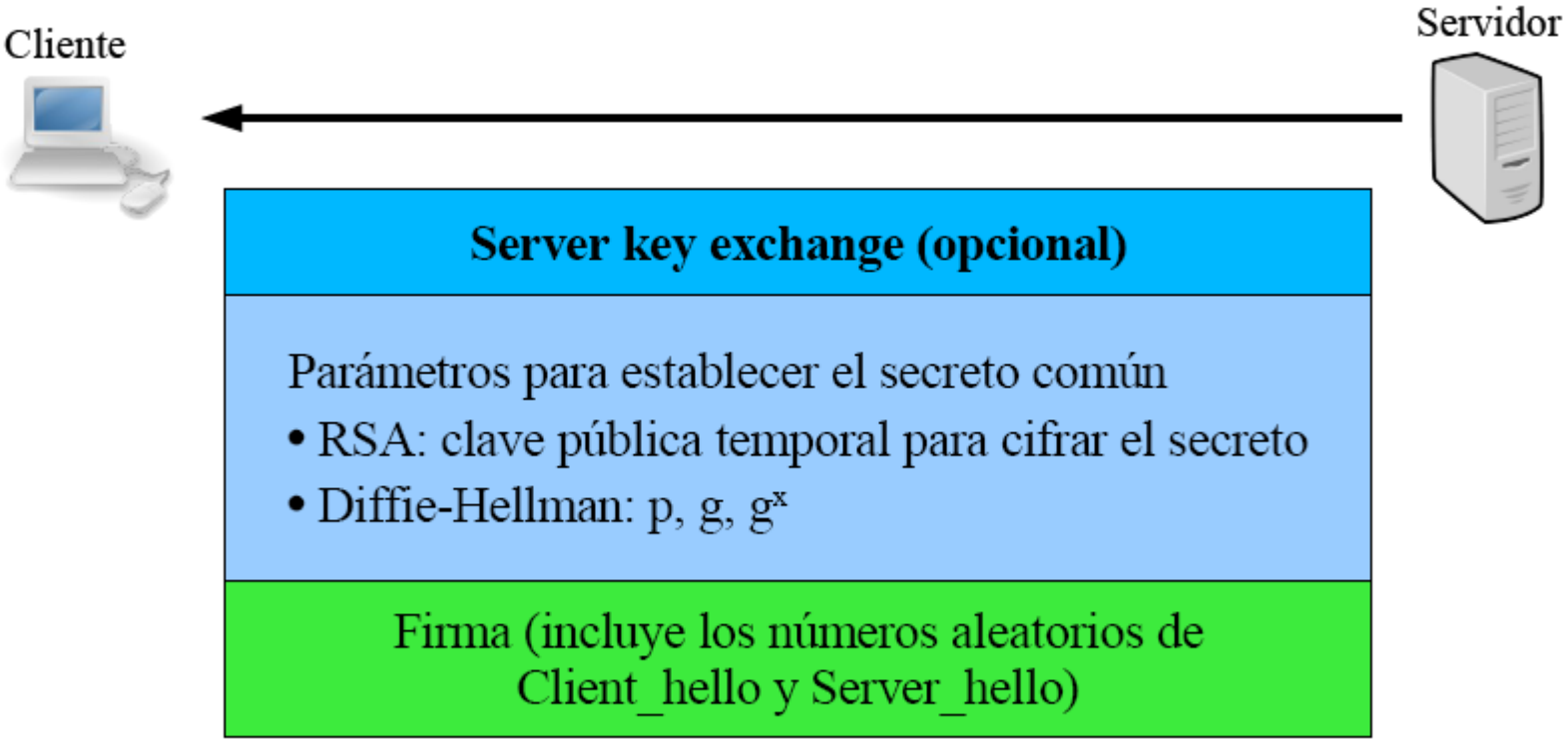
- El servidor elige un algoritmo criptográfico de la lista y se lo envía al cliente junto con su certificado digital (opcional) y su número aleatorio que había enviado al cliente firmado con su clave privada.



