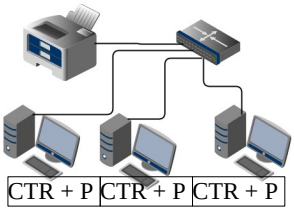
Resumen conceptual PSP Segunda evaluación

<u>Servicio</u> ⇒

- Programa auxiliar utilizado en un sistema de computadoras
- Gestionar una colección de recursos
- Prestar su funcionalidad a los usuarios y aplicaciones
- Ej: Cuando se envía un documento a una impresora que forma parte de una red



TCP / IP → **Modelo Cliente / Servidor**

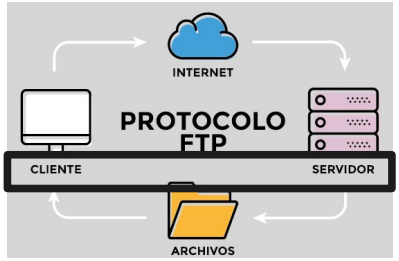
Capa de Aplicación	FTP	– Define las aplicaciones de red y los sercicios de internet
Capa de Transporte	TC	– Utiliza la capa de transporte para enviar y recibir datos
Capa de Internet	IP	
Capa de Interfaz	Ethernet	

e**STH**er-**F**ou**NDS-T**ranquiliy

SMTP	*(Simple Mail Transport Protocol)*	Administra el correo electrónico
TELNET	*(Telecomunication NetWork)*	Permite acceder a una máquina remota Se usa para arreglar los fallos en ellas Problema: La Seguridad, contraseñas y nombres viajan por la red en texto plano
HTTP	*(HyperText Transport Protocol)*	Realiza peticiones y recibe datos de los servidores web
FTP	*(File Transport Protocol)*	Servicio confiable <u>orientado a conexión</u> para transferir ficheros
NFS	*(Network File System)*	Acceso en línea a ficheros remotos
DNS	*(Domain Name System)*	Resuelve el nombre de un host en una dirección IP para evitar recordarla
SNMP	*(Simple Network Management Protocol)*	Permite a los administradores monitorear, controlar y supervisar el funcionamiento de una red
TFTP	*(Trivial File Transfer Protocol)*	Servicio <u>no orientado a conexión</u> que utiliza el <u>protocolo UDP</u> Semejante al FTP Utilizado por aplicaciones que no necesitan mucha interacción cliente-servidor

Resumen conceptual PSP Segunda evaluación

FTP ⇒



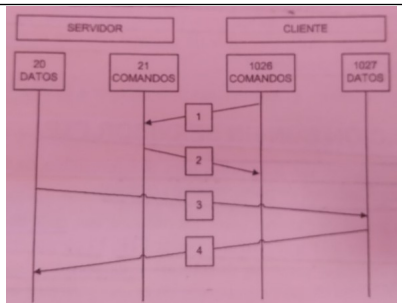
- De las herramientas más utiles para el **intercambio de ficheros**
- Es la forma habitual de publicación en Internet
- Existen dos tipos de **accesos**:
  - **Anónimo:**
    - Sin privilegios
    - Se le recluye a un directorio público donde solo se le permite descargar ficheros
  - **Autorizado**:
    - Tiene ciertos privilegios
    - Se le dirige a su directorio personal, donde puede subir y bajar ficheros, teniendo una cuota de espacio asignado
- Utiliza dos **conexiones** distintas:
  - **Conexión de Control:**
    - Encargada de iniciar y mantener la comunicación con el servidor
    - El cliente se conecta a un puerto aleatorio
    - El servidor se conecta al puerto 21 (*Command port*)
      - Puerto de comandos para transferir las órdenes
  - **Conexión de Transferencia:**
    - Existe únicamente cuando hay datos a transmitir
    - Se encarga de enviar datos entre cliente y sevidor
    - El cliente obtiene un nuevo puerto
    - El servidor se conecta al puerto 20
      - Puerto para la trasferencia de datos

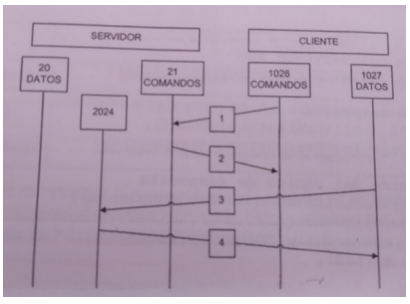
En el proceso de **comunicación** el cliente puede actuar de dos formas:

Modo Activo *Estandar o Port*	
Cliente	Manda comando PORT al puerto 21 y le indica su número de puerto
Servidor	Envía un ACK de vuelta al cliente
Servidor	Inicia la conexión del puerto 20 con el puerto del cliente
Cliente	Envia un ACK al servidor

Modo Pasivo *PASV*	
Cliente	Manda un comando PASV al puerto 21
Servidor	Le indica al cliente el puerto por el que lo va a escuchar
Cliente	Inicia la conexión de datos al puerto que le ha dicho el servidor
Servidor	Envia un ACK al puerto del cliente

El modo Activo tiene un problema de seguridad, ya que se abre una conexión de datos a la máquina cliente desde fuera para dentro y si el cliente está conectado a una red insegura, puede ser atacado fácilmente
El uso de un cortafuegos rechazará estas conexiónes
Por esta razón se creó el modo pasivo





Resumen conceptual PSP Segunda evaluación

Para comunicarfse con un servidor FTP → **commons – net – x . y . jar**

Clase FTPClient:

- Encapsula toda la funcionalidad necesaria para almacenar y recuperar ficheros de un servidor FTP
- Se encarga de todos los detalles de bajo nivel de la interacción
- Para utilizarla:
  1. Se realiza la conexión con el método **connect ( )**
  2. Se comprueba el **código de respuesta** → **Clase FTPReply**
  3. Realiza las operaciones de **transferencia**
  4. Cierra la conexión con **disconnect ( )**

