

Programación de Servicios y Procesos

Tema 4 Técnicas de Programación Segura

José Luis González Sánchez





Contenidos

1. Introducción a la seguridad en la comunicaciones
2. Cifrado Unidireccional
3. Cifrado Simétrico
4. Cifrado Asimétrico
5. Firma Digital
6. Certificado Digital
7. Cifrado de Sesión
8. Comunicaciones Seguras
9. SSL/TSL
10. Secure Sockets

Introducción a la Seguridad en las Comunicaciones

Cómo cifrar nuestra información



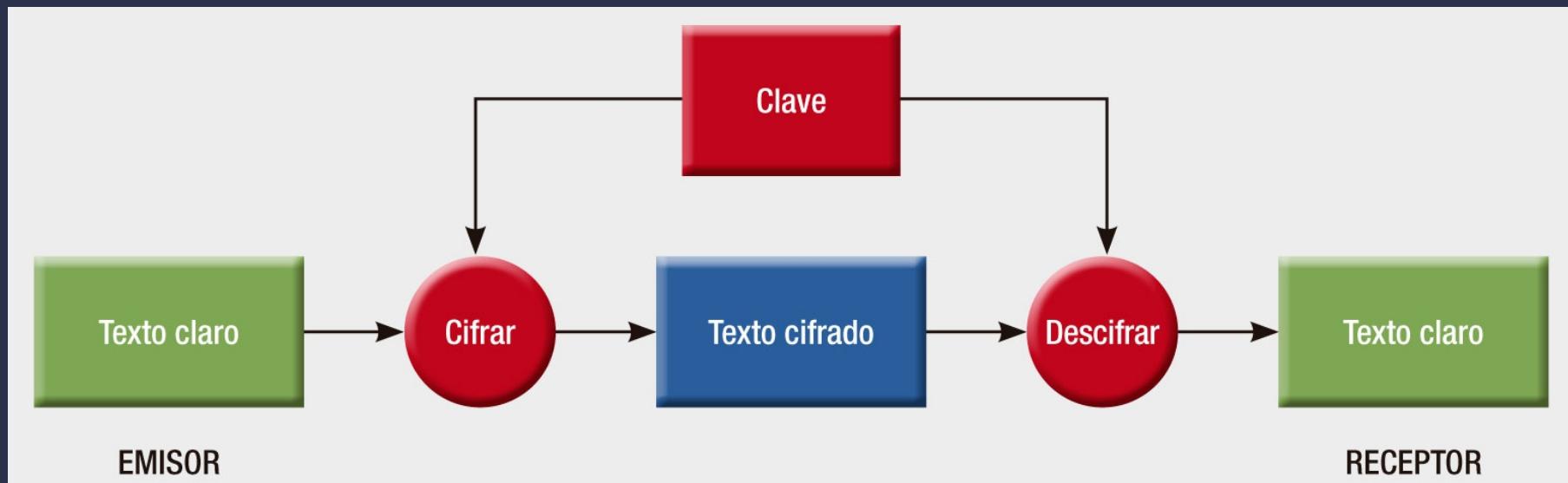
Sistemas criptográficos

- La **Criptografía** se ocupa de las técnicas de cifrado o codificado destinadas a alterar las representaciones lingüísticas de ciertos mensajes con el fin de hacerlos ininteligibles a receptores no autorizados.
- Por tanto el tipo de propiedades de las que se ocupa la criptografía son, por ejemplo:
 - **Confidencialidad:** garantiza que la información sea accesible únicamente a personal autorizado.
 - **Integridad:** garantiza la corrección y completitud de la información. Para conseguirlo puede usar por ejemplo funciones hash criptográficas MDC, protocolos de compromiso de bit, o protocolos de notarización electrónica.
 - **Vinculación:** permite vincular un documento o transacción a una persona o un sistema de gestión criptográfico automatizado. Cuando se trata de una persona, se trata de asegurar su conformidad respecto a esta vinculación (content commitment) de forma que pueda entenderse que la vinculación gestionada incluye el entendimiento de sus implicaciones por la persona. Antiguamente se utilizaba el término "No repudio"
 - **Autenticación:** proporciona mecanismos que permiten verificar la identidad del comunicador. Para conseguirlo puede usar por ejemplo función hash criptográfica MAC o protocolo de conocimiento cero.
 - Soluciones a problemas de la falta de simultaneidad en la telefirma digital de contratos. Para conseguirlo puede usar por ejemplo protocolos de transferencia inconsciente.



Sistemas criptográficos

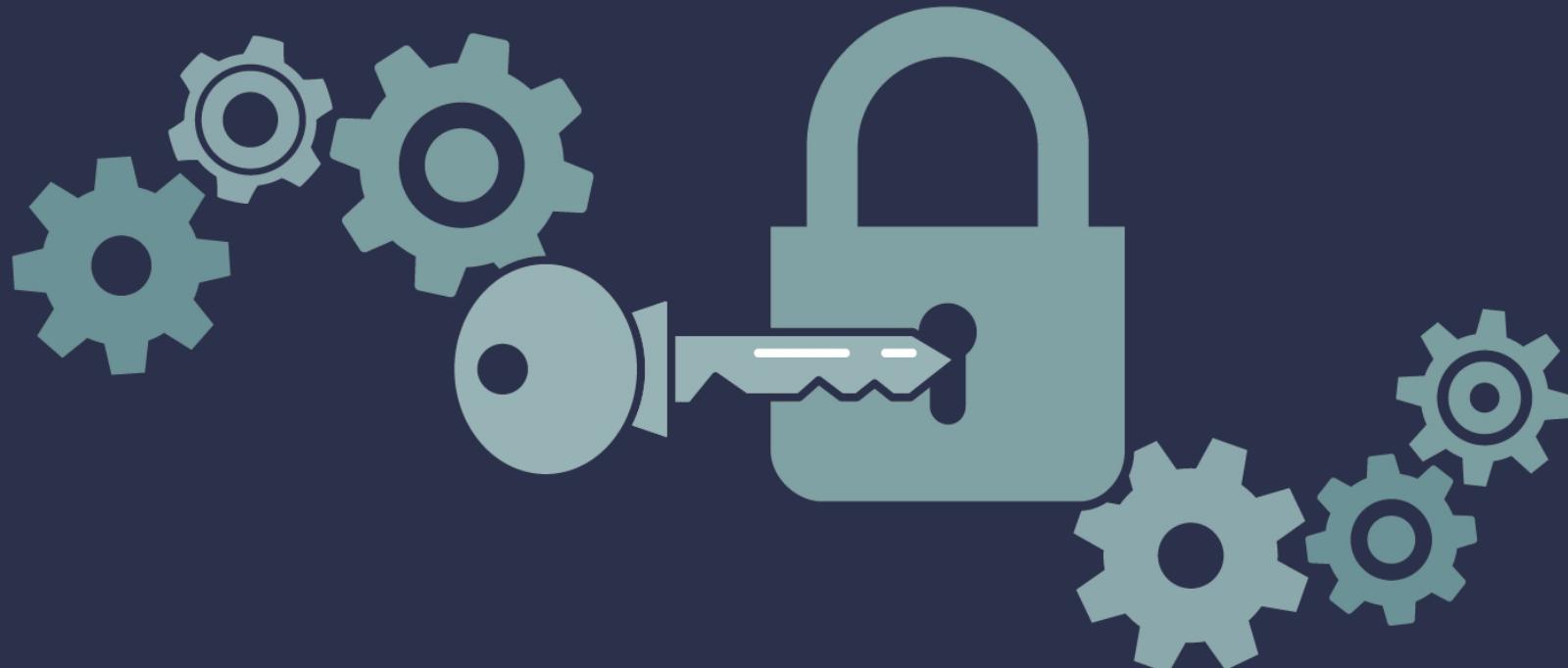
- Tipos de cífrados
 - **De una sola vía:** Una vez cifrados no se pueden descifrar. Se usa para comprobar la integridad del fichero o como parte de la firma digital. Ejemplo MD5 o SHA
 - **Simétricos:** Se usa la misma clave para cifrar y descifrar. Una sola clave, por ejemplo DES o AES
 - **Asimétricos:** Se usa una clave para cifrar y otra clave para descifrar, por lo tanto hay que tener un par de claves. Por ejemplo RSA





El proyecto

- El proyecto llamado criptografía, hace uso de una clase cifrador donde se ha programado las cosas más comunes con ejemplos para que practiquéis y os sirva de base para tener el código básico para futuros proyectos.
- Estúdialo con calma. Son tus apuntes.



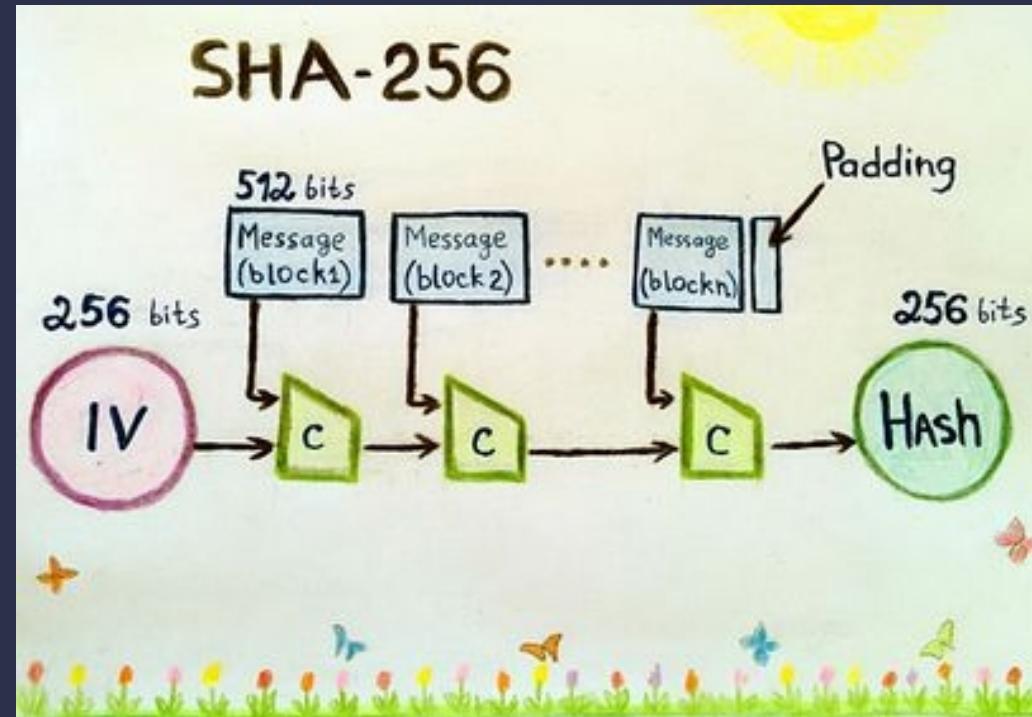
Cifrado Unidireccional

Código que identifica de forma inequívoca a un contenido



Cifrado HASH o una sola vía

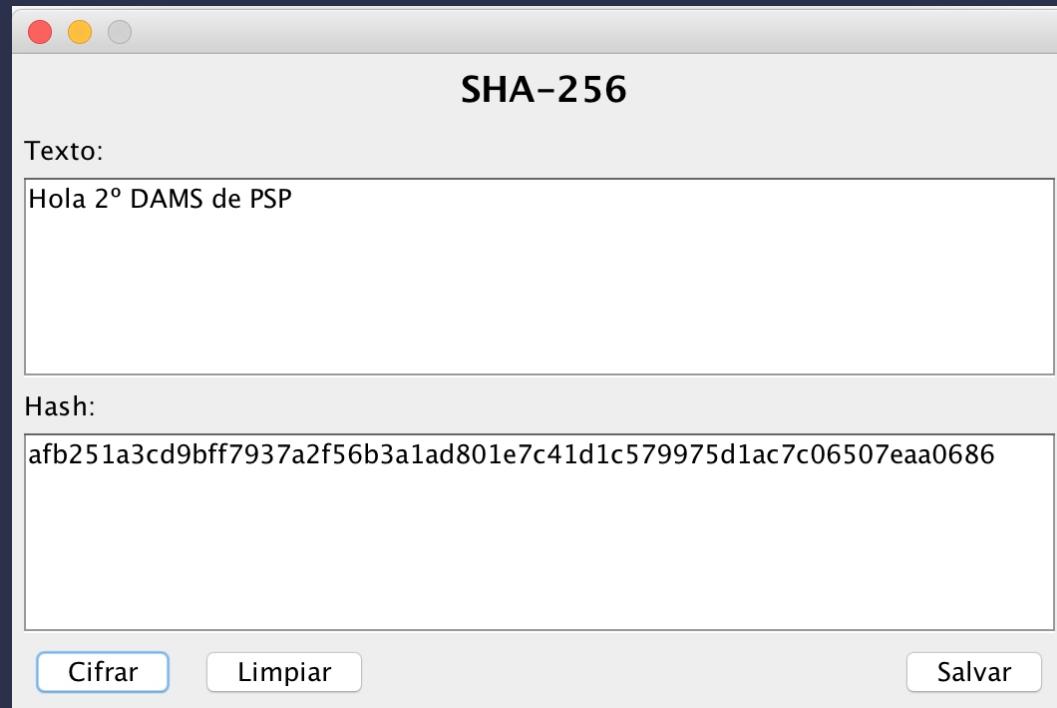
- Una vez cifrado “no se puede” descifrar. Nos sirve para calcular la integridad de un fichero. Muy común para firmas digitales o para saber si te has descargado bien un fichero. Usaremos SHA en vez de md5 porque es más moderno y seguro





Cifrado HASH o una sola vía

- Se usa la clase [MessageDigest](#) de Java.
- Le he hecho un apaño para que cincida con el SHA de PHP/MySQL y uso la versión 256/512 bits (se podría hacer fácilmente para cualquier otra es su instanciador Singleton. Puedes verlo en Cifrador->SHA256/512