Envío del certificado del servidor al cliente

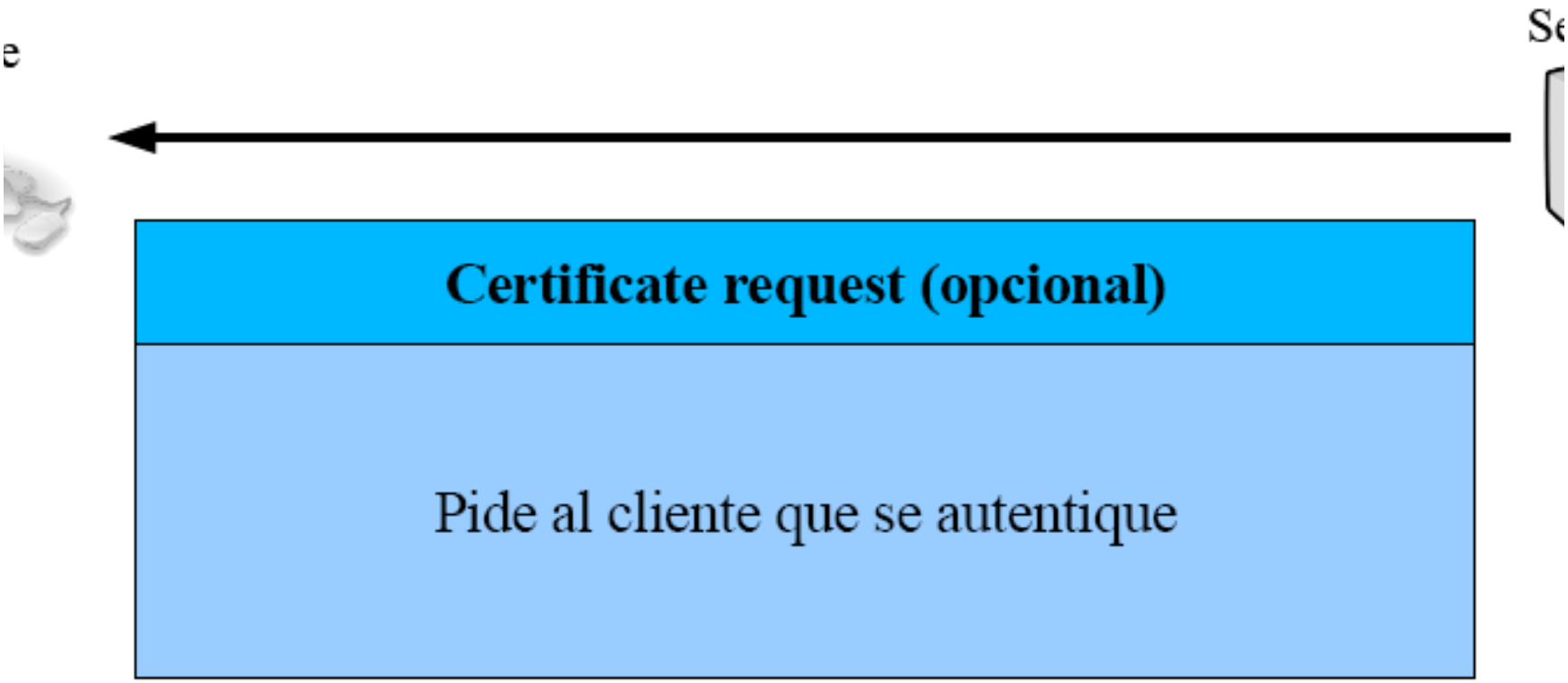
El cliente comprueba la validez del certificado

El cliente extrae la clave pública y comprueba que el número haya sido firmado correctamente.

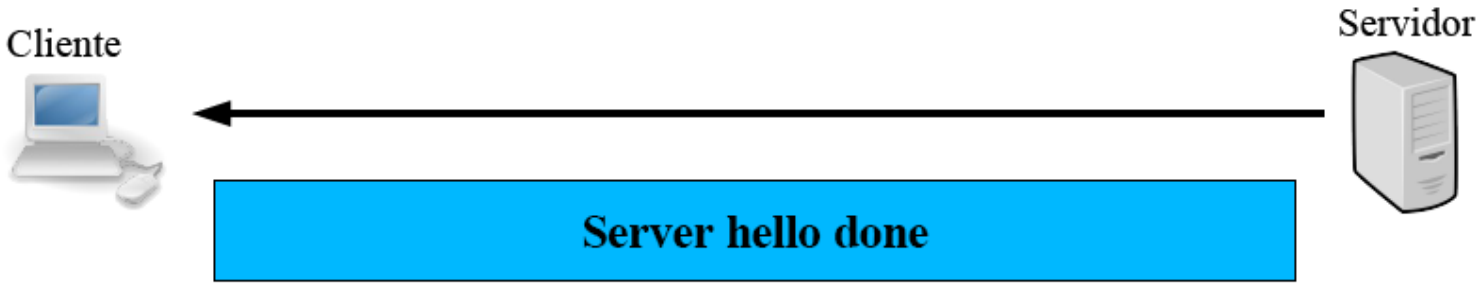


Este mensaje solo se envía si se considera que la información del certificado no es suficiente, o bien, este no ha sido enviado