  ○ Almacena un conjunto de constantes para **códigos de respuesta**
  ○ Los códigos son nombres nemotécnicos transcritos de las descripciónes de los mismos de la **RFC 959**

El método **isPositiveCompletion (int respuesta)** devuelve *true* si el código de respuesta a terminado positivamente		
El **código 220** significa que el servicio está preparado		
1yz	El servidor inició la acción	
2yz	El servidor terminó con éxito la acción	
3yz	El servidor aceptó el comando pero la acción necesita información	
4yz	El servidor no aceptó el comando y la acción no ocurrió	**Terminación Negativa Transitoria**
5yz	El servidor no aceptó el comando y la accion no ocurrió	**Terminación Negativa Permanente**

- Todos los métodos de comunicación con el servidor pueden lanzar **IOException**
- El servidor puede optar por cerrar la sesión antes de tiempo si el cliente ha estado inactivo más de un periodo (generalmente 900 segundos)
  - FTPClient detectará el cierre prematuro y puede lanzar la excepción **FTPConnectClosedException**
- Lo normal es:
  - Conectarse al servidor con un nombre y una clave de usuario mediante el método **login ( )** *(devuelve true si la conexión se realiza correctamente)*
  - Conectarse al servidor en modo pasivo utilizando el método **enterLocalPassiveMode ( )**
  - Desconectarse usando el método **logout ( )**

Métodos de la clase FTP ⇒ void connect (String host) // int getReplyCode( ) // String getReplyString ( )

Metodos de la clase FTPClient	
boolean	login (String user, String password) // logout ( ) // enterLocalActiveMode ( ) // enterLocalPasiveMode ( ) // changeWorkingDirectory (String pathname) // chaneToParentDirectory ( ) // setFileType (int FileType) // storeFile (String nombre, InputStream local) // retrieveFile (String nombre, OutputStream local) // deleteFile (String pathname) // rename (String antiguo, String nuevo) // remove Directory (String pathname) // makeDirectory (String pathname)
void	disconnect ( )
String	printWorkingDirectory ( )
FTPFile []	listFiles ( ) // listFiles (String path) // listDirectories ( ) // listDirectories (String parent)
String []	listNames ( )

Resumen conceptual PSP Segunda evaluación

FTPFile:

- Utilizada para representar información acerca de los ficheros almacenados

Metodos de la clase FTPFile	
boolean	isDirectory ( ) // is File ( ) // isSymbolicLink ( )
long	getSize ( )
String	getName ( ) // getUser ( )
int	getType ( )

Para subir un fichero a un FTP:

1. Se necesitan un nombre de usuario y una clave para tener el espacio y los permisos necesarios
2. Tener a disposicion Filezilla Server o un hosting web con servicio FTP
3. Indicar el tipo de fichero a subir con **setFileType ( )** e indicando **BINARY_FILE_TYPE**, que permite enviar ficheros de cualquier tipo
4. El String de entrada creado en el punto anterior, se pasa al método **storeFile ( )** indicando el nombre que le vamos a dar en el primer String y el InputStream en el segundo *(devuelve true si todo es correcto)*
5. Cerrar el flujo de entrada

Para descargar un fichero de un FTP:

- Se utilizará el método **retrieveFile (String remote, OutputStream local)**
  - Necesitamos saber el directorio donde está ubicado y crear un stream de salida a nuestro disco duro

SMTP ⇒

- Protocolo estándar de internet para el **intercambio de correo**
- Por defecto utiliza el **puerto 25**
- Funciona con comandos de texto
  - A cada comando que envía el cliente le sigue una respuesta del servidor compuesta por un número y un mensaje descriptivo

Clase SMTPClient:

- Encapsula toda la funcionalidad necesaria para enviar ficheros a través de un servidor MTP
- Es necesario:
  1. Antes de iniciar cualquier operación es necesario conectarse (como en las clases derivadas de SocketClient) con el método **connect ( )**
     - Si se desea conectar a otro puerto distinto al 25, se debe indicar con **connect (host, puerto)**
  2. Comprotar el **código de respuesta** → **Clase SMTPReply:**
     - Almacena un conjunto de constantes para **códigos de respuesta**
     - Los códigos se pueden consultar en la **RFC 2821**
  3. Cerrar la conexión con **disconnect ( )**

El método **isPositiveCompletion (int respuesta)** devuelve *true* si el código de respuesta a terminado positivamente
El metodo **getReplyString ( )** devuelve el valor entero del código de respuesta
El metodo **getReplyCode ( )** devuelve el valor entero del código de respuesta

} Como en la clase FTP

- Constructores de la clase (el uso de uno u otro depende de los danos que nos proporcione el servidor:

Resumen conceptual PSP Segunda evaluación

Si se pone *false* será explícito

- **SMTPClient (booleano implícito)**:
  - Modo de seguridad implicíto, (*true*)
  - En este modo la negociacion SSL/TLS comienza después de que se haya establecido la conexión
- **SMTPSClient ( ) :**
  - Modo se seguridad explicito, (*false*)
    En éste modo la negociación SSL/TLS se inicia cuando el usuario llama al método execTLS( ) y el servidor lo acepta

Clase AutenticatingSMTPClient:

- Con soporte de autentificación al conectarse al servidor SMTP
  - Sobre el protoclo SSL (*Secure socket Layer*)
    - Protocolo criptográfico empleado para conexiones seguras entre cliente y servidor **TLS** (*Transport Layer Security*)
      - Protocolo sucesor de SSL

STARTTLS:

- Extensión que permite cifrar las comunicaciones entre cliente y servidor
- El protocolo SMTP por defecto no utilia cifrado pero todos los mensajes que utilicen esta extensión lo estarán
- Cuando el servidor permite utilizarlo se lo comunica al cliente y el cliente le responde para iniciar la sesión e intercambiar los correos que ya estarán cifrados en el tránsito
- Se incorporó como nuevo mecanismo a través del RFC 3207