Cifrado Simétrico

La misma clave para todo

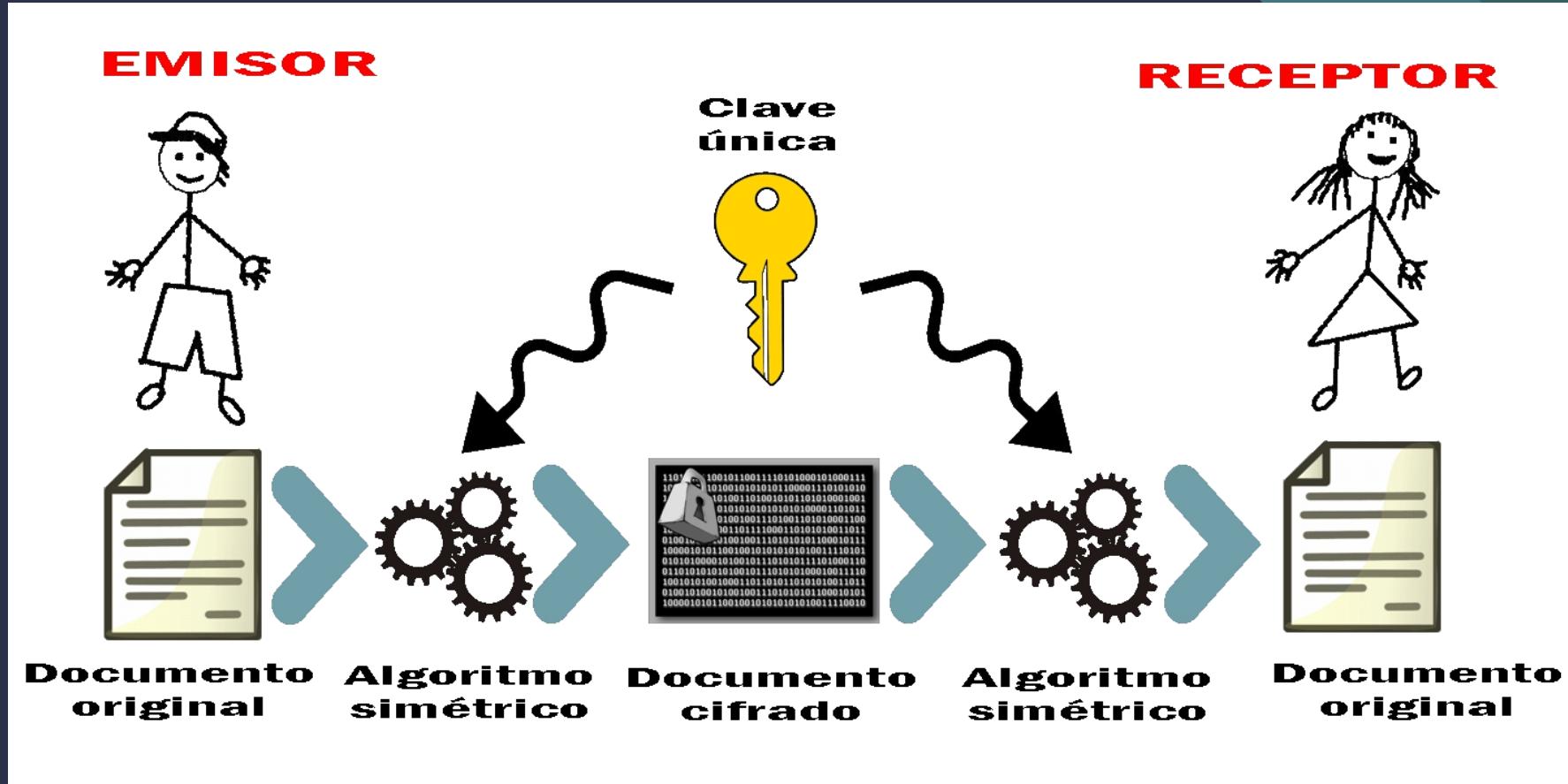


Cifrado simétrico

- Hay muchas opciones en [Java](#):
 - AES/CBC/NoPadding (128)
 - AES/CBC/PKCS5Padding (128)
 - AES/ECB/NoPadding (128)
 - AES/ECB/PKCS5Padding (128)
 - DES/CBC/NoPadding (56)
 - DES/CBC/PKCS5Padding (56)
 - DES/ECB/NoPadding (56)
 - DES/ECB/PKCS5Padding (56)
 - DESede/CBC/NoPadding (168)
 - DESede/CBC/PKCS5Padding (168)
 - DESede/ECB/NoPadding (168)
 - DESede/ECB/PKCS5Padding (168)
 - RSA/ECB/PKCS 1Padding (1024, 2048)
 - RSA/ECB/OAEPWithSHA-1AndMGF1Padding (1024, 2048)
 - RSA/ECB/OAEPWithSHA-256AndMGF1Padding (1024, 2048)
- Los modos son la forma de trabajar del algoritmo. El más sencillo es el modo **ECB** (*Electronic Cookbook Mode*), en el cual los mensajes se dividen en bloques y cada uno de ellos es cifrado por separado utilizando la misma clave K. La desventaja de este método es que a bloques de texto plano o claro idénticos les corresponden bloques idénticos de texto cifrado, de manera que se pueden reconocer estos patrones como guía para descubrir el texto en claro a partir del texto cifrado. De ahí que no sea recomendable para protocolos cifrados.
- En el modo **CBC** (*Cipher Block Chaining*), a cada bloque de texto plano se le aplica la operación XOR con el bloque cifrado anterior antes de ser cifrado. De esta forma, cada bloque de texto cifrado depende de todo el texto en claro procesado hasta este punto. Para hacer cada mensaje único se utiliza asimismo un vector de inicialización.

Cifrado Simétrico

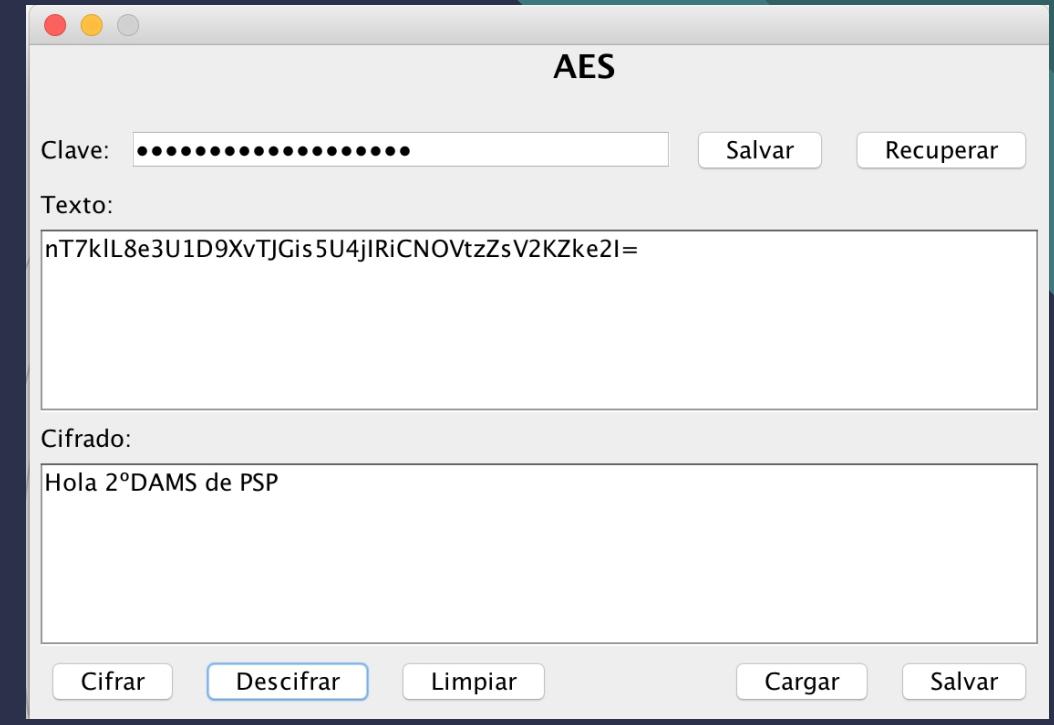
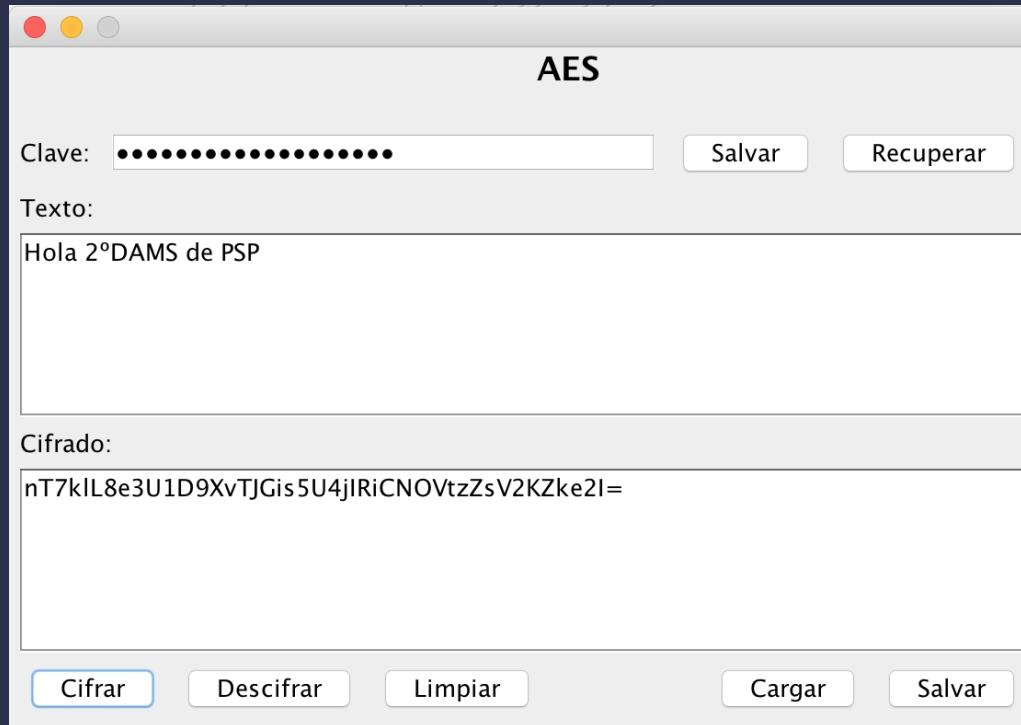
- Se usa la misma contraseña para cifrar y descifrar. Ojo no la vayas regalando por ahí





Cifrado simétrico

- La mejor opción es AES en algunos de sus modos. Ideal para ficheros. Para ello debemos crear una clave de la longitud requerida, luego iniciar el Cifrador indicándole cómo debe comportarse y cifrar/cifrar el mensaje y Puedes verlo funcionando en Cifrador->cifrarAES y Cifrador->descifrarAES



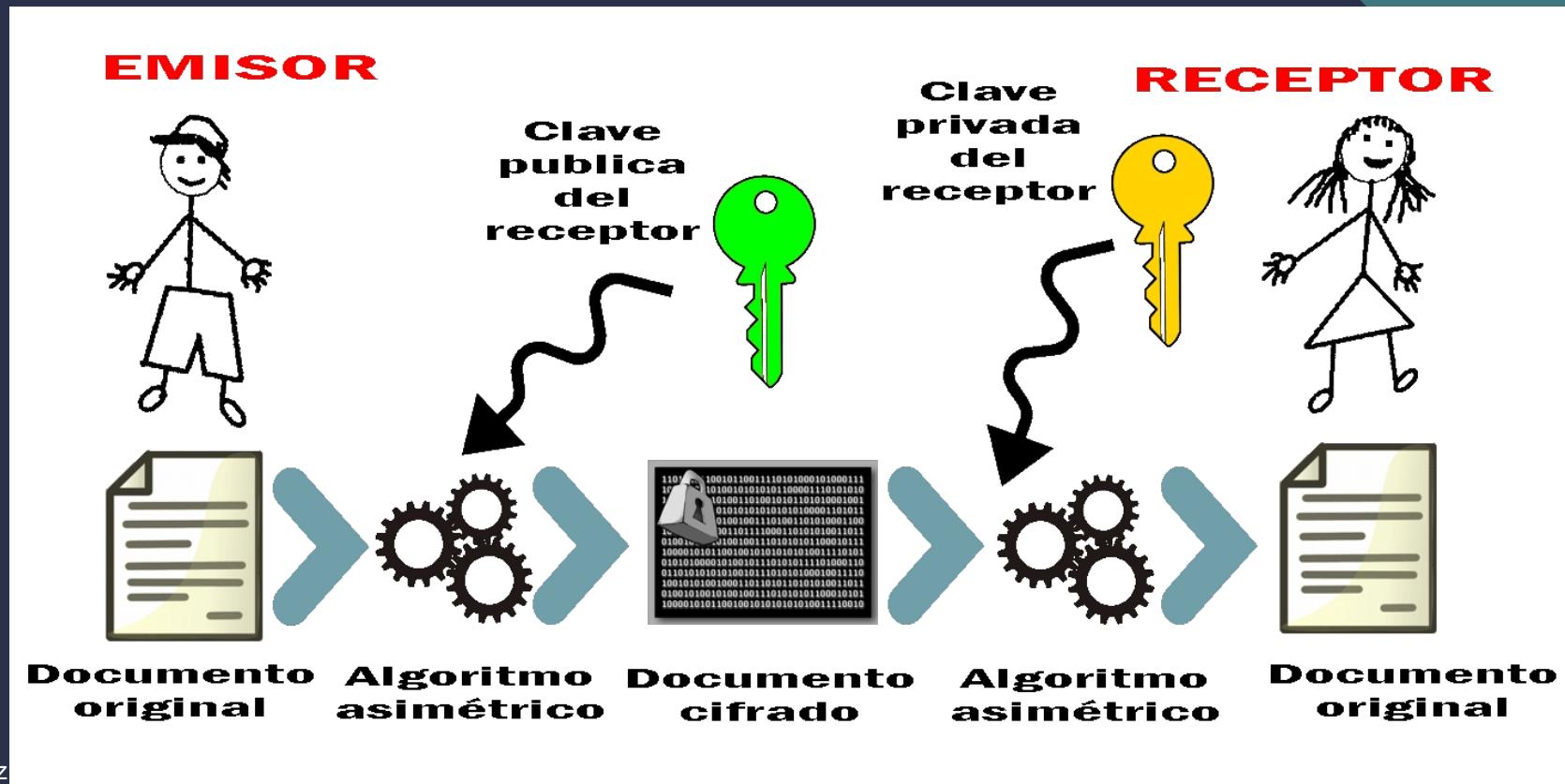
Cifrado Asimétrico

Un par de clave: Privada y Pública



Cifrado asimétrico

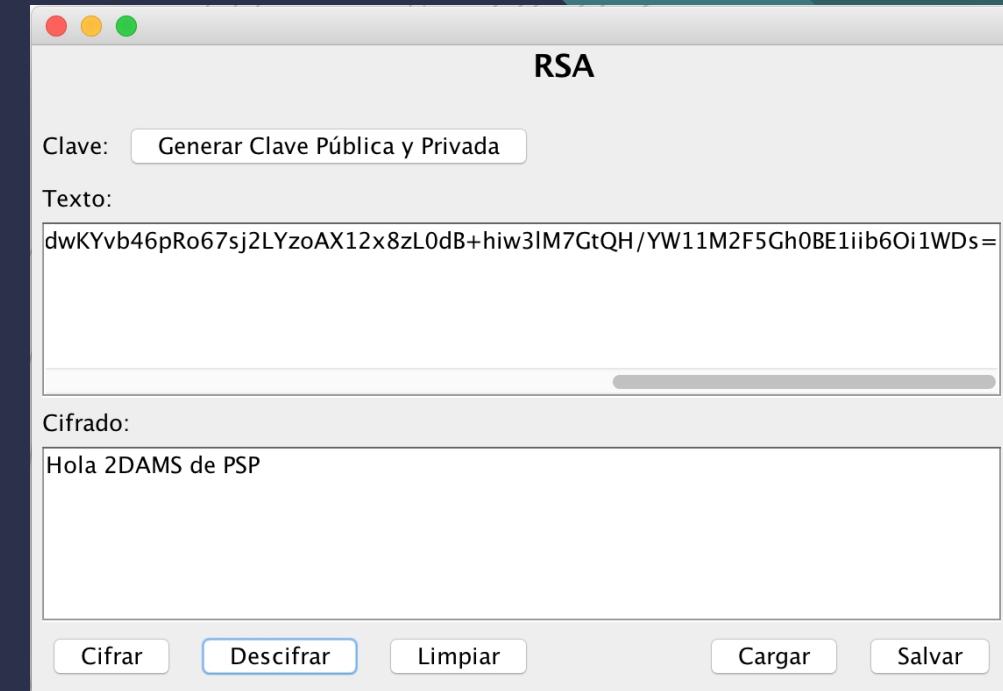
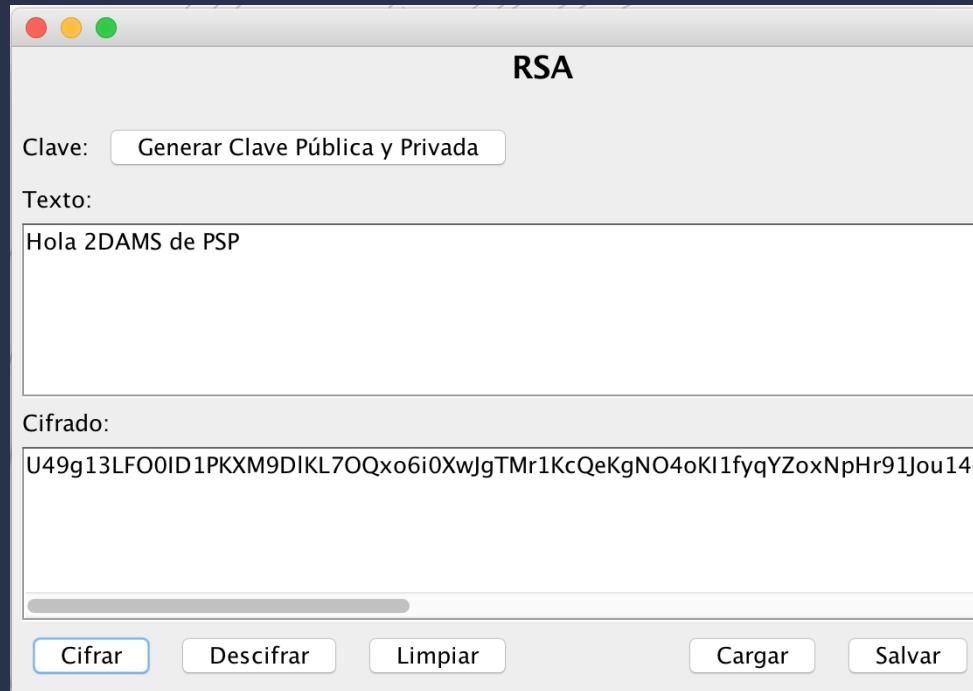
- Consiste en tener dos pares de claves: pública y privada para cifrar y descifrar.
- El emisor cifra con su clave pública del receptor (para eso es pública) y el receptor lo descifra con su clave privada. Solo podrá hacerlo él que tiene dicha clave privada (por lo tanto no la vayas dando por ahí, solo ofrece la pública).