El servidor puede enviar una petición de certificado al cliente

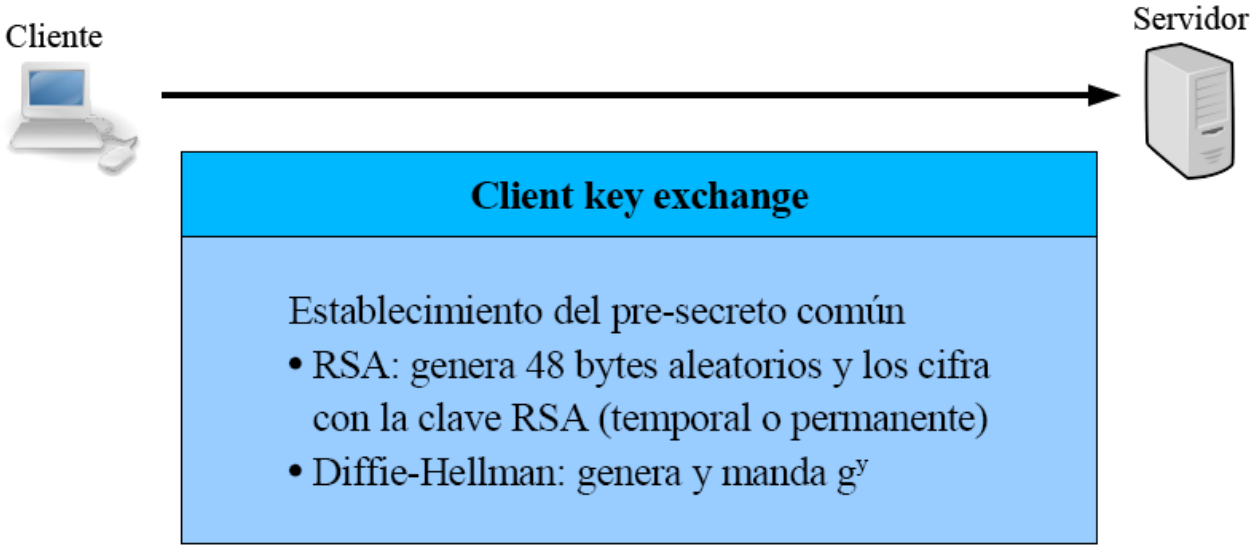


El servidor envía un mensaje “hello done” y el cliente verifica la validez del certificado del servidor

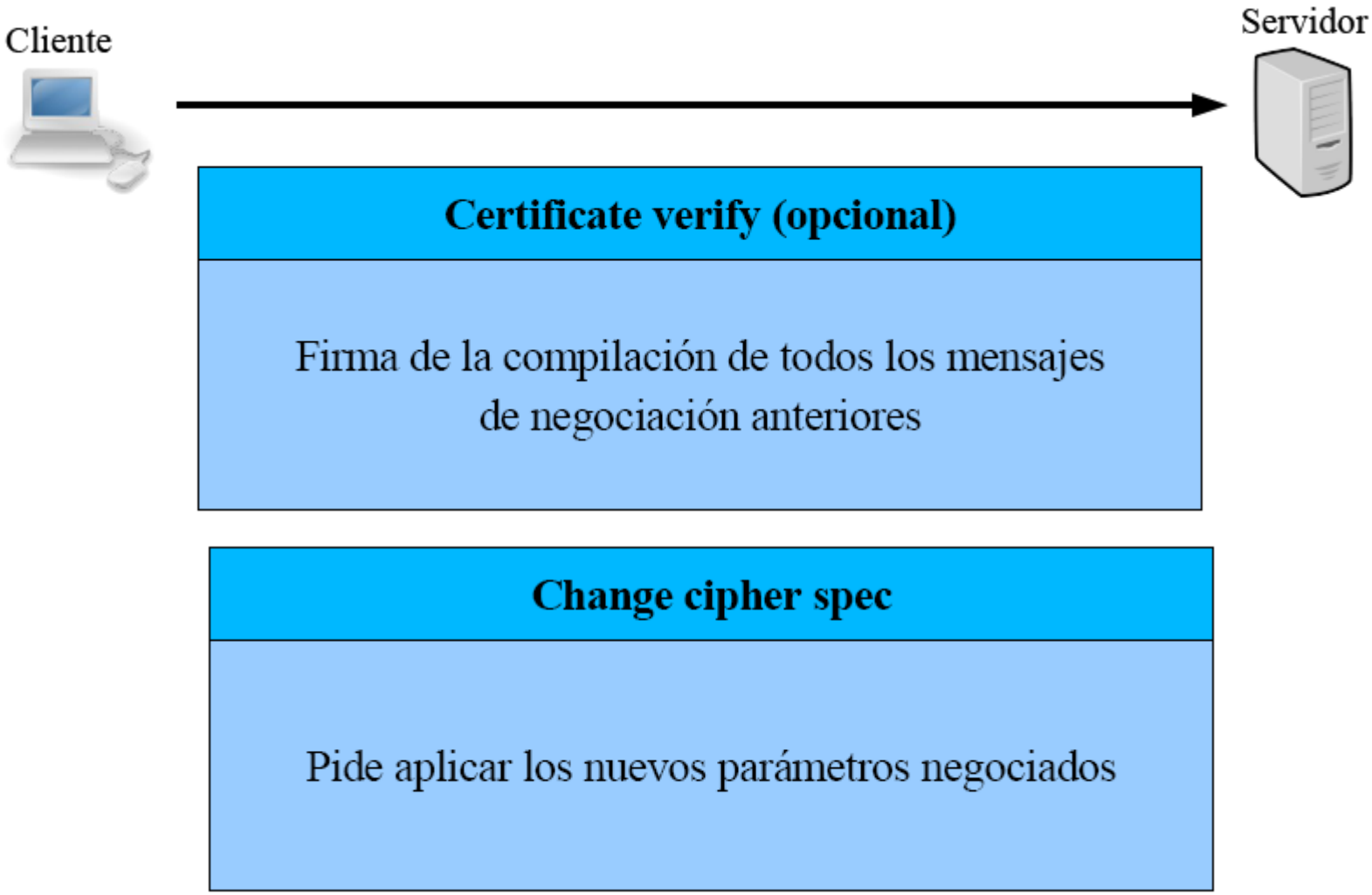


Si el servidor realizo una petición: Certificate request, el cliente envía su certificado.

- El cliente genera un clave de sesión, la encripta con la clave pública del servidor y se la envía.



