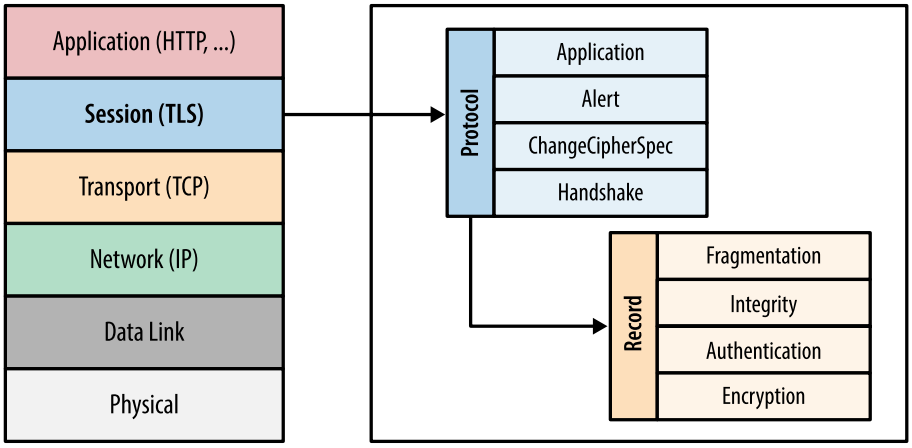
TLS 1.2

1 – ¿Qué es TLS y cómo funciona?

TLS (Transport Layer Security) es un protocolo de seguridad que actualmente se emplea para encriptar la mayoría de movimientos que se realizan a través de Internet. TLS es la evolución de SSL (Security Sockets Layer).

Con TLS se decide el tipo de cifrado que se va a emplear para encriptar la información, lo que se conoce como cipher suit. Los tipos de cifrado pueden variar dependiendo la versión de TLS. Para que la comunicación pueda producirse, el cliente y el servidor deben tener al menos un tipo de cifrado en común (si no, no se "entenderían").

El TLS será un paquete corto de datos que se transmite antes que el resto de la información.



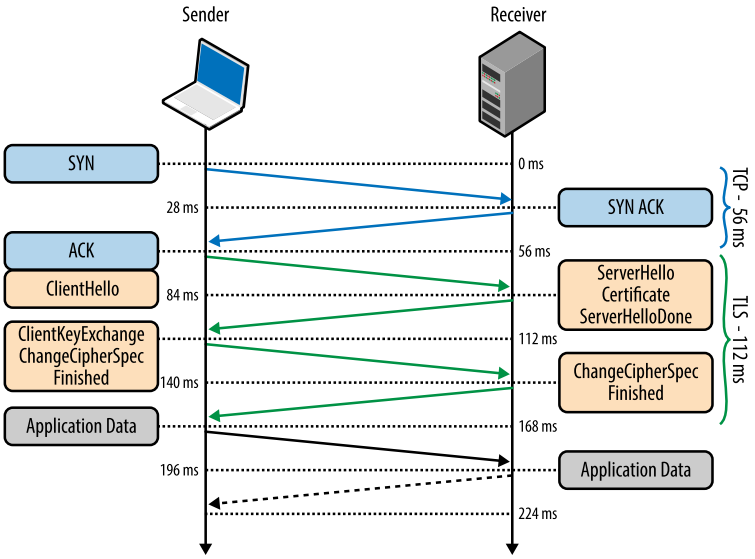
2 – TLS handshake

Antes de que se produzca ningún intercambio de datos, se lleva a cabo el TLS handshake:

1. "**Client Hello**" del cliente al servidor. Este paquete contendrá:
   - "Max TLS versión": última versión de TLS que soportará el cliente.
   - Número aleatorio: asegurará que no se produzcan ataques de respuesta.
   - Lista con todos los cipher suits que el cliente soporta.
2. El servidor responde con un "**Server Hello**", en el que habrá elegido entre las opciones que el cliente tenía disponibles. Así, incluirá:
   - Versión de TLS escogida.
   - Otro número aleatorio.
   - Un cipher suit escogido dentro de las opciones del cliente.
   - Certificado que incluirá la clave pública
   - "Server-Key Exchange Message": contiene los parámetros para realizar el intercambio de claves Diffie-Hellman.
   - Firma digital: resumen de los mensajes anteriores mediante una función hash y una clave privada del servidor ligada a la clave pública del certificado.
   - "Server Hello Done": indica que el servidor ha terminado el "Server Hello".