Cifrado asimétrico

- La mejor opción es usar RSA. Pero ojo RSA se usa para cosas concretas, cuidado de cifrar ficheros muy grandes, pues tarda más de la cuenta y el tamaño puede ser mayor. Puede verlo en Cifrador->cifrarRSA y Cifrador->descifrarRSA, así como los métodos para crear/leer/salvar claves privadas y públicas (fundamental porque hay que generarlas)



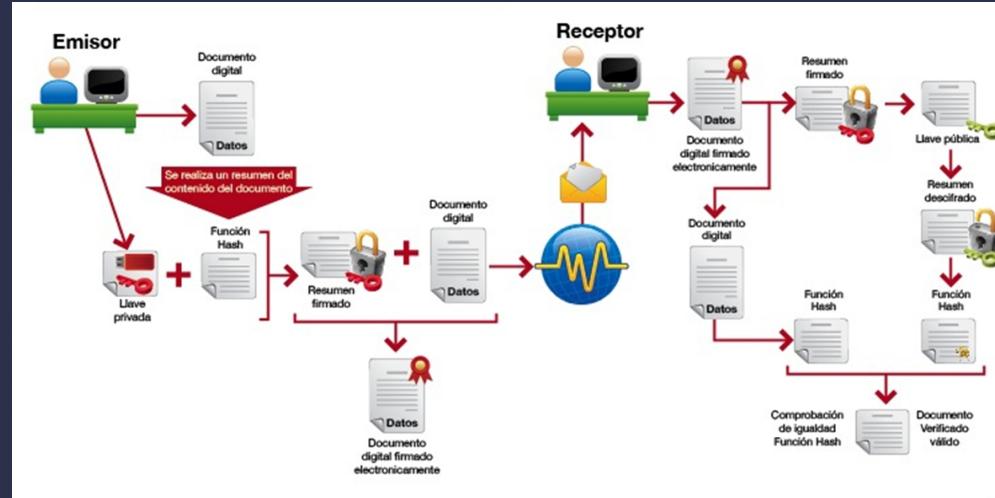
Firma Digital

Eres quien dices ser



Firma digital

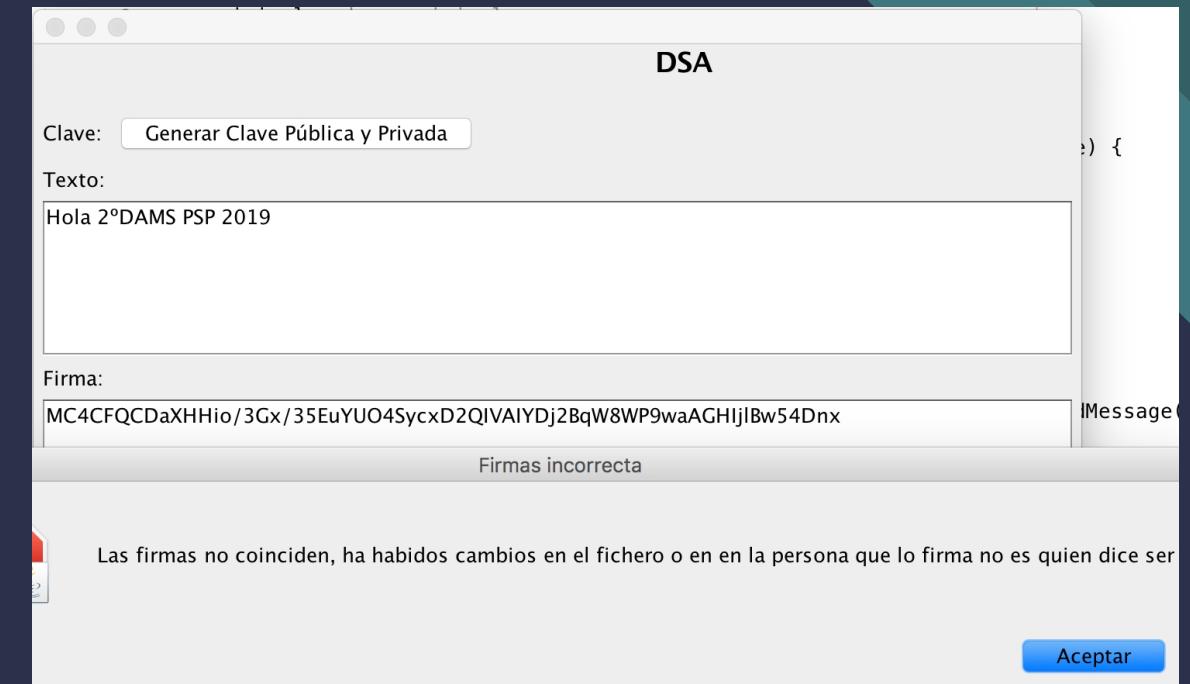
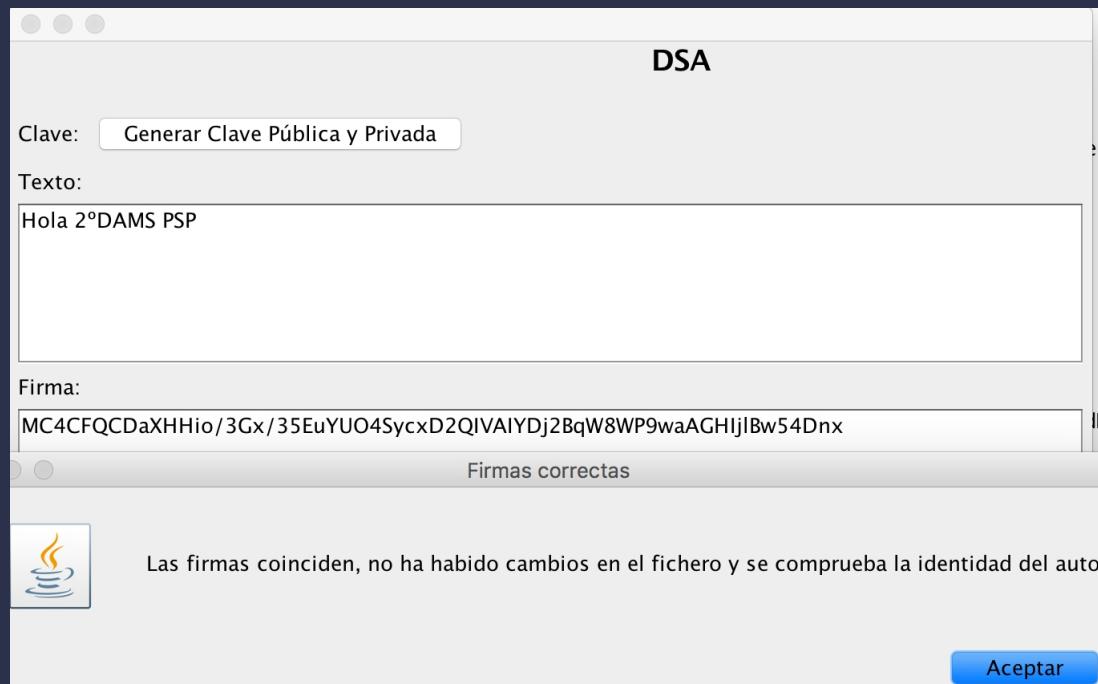
- Se puede decir que es una mezcla de cifrado asimétrico con funciones hash con el objetivo de:
 - Comprobar la integridad de la información (que no haya cambiado)
 - Comprobar la identidad del emisor
- El concepto es sencillo. El emisor, sobre el fichero a firmar se crea un resumen hash. Este resumen se cifra con nuestra clave privada y se adjunta junto al fichero original. El receptor recibe el mensaje y descifra con la clave pública del emisor (si se sabe ya que es él quien lo envía) posteriormente calcula el resumen hash del mensaje, si ambos resúmenes coinciden (el que ha descifrado y el que ha calculado) es que el mensaje no se ha modificado hasta la entrega





Firma digital

- Vamos a usar DSA, para ello debemos crear un juego de claves de firma pública y privadas y salvarlas. Puedes verlo en acción en Cifrador->firmarDSA y Cifrador->verificarDSA y sus métodos asociados para crear/leer/salvar claves privadas y públicas.



Certificado Digital

Tu identidad en la red



Certificado digital

- Un **certificado digital** o certificado electrónico es un fichero informático firmado electrónicamente por un **prestashop de servicios de certificación**, **considerado por otras entidades como una autoridad para este tipo de contenido**, que vincula unos datos de verificación de firma a un firmante, de forma que únicamente puede firmar este firmante, y confirma su identidad.
- Tiene una estructura de datos que contiene información sobre la entidad (por ejemplo una clave pública, una identidad o un conjunto de privilegios).
- La firma de la estructura de datos del certificado agrupa la información que contiene de forma que no puede ser modificada sin que esta modificación sea detectada.
- Veremos su uso en otra presentación y nos servirá para evitar para cifrar usando unas claves auto generadas, es otra opción existente, pero el proceso a realizar con ellos es el mismo que hemos visto: firmar, cifrar, claves de sesión, etc.



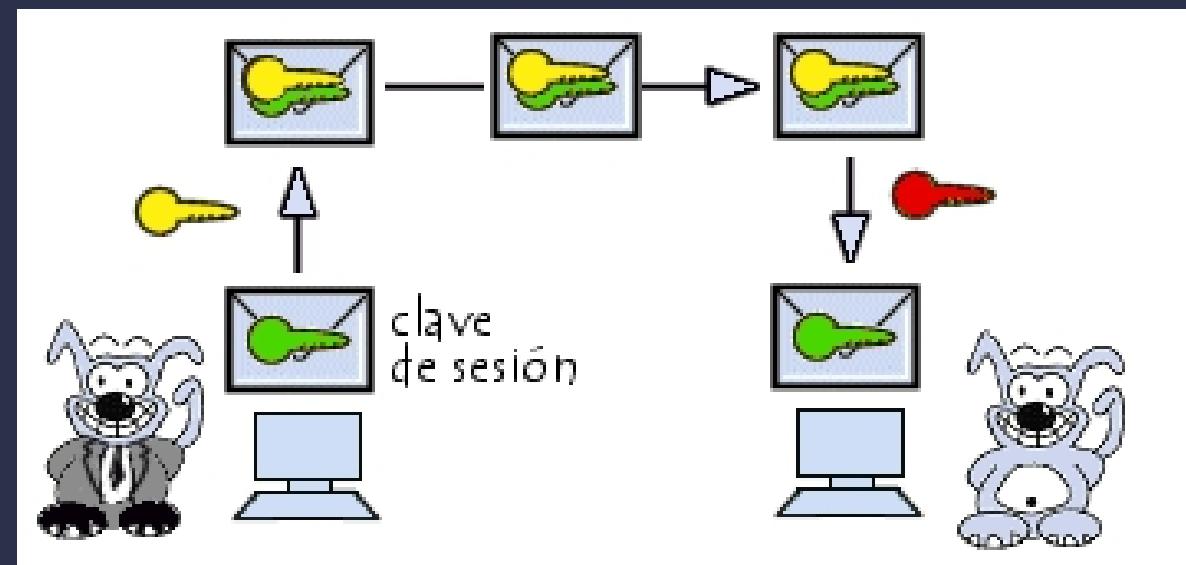
Cifrado de Sesión

Intercambiemos datos de forma segura



Claves de sesión

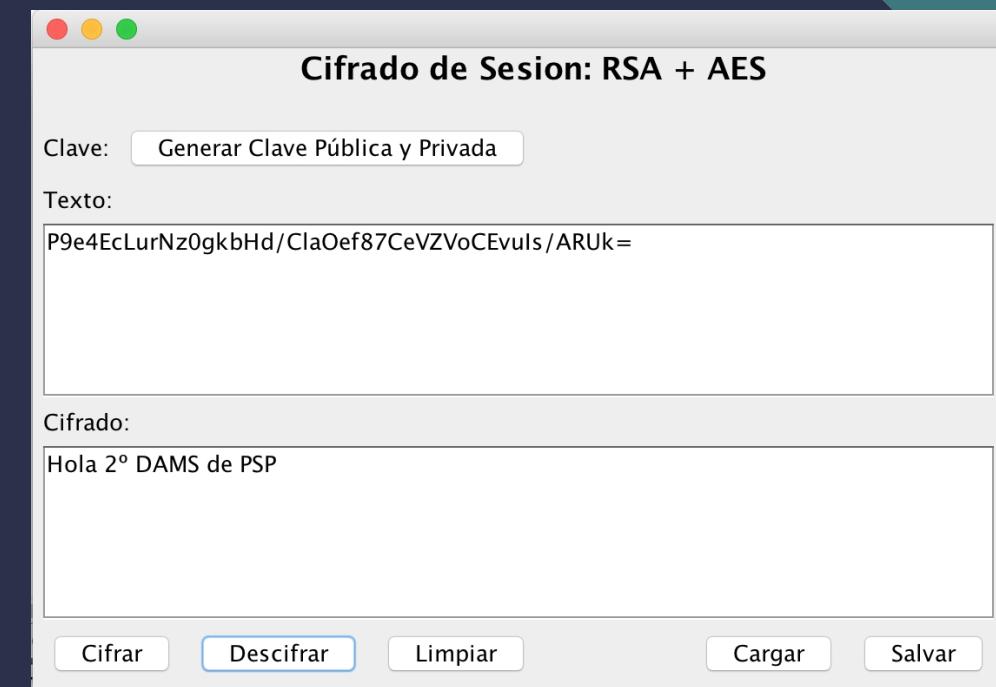
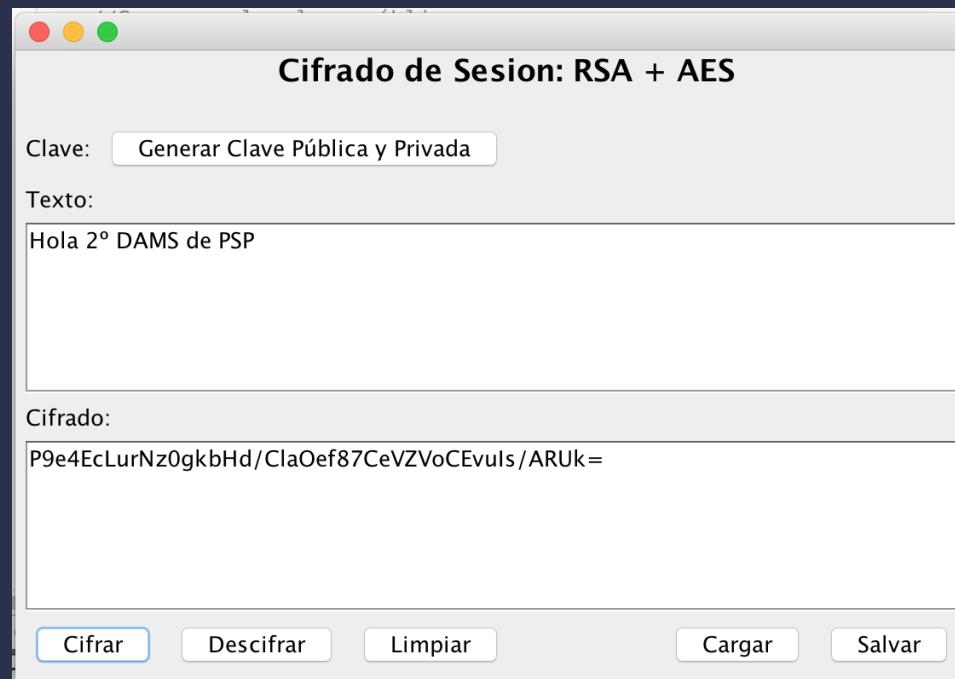
- Las claves de sesión se usan para intercambiar datos a través de un medio, por ejemplo socket y servicios webs (es decir, por medios de comunicación compartidos). Combina cifrado asimétrico y simétrico
- El funcionamiento es el siguiente. El emisor crea una clave de sesión compartida simétrica. Esta clave se cifra con la clave pública del receptor. El receptor recibe la clave privada cifrada y la descifra con su clave pública. DE esta manera ya han compartido su clave privada de sesión y la pueden utilizar para enviar y recibir la información que se intercambian de manera segura a través del medio compartido.