Metodos de la clase SMTPClient	
boolean	addRecipient (String address) // completePendingCommand ( ) // login ( ) // login (String hostname) // logout ( ) //sendSortMessaeData (String message) // sendSimpleMessage (String remitente, String [] destinatarios, String message) // sendSimpleMessage (String remitente, String destinatario, String message) // sentSender (String address) // verify (String username)
Writer	sendMessageData ( )

Clase SMTPHeader:

- Utiliado para la construcción de una cabecera mínima aceptable en el envío de un mensaje de correo
- Se debe indicar el *from*, el *to* y el *subject*

Metodos de la clase SMTPHeader	
void	void addCC(String address) // addeaderField(String headerField, String value)
String	toString( )

Para autentificarse en SMTP (SMTP AUTH):

- Extensión mediante la cual un cliente puede iniciar sesión mediante uno de los mecanismos admitidos por el servidor
- La clase **AuthenticatinSMTPClient** poroporciona este soporte de autenticación
  - Está reflejado en **RFC 4954**
  - Su contructor por defecto lanza la excepcion *NoSucAlgoritException*, que se produce cuando, despues de solicitarse algún algoritmo criptográfico, éste no está disponible en el entorno
  - Los valores para éste método son:
    - **CRAM_MD5 :** La contraseña se envia encriptada
    - **LOGIN** y **PLAIN**: La constraseña se envia desencriptada (como texto plano) pero no es importante ya que todo el proceso se realiza sobre un canal cifrado
    - **XOAUT**: Mecanismo de autentificación SASL que se basa en firmas Oauth

Acceso a los mensajes de un servidor SMTP:

- Otros protocolos para funciones adicionales:

Resumen conceptual PSP Segunda evaluación

- **MIME** (*Multiprupose Internet Mail Extension)*
  - Permite la insercción de documentos en un mensaje
  - Su versión segura se denomina **S/MIME**
- **POP** *(Post Office Protocol*)
  - La última versión es **POP3**
  - **Proporicona acceso a los mensajes, es el servidor del correo electrónico entrante**
  - Su puerto es el 110
- **IMAP** (*Internet Message Access Protocol)*
  - Su última versión es la **IMAP4**
  - Proporciona acceso a los mensajes, es tambien un servidor de correo electrónico entrant

Diferencias entre POP3 e IMAP4:

- IMAP4 permite que los usuarios organicen sus mensajes en carpetas
- Con IMAP4 lods mensajes continúan siempre almacenados en el servidor, cosa que con POP no ocurre

El uso de TELNET para comunicarse con un servidor SMTP:

- Se ha de tener instalado el cliente Telnet
- En la **línea de comandos del DO**
  1. Se escribe: telnet
  2. despues: open localhost25
- El servidor nos responde con un número de 3 dígitos (con sintaxis y significados como en FTP)
- Introducimos los comandos:
  1. Se abre la sesión con: **HELO**
  2. Escribimos el origen con: **MAIL FROM:**
  3. Indicamos el destino con: **RCPT TO:**
  4. Con el comando **DATA** enviamos el mensaje
  5. Finalizamos con: **QUIT**

Por cada línea que se escribe el servidor va respondiendo

El uso de TELNET para comunicarse con un servidor POP3:

1. Es necesario tener un usuario en el servidor local de correo
2. En primer lugar se envia: **USER** con el nombre de usuario
3. A continuación se envía **PASS** con la clave
4. Después el comando **STAT** muestra el número de mensajes del usuario y el tamaño
5. El comando **LIST** muestra la lista de mensajes
6. Con **RETR(número)** muewstra el contenido del mensaje seleccionado

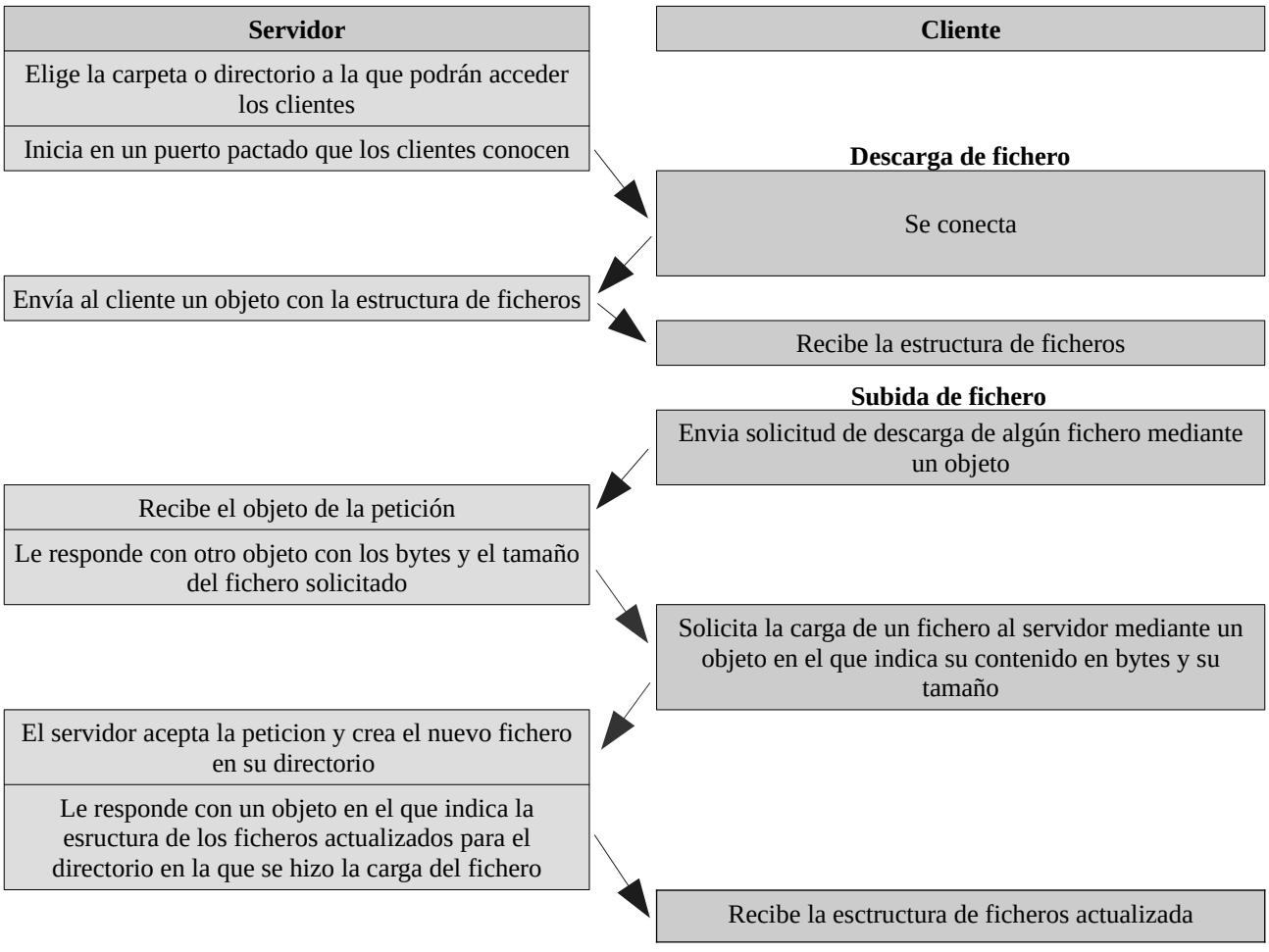
El uso de Java para comunicarse con un servidor POP3:

- **Clase POP3Client:**
  - Implementa el lado cliente del protocolo POP3 de internet definido en la **RFC 1939**
- **POP3SClient:**
  - POP3 con soporte SSL, extiende **POP3Client**
  - Modo implícito: **POP3SClient (boolean implicit)**
  - Modo explícito; **POP3SClient**
- **POP3MessageInfo:**
  - Utilizado para devolver información acerca de los mensajes almacenados en el servidor POP3
  - Sus campos son: *identifier, number* y *size*, los cuales tienen distintos singnificados dependiendo de la información que devuelve

Resumen conceptual PSP Segunda evaluación

	Identifier	Number	Size
Comando de estado	*NULO*	Número de mesajes en el buzón de correo	Tamaño del buzón de correo en bytes
Lista de mensajes	*NULO*	Número de mensajes	Tamaño del mensaje en bytes
Lista de un único mensaje	Identificador único del mensaje	Número del mensaje	*NO DEFINIDO*

Programación de servidores con Java (Ejercicio de entrega)



Resumen conceptual PSP Segunda evaluación

Lista de buena praxis en la seguridad al escribir un código:

Informarse
- Estudiar y comprender los errores que otros hayan cometido
- Recopilar problemas de vulnerabilidad que se comparten en los foros
- Leer libros, artículos y análisis
- Explorar en software de código abierto y buscar ejemplos

Precaución en el manejo de datos (Se debe de verificar cada pieza de entrada de datos al programa)
- Limpiar los datos. Proceso de examen de los datos de entrada
- Realizar la comprobación de límites.
- Revisar los ficheros de configuración.
- Comprobar los parámetros de línea de comandos
- No fiarse de las URLs web para insertar variables y sus valores.
- Cuidado con los contenidos web.
- Comprobar las cookies web.
- Comprobar las variables de entorno.
- Establecer valores iniciales válidos para los datos.
- Comprender todas las referencias de nombre de fichero y utilizarlas correctamente dentro de los programas
- Especial atención al almacenamiento de información sensible.

Reutilización de código bueno
- Reutilización de software que ha sido completamente revisado y probado, y ha resistido las pruebas del tiempo y de los usuarios

Insistir en la revisión de los procesos (Si un programa es confiado a varias personas, todas deben participar en la revisión)
- Realizar una revisión por pares (dos o más revisores). Para entornos de desarrollo informales.
  - Es recomendable el desarrollo de una lista de cosas que buscar, esta tiene que ser mantenida y actualizada con los nuevos fallos de programación encontrados
- Realizar una validación y verificacion independiente. Revisión más formal
  - Consiste en la revision del código fuente línea por línea para ganrantizar que se ajusta a su diseño, asi como a otros criterios
- Utilizar las herramientas herramientas de software disponibles para ayudar a la revision de fallos en el código fuente. Útiles para capturar de errores comunes, pero no tanto para detectar cualquier otro error

Utilizar listas de control de seguridad (Útiles para asegurarse de que se han cubierto todas las fases
- Requiere de una contraseña para que los usuarios puedan acceder
- Inicios de sesión son únicos
- Sistema de control de acceso basado en roles
- La contraseña no se transmite a través de la red en texto plano
- Se utiliza cifrado para proteger los datos entre servidores y clientes

Amabilidad con los mantenedores (El mantenimiento del código es de vital importancia para la seguridad del software en el transcurso de su vida útil)
- Utilizar normas con respecto a cosas como la documentación en línea del código fuente, etc. para que hagan la vida más fácil a los que posteriormente mantendrán el código.
- El código modular, que está bien documentado y es fácil de seguir es más fácil de mantener
- Retirar código obsoleto
- Analizar todos los cambios en el código. Hemos de asegurarnos de probar a fondo los cambios en el código antes de entrar en producción