   En caso de que el servidor no soporte alguno de los requisitos del cliente, se enviará un mensaje de error.

3. El cliente responde con un "**Client Key Exchange**", que contiene:
   - "Change key spec message": indica que está preparado para comenzar a encriptar con toda la información recibida, por lo que el siguiente mensaje que mande el cliente va a estar encriptado.
   - Este último mensaje será el "finished message", que contendrá un resumen de todo el TLS handshake pero encriptado y empleando la clave.
4. El servidor responderá con un **"Server Key Exchange"** de la misma manera que el cliente al final de su último mensaje, es decir, con un "change cipher spec" y con un "finished message".



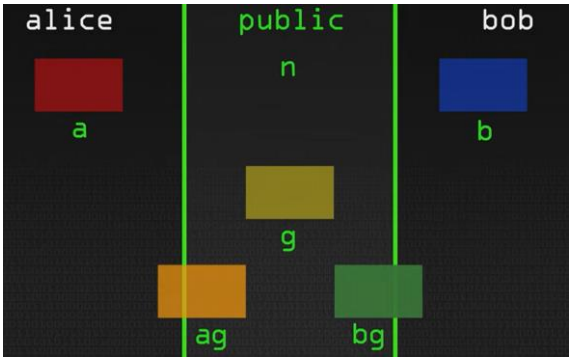
TLS 2.0 handshake. Cabe destacar que antes de que esto se produzca, tiene lugar el three way handshake del protocolo TCP.

3 – Intercambio de claves (Diffie-Hellman)

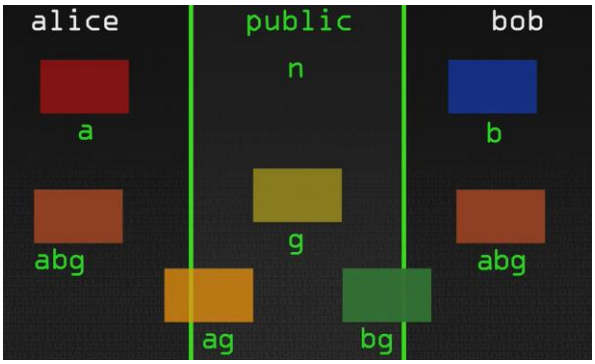
Diffie-Hellman se emplea actualmente en prácticamente cualquier proceso de criptografía que conlleve una clave simétrica.

Diffie-Hellman llevará a cavo complejos procesos matemáticos mediante los que se intercambiarán variables públicas combinadas con otras variables privadas para poder crear en ambos lados de la comunicación la misma clave.

1. Emisor y receptor se ponen de acuerdo en los procesos matemáticos que se van a emplear.
2. Las variables públicas se mezclarán con las claves privadas para crear las claves públicas. De esta forma, a partir de la mezcla no se puede saber su origen.



3. Se intercambian las claves públicas. Estas estarán en el área pública, por lo que cualquiera puede verlas, pero no se puede saber su origen.
4. Estas claves públicas pasan de nuevo a la parte privada, pero del lado contrario de la comunicación del que provienen, y se mezclan esta vez con la otra clave privada.



4 – Criptografía de la clave pública

Antiguamente se empleaba la misma clave en el proceso de encriptado y en el proceso de desencriptado (sistema simétrico). El problema que había es que ambas partes se tenían que poner de acuerdo en una misma clave, lo que es poco seguro y a veces complicado de realizar.