Claves de sesión

- En nuestro ejemplo se puede ver el funcionamiento en Cifrador->cifrarSesion y Cifrador->descifrarSesion. Como es un ejemplo la clave compartida secreta se crea y se almacena en un fichero. Pero debemos pensar que este método se debe hacer a través de un medio compartido y se envía y se almacena en memoria (lo que dure la sesión de intercambio de información)





Otros ejemplos

- Como he estado trabajando con texto y no ficheros por eso he utilizado Base64 para poder implementar encima otra codificación que me guarde cadenas. Ahora en los siguientes ejemplos veremos que no lo uso para trabajar directamente en binario
 - **Firmar/Verificar fichero:** Firma un fichero y obtiene el recibo de firma. Para verificar un fichero, cargamos el fichero y el recibo de firma y comprueba si coincide.
 - **Cifrar/Descifrar fichero AES.** Cifraremos/Descifraremos un fichero con AES
 - **Cifrar/Descifrar fichero con RSA.** Cifraremos/Descifraremos un fichero con RSA (poco recomendable)



Firmar/Verificar fichero

The screenshot shows a digital signature application window titled "Firmar / Validar Fichero". The window has three main sections: a "Clave:" field with a button "Generar Clave Pública y Privada", a "Fichero:" field containing the path "/Users/link/prueba.txt" with buttons "Abrir", "Firmar", and "Validar", and a status message "Firmas correctas" below it. The background of the application window is white, and the overall interface is clean and modern.

1 Hola 2º DAMS de PSP 2

2

3

4 LOG START

5 19:09:29 version 20180801-mac

6

7

8

9

10

11

12

13

Clave:

Fichero:

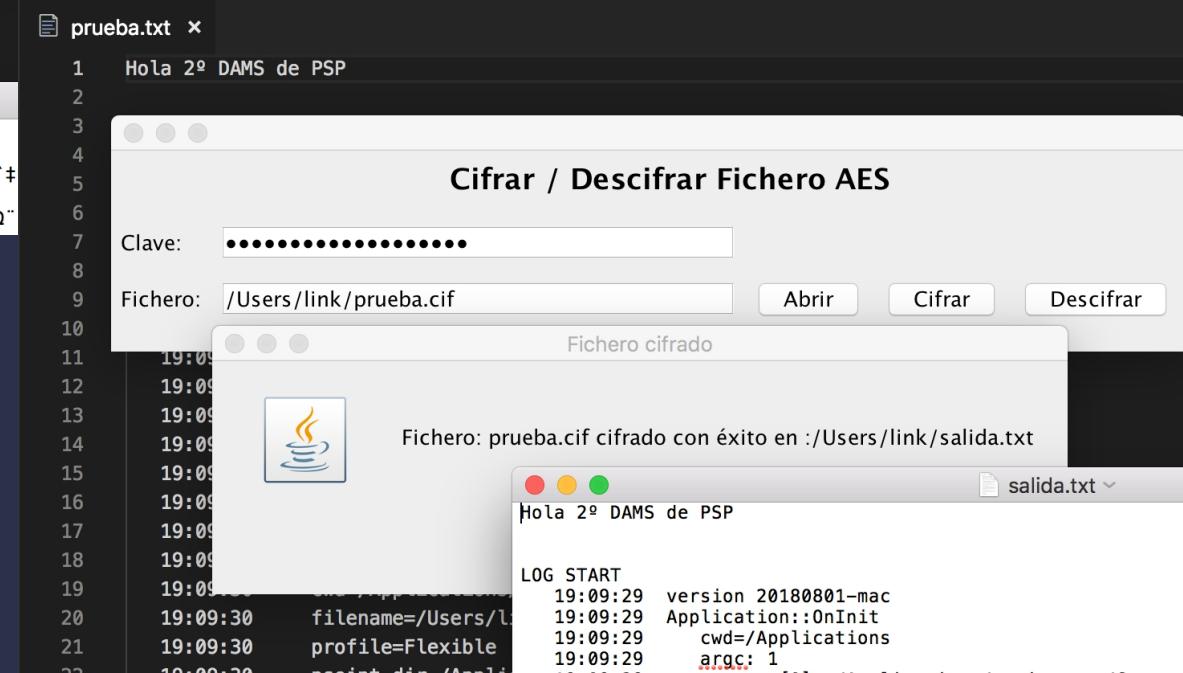
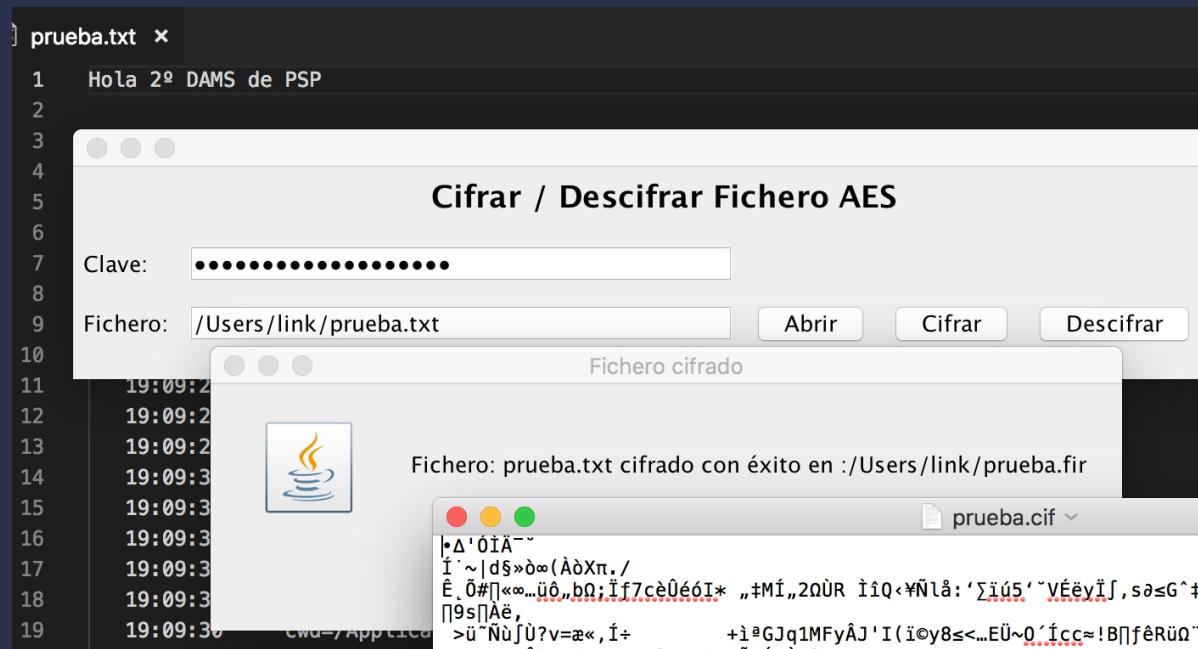
Firmas incorrecta

! Las firmas no coinciden, ha habidos cambios en el fichero o en la persona que lo firma no es quien dice ser

Aceptar



Cifrar/Descifrar fichero AES





Cifrar/Descifrar fichero RSA

- Echaros a dormir. ¿Por qué? RSA es muy lento, pues el proceso es costoso para hacerlo byte a byte, es por eso por lo que no se usa para este tipo de menesteres. ¿qué hacer? El sistema de sesiones empaquetado o usar el cifrado AES. Pero por probar que no quede.
- Pocas cosas veréis así de bestia.... Fracaso del profesor!!



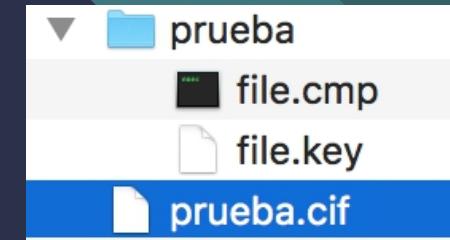
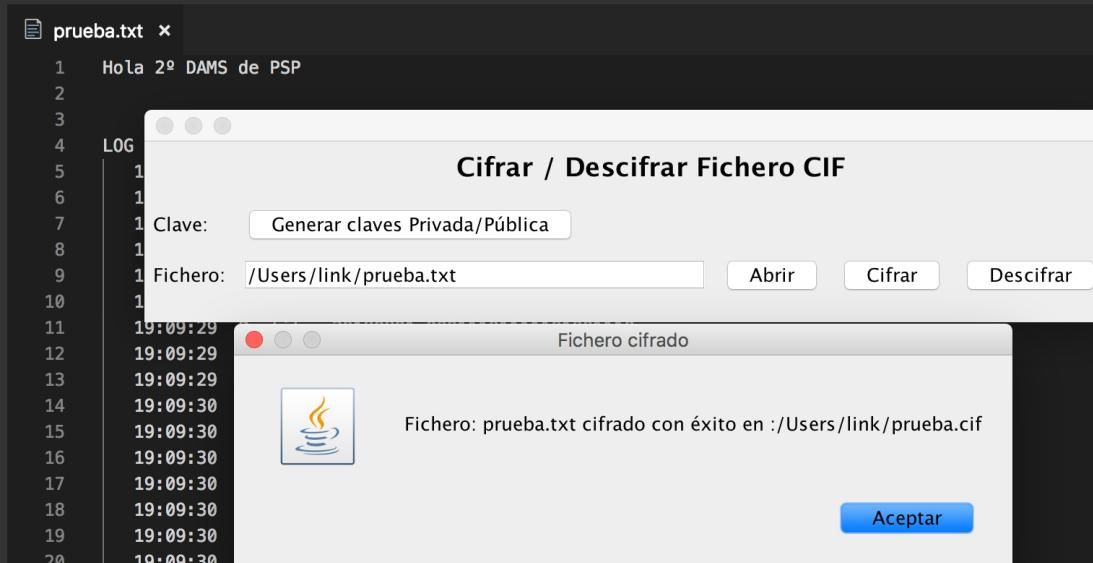


¿Y si...?

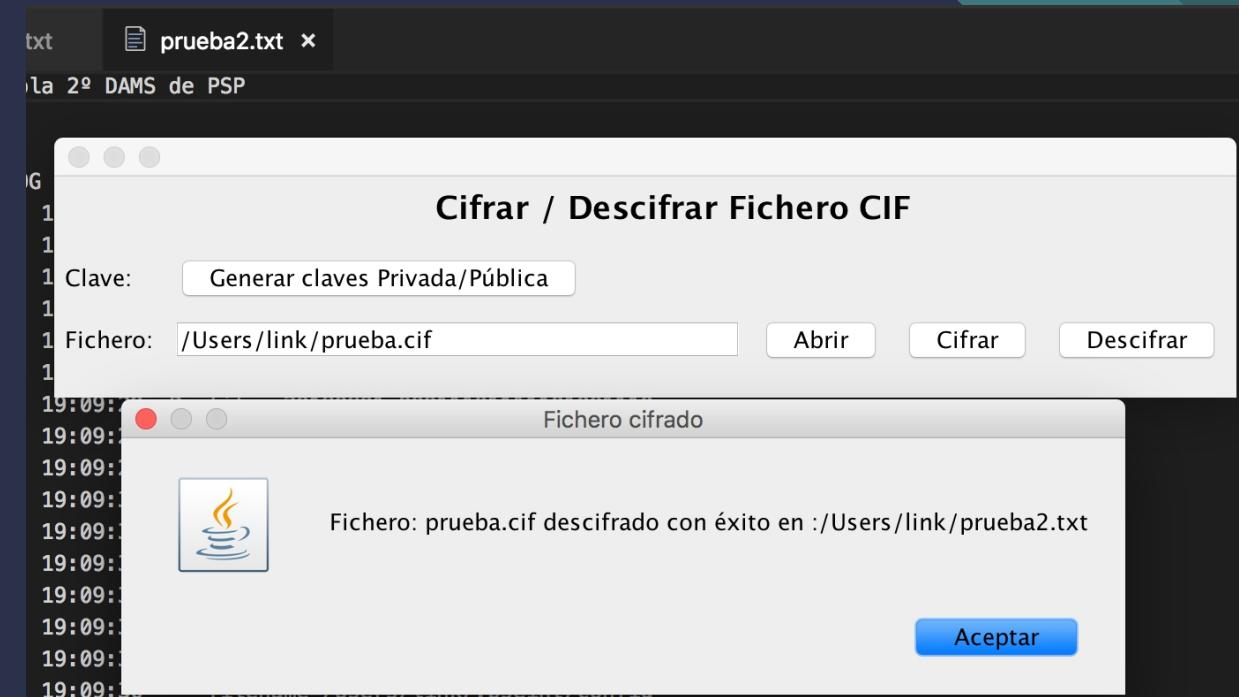
- Cogemos la idea de la clave de sesión. Es decir, ciframos el fichero con una clave de sesión AES y esta la ciframos con la clave publica del receptor. Lo empaquetamos en un solo fichero (.zip). Para abrirlo, lo descomprimimos, desciframos la clave de sesión con nuestra clave privada y el fichero con la clave de sesión. On fire!!!



¿Y si...?



Descomprimo el fichero cifrado, para que lo veáis.
Ambos por separado son inútiles



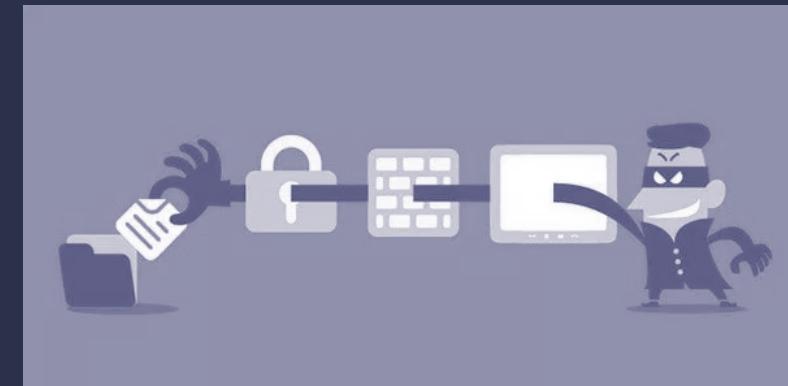
Comunicaciones Seguras

Asegurando nuestras comunicaciones por red



Comunicaciones seguras

- Existe un problema cuando trabajamos con servicios y procesos que estén en distintas máquinas (o a veces en la misma). Si estos servicios (o procesos) se están comunicando con otros a través de un espacio común (por ejemplo un puerto de comunicaciones de internet) estos son vulnerables.
- Esto que quiere decir, que cuando usamos una red o internet para intercambiar datos, dichos datos circulan libremente y es por ello que nuestra información puede ser interceptado, vista o robada por personas (o bots) que no deberían tener acceso a ella.
- Es fundamental que protejamos nuestras comunicaciones y los sistemas de transacción de nuestros mensajes dependiendo de la finalidad de los mismos: cuanto más seguros queramos que sea nuestro sistema más tendremos que esforzarnos en la seguridad de las comunicaciones y de los protocolos de transmisión/recepción de la información.
- No debemos escatimar. Nunca será suficiente