- Secreto principal = 48 bytes de PRF(pre-secreto,"*master secret*", No. aleatorio cliente + No. aleatorio servidor)


Cliente
Servidor
Certificate verify (opcional)
Firma de la compilación de todos los mensajes de negociación anteriores
Change cipher spec
Pide aplicar los nuevos parámetros negociados

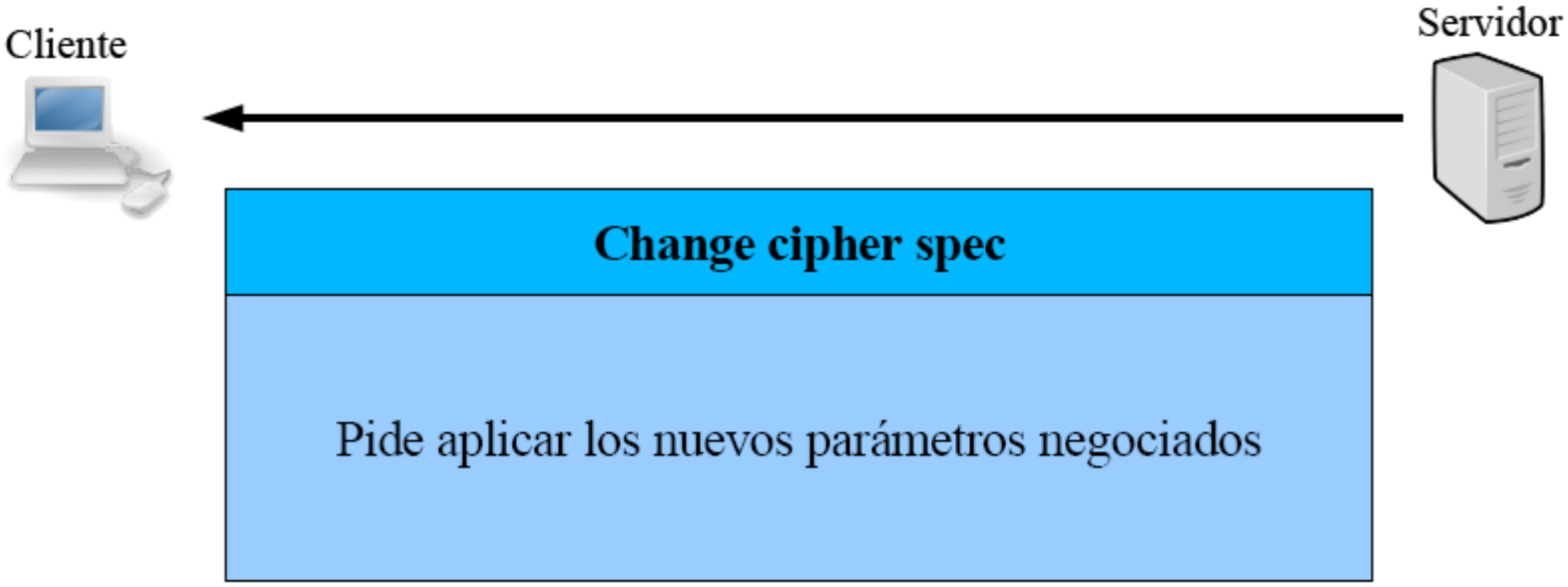
El cliente notifica el fin de la secuencia y
permite verificar que los nuevos parámetros funcionan, calcula el MAC (Message Authentication Code)de toda la comunicación y