Un sistema asimétrico con 2 claves proporciona más seguridad y versatilidad. Para ello, además de estas 2 claves privadas, se emplearán 2 claves públicas a las que cualquiera puede tener acceso. De esta forma, supongamos que A quiere enviar un mensaje encriptado a B. Para ello, A encriptará el mensaje con la clave pública de B, ya que sabe que B lo podrá desencriptar con su clave privada. Sin embargo, esto se puede realizar de forma opuesta: A encripta el mensaje con su clave privada y B lo desencriptará con la clave pública de A. De esta forma se asegura que solamente A puede haber mandado el mensaje ya que solamente A tiene acceso a su clave privada. Con todo ello, llegamos al mejor de los casos en el que A encriptará su mensaje tanto con su clave privada como con la clave pública de B. De esta forma se asegura una conversación al 100% privada e inmodificable entre ambos.

5 – La huella digital y el hash

Una huella digital es un conjunto de datos asociados a un mensaje que permiten asegurar que el mensaje no fue modificado.

La huella digital se obtiene aplicando una función al mensaje denominada hash.

Características del hash:

1. Dos mensajes iguales producen idénticas huellas digitales.
2. Dos mensajes iguales producen huellas digitales iguales.
3. Dos huellas digitales idénticas pueden ser el resultado de dos mensajes iguales o de dos mensajes completamente diferentes.
4. Una función hash es irreversible, por tanto, su comprobación se realizará aplicando de nuevo la misma función hash al mensaje.

6 – Firmas y certificados digitales

Las firmas digitales se emplean en los sistemas de cifrado asimétricos. Estas firmas son de gran utilidad para conocer quién fue el emisor de la información, ya que cada emisor tiene su firma digital.

El proceso para en el que se emplea la firma digital tiene más pasos:

1. La información se transforma mediante un complejo algoritmo denominado “SHA256” para crear el hash, mencionado anteriormente. Menciono el término transformar ya que, al ser un proceso prácticamente imposible de revertir, no se considera encriptar.
2. Una vez se tiene el hash, este ya sí se encripta mediante la clave privada del emisor.
3. Este mensaje encriptado se une al mensaje original y se manda al receptor. Esto es la firma digital.
4. El receptor del mensaje desencripta este mediante la clave pública del emisor (si lo puede desencriptar, sabe que solo ha podido venir de ese emisor).
5. Después, el receptor emplea el SHA256 sobre el mensaje original para calcular el hash de nuevo. Si este hash coincide con el valor que obtuvo al desencriptar el otro mensaje, se puede confirmar que el mensaje original no ha sido manipulado.

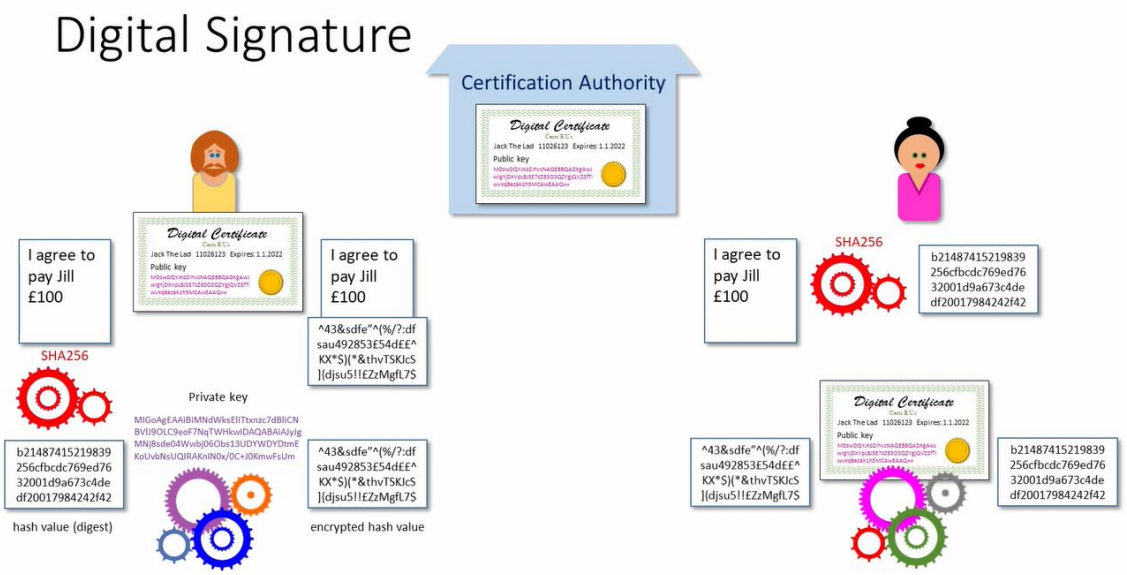
Todo este proceso no proporciona ningún tipo de seguridad. Es decir, otra persona podría haberse hecho pasar por el emisor. Para asegurar que solo el emisor real ha mandado el mensaje, se emplea el certificado digital.