El proyecto

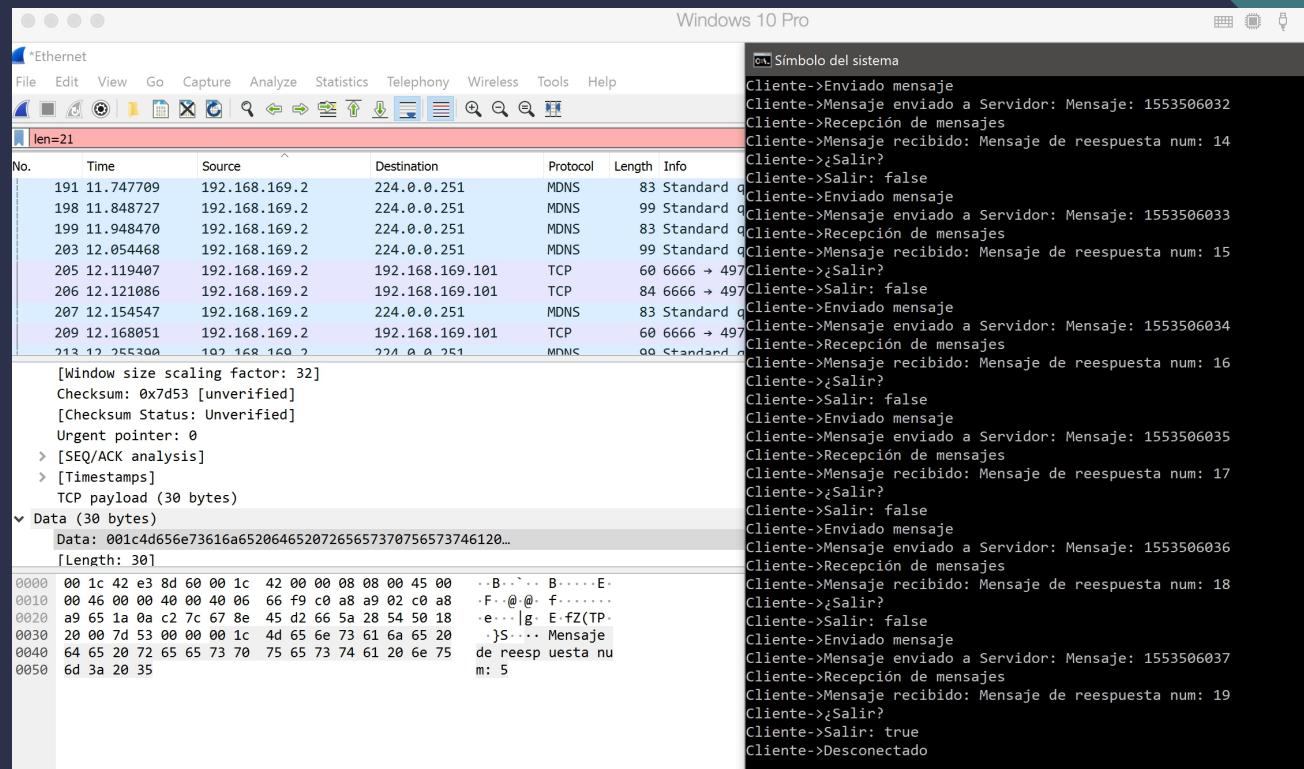
- Vamos a partir de un proyecto muy similar a los hechos ya anteriormente, un servidor al que se conectan uno o varios clientes a través de un puerto determinado e intercambian información. Luego veremos los problemas que tenemos cómo podemos solucionarlos
- Nuestro servidor escucha por el puerto 6666, recibe una cadena y responde con otra cadena y un número. Cuando haya recibido 20 mensajes indica al cliente que corte la conexión. El cliente se conecta, envía un mensaje y si recibe salir, sale, si no espera un segundo y repite hasta salir.
- Repasa y estúdialo con calma, hay distintas versiones según la solución





Problemas

- El utilizar la red y a hacerlo a través de un puerto, todos nuestros mensajes se desplazan libremente a través de ella, por lo que pueden ser fácilmente visualizados con un capturador de paquetes. Ver la versión 1





Problemas

Time	Source	Destination	Protocol	Length	Info
476 17.625926	192.168.169.2	192.168.169.101	TCP	85	6666 → 49788 [PSH, ACK] Seq=375 Ack=274 Win=262144 Len=31
477 17.659362	192.168.169.2	224.0.0.251	MDNS	83	Standard query response 0x0000 A 127.0.0.1
479 17.668599	192.168.169.2	192.168.169.101	TCP	60	6666 → 49788 [PSH, ACK] Seq=406 Ack=274 Win=262144 Len=1[Ma
481 17.758773	192.168.169.2	224.0.0.251	MDNS	99	Standard query response 0x0000 A 127.0.0.1 A 127.0.0.1
482 17.868038	192.168.169.2	224.0.0.251	MDNS	83	Standard query response 0x0000 A 127.0.0.1
483 17.968439	192.168.169.2	224.0.0.251	MDNS	99	Standard query response 0x0000 A 127.0.0.1 A 127.0.0.1
484 18.068884	192.168.169.2	224.0.0.251	MDNS	83	Standard query response 0x0000 A 127.0.0.1
485 18.172317	192.168.169.2	224.0.0.251	MDNS	99	Standard query response 0x0000 A 127.0.0.1 A 127.0.0.1
486 18.271538	192.168.169.2	224.0.0.251	MDNS	83	Standard query response 0x0000 A 127.0.0.1

Frame 476: 85 bytes on wire (680 bits), 85 bytes captured (680 bits) on interface 0
 Ethernet II, Src: Parallel_00:00:08 (00:1c:42:00:00:08), Dst: Parallel_e3:8d:60 (00:1c:42:e3:8d:60)
 Internet Protocol Version 4, Src: 192.168.169.2, Dst: 192.168.169.101
 Transmission Control Protocol, Src Port: 6666, Dst Port: 49788, Seq: 375, Ack: 274, Len: 31
 Data (31 bytes)

```
00 1c 42 e3 8d 60 00 1c 42 00 00 08 08 00 45 00 ..B...`.. B.....E.
00 47 00 00 40 00 40 06 66 f8 c0 a8 a9 02 c0 a8 ..G..@@.. f.....
a9 65 1a 0a c2 7c 67 8e 46 ad 66 5a 28 e7 50 18 ..e...|g.. F-fZ(.P.
20 00 49 e7 00 00 00 1d 4d 65 6e 73 61 6a 65 20 ..I..... Mensaje
64 65 20 72 65 65 73 70 75 65 73 74 61 20 6e 75 de reespuesta nu
6d 3a 20 31 32 m: 12
```

Wireshark · Packet 516 · Ethernet

- ▶ Frame 516: 85 bytes on wire (680 bits), 85 bytes captured (680 bits) on interface 0
- ▶ Ethernet II, Src: Parallel_00:00:08 (00:1c:42:00:00:08), Dst: Parallel_e3:8d:60 (00:1c:42:e3:8d:60)
- ▶ Internet Protocol Version 4, Src: 192.168.169.2, Dst: 192.168.169.101
- ▼ Transmission Control Protocol, Src Port: 6666, Dst Port: 49788, Seq: 439, Ack: 316, Len: 31
 - Source Port: 6666
 - Destination Port: 49788
 - [Stream index: 0]
 - [TCP Segment Len: 31]
 - Sequence number: 439 (relative sequence number)
 - [Next sequence number: 470 (relative sequence number)]
 - Acknowledgment number: 316 (relative ack number)
 - 0101 = Header Length: 20 bytes (5)
 - ▶ Flags: 0x018 (PSH, ACK)
 - Window size value: 8192

0000	00 1c 42 e3 8d 60 00 1c 42 00 00 08 08 00 45 00 ..B...`.. B.....E.
0010	00 47 00 00 40 00 40 06 66 f8 c0 a8 a9 02 c0 a8 ..G..@@.. f.....
0020	a9 65 1a 0a c2 7c 67 8e 46 ed 66 5a 29 11 50 18 ..e... g.. F-fZ(.P.
0030	20 00 47 7d 00 00 00 1d 4d 65 6e 73 61 6a 65 20 ..G}.... Mensaje
0040	64 65 20 72 65 65 73 70 75 65 73 74 61 20 6e 75 de reespuesta nu
0050	m: 14



Problemas

*Ethernet

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

Apply a display filter ... <Ctrl-/>

No.	Time	Source	Destination	Protocol	Length	Info
18	1.707835	192.168.169.101	239.255.255.250	SSDP	179	M-SEARCH * HTTP/1.1
46	4.358052	192.168.169.101	192.168.169.2	TCP	66	49788 → 6666 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
48	4.358334	192.168.169.101	192.168.169.2	TCP	54	49788 → 6666 [ACK] Seq=1 Ack=1 Win=525568 Len=0
50	4.364695	192.168.169.101	192.168.169.2	TCP	75	49788 → 6666 [PSH, ACK] Seq=1 Ack=1 Win=525568 Len=21
54	4.416079	192.168.169.101	192.168.169.2	TCP	54	49788 → 6666 [ACK] Seq=22 Ack=31 Win=525568 Len=0
56	4.462604	192.168.169.101	192.168.169.2	TCP	54	49788 → 6666 [ACK] Seq=22 Ack=32 Win=525568 Len=0
60	4.710755	192.168.169.101	239.255.255.250	SSDP	179	M-SEARCH * HTTP/1.1
69	5.428892	192.168.169.101	192.168.169.2	TCP	75	49788 → 6666 [PSH, ACK] Seq=22 Ack=32 Win=525568 Len=21
72	5.496165	192.168.169.101	192.168.169.2	TCP	54	49788 → 6666 [ACK] Seq=23 Ack=62 Win=525312 Len=0

> Frame 50: 75 bytes on wire (600 bits), 75 bytes captured (600 bits) on interface 0
> Ethernet II, Src: Parallel_e3:8d:60 (00:1c:42:e3:8d:60), Dst: Parallel_00:00:08 (00:1c:42:00:00:08)
> Internet Protocol Version 4, Src: 192.168.169.101, Dst: 192.168.169.2
▼ Transmission Control Protocol, Src Port: 49788, Dst Port: 6666, Seq: 1, Ack: 1, Len: 21
 Source Port: 49788
 Destination Port: 6666
 [Stream index: 0]
 [TCP Segment Len: 21]
 Sequence number: 1 (relative sequence number)
 [Next sequence number: 22 (relative sequence number)]
 0000 00 1c 42 00 08 00 1c 42 e3 8d 60 08 00 45 00 ..B.....B...`E..
 0010 00 3d 57 2e 40 00 80 06 00 00 c0 a8 a9 65 c0 a8 =W.@.....e..
 0020 a9 02 c2 7c 1a 0a 66 5a 27 d6 67 8e 45 37 50 18 ...|..fZ 'g.E7P.
 0030 08 05 d3 e8 00 00 00 13 4d 65 6e 73 61 6a 65 3a Mensaje:
 0040 20 31 35 35 33 35 30 36 30 31 36 1553506 016

Wireshark · Packet 50 · Ethernet

> Frame 50: 75 bytes on wire (600 bits), 75 bytes captured (600 bits) on interface 0
> Ethernet II, Src: Parallel_e3:8d:60 (00:1c:42:e3:8d:60), Dst: Parallel_00:00:08 (00:1c:42:00:00:08)
> Internet Protocol Version 4, Src: 192.168.169.101, Dst: 192.168.169.2
▼ Transmission Control Protocol, Src Port: 49788, Dst Port: 6666, Seq: 1, Ack: 1, Len: 21
 Source Port: 49788
 Destination Port: 6666
 [Stream index: 0]
 [TCP Segment Len: 21]
 Sequence number: 1 (relative sequence number)
 [Next sequence number: 22 (relative sequence number)]
 Acknowledgment number: 1 (relative ack number)
 0000 00 1c 42 00 08 00 1c 42 e3 8d 60 08 00 45 00 ..B.....B...`E..
 0010 00 3d 57 2e 40 00 80 06 00 00 c0 a8 a9 65 c0 a8 =W.@.....e..
 0020 a9 02 c2 7c 1a 0a 66 5a 27 d6 67 8e 45 37 50 18 ...|..fZ 'g.E7P.
 0030 08 05 d3 e8 00 00 00 13 4d 65 6e 73 61 6a 65 3a Mensaje:
 0040 20 31 35 35 33 35 30 36 30 31 36 1553506 016



Soluciones

- La primera solución sería usar un sistema de cifrado de clave simétrica...
 - MAL: deberían tener todos los clientes la claves y el servidor. Si nos enganchan la clave, se acabó. ¿Cómo la intercambiamos?



Soluciones

- Solo usar clave asimétrica RSA
 - MAL: sistema realmente lento que puede hacer desesperar a todo el mundo.



Soluciones: Cifrado de Sesión

- Sistema híbrido RSA/AES, clave de sesión
 - Buena opción. El cliente tiene el certificado del servidor, lo usa para cifrar una clave de sesión que se intercambia a través de la red.
 - Una vez intercambiada los datos cruciales se cifran y se transmiten.
 - Problemas: El cliente siempre tiene que tener la clave pública del servidor. Si esta cambia debe informar de ello.
- Vamos a realizar lo siguiente:
 - Cogemos el programa Cifrador, y en RSA generamos un par de claves
 - En el proyecto creamos la carpeta cert y copiamos dichas claves allí.
 - En el cliente, vamos a crear dicha carpeta también y copiamos solo la clave pública allí.
 - Ahora procedemos como uno de los ejemplos que ya hemos hecho con anterioridad en Cifrador, pero metiendo el socket por medio.
 - Al conectarnos, antes de enviar nada le enviamos al servidor una clave de sesión AES aleatoria cifrada con RSA (público)
 - El servidor recibe el mensaje, lo descifra con su clave privada y ya tiene la clave de sesión AES.
 - El resto de mensajes se cifra y descifra con dicha clave.
 - Usaremos métodos realizados en cifrador, repásalos si no los tienes claros.
 - Vamos a mandar todos los datos cifrados (y recibirlos) pero solo habría que hacerlo con los cruciales.
 - Ver la versión 2.



Soluciones: Cifrado de Sesión

The screenshot shows a Wireshark capture window titled "Ethernet" and a separate "Símbolo del sistema" window showing application logs.

Wireshark Window:

- File menu:** File, Edit, View, Go, Capture, Analyze, Statistics, Telephony, Wireless, Tools, Help.
- Toolbar:** File, Open, Save, Print, Capture, Analyze, Statistics, Tools, Help.
- Search Bar:** Expression... (highlighted).
- Table Headers:** No., Time, Source, Destination, Protocol, Length, Info.
- Table Data:** Shows 14 rows of network traffic. Row 72 is selected, showing details:
 - Checksum: 0x6648 [unverified]
 - [Checksum Status: Unverified]
 - Urgent pointer: 0
 - > [SEQ/ACK analysis]
 - > [Timestamps]
 - TCP payload (46 bytes)
 - Data (46 bytes):** Data: 002c65417068335842436e6a2f6c63584776447735317441...
 - Length: 461
- Hex and ASCII panes:** Below the table, showing the raw hex and ASCII representation of the selected packet's payload.

Símbolo del sistema Window:

- Logs corresponding to the captured traffic:
 - Cliente->Enviado mensaje
 - Cliente->Mensaje enviado a Servidor: Mensaje: 1553513125
 - Cliente->Recepción de mensajes
 - Cliente->Mensaje recibido: Mensaje de reespuesta num: 14
 - Cliente->¿Salir?
 - Cliente->Salir: false
 - Cliente->Enviado mensaje
 - Cliente->Mensaje enviado a Servidor: Mensaje: 1553513126
 - Cliente->Recepción de mensajes
 - Cliente->Mensaje recibido: Mensaje de reespuesta num: 15
 - Cliente->¿Salir?
 - Cliente->Salir: false
 - Cliente->Enviado mensaje
 - Cliente->Mensaje enviado a Servidor: Mensaje: 1553513127
 - Cliente->Recepción de mensajes
 - Cliente->Mensaje recibido: Mensaje de reespuesta num: 16
 - Cliente->¿Salir?
 - Cliente->Salir: false
 - Cliente->Enviado mensaje
 - Cliente->Mensaje enviado a Servidor: Mensaje: 1553513129
 - Cliente->Recepción de mensajes
 - Cliente->Mensaje recibido: Mensaje de reespuesta num: 17
 - Cliente->¿Salir?
 - Cliente->Salir: false
 - Cliente->Enviado mensaje
 - Cliente->Mensaje enviado a Servidor: Mensaje: 1553513130
 - Cliente->Recepción de mensajes
 - Cliente->Mensaje recibido: Mensaje de reespuesta num: 18
 - Cliente->¿Salir?
 - Cliente->Salir: false
 - Cliente->Enviado mensaje
 - Cliente->Mensaje enviado a Servidor: Mensaje: 1553513131
 - Cliente->Recepción de mensajes
 - Cliente->Mensaje recibido: Mensaje de reespuesta num: 19
 - Cliente->¿Salir?
 - Cliente->Salir: true
 - Cliente->Desconectado



Soluciones: Cifrado de Sesión

Time	Source	Destination	Protocol	Length	Info
112 8.471825	192.168.169.101	192.168.169.2	TCP	54	50298 → 6666 [ACK] Seq=271 Ack=217 Win=525312 Len=0
123 9.512999	192.168.169.101	192.168.169.2	TCP	100	50298 → 6666 [PSH, ACK] Seq=271 Ack=217 Win=525312 Len=46
124 9.513145	192.168.169.2	192.168.169.101	TCP	60	6666 → 50298 [ACK] Seq=217 Ack=317 Win=262080 Len=0
125 9.515113	192.168.169.2	192.168.169.101	TCP	100	6666 → 50298 [PSH, ACK] Seq=217 Ack=317 Win=262144 Len=46
127 9.570020	192.168.169.101	192.168.169.2	TCP	54	50298 → 6666 [ACK] Seq=317 Ack=263 Win=525312 Len=0
128 9.573506	192.168.169.2	192.168.169.101	TCP	80	6666 → 50298 [PSH, ACK] Seq=263 Ack=317 Win=262144 Len=26
129 9.616213	192.168.169.101	192.168.169.2	TCP	54	50298 → 6666 [ACK] Seq=317 Ack=289 Win=525312 Len=0
143 10.937444	192.168.169.101	192.168.169.2	TCP	100	50298 → 6666 [PSH, ACK] Seq=317 Ack=289 Win=525312 Len=46
144 10.937602	192.168.169.2	192.168.169.101	TCP	60	6666 → 50298 [ACK] Seq=289 Ack=363 Win=262080 Len=0

Checksum: 0x770a [unverified]

[Checksum Status: Unverified]

Urgent pointer: 0

➤ [SEQ/ACK analysis]

➤ [Timestamps]

TCP payload (46 bytes)

Data (46 bytes)

Data: 002c65417068335842436e6a2f6c63584776447735317443...