Resumen conceptual PSP Segunda evaluación

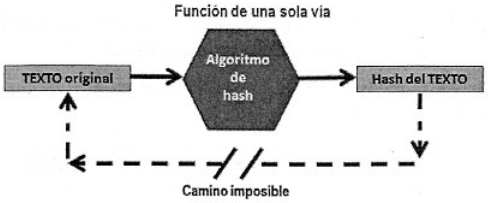
Es muy importante **hacer una lista** con las cosas que se deben y no hacer a la hora de aplicar codigo seguro	
Lo que NO se debe de hacer	**Lo que SI se debe de hacer**
Escribir código que utilize nombres de ficheros relativos	La referencia a un nombre de fichero debe de ser completa
Referirse dos veces en el mismo programa a un fichero por su nombre	Se crecomienda abrir el fichero una vez por su nombre y utilizar el identificador a partir de entonces
Invocar programas no confiables dentro de los programas	
Asumir que los usuarios no son maliciosos	
Dar por sentado el éxito	Cada vez que se realice una llamada al sistema (abrir un fichero, leer un fichero, etc) comprobar el valor de retorno por si la llamada falló
Invocar un shell o una línea de comandos *(Se refiere a un sistema que proporcione acceso a los servicios del kernel)*	
Autenticarse en criterios que no sean de confianza	
Utilizar áreas de almacenamiento con permisos de escritura	Hay que suponer que la información puede ser manipulada, alterada o destruida por cualquier persona o proceso que así lo desee
Guardar datos confidenciales en bases de datos sin protección de contraseña	
Hacer eco de las contraseñas o mostrarlas en la pantalla del usuario	
Emitir contraseñas via e-mail	
Distribuir mediante programación informacion confidencial a través de correo electrónico	
Guardar las contraseñas sin cifrar en disco en un formato facil de leer	Se debe de utilizar en su lugar certificados, encriptación fuerte, o transmisión segura entre host de confianza
Transmitir entre los sistemas contraseñas sin encriptar	Se debe de utilizar como antes, certificados, encriptación fuerte, o transmisión segura entre host de confianza
Tomar decisiones de acceso basadas en variables de entorno o parámetros de línea de comandos que se pasan tiempo en ejecución	
Evitar confiar en el software o los servicios de terceros para operaciones críticas	

Resumen conceptual PSP Segunda evaluación

Técnicas y mecanismos de seguridad más importantes

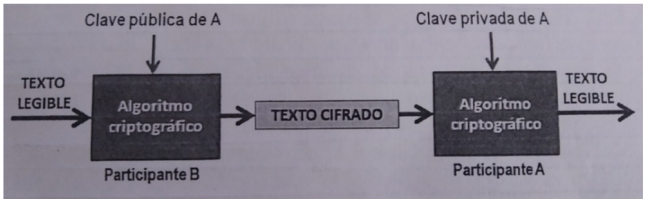
Criptografía:

- Su objetivo es ocultar el significado de un mensaje mediante el cifrado o codificación del mensaje
- Existen tres clases de algoritmos criptográficos:
  - **Funciones de una sola vía** (o funciones Hash)
    - Permiten mantener la integridad de los datos
    - Son de naturaleza matemática
    - Tienen un amplio abanico de usos
    - Cualquier protocolo las utiliza
    - Los dos algoritmos más utilizados son: **MD5** *(Message Digest)* y **SHA-1**
    - Su funcionamiento:
      - *Dado un mensaje "x" se calcula el resultado de f(x)*
      - *A ese f(x) se le denomina **hash de x** o **message digest de x***
      - *Es practicamente imposible calcular x a partir de f(x)*



  - **Algoritmos de clave secreta o de criptografía simétrica**
    - El emisor y el receptor comparten una clave
    - Esta clave se utiliza tanto para encriptar como para desencriptar
    - Los algoritmos más populares son:
      - **DES** *(Data Encryptation Standar)*
        - Claves de 56 bits y cifrado bloques de 64
      - **Triple DES** (variante de DES)
        - Claves de 128 bits (112 de clave y 16 de paridad)
      - **AES** *(Advance Encryption Standard)*
        - Tamaño variable siendo 256 bits el estándar
    - *Lo esencial es la clave K*

  - **Algoritmos de clave pública o de criptografía asimétrica**
    - El emisor emplea una **clave pública** difundida por el receptor
    - El receptor emplea una **clave privada** para desencriptar y sólo el puede hacerlo
    - El algoritmo más popular es: **RSA**
      - Su uso es universal como método de autentificación y firma digital y componente de control de protocolos y sistemas
    - Su funcionamiento:
      - *Se basa en que cada participante genera una pareja de claves relacionadas entre sí*
        - *Una pública (que todo el mundo puede conocer) y una privada (propia de cada participante)*
      - *Cualquiera que conozca la clave pública podría cifrar pero descifrarlo solo puede ser posible con la clave privada*

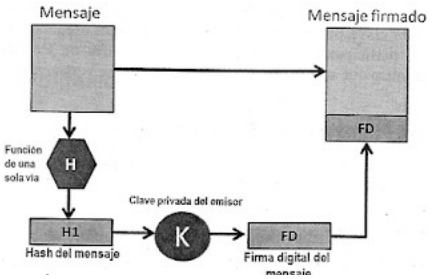


Resumen conceptual PSP Segunda evaluación

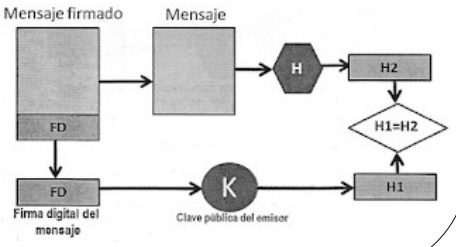
	simétrica	asimétrica
puntos fuertes	– Cifra mas rápido que la clave asimétrica – Sirve como base para los sistemas criptográficos basados en hardware	– Permiten conseguir autenticación y no repudio para muchos protocolos criptográficos – Pueden emplearse junto con otros métodos – Permiten una administración sencilla de claves
puntos débiles	– Requiere un sistema de distribución de claves muy seguro – Si la clave cae en manos no autorizadas deja de funcionar, por lo que requiere una administración compleja – El número total de claves crece rápidamente con el número de usuarios	– Son algoritmos más lentos que los de clave simetrica por lo que no se usan para cifrar grandes cantidades de datos – Sus implementaciones estan hechas en sistemas de software – Para una gran red de usuarios y/o máquinas se requiere un sistema de certificación de la autenticidad de las claves