---

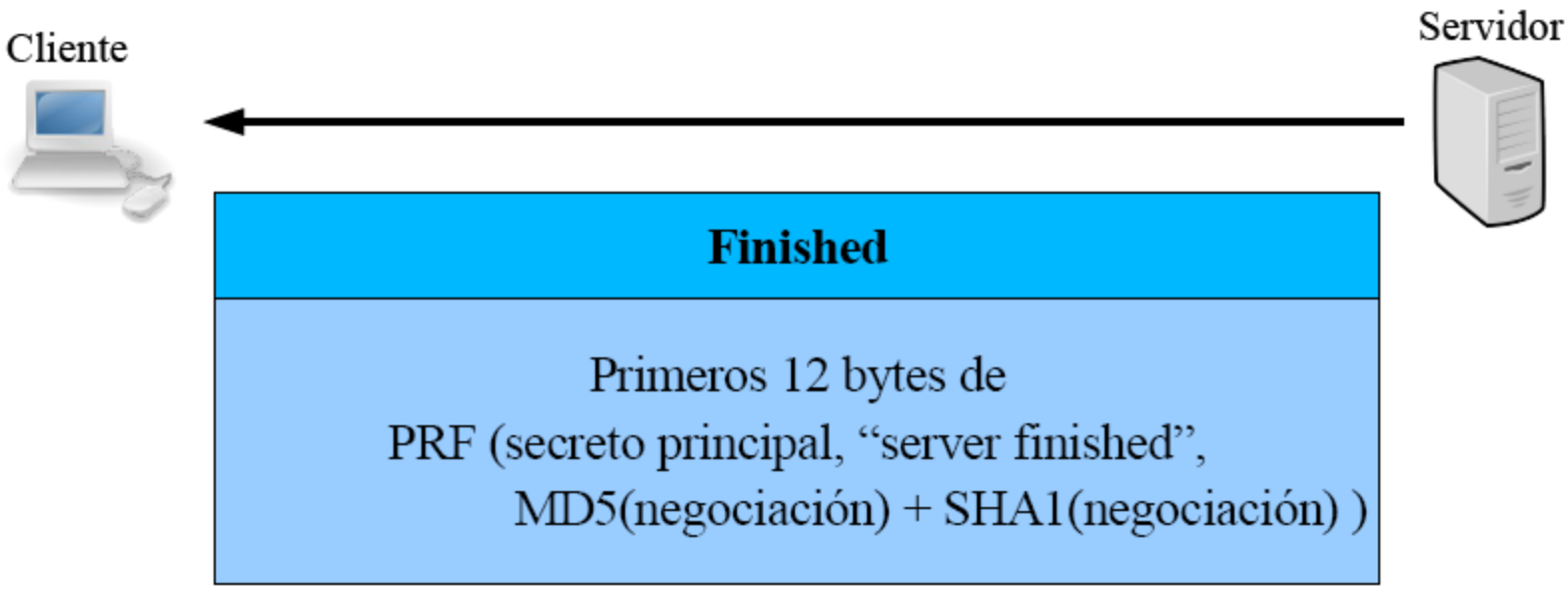
Finished

Primeros 12 bytes de
PRF (secreto principal, "client finished",
MD5(negociación) + SHA1(negociación)

- El servidor usa la clave de sesión para calcular el MAC de toda la comunicación (utilizando la clave de sesión) y se la envía al cliente.



El servidor indica el fin de secuencia enviando un mensaje cifrado



Transmisión segura de datos

Transmisión segura de datos

- Tiene principalmente dos objetivos:
  - Encriptación
    - Evitar develar el texto transmitido
  - Integridad
    - Evitar que los datos transmitidos se modifiquen

Para la transmisión de datos seguros, se requiere:

- Protocolo de alertas:
  - Manda mensajes de control
    - Errores (advertencias o fatales)
    - Terminación
  - Paquetes comprimidos, cifrados por la capa del protocolo de paquetes

- Protocolo de la datos de aplicación
  - Pasa los datos de la capa superior (HTTP, FTP...) a la capa de protocolo de paquetes para ser comprimidos, cifrados, etc.

- Protocolo de paquetes (record protocol):
  - Proporciona privacidad
    - Encripción simétrica
  - Proporciona integridad
    - MAC con clave (HMAC_MD5, HMAC_SHA1)
  - Encapsula los datos de la capa superior
    - Fragmenta en bloques menores que 16384 bytes
    - Comprime (opcional)
    - Calcula MAC
    - Cifra

Protocolo de paquetes

- SSL transporta datos en bloques llamados registros SSL los cuales se derivan del protocolo de paquetes:

1. Los datos se parten en bloques llamados fragmentos, los cuales pueden tener longitud variable.

2. A cada fragmento se le calcula su MAC (usando la clave de sesión) y se le concatena al fragmento.

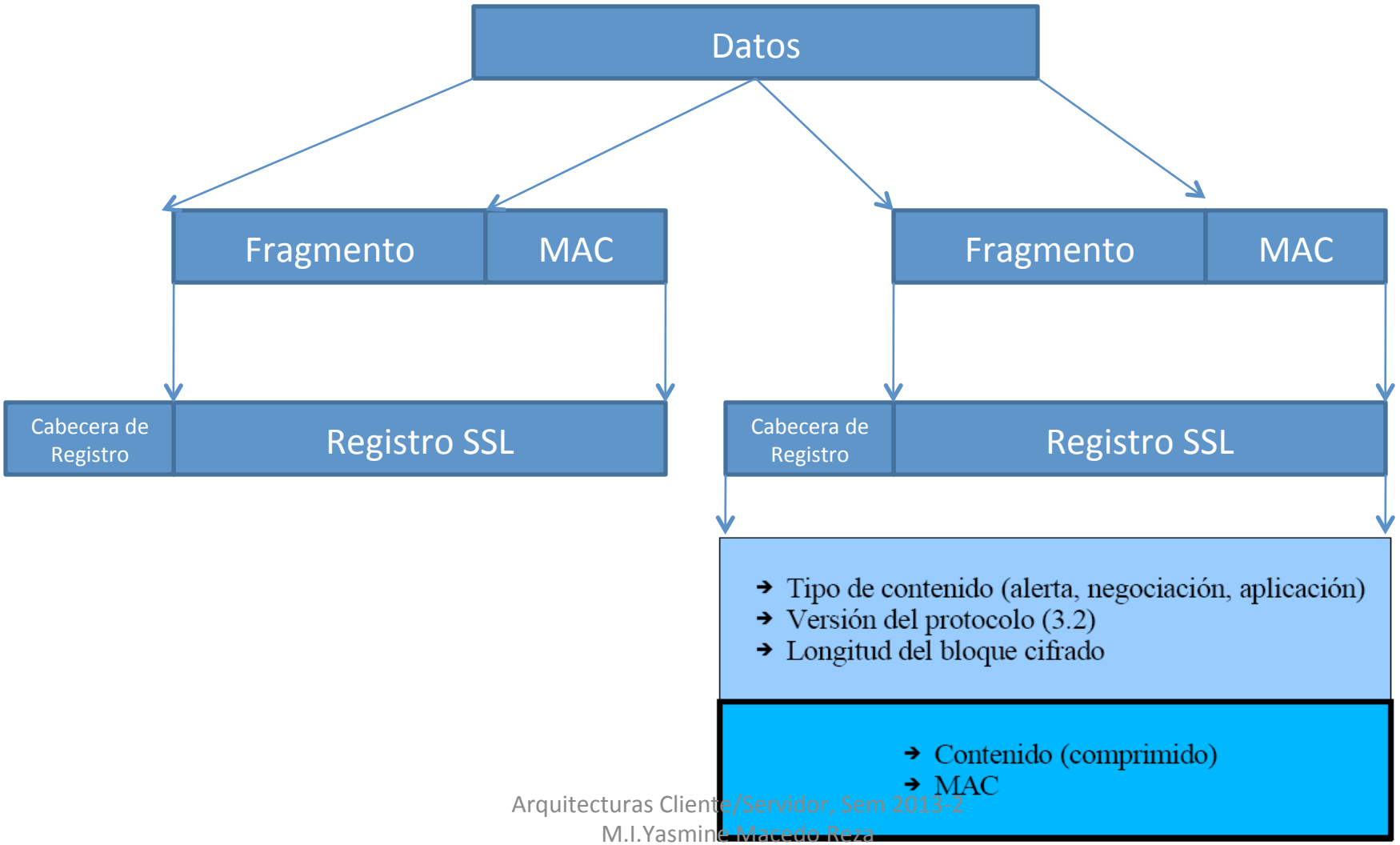
3. Se encripta el fragmento y el MAC usando la clave de sesión, para obtener el registro SSL