[Length: 46]

```
00 00 1c 42 e3 8d 60 00 1c 42 00 00 08 08 00 45 00 ..B...`.. B.....E.
10 00 56 00 00 40 00 40 06 66 e9 c0 a8 a9 02 c0 a8 .V..@..@. f.....
20 a9 65 1a 0a c4 7a 53 f7 67 6a 67 f8 23 af 50 18 .e...zS. gjg.#.P.
30 20 00 77 0a 00 00 00 2c 65 41 70 68 33 58 42 43 .w..., eAph3XBC
40 6e 6a 2f 6c 63 58 47 76 44 77 35 31 74 43 6c 33 nj/lcXGv Dw51tC13
50 6c 2b 59 78 59 35 56 35 59 76 50 50 45 6d 48 30 1+YxY5V5 YvPPEmH0
60 4c 67 38 3d Lg8=
```

Wireshark · Packet 125 · Ethernet

- Frame 125: 100 bytes on wire (800 bits), 100 bytes captured (800 bits) on interface 0
- Ethernet II, Src: Parallel_00:00:08 (00:1c:42:00:00:08), Dst: Parallel_e3:8d:60 (00:1c:42:e3:8d:60)
- Internet Protocol Version 4, Src: 192.168.169.2, Dst: 192.168.169.101
- Transmission Control Protocol, Src Port: 6666, Dst Port: 50298, Seq: 217, Ack: 317, Len: 46
 - Source Port: 6666
 - Destination Port: 50298
 - [Stream index: 0]
 - [TCP Segment Len: 46]
 - Sequence number: 217 (relative sequence number)
 - [Next sequence number: 263 (relative sequence number)]
 - Acknowledgment number: 317 (relative ack number)

0000	00 1c 42 e3 8d 60 00 1c 42 00 00 08 08 00 45 00	..B...`.. B.....E.
0010	00 56 00 00 40 00 40 06 66 e9 c0 a8 a9 02 c0 a8	.V..@..@. f.......
0020	a9 65 1a 0a c4 7a 53 f7 67 6a 67 f8 23 af 50 18	.e...zS. gjg.#.P.
0030	20 00 77 0a 00 00 00 2c 65 41 70 68 33 58 42 43	.w..., eAph3XBC
0040	6e 6a 2f 6c 63 58 47 76 44 77 35 31 74 43 6c 33	nj/lcXGv Dw51tC13
0050	6c 2b 59 78 59 35 56 35 59 76 50 50 45 6d 48 30	1+YxY5V5 YvPPEmH0
0060	4c 67 38 3d	Lg8=



Soluciones: Cifrado de Sesión

*Ethernet

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

tcp

No.	Time	Source	Destination	Protocol	Length	Info
112	8.471825	192.168.169.101	192.168.169.2	TCP	54	50298 → 6666 [ACK] Seq=271 Ack=217 Win=525312 Len=0
123	9.512999	192.168.169.101	192.168.169.2	TCP	100	50298 → 6666 [PSH, ACK] Seq=271 Ack=217 Win=525312 Len=46
124	9.513145	192.168.169.2	192.168.169.101	TCP	60	6666 → 50298 [ACK] Seq=217 Ack=317 Win=262080 Len=0
125	9.515113	192.168.169.2	192.168.169.101	TCP	100	6666 → 50298 [PSH, ACK] Seq=217 Ack=317 Win=262144 Len=46
127	9.570020	192.168.169.101	192.168.169.2	TCP	54	50298 → 6666 [ACK] Seq=317 Ack=263 Win=525312 Len=0
128	9.573506	192.168.169.2	192.168.169.101	TCP	80	6666 → 50298 [PSH, ACK] Seq=263 Ack=317 Win=262144 Len=26
129	9.616213	192.168.169.101	192.168.169.2	TCP	54	50298 → 6666 [ACK] Seq=317 Ack=289 Win=60 Len=46
143	10.937444	192.168.169.101	192.168.169.2	TCP	100	50298 → 6666 [PSH, ACK] Seq=317 Ack=289 Win=60 Len=46
144	10.937603	192.168.169.2	192.168.169.101	TCP	60	6666 → 50298 [ACK] Seq=289 Ack=317 Win=60 Len=46

Checksum: 0xd401 [unverified]
[Checksum Status: Unverified]
Urgent pointer: 0
↳ [SEQ/ACK analysis]
↳ [Timestamps]
TCP payload (46 bytes)

↳ Data (46 bytes)
Data: 002c526875744671694a575875644c52776f453170687653...
[Length: 46]

Hex View:

```

0000  00 1c 42 00 00 08 00 1c  42 e3 8d 60 08 00 45 00  ..B.....B...`..E..
0010  00 56 58 58 40 00 80 06  00 00 c0 a8 a9 65 c0 a8  .VXX@.....e...
0020  a9 02 c4 7a 1a 0a 67 f8  23 af 53 f7 67 b2 50 18  ...z..g. #.S.g.P.
0030  08 04 d4 01 00 00 00 2c  52 68 75 74 46 71 69 4a  ......., RhutFqij
0040  57 58 75 64 4c 52 77 6f  45 31 70 68 76 53 50 62  WXudLRwo E1phvSPb
0050  69 32 66 79 35 71 4d 4b  68 46 54 70 56 63 63 6b  i2fy5qMK hFTPvCck
0060  2b 70 6b 3d              +pk=

```

Frame 143: 100 bytes on wire (800 bits), 100 bytes captured (800 bits) on interface 0
Ethernet II, Src: Parallel_e3:8d:60 (00:1c:42:e3:8d:60), Dst: Parallel_00:00:08 (00:1c:42:00:00:08)
Internet Protocol Version 4, Src: 192.168.169.101, Dst: 192.168.169.2
Transmission Control Protocol, Src Port: 50298, Dst Port: 6666, Seq: 317, Ack: 289, Len: 46
Source Port: 50298
Destination Port: 6666
[Stream index: 0]
[TCP Segment Len: 46]
Sequence number: 317 (relative sequence number)
[Next sequence number: 363 (relative sequence number)]
Acknowledgment number: 289 (relative ack number)

```

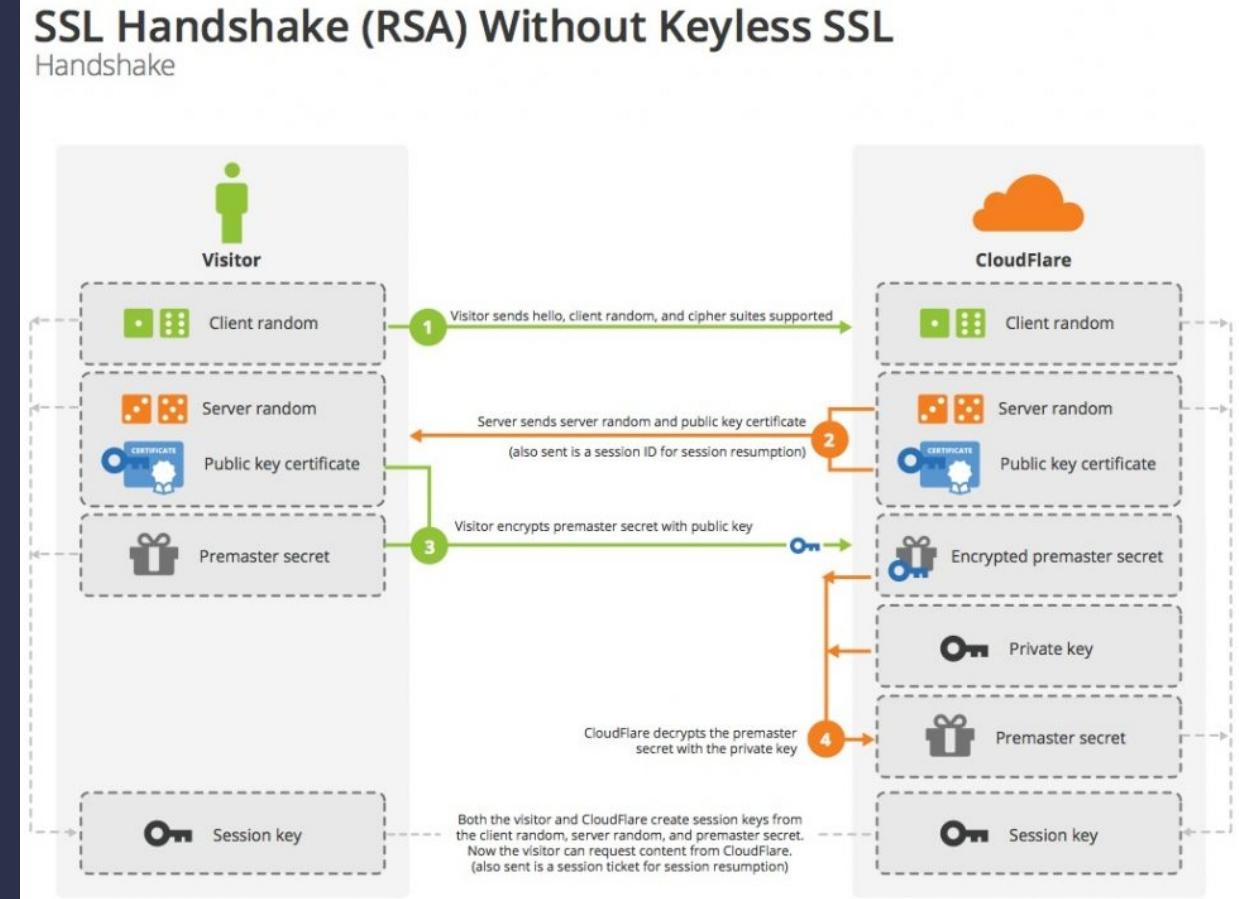
0000  00 1c 42 00 00 08 00 1c  42 e3 8d 60 08 00 45 00  ..B.....B...`..E..
0010  00 56 58 58 40 00 80 06  00 00 c0 a8 a9 65 c0 a8  .VXX@.....e...
0020  a9 02 c4 7a 1a 0a 67 f8  23 af 53 f7 67 b2 50 18  ...z..g. #.S.g.P.
0030  08 04 d4 01 00 00 00 2c  52 68 75 74 46 71 69 4a  ......., RhutFqij
0040  57 58 75 64 4c 52 77 6f  45 31 70 68 76 53 50 62  WXudLRwo E1phvSPb
0050  69 32 66 79 35 71 4d 4b  68 46 54 70 56 63 63 6b  i2fy5qMK hFTPvCck
0060  2b 70 6b 3d              +pk=

```



Problema y solución

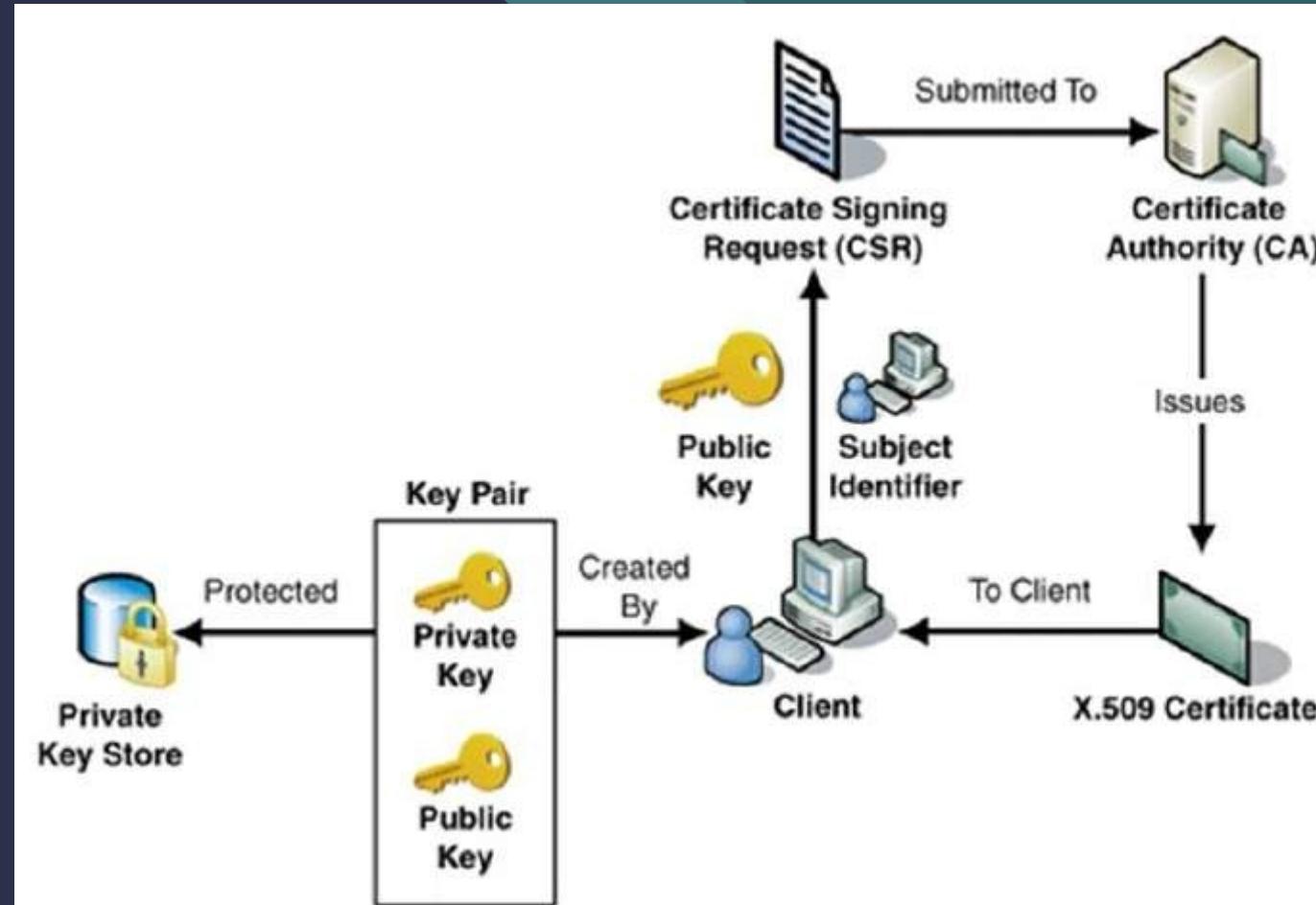
- El problema que tenemos es que siempre debemos tener la clave pública en el cliente.
 - Solución: Enviársela como vector de Bytes y que la almacene en el disco (o no) antes de leerla.
 - Ver Versión 3
- Y con esto sin querer, nos hemos montado una estructura de túnel de cifrado muy parecida al SSL salvando algunas cosas:
 - Uso de certificados (aunque estos tiene las claves)
 - ¿dónde se genera la clave de sesión?
 - Si es el cliente la mandamos cifrada al servidor con su clave pública del servidor
 - Si es en el servidor la mandamos cifrada al cliente con la clave privada del servidor





Certificado digital

- Un **certificado digital** o certificado electrónico es un fichero informático firmado electrónicamente por un **prestashop de servicios de certificación**, considerado por otras entidades como una autoridad para este tipo de contenido, que vincula unos datos de verificación de firma a un firmante, de forma que únicamente puede firmar este firmante, y confirma su identidad.
- Tiene una estructura de datos que contiene información sobre la entidad (por ejemplo una **clave pública**, una identidad o un conjunto de privilegios).
- La firma de la estructura de datos del certificado agrupa la información que contiene de forma que no puede ser modificada sin que esta modificación sea detectada.
- Veremos su uso en otra presentación y nos servirá para evitar para cifrar usando unas claves auto generadas, es otra opción existente, pero el proceso a realizar con ellos es el mismo que hemos visto: firmar, cifrar, claves de sesión, hacer las conexiones seguras, etc.