Firma digital:

- Compuesto por una serie de datos asociados a un mensaje
- Permiten asegurar la identidad del firmante y la integridad del mensaje
- El método más extendido es el **RSA**
- **La firma** sigue el siguiente procedimiento:
  1. El emisor genera un hash del mensaje mediante una función acordada
     - Este mensaje es cifrado con su clave privada y el resultado es lo que se conoce como firma digital
  2. El emisor envia el mensaje junto a su firma digital al receptor



- **La comprobación** por parte del receptor sigue este procedimiento:
  1. Separa el mensaje recibido de la firma
  2. Genera el resumen del mensaje usando la misma función que el emisor
  3. Descifra la firma mediante la clave pública obteniendo el hash original
  4. Si los dos resúmenes coinciden, el mensaje a sido enviado por quien dice ser



Certificado digital:

- Documento que certifica que una entidad determinada tiene una clave pùblica determinada. El certificado es capa de enlazar la clave pública junto a ese nombre del titular
- Para certificar estos documentos se acude a las **Autoridades de Certificacion (AC)** (*de alta confianza)*
- Al aplicar el algoritmo de firma digital al documento se obtiene un texto, una secuencia de datos
- Tiene un formato estándar universalmente aceptado como **X.509**:
  - **Versión**
  - **Número de serie**
  - **Algoritmo de firma y parámetros**
  - **Emisor del certificado**
  - **Fechas de inicio y final**
  - **Nombre del propietario de la clave pública**
  - **Identificador del algoritmo que se está utilizando**
  - **La firma digital de la AC**
- Principales aplicaciones:
  - Autentificar la identidad del usuario
  - Tramites electrónicos ante organismos públicos
  - Trabajar con facturas electrónicas
  - Firmar digitalmente e-mails y todo tipo de documentos
  - Cifrar datos para que sólo el destinatario pueda acceder a su contenido

Resumen conceptual PSP Segunda evaluación

Control de acceso:

- Permitiendo la entrada a usuarios correctamente autenticados y impidiendosela a los demás
- Es más que un simple requerimiento de nombres de usuarios y contraseñas
- Tiene tres componentes importantes:
  - **Identificacion:**
    - El sujeto suministra la información diciendo quien es
  - **Autentificación:**
    - Cualquier proceso que verifique que alquien es quien dice ser
  - **Autorizacion:**
    - Proceso que determina, una vez autentificado, que el sujeto tiene acceso a los recursos
- Debería de recoger un registro de todas las entradas e intentos fallidos

Estas medidas se centran en estas tres formas:
– *Contraseñas:* Algo que se sabe (la mas extendida)
– *Biometría:* Algo que se es
– *Access Tokens:* Algo que se tiene

Seguridad en el entorno Java

- El punto en el que se implementa la seguridad interna de Java es antes de que la máquina virtual de Java comience con el proceso de interpretación
- Hay tres componentes del proceso:
  1. **Cargador de clases:**
     - Responsable de encontrar y cargar los bytecodes que definen las clases
     - Como mínimo tres cargadores:
       - **Bootstrap:**
         - Carga las clases del sistema desde el fichero JAR
       - **Extensión:**
         - Carga una extension estándar desde jre/lib/ext
       - **Aplicación:**
         - Localiza las clases en los ficheros según lo establecido en el CLASSPATH
  2. **Verificador de ficheros de clases:**
     - Válida los bytecodes
     - Una comprobación de ejemplo sería:
       - Que las llamadas a un método coincidan con los tipos de referencias de objeto
  3. **Gestor de seguridad:**
     - Controla si está permitida una determinada operación
     - Un ejemplo sería:
       - Que el hilo actual pueda actuar a un paquete específico

Por defecto no se instala de forma automática ningún gestor de seguridad cuando se ejecuta una aplicación Java. Para instalar un gestor de seguridad en nuestro programa:

- Incluir al iniciar la opción **-Djava.security.manager**
- Invocar al método **setSecurityManager ( )** de la clase System

Tras ésto, al ejecutar el programa se cara por defecto un **archivo de políticas** predeterminado y otorga todos los permisos al código para acceder a algunas propiedades tales como **os.name** y **java.version**. (Estas propiedades no son sensibles a la seguridad, por lo que su concesión no representa un riesgo)

Existen otras propiedades como **user.home** y **java.home** las cuales no están entre las propiedades a las que se le otorga permiso de lectura al fichero de politicas. Al intentar acceder a ellas lanzará una excepción: *AccessControlException*

La plataforma Java define un conjunto de APIS que abarca las principales áreas de seuridad incluyendo la criptografía, clave pública, autenticación, comunicación segura y control de acceso. Éstas permiten a los desarrolladores integrar fácilmente la seguridad en sus aplicaciones

Resumen conceptual PSP Segunda evaluación

APIS para la seguridad en las aplicaciones		
JCA *Arquitectura Criptográfica de Java*	**JSSE** *Extensión de Sockets Seguros*	**JAAS** *Servicio de Autentificación y Autorización de Java*
Infraestructura para la ejecución de los principales **servicios de cifrado**, incluye: – firmas digitales, – resúmenes de mensajes hash, – certificados y validación de certificados, – encriptacion (simétrico y asimétrico), – generación y gestión de claves – generacion segura de números aleatorios	Conjunto de paquetes Java provistos para la comunicación segura de Internet. Implementa una versión Java de los protocolos **SSL y TLS,** además incluye funcionalidades como: – cifrado de datos, – autenticación del servidor, – integridad de mensajes – autenticación del cliente	Interfaz que permite a las aplicaciones Java . Pduede usarse con dos fines: 1. Conocer quien está ejecutando el código 2. Garantizar que quién lo esté ejecutando tiene los permisos para ello