Cabecera de Registro SSL

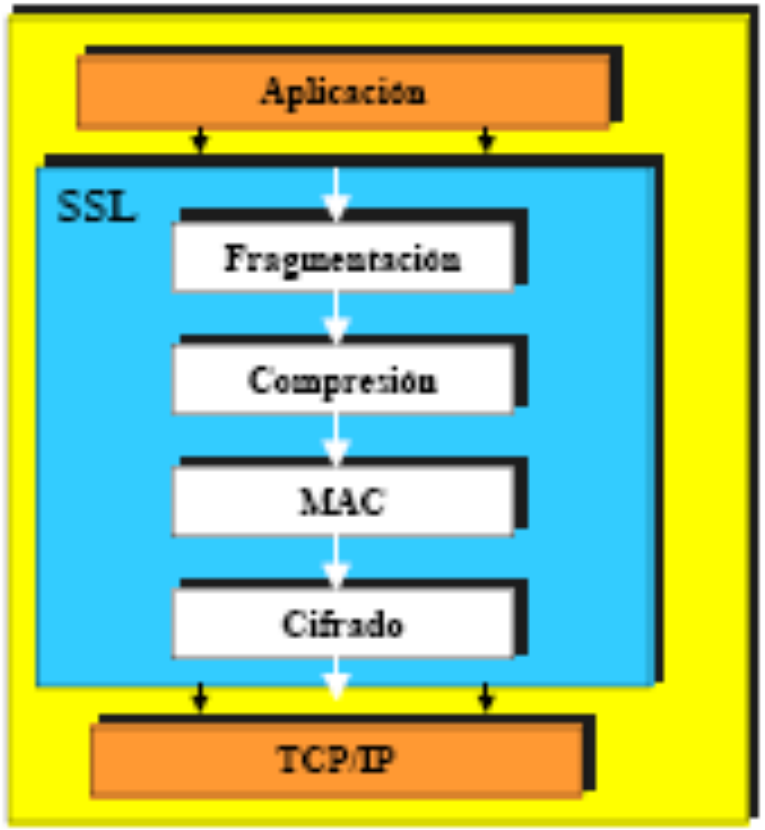
- Al registro SSL se le agrega una cabecera de registro con información sobre el contenido del registro, el cual tiene tres campos:

Content Type	Descripción
application_data	Datos normales
handshake	Renegociar el handshake durante la comunicación
change_cipher_spec	Permite cambiar el algoritmo criptográfico durante la comunicación
alert	Se utiliza para enviar mensajes de error y para finalizar la comunicación

Obtención de Fragmentos SSL



En resumen

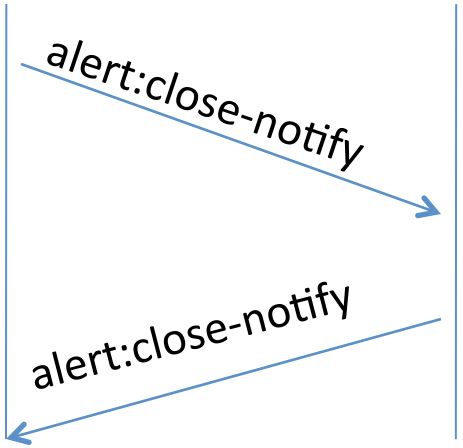

Aplicación
SSL
Fragmentación
Compresión
MAC
Cifrado
TCP/IP

Cierre de la Conexión

Cierre de la Conexión

- Es importante que las conexiones SSL se cierren con un par de mensajes **alert** con en subtipo **close_notify,** de lo contrario se puede realizar un ataque en donde el receptor solo reciba parte de los datos.

En caso de que no se cierre correctamente la conexión, los datos recibidos no se deben considerar válidos, aunque hay implementaciones de SSL que ignoran esta condición.