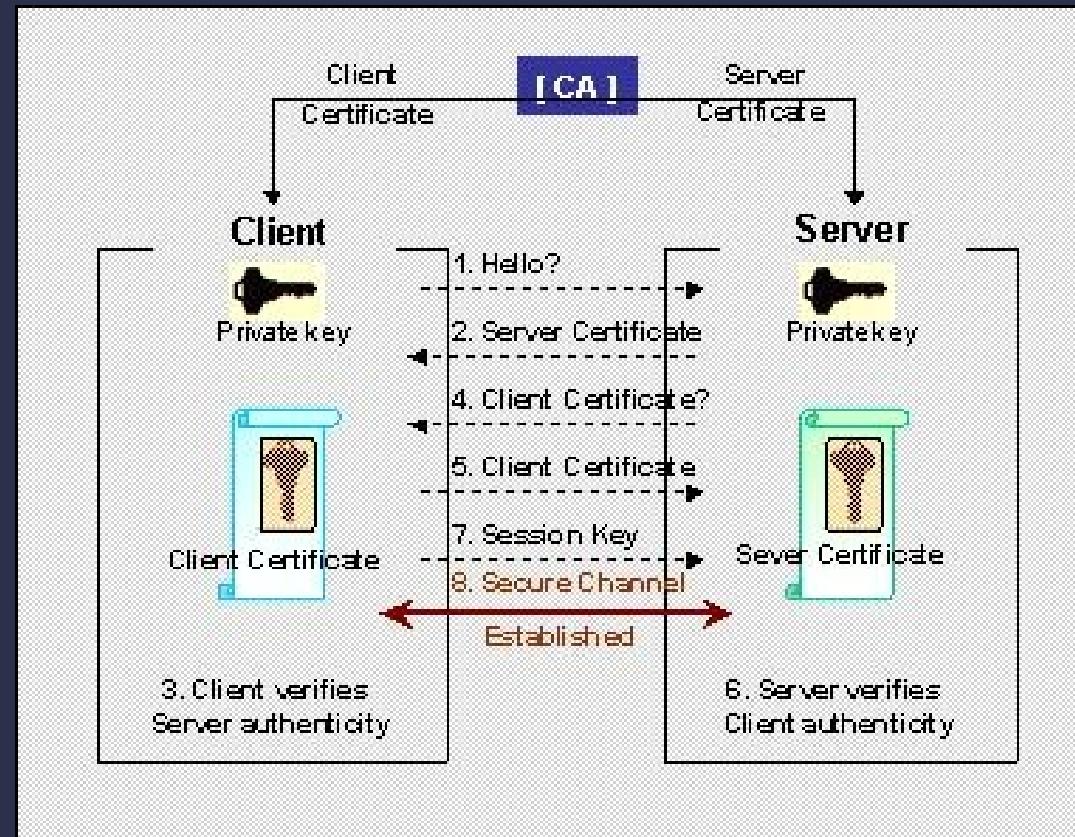
Certificado digital

Version	Version of X.509 to which the Certificate conforms
Serial Number	A number that uniquely identifies the Certificate
Signature Algorithm ID	The names of the specific Public Key algorithms that the CA has used to sign the Certificate (Ex.- RSA with SHA-1)
Issuer (CA) X.500 Name	The identity of the CA Server who issued the Certificate
Validity Period	The period of time for which the Certificate is valid with start date and expiration date
Subject X.500 Name	The owner's identity with X.500 Directory format (Ex.- cn=auser, ou=SP, o=Alphawest)
Subject Public Key Info	Algorithm ID Public Key Value
Issuer Unique ID	The Public Key of the owner of the Certificate and the specific Public Key algorithms associated with the Public Key
Subject Unique ID	Information used to identify the issuer of the Certificate
Extension	Information used to identify the Owner of the Certificate
CA Digital Signature	Additional information like Alternate name, CRL Distribution Point (CDP) The actual digital signature of the CA



Conexiones seguras con Certificados

- A partir del uso de certificados podemos hacer lo visto en el ejemplo anterior



SSL/TSL

Asegurando TCP/IP

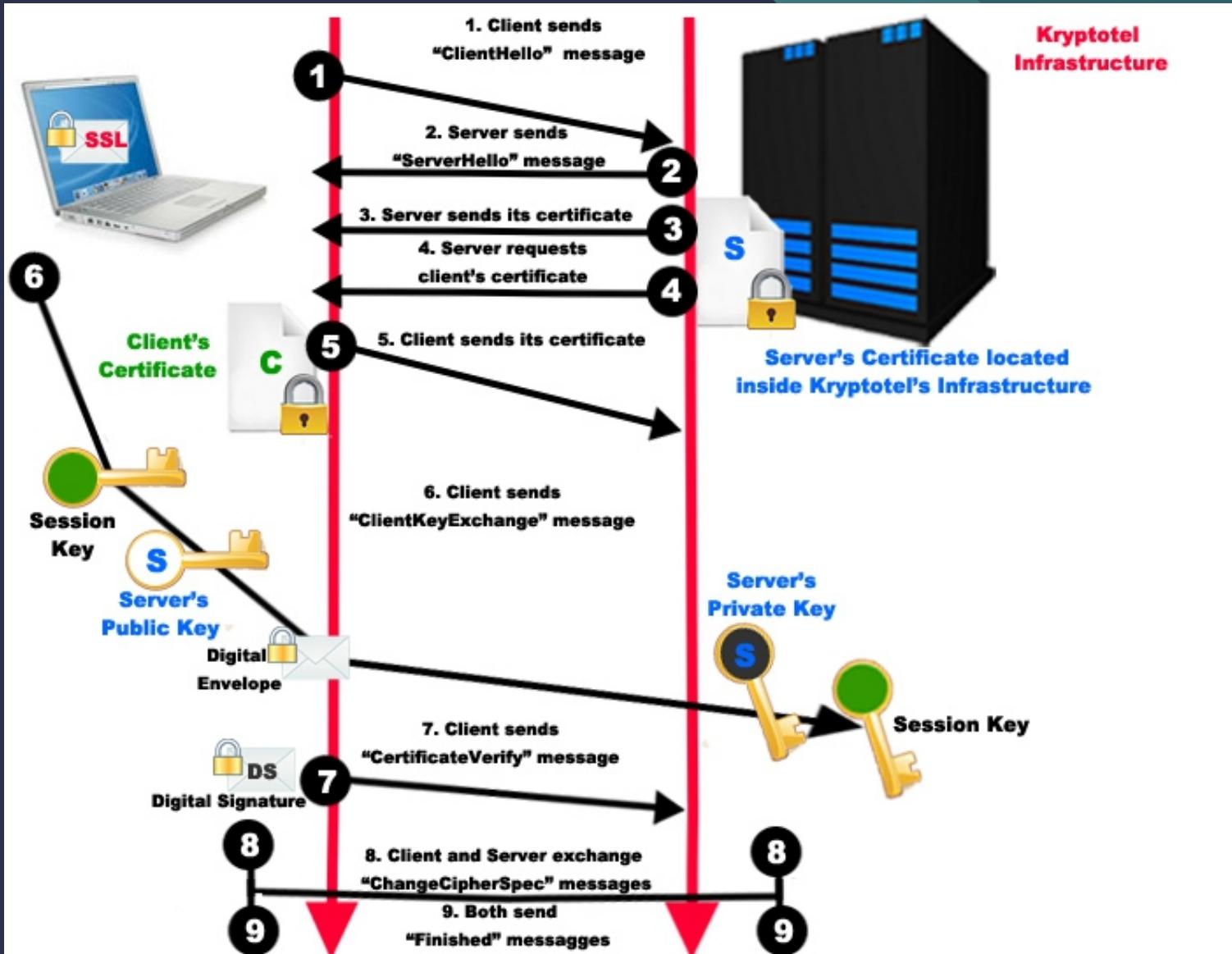


SSL/TSL

- SSL es el acrónimo de Secure Sockets Layer (capa de sockets seguros), la tecnología estándar para mantener segura una conexión a Internet, así como para proteger cualquier información confidencial que se envía entre dos sistemas e impedir que los delincuentes lean y modifiquen cualquier dato que se transfiera, incluida información que pudiera considerarse personal.
- El protocolo TLS (Transport Layer Security, seguridad de la capa de transporte) es solo una versión actualizada y más segura de SSL. Si bien aún denominamos a nuestros certificados de seguridad SSL porque es un término más común, en realidad se compran los certificados TLS más actualizados con la opción ECC, RSA y DSA.

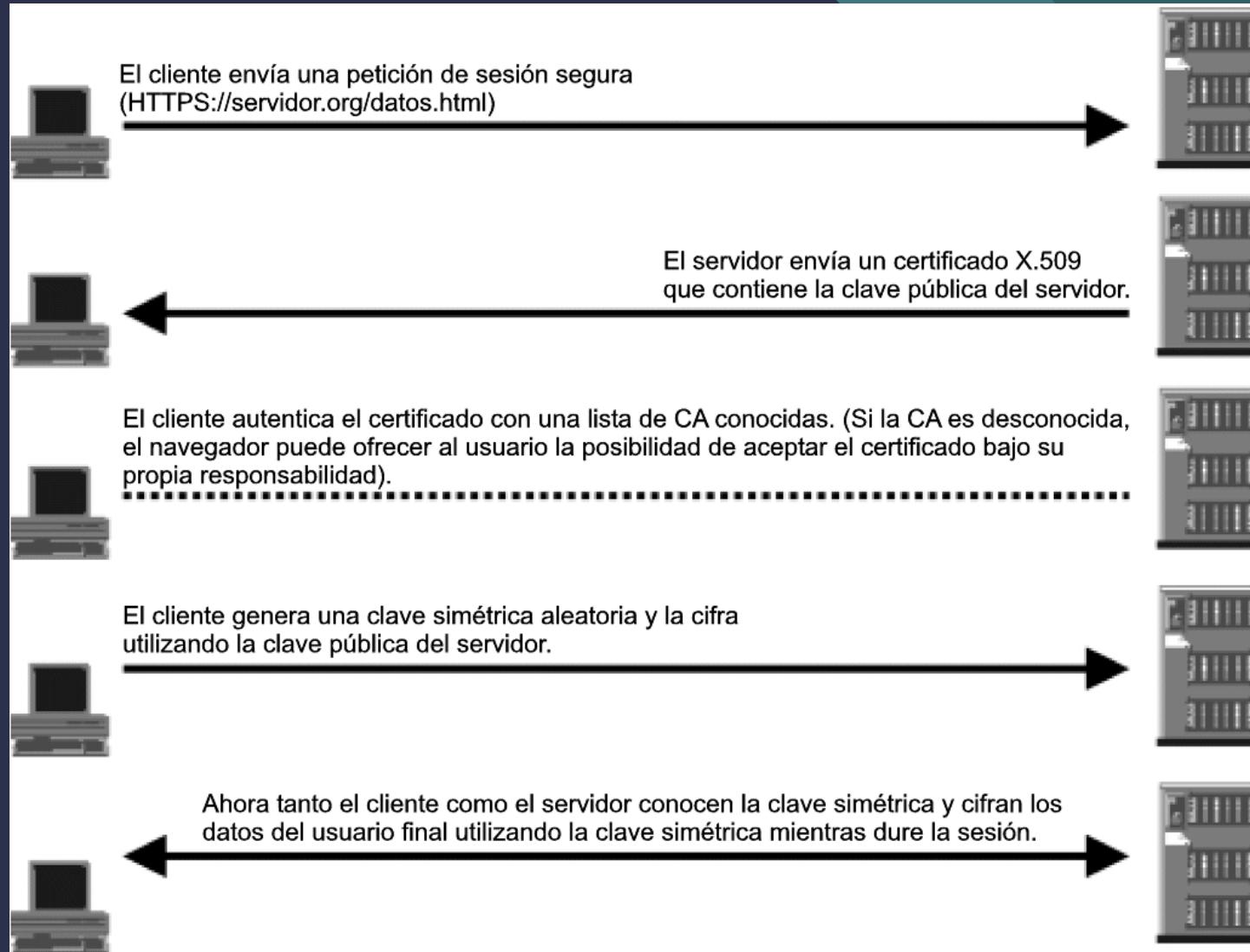


SSL/TSL





SSL ejemplo en HTTPS

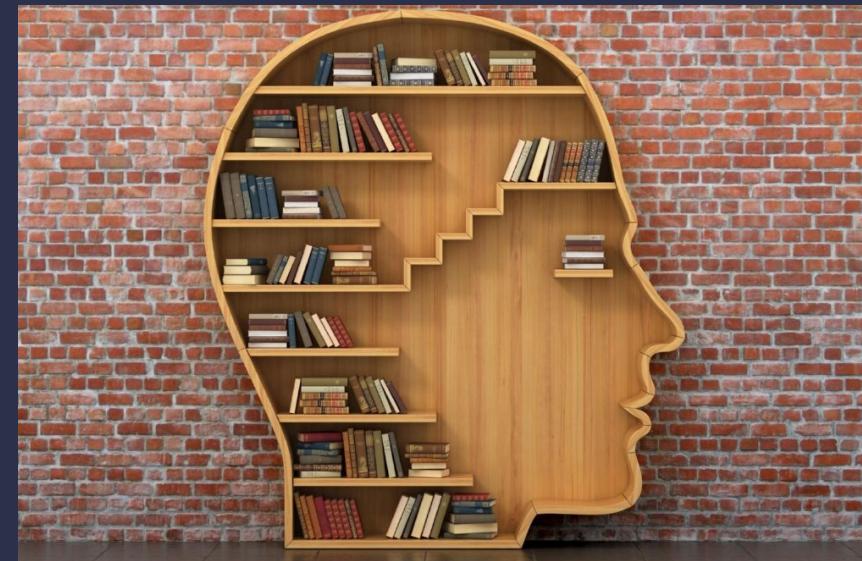




Nuestro programa con SSL

- Lo primero que vamos a ver es cómo trasformar a versión 1 de nuestro programa (vulnerable) en una versión que haga SSL usando JAVA
- No tenemos que perder de vista que aunque ahora no hagamos las cosas manuales (claves públicas, privadas y de sesión) en el fondo, lo que se está haciendo es todo lo visto en los ejemplos 2 y 3. Es por eso que lo hemos hecho, para comprender bien cómo funciona la estructura SSL/TSL por si alguna vez no podemos usarla directamente sepamos cómo implementarla.

El saber no ocupa lugar



Secure Socket

Sockets Seguros en JAVA



Programando Secure Sockets

- Lo primero que necesitamos es crear un juego de claves y nuestro certificado. Recordemos que nuestro certificado es donde tendremos la clave pública y parte de nuestra identidad y es lo que tendrán nuestros clientes. La clave privada nunca la tendrán (pero eso ya lo sabemos)
- Para ello usaremos el comando **keytool** para crear nuestro fichero jks
- **keytool -genkey -keyalg RSA -alias serverKey -keystore serverKey.jks -storepass servpass**
 - **keytool** está en el directorio bin de donde tengamos instalado java.
 - Con la opción **-genkey** le estamos diciendo que genere un certificado.
 - **-keyalg RSA** le indicamos que lo queremos encriptado con el algoritmo RSA
 - **-alias serverKey**. El certificado se meterá en un fichero de almacén de certificados que podrá contener varios certificados. Este alias es el nombre con el que luego podremos identificar este certificado concreto dentro del almacén. Podemos poner cualquier nombre que nos de una pista de qué es ese certificado.
 - **-keystore serverKey.jks**. Este es el fichero que hará de almacén de certificados. Si no existe se crea, si ya existe se añade el certificado con el alias que se haya indicado.
 - **-storepass servpass**. El almacén está protegido con contraseña, para acceder a él necesitamos la contraseña. Si el almacén no existe, se crea usando esta contraseña, por lo que deberemos recordarla. Si ya existe, debemos proporcionar la contraseña que tuviera ese almacén.
 - **OJO** puede dar problemas en estos momentos con JAVA 17, hacer los certificados con JAVA 11



Programando Secure Sockets

```
/ServidorSeguro/cert/> keytool -genkey -alias claveSSL -keyalg RSA -keystore AlmacenSSL.jks -storepass 1234567
```

¿Cuáles son su nombre y su apellido?