Fichero de políticas predeterminado java.policy

- Están situados en la carpeta java.home/conf/security/
- Especifica los permisos disponibles para el código en varias fuentes
- Utilizado para conceder permisos al sistema
- Contiene una secuencia de entrada **grant**
  - Cada una de ellas tiene una o más entradas de permisos
  - Su signficado:
    - **codeBase:**
      - Ubicación del código base sobre el que se van a definir los permisos
      - Su valor es una URL
      - Si se omite, los permisos se aplican a todas las fuentes
    - **Nombre_clase:**
      - Nombre de la clase de los permisos
    - **Nombre_destino:**
      - Especifica el destino del permiso, dependiendo de la clase de permiso
    - **Acción:**
      - Indica una lista de acciones separadas por comas
        - *read, write, delete o execute : java.ioFilePermission*
        - *accept, listen, connect, resolve : java.net.SocketPermission*
        - *read, write : java.util.PropertyPermission*

Herramienta policytool:

- Sirve para editar los ficheros de politicas
- Se recomienda su uso ya que verifican la sintaxis de su contenido
- Los errores provocan una excepcion *accessControlEsception*
- Para ejecutarla se escribe la orden policytool desde la línea de comandos
- Se abre una interfaz donde se pueden agregar, editar o borrar permisos

Resumen conceptual PSP Segunda evaluación

Criptografía con JCA:

- El **API JCA** incluye una extension **JAVAJCE** que incluye a su vez dos componentes de software:
  1. El marco que define y soporta los servicios criptográficos
     - Incluye paquetes como:
       - **Java.security.***
       - **java.crypto**
  2. Proveedores reales que contienen las implementaciones criptográficas
     - El proveedor es el encargado de proporcionar la implementación de uno o varios algoritmos
     - Se definen en **java.security**
     - Un ejemplo sería: Sun
- Está estructurado en torno a algunas clases e interfaces centrales de propósito general
  - La funcionalidad detrás de estas interfaces es proporcionada por los proveedores
  - Se puede utilizar la clase **Cipher** para cifrar y descifrar algunos datos
- Define el concepto de proveedor mediante la clase **Provider** del paquete **Java.security**
  - Se trata de una clase abstracta que debe ser redefinida por clases proveedor especificas
  - Tiene metodos para acceder a diferentes datos, como el proveedor, el numero de version y otra información

La función hash o message digest:

- Es una marca digital de un bloque de datos
- Los más conocidos son **MD5** y **SHA-1**
- La **clase MessageDigest:**
  - Permite a las aplicaciones implementar algoritmos de resumen de mensajes criptográficos seguros
    - Esos resumenes toman una entrada de tamaño arbitrario y generan un resultado de tamaño fijo ◀ **DIGEST**
      - Tiene dos propiedades:
        - Inviabilidad en encontrar dos mensajes con el mismo valor
        - No debe revelar nada sobre la entrada utiliada para generarlo
    - Son utilizados para generar identificadores de datos únicos y confiables ◀ **CHECKSUMS**
    - Los cambios en un solo bit del mensaje deberían producir un valor de resumen diferente
    - Para crear un objeto MessageDigest se utiliza el metodo **getInstance (String algoritmo)**

Generando Firmas digitales:

- Para crear una firma digital se necesita una clave privada y una publica
- En algunos casos estas se encuentran en ficheros, en otras pueden ser importadas y utilizadas para firmar y en otros casos necesitan ser generadas
  - **Clase KeyPairGenerator:**
    - Permite generar ese par de claves
    - Dispone de un constructor protegido
    - Para crear un objeto se invoca al método **getInstance (String algoritmo)**
  - **Clase KeyPair:**
    - Clase soporte para generar claves

Ambas clases son interfaces que agrupan todas las interfaces de clave privada y pública

a) Creando las claves:

1. El primer paso para generar un par de claves es obtener un objeto generador
2. Se debe inicializar ese generador con el metodo **inicialize ( )**
3. Pasarle dos argumentos
   - El tamaño clave
     - Multiplo de 64
     - Si se pasa un numero erroneo se produce la excepción *InvalidParameterException*
   - Un generador de números aleatorios, **SecureRandom**
4. El ultimo paso es almacenarlas en los objetos **PrivateKey** y **PublicKey**

b) Firmando los documentos:

- Se realiza mediante la clase **Signature**
  - Se inicializa el objeto para firmar con una clae privada y se le asignan los datos que debe firmar
  - Los bytes resultantes de la firma se guardan con los datos firmados

c) Verificando la firma:

- Se crea e inicializa otro objeto **Signature** y se le da la clave pública correspondiente
- Si coinciden los datos el objeto Signature informa del éxito:
- Consta de tres fases:
  1. *Inicialiacion:*
     - **initVerify( )** para pública
     - **initSignt ( )** para privada
  2. *Actualización:*
     - **método update ( )**
  3. *Firma o verificacion:*
     - **sign ( )** para firma
     - **verify ( )** para verificar

Para almacenar la clave privada en disco ⇒ Es necesario codificarla en formato PKCS8 usando la clase **PKCS8EncodedKeySpec**

Para almacenar la clave pública en disco ⇒ Es necesario codificarla en formato X.509 usando la clase **X509EncodedKeySpec**