Otras características

- Sesiones
  - El proceso de handshake es costoso ya que utiliza el algoritmo de clave pública. Para se más eficiente SSL permite que solamente se realice una vez, de forma que las conexiones posteriores utilicen los mismos parámetros.

  - Para conseguir esto , la primera vez que nos conectamos al servidor, éste devuelve al cliente un **session_id** que el cliente utilizará en las siguientes conexiones.

Lo más sobresaliente de funciones Hash (o message digest)

- Reciben un texto de tamaño variable y devuelven un resumen

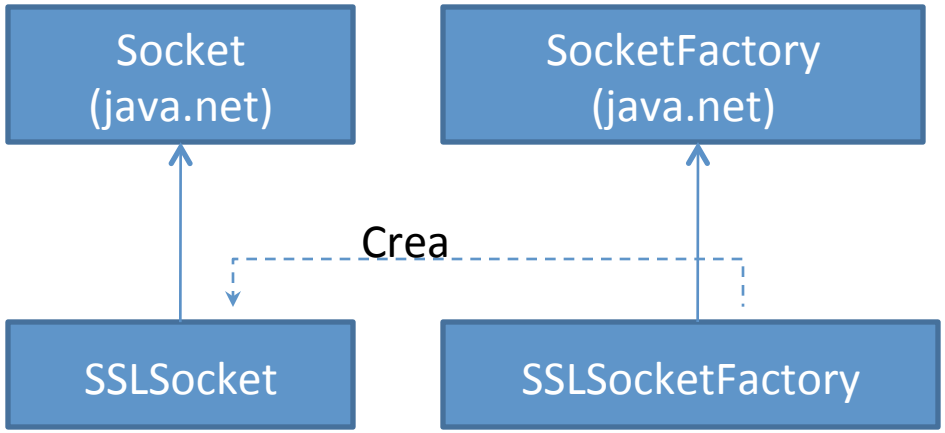
- En criptografía generalmente se utilizan funciones hash unidireccionales…

- Muchas entradas pueden dar la misma salida, pero es complicado que dada una salida encontremos una entrada que de esa misma salida.

SSL en java

- JSSE (Java Secure Socket Extensions) proporciona un API estándar para acceder a conexiones SSL y TLS.
- Hasta Java 1.3 JSSE era un extensión al API estándar.
- En la versión1.4 ya se integró al API.
- Las clases de JSSE se encuentran en el paquete `javax.net.ssl.*`

Principales clases de JSSE



- Las clases SSLSocketFactory no tienen constructor, cuenta con métodos estáticos

```
static  SocketFactory  getDefault()
```

- Crear un socket seguro desde la clase Factory

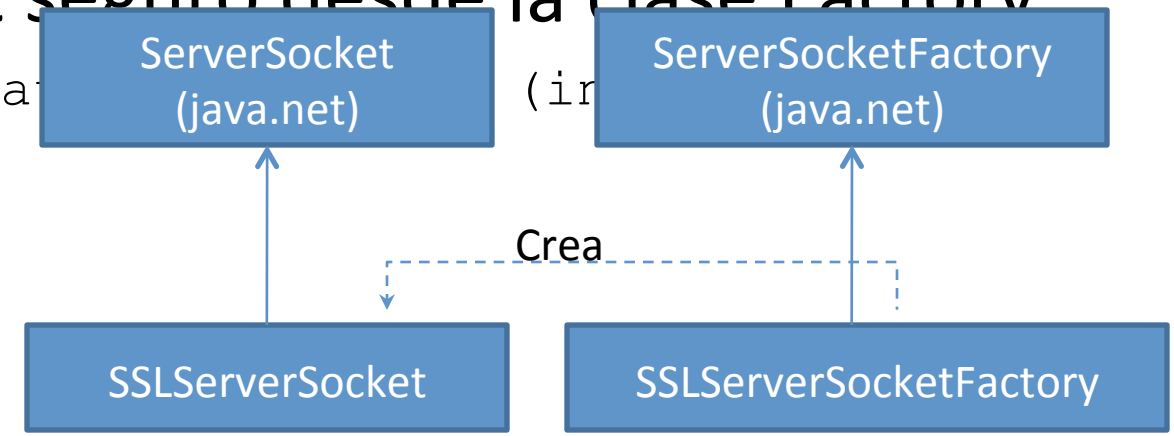
```
Socket createSocket (InetAddress host, int port)
```

- Las clases SSLServerSocketFactory no tienen constructor, cuenta con métodos estáticos

```
static  ServerSocketFactory
  getDefault()
```

- Crear un socket seguro desde la clase Factory

```
ServerSocket crea                (ir
```



Ejemplo:

- Para obtener un SSLServerSocket :

```
ServerSocketFactory ssf =
  SSLServerSocketFactory.getDefault();
ServerSocket ss =
  ssf.createServerSocket(9999);
```

- Para obtener un SSLSocket :

```
SocketFactory sf =
  SSLSocketFactory.getDefault();
Socket s =
  sf.createServerSocket(servidor,
  9999);
```