Los certificados digitales los proporcionan empresas fiables denominadas “autoridades de certificación” como Global Sign o el propio Google con sus productos.

El proceso de certificación es el siguiente:

1. El emisor proporciona su clave pública a la autoridad de certificación.
2. La autoridad de certificación examina cuidadosamente que el emisor sea quien dice ser.
3. Una vez verificada la identidad del emisor, se le proporciona un certificado digital que acredita que el emisor es quien dice ser, junto con su clave pública y las credenciales de la autoridad de certificación (verificadas por una autorización aún superior).

Si en el proceso anterior, además de que el emisor envíe el mensaje original y el mensaje hasheado y encriptado, adjunta una copia del certificado digital, el receptor tendrá claro que el emisor de la información es quien dice ser y podrá desencriptar el mensaje con la clave pública del emisor proporcionada mediante el certificado digital.

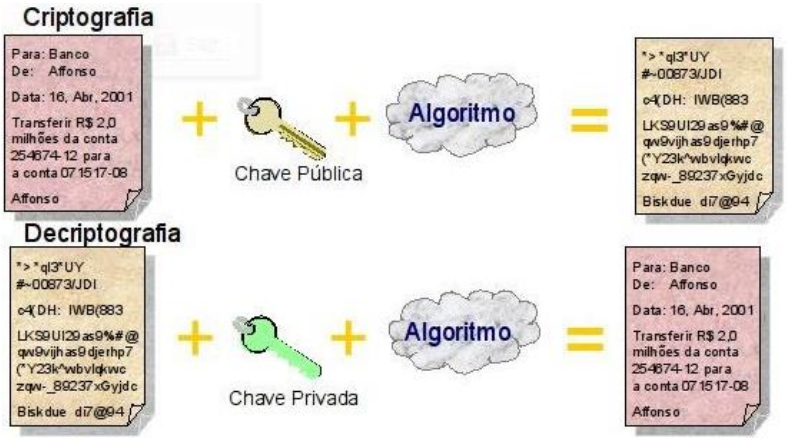


7 – RSA

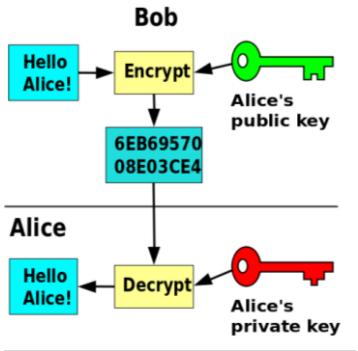
RSA (Rivest, Shamir y Adleman) es un sistema criptográfico de clave pública que utiliza factorización de números enteros. Este es válido tanto para cifrar como para firmar digitalmente, lo que va relacionado con puntos anteriormente mencionados.

Los mensajes enviados se representan mediante números, y el funcionamiento se basa en el producto de dos números primos grandes elegidos al azar y mantenidos en secreto.

Como en todo sistema de clave pública, cada usuario posee dos claves de cifrado: una pública y otra privada. Cuando se quiere enviar un mensaje confidencial, el emisor busca la clave pública del receptor, cifra su mensaje con esa clave, pero también con el algoritmo RSA, y una vez que el mensaje cifrado llega al receptor, este se ocupa de descifrarlo usando su clave privada y el mismo algoritmo.