[Unknown]: José Luis González

¿Cuál es el nombre de su unidad de organización?

[Unknown]: PSP

¿Cuál es el nombre de su organización?

[Unknown]: IES Luis Vives

¿Cuál es el nombre de su ciudad o localidad?

[Unknown]: Leganés

¿Cuál es el nombre de su estado o provincia?

[Unknown]: Madrid

¿Cuál es el código de país de dos letras de la unidad?

[Unknown]: ES

¿Es correcto CN=José Luis González, OU=PSP, O=IES Luis Vives, L=Leganés, ST=Madrid, C=ES?

[no]: si

Generando par de claves RSA de 2.048 bits para certificado autofirmado (SHA256withRSA) con una validez de 90 días
para: CN=José Luis González, OU=PSP, O=IES Luis Vives, L=Leganés, ST=Madrid, C=ES



Programando Secure Sockets

- Como el certificado del servidor está dentro de un almacén, tenemos que sacarlo de ahí a un fichero. El comando es keytool
- **keytool -export -keystore serverkey.jks -alias serverKey -file ServerPublicKey.cer**
 - **-export** es para exportar el certificado
 - **-keystore serverkey.jks** indica en qué almacén está el certificado que queremos exportar
 - **-alias serverKey** es el identificador del certificado dentro del almacén. Debe ser el mismo alias que pusimos cuando lo creamos.
 - **-file ServerPublicKey.cer** es el nombre del fichero donde queremos que se guarde el certificado que vamos a extraer.



Programando Secure Sockets

```
/ServidorSeguro/cert/> keytool -export -alias claveSSL -keystore AlmacenSSL.jks -storepass 1234567 -file CertificadoSSL.cer  
Certificado almacenado en el archivo <CertificadoSSL.cer>
```

- Una vez que tenemos el fichero con el certificado, debemos meterlo dentro de un almacén de certificados de confianza del cliente. Cómo no, el comando a utilizar es keytool.
- **keytool -import -alias serverKey -file ServerPublicKey.cer -keystore clientTrustedCerts.jks -keypass clientpass -storepass clientpass**
 - **-import** para indicar que queremos meter un certificado existente en un almacén.
 - **-alias serverKey** es el identificador que queremos dar al servidor dentro del almacén de certificados de confianza del cliente. Hemos puesto otra vez serverKey, pero al ser un almacén distinto de el del servidor, podríamos poner otro nombre.
 - **-file ServerPublicKey.cer** El certificado que queremos importar
 - **-keystore clientTrustedCerts.jks** el almacén de certificados de confianza del cliente, se creará si no existe.
 - **-keypass clientpass** Clave para proteger el certificado dentro del almacén. Debe ser la misma que la del almacén.
 - **-storepass clientpass** Clave para el almacén, si el almacén existe debe ser la que diéramos en el momento de crearlo. Si no existe, el almacén se creará protegido con esta clave. Esta clave debe coincidir además con la que pongamos en la opción **-keypass** porque luego el código java lo exigirá así.



Programando Secure Sockets

```
/ClienteSeguro/cert/> keytool -import -alias claveSSL -file CertificadoSSL.cer -keystore UsuarioAlmacenSSL.jks -storepass 0987654
```

Propietario: CN=José Luis González, OU=PSP, O=IES Luis Vives, L=Leganés, ST=Madrid, C=ES

Emisor: CN=José Luis González, OU=PSP, O=IES Luis Vives, L=Leganés, ST=Madrid, C=ES

Número de serie: 61802abdab152e6c

Válido desde: Mon Jun 06 20:47:04 CEST 2022 hasta: Sun Sep 04 20:47:04 CEST 2022

Huellas digitales del certificado:

SHA1: EE:9C:05:38:81:99:74:57:36:74:F5:8C:71:81:84:86:00:8E:EF:22

SHA256: A8:FD:63:49:9D:91:08:25:15:0F:83:9D:F7:5E:22:1A:13:17:FC:0F:0A:3D:04:1C:BD:D3:4E:F3:64:B1:6B:BE

Nombre del algoritmo de firma: SHA256withRSA

Algoritmo de clave pública de asunto: Clave RSA de 2048 bits

Versión: 3

Extensiones:

```
#1: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: F7 02 65 7E F3 20 BA 17  49 2C 0D 90 7E 2A CE F2  ..e...I,...*..
0010: CC 0D BB ED
]
]
```

```
¿Confiar en este certificado? [no]: si
Se ha agregado el certificado al almacén de claves
```



Programando Secure Sockets

```
SSLServerSocketFactory serverFactory = (SSLServerSocketFactory) SSLServerSocketFactory.getDefault();  
ServerSocket serverSocket = serverFactory.createServerSocket(port);
```

```
SSLSocketFactory clientFactory = (SSLSocketFactory) SSLSocketFactory.getDefault();  
Socket client = clientFactory.createSocket(server, port);
```



Programando Secure Sockets

- Si ahora los ejecutáramos obtendríamos errores pues en ningún lado le hemos indicado los certificados, para ello debemos llamarlos desde la línea de comandos de la siguiente manera:
 - Servidor: `java -Djavax.net.ssl.keyStore= path/a/AlmacenSSL.jks -Djavax.net.ssl.keyStorePassword=1234567 path/fichero/java.java`
 - Cliente: `java -Djavax.net.ssl.trustStore=path/a/UsuarioAlmacenSSL.jks -Djavax.net.ssl.trustStorePassword=0987654 path/fichero/java.java`
- Como esto es una locura, vamos a usar otras propiedades que nos ofrece java
 - `System.setProperty("javax.net.ssl.keyStore", "/cert/AlmacenSSL.jks");`
 - `System.setProperty("javax.net.ssl.keyStorePassword", "1234567");`
 - `System.setProperty("javax.net.ssl.trustStore", "/cert/UsuarioAlmacen.jks");`
 - `System.setProperty("javax.net.ssl.trustStorePassword", "0987654");`
- Puedes verlo en la versión 4



Programando Secure Sockets

- Opcionalmente podemos registrar un proveedor SSL en los programas añadiendo esta línea:
 - `java.security.Security.addProvider(new com.sun.net.ssl.internal.ssl.Provider());`
- Ahora para ejecutar los programas cliente y servidor basta con escribir: `java ServidorSeguro` y `java ClienteSeguro` desde la línea de comandos.
- El método `getSession()` de la clase `SSLocket` devuelve un objeto `SSLSession` con la sesión SSL utilizada por la conexión, a partir de ella podemos obtener información como el identificador de la sesión, el cifrado SSL, el protocolo, el certificado, etc. El siguiente código dentro del programa cliente muestra información sobre la sesión y el certificado utilizado. Puedes verlo en el método sesión del clienteSeguro

Cliente->Conectado al servidor...

Servidor: 192.168.169.2

Cifrado: TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256

Protocolo: TLSv1.2

IDentificador:41883206041376217492932411480799447318300783466254228749482997986814162506369

Creación de la sesión: 1553534454678

Propietario : CN=@joseluisgs, OU=Programacion de Servicios y Procesos, O=IES Trassierra, L=Córdoba, ST=Córdoba, C=ES

Algoritmo: SHA256withRSA

Tipo: X.509

Emisor: CN=@joseluisgs, OU=Programacion de Servicios y Procesos, O=IES Trassierra, L=Córdoba, ST=Córdoba, C=ES

Número Serie: 1238454211



Programando Secure Sockets

The screenshot displays a Wireshark capture session titled "*Ethernet". The left pane shows a list of network packets, and the right pane shows a text log of the communication between a Client and a Server.

Packets List:

No.	Time	Source	Destination	Protocol	Length	Info
412	24.013802	192.168.169.2	224.0.0.251	MDNS	147	Star...
413	24.068385	192.168.169.101	192.168.169.2	TLSv1.2	139	App...
414	24.068978	192.168.169.2	192.168.169.101	TCP	60	6666 > 4969 [TCP SYN]
415	24.073248	192.168.169.2	192.168.169.101	TLSv1.2	139	App...
416	24.073848	192.168.169.2	192.168.169.101	TLSv1.2	192	App...
417	24.074055	192.168.169.101	192.168.169.2	TCP	54	4969 <--> 6666 [TCP ACK]
418	24.115919	192.168.169.2	224.0.0.251	MDNS	131	Star...
419	24.216206	192.168.169.2	224.0.0.251	MDNS	147	Star...

Text Log:

```
Ciente->Enviado mensaje
Ciente->Mensaje enviado a Servidor: Mensaje: 1553534928
Ciente->Recepción de mensajes
Ciente->Mensaje recibido: Mensaje de reespuesta num: 14
Ciente->¿Salir?
Ciente->Salir: false
Ciente->Enviado mensaje
Ciente->Mensaje enviado a Servidor: Mensaje: 1553534929
Ciente->Recepción de mensajes
Ciente->Mensaje recibido: Mensaje de reespuesta num: 15
Ciente->¿Salir?
Ciente->Salir: false
Ciente->Enviado mensaje
Ciente->Mensaje enviado a Servidor: Mensaje: 1553534930
Ciente->Recepción de mensajes
Ciente->Mensaje recibido: Mensaje de reespuesta num: 16
Ciente->¿Salir?
Ciente->Salir: false
Ciente->Enviado mensaje
Ciente->Mensaje enviado a Servidor: Mensaje: 1553534931
Ciente->Recepción de mensajes
Ciente->Mensaje recibido: Mensaje de reespuesta num: 17
Ciente->¿Salir?
Ciente->Salir: false
Ciente->Enviado mensaje
Ciente->Mensaje enviado a Servidor: Mensaje: 1553534932
Ciente->Recepción de mensajes
Ciente->Mensaje recibido: Mensaje de reespuesta num: 18
Ciente->¿Salir?
Ciente->Salir: false
Ciente->Enviado mensaje
Ciente->Mensaje enviado a Servidor: Mensaje: 1553534933
Ciente->Recepción de mensajes
Ciente->Mensaje recibido: Mensaje de reespuesta num: 19
Ciente->¿Salir?
Ciente->Salir: true
Ciente->Desconectado
```



Programando Secure Sockets

- Se puede ver que ahora todo el tráfico está cifrado bajo SSL/TSL, siendo prácticamente ilegible

The screenshot shows the Wireshark interface capturing traffic on the 'Ethernet' interface. The main window displays a list of 221 captured packets, with packet 215 selected. The selected packet's details are shown in the center pane, revealing its structure: Ethernet II frame, IP Version 4, TCP segment, and TLSv1.2 record layer. The packet payload is identified as 'Application Data'. The bottom pane shows the raw hex and ASCII data for the selected packet. The ASCII dump is heavily redacted, appearing as a series of question marks and other non-printable characters, demonstrating that the traffic is encrypted.



Programado Secure Sockets

- Incluso podríamos tener el certificado instalado en el sistema. Además podéis usar [KeyStore Explorer](#)

The image shows two windows from the KeyStore Explorer application. The left window displays the details of a certificate file named 'CertificadoSSL17.cer'. It shows the subject as 'José Luis González Sánchez' and the issuer as 'CN=José Luis González Sánchez,OU=IES Luis Vives,O=2DAM,L=Leganés,S'. The certificate is valid from 06/09/2022 to 09/09/2022. The right window shows the 'AlmacenSSL17.jks' keystore, listing a single entry named 'clavessl17' which is an RSA 2048-bit key with SHA-256 signature algorithm, valid until 09/09/2022.

CertificadoSSL17.cer

José Luis González Sánchez
Identity: José Luis González Sánchez
Verified by: José Luis González Sánchez
Expires: 04/09/22

Details

Subject Name

C (Country): ES
ST (State): Madrid
L (Locality): Leganés
O (Organization): 2DAM
OU (Organizational Unit): IES Luis Vives
CN (Common Name): José Luis González Sánchez

Issuer Name

C (Country): ES
ST (State): Madrid
L (Locality): Leganés
O (Organization): 2DAM
OU (Organizational Unit): IES Luis Vives
CN (Common Name): José Luis González Sánchez

Issued Certificate

Version: 3
Serial Number: 00 F6 4F DD 32 ED F1 9A 77
Not Valid Before: 2022-06-06
Not Valid After: 2022-09-04

Certificate Fingerprints

SHA-1: 51:3A:36:68:07:F5:F2:E9:4E:BC:12:A0:4F:96:3C:38:B4:4

AlmacenSSL17.jks - KeyStore Explorer 5.5.1

File Edit View Tools Examine Help

AlmacenSSL17.jks

Entry Name	Algorithm	Key Size	Certificate Expiry	Last Modified
clavessl17	RSA	2048	4/9/2022 23:20:59 ...	-

OK KeyStore Type: PKCS #12, Size: 1 entry , Selected: 1 entry, Path: '/home/joseluisgs/Proyectos/ServidorSeg...'



Programado Secure Sockets

- Vamos a usar OpenSSL Para ver las diferencias cargando los certificados realizados con Openssl. Ejemplo 4 SSL
- Creamos el ceretificado del cliente y servidor

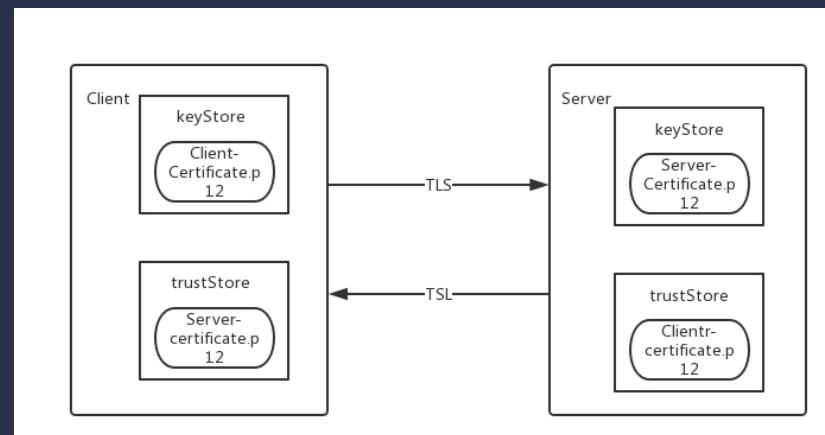
```
openssl req -newkey rsa:2048 -nodes -keyout client-key.pem -x509 -days 365 -out client-certificate.pem
```

```
openssl req -newkey rsa:2048 -nodes -keyout server-key.pem -x509 -days 365 -out server-certificate.pem
```

Creamos los certificados del cliente y servidor

```
openssl pkcs12 -inkey client-key.pem -in client-certificate.pem -export -out client-certificate.p12
```

```
openssl pkcs12 -inkey server-key.pem -in server-certificate.pem -export -out server-certificate.p12
```





Una capa más de seguridad

- Aunque a veces con esto suele ser diferentes, distintos desarrolladores incluyen una capa más, es decir, cifrar de nuevo con otra clave de sesión (doble cifrado)
- Para ello voy a recuperar el ejemplo 3, y voy a añadirle la capa de seguridad que creamos. Esta vez, la clave pública no la saco de un fichero, si no que la obtengo de leer el certificado del servidor, el cual envío al cliente (aunque ya lo tiene en su almacén).
- Con ello lo que hacemos es jugar un poco con los métodos para conocer su estructura. Obviamente la clave privada del servidor la obtengo de leer mi almacén.
- Todo esto se puede ver en la última versión, la v5

“

"El único sistema seguro es aquél que está apagado en el interior de un bloque de hormigón protegido en una habitación sellada rodeada por guardias armados"

-- Gene Spafford

”



Recursos

- Twitter: <https://twitter.com/joseluisgonsan>
- GitHub: <https://github.com/joseluisgs>
- Web: <https://joseluisgs.github.io>
- Discord: <https://discord.gg/kyhx7kGN>
- Aula Virtual:
<https://aulavirtual33.educa.madrid.org/ies.luisvives.leganes/course/view.php?id=248>



Gracias

José Luis González Sánchez