ServidorSSL

```
import java.io.*;
import java.net.*;
import javax.net.*;
import javax.net.ssl.*;
public class ServidorSSL extends Thread
{
  Socket s;
      static int nCliente = 1;
      public static void main(String[] args) throws Exception
      {
        ServerSocketFactory ssf = SSLServerSocketFactory.getDefault();
        ServerSocket ss = ssf.createServerSocket(9999);
        System.out.println("Servidor Lanzado");
        while(true)
        {
                new ServidorSSL(ss.accept()).start();
        }
      }
```

```
public ServidorSSL(Socket s)
{
      this.s = s;
}
public void run()
      {
            try
            {
                  System.out.println("Cliente"+ nCliente + "conectado");
                  PrintWriter pw = new PrintWriter(s.getOutputStream());
                  pw.println("Hola cliente numero"+nCliente+ "soy el servidor");
                  pw.close();
                  s.close();
                  System.out.println("Cliente"+ nCliente + "conectado");
                  }
                  catch(Exception ex){
                      ex.printStackTrace();
                  }
        }
        }
```

Ejemplo SSL: Cliente SSL

```
import java.io.*;
import java.net.*;
import javax.net.*;
import javax.net.ssl.*;
public class ClienteSSL
{
    public static void main (String[] args) throws Exception
     {
        if(args.length!=2){
            System.out.println(" ClienteSSL <host><puerto>");
           return;
        }
        SocketFactory sf = SSLSocketFactory.getDefault();
        Socket s =sf.createSocket(args[0], Integer.parseInt(args[1]));
        BufferedReader br = new BufferedReader( new
           InputStreamReader(s.getInputStream()));
        System.out.println(br.readLine());
        br.close();
        s.close();
    }
    }
```

Keystore

- El almacén de claves es una base de datos protegida que contienen claves y certificados para un usuario o grupo de usuarios. El acceso al mismo está protegido por un sistema de contraseñas y además, cada clave privada puede protegerse con una contraseña individual.

- El keystore es donde el programa almacena sus certificados para identificarse en el establecimiento de una conexión SSL.

Herramienta Keytool

- Java proporciona una herramienta llamada "keytool" que se emplea para gestionar un almacén de claves (*keystore*), entre otras cosas permite:
  - Crear pares de claves (pública y privada).
  - Emitir solicitudes de certificados (que se envían al CA apropiado).
  - Importar replicas de certificados (obtenidos del CA contactado).
  - Designar claves públicas de otros como de confianza.

jdk1.6.0_03\bin>keytool
sintaxis de keytool:

- -certreq .....
- -changealias …
- -delete …
- -exportcert …
- -genkeypair …
- -genseckey …
- -help
- -importcert …
- -importkeystore ….
- -keypasswd ….
- -list …
- -printcert …
- -storepasswd …

Operación	Descripción
-certreq	Permite crear un CSR
-delete	Borra una entrada del keystore
-export	Permite exportar certificados keystore
-genkey	Permite crear un certificado self-signed en el que se almacena el par clave privada/ clave pública
-help	Muestra un mensaje de ayuda con las distintas opciones de keytool
-identitydb	Convierte un keystore del jdk 1.1 a jdk 1.2
-import	Permite importar un certificado de un trusting point.
-keyclone	Realiza una copia de un certificado con otro alias
-keypasswd	Permite indicar el password de una entrada determinada
-list	Muestra el contenido de una keystore
-printcert	Imprime el contenido de un certificado pasado como un argumento con la opcion -file
-selfcert	Crea un certificado self-signed a partir de un certificado creado con -genkey
-storepasswd	Indica el password de todo el keystore

Truststores

- JSSE utiliza un tipo de keystore llamado trutstores, que es una base de datos con los certificados de las principales CA

- Se utiliza para comprobar la validez de los certificados que se reciben

- **keytool –list –keystore $JRE_HOME/lib/security/cacerts**

```
Utilizando el password: changeit
Para mac password: changeme, y ruta: /System/Library/Frameworks/
   JavaVM.framework/Home/lib/security
```

- **Para generar un nuevo keystore:**

- **keytool -genkey -keystore programakeystore**

Ejecución de programas SSL

Ahora se puede indicar el keystore y password a usar por el programa en Java, en las propiedades del sistema, en el caso del Servidor:

```
$java
    -Djavax.net.ssl.keyStore=programakeystore
    -Djavax.net.ssl.keyStorePassword=pwdProg
    ServidorSSL
```

```
$java -Djavax.net.ssl.keyStore=programakeystore -Djavax.net.ssl.keyStorePassword=pwdProg ServidorSSL
```

Para ejecutar el cliente necesitamos colocar el certificado en un keystore de confianza de usuario

- Lo primero es exportar el certificado del archivo programakeystore a un archivo programa.cer

```
$keytool –export –alias mykey –file programa.cer –
   keystore programakeystore
```

- El alias mykey es el que se crea por default
- Ahora se importa el certificado a un archivo progtruststore con:

```
$keytool –import –file programa.cer –
  keystore progtruststore
```

Y ahora se puede ejecutar el cliente:

```
$java
    -Djavax.net.ssl.trustStore=progtruststore
    -Djavax.net.ssl.trustStorePassword=pwdProg
      ClienteSSL localhost 9999
```

```
$java -Djavax.net.ssl.trustStore=progtruststore -Djavax.net.ssl.trustStorePassword=pwdProg
    ClienteSSL localhost 9999
```

Opcional

- Realizar una aplicación **segura.**

- Haciendo una aplicación cliente/servidor seguro utilizando sockets SSL (proyecto 1)

.