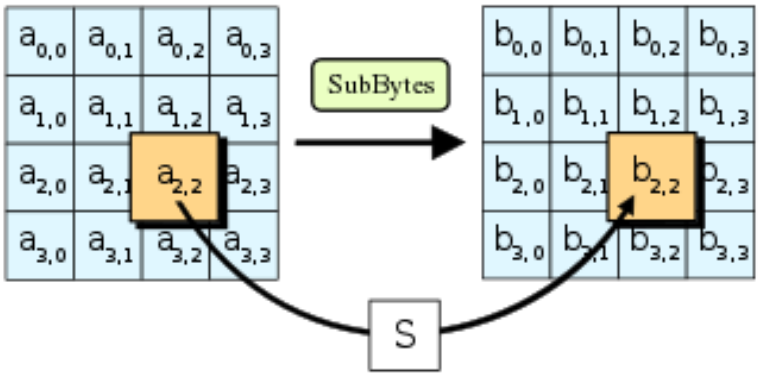
En el caso de querer firmar, el emisor obtiene un hash del mensaje a firmar y lo procesa con su clave privada obteniendo así la firma; se envía el mensaje y la firma; el receptor recalcula el hash y descifra el hash original con la clave pública del emisor, validando así (o no) la firma del mensaje.



Se cree que RSA será seguro mientras no se conozcan formas rápidas de descomponer un número grande en producto de primos.

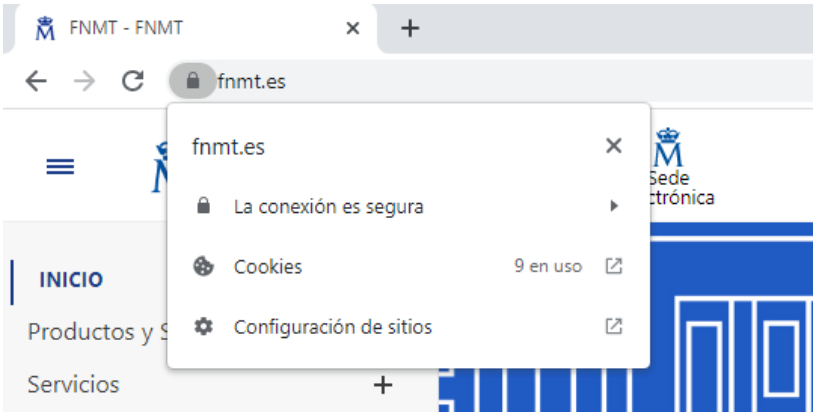
8 – AES

AES (Advanced Encryption Standard) es un sistema de cifrado por bloques que actualmente emplea como estándar de cifrado del gobierno de los Estados Unidos. Es uno de los algoritmos de cifrado más usados basado en criptografía simétrica.