Para recuperar las claves de los ficheros ⇒ Se necesita la clase **KeyFactory**

<table>
<tr><th colspan="2">Pasos para encriptar y desencriptar (Ejercicio de entrega)</th></tr>
<tr><th>con clave secreta (Asimétrico)</th><th>con clave publica (Simétrica)</th></tr>
<tr><td>1. Se crea una clave secreta:
– Con el algoritmo AES y
– Tamaño 128bits
2. Se crea un objeto Cipher con el algoritmo elegido
3. Se inicializa el modo encriptacion con la clave creada
4. Se realiza el cifrado con el método doFinal ( )
5. Se configura el objeto Cipher para desencriptación
– Con la clave anterior usando el metodo doFinal ( )</td><td>1. En primer lugar se crea el par de claves
2. Se crea e objeto Cipher con el algoritmo
3. Se inicializa en modo encriptación co nla clave publica
4. Se realiza el cifrado con el método doFinal ( )
5. Se configura el objeto Cipher en modo desencriptacion:
– Con la clave privada usando el método doFinal( )</td></tr>
<tr><th>convinación de simétrico y asimétrico
para la solución de rapidez en grandes cantidades de información</th><th>con flujo de datos</th></tr>
<tr><td>1. El participante A:
– Crea una clave simétrica y la utiliza para encriptar
– Encripta la clave con la asimétrica del participante B
– Lo envía todo al participante B
2. El participante B:
– Utiliza su clave asimétrica y desencripta la simétrica
– Utiliza la simétrica para desencriptar el mensaje</td><td rowspan="2">Se realiza mediante las clases proporcionadas con JCA:
– CipherOutputStream: Compuesto por:
– Objeto Cipher, que debe estar inicializado con anterioridad y que procesa los datos
– OutputStream: escribe los datos en subyacente
– CipherInputStream: Compuesto por:
– Objeto Cipher, que debe estar inicializado con anterioridad y que procesa los datos
– InputStream: Provee de los datos al Cipher</td></tr>
<tr><td>El concepto clave de sesión es un término medio entre ambos cifrados, ya que combina las dos técnicas. Consiste en enerar una clave de sesión K y cifrarla usando la clave publica del receptor. El receptor descifra la clave de sesión usando su clave privada. El emisor y el receptor comparten una clave que sólo ellos conocen y pueden cifrar sus comunicaciones usando la misma clave de sesión</td></tr>
</table>

Resumen conceptual PSP Segunda evaluación

Comunicaciones seguras con Java.JSSE

- Utilizado cuando viajan por la red datos o información confidencial
- Utiliza los protocolos **SSL** *(Secure Sockets Layer)* y **TLS** *(Transport Layer Security)*
- JSE es un conjunto de paquetes que permiten el desarrollo de aplicaciones seguras
- Sus clases se encuentran en los paquetes **javax.net** y **javax.net.ssl**
- Tiene dos clases **SSLServerSocketFactory** y **SSLSocketFactory** (representan sockets seguros)
  - No tienen constructor
  - Tienen el método estático **getDefault ( )**
- Cuando dos sockets SSL quieren establecer conexión tienen que presentarse el uno al otro y comprobar que son de confianza, y si todo va bien la conexión se establece.
  - Para que exista confianza se debe de crear un certificado para cada socket con la herramienta **keytool,** la cual se debe de usar en cada caso como indica:
    - Para crear un certificado del servidor se utilizan las sigiuentes opciones:
      - **-genkey**
      - **-keyalg RSA**
      - **-alias servidor**
      - **-keystore Almacensrv** (Fichero que hará de almacén de certificados
      - **-storepass ServidorPassAlmacen**
    - Para que el servidor saque su certificado para presentarselo al cliente
      - **-exportcert**
      - **-keystore AlmacenSrv**
      - **-alias servidor**
      - **-file CertivificadoServ.cer**
      - **-storepass ServidorPassAlmacen**
    - El cliente debe:
      - **-importcert:**
      - **-alias servidor**
      - **-file CertificadoServ.cer**
      - **-keystore CliCertConfianza**
      - **-storepass ClientePassAlmacen**
- El programa servidor para obtener la información del certificado debe usar el método **getLocalCertificates ( )**
- Para indicar los certificados de confianza lo habitual es indicar los certificados de los sockets SSL desde código Java con una serie de clases (aunque puede utiliar **System.setProperty ( )**:
  - **KeyStore**
  - **KeyManagerFactory**
  - **TrustManagerFactory**
  - **SSLContext**

Control de acceso con Java.JAAS

Estas clases e interfaces están disponbles en:
javax. security.auth.*
javax.security.auth.callback.*
javax.security.auth.login.*
javax.security.auth.spi.*

- Están incolucradas las siguientes clases o interfaces:
  - **LoginContext**:
    - Contexto de inicio de sesisón mediante la creación de un **Subject**
    - Se realiza llamando al metodo **login( )**
  - **LoginModule:**
    - Es la interfaz que debe implementarse para definir y validar los mecanismos de autentificación
    - Se deben implementar los métodos: *inicialize ( ), login ( ), commit ( ), abort ( ) y logout ( )*
  - **Subject:**
    - Clase que presenta a un ente autenticable dentro de la aplicación: *entidad, usuario y sistema*
  - **Principal:**
    - Interfaz que representa los atributos que posee cada **Subject** recuperado
    - Un principal puede contener varios **Subject**
  - **CallBackHandler:**
    - Interfaz que se debe implementar cuando se necesar recibir información del usuario

Resumen conceptual PSP Segunda evaluación

Pasos para la autenticación	Pasos para la autorización
1. Creación de una instancia de **LoginContext ( )** 2. Uno o más **LoginModule** (basándose en el archivo de configuración de JAAS) 3. Instalación de cada LoginModule opcionalmente provista de un **CallBackHandler** 4. Invocación del método **login ( )** del **LoginContext ( )** el cual a su ve invocará el método del **LoginModule** 5. Los datos del usuario se obtienen por medio del **CallBackHandler** 6. El **LoginModule** comprueba los datos introducidos por el usuario y los valida 7. Si la validación tiene éxito el usuario queda autenticado	– Extiende la arquitectura de seuridad de Java centrada en el código y se basa en el uso de políticas de seguridad para especificar cuales son los permisos de control de acceso – Se otorgarán no solo en función del código que se esté ejecutando, sino también de quién lo essté ejecutando – Se crea un Subject como resultado que representará al usuario autenticado – Este subject se compone de un conjunto de principales, donde cada principal representa a un atributo del usuario – Para que la autoriación JASS tenga lugar se requiere: – El usuario debe autenticarse – En el fichero de politicas se deben configurar entradas para los participantes – Se debe asociar al Subject el contexto de control de acceso actual usando los métodos **doAS ( )** o **doAsPrivileged ( )** de la clase Subject