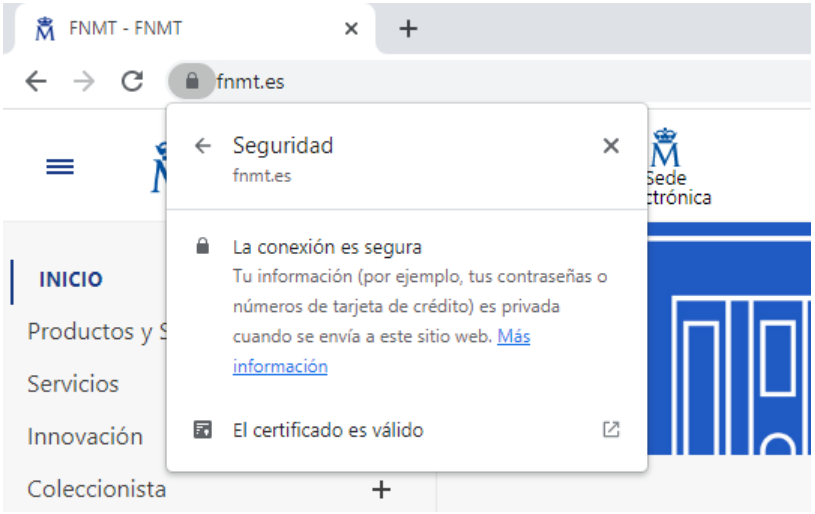


Visualización del TLS en el navegador

1. Seleccionamos el candado de la barra de URL y acto seguido en “la conexión es segura”.

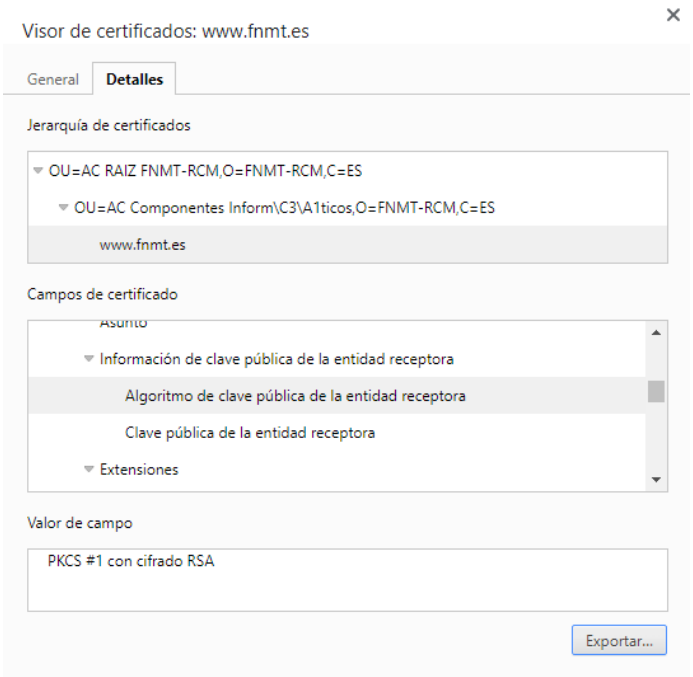


2. Aquí podremos comprobar los detalles sobre el certificado digital.

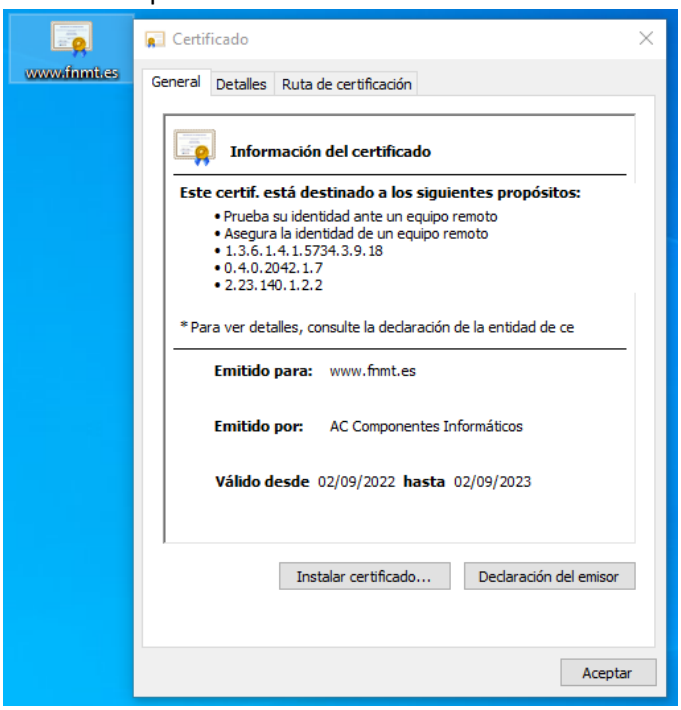


3. Hacemos clic en “el certificado es válido” y podremos visualizar información mencionada anteriormente como la autoridad de certificación, el periodo de validez del certificado o la huella digital.

4. Presionamos en detalles para visualizar más información. Podremos visualizar información como el tipo de cifrado (RSA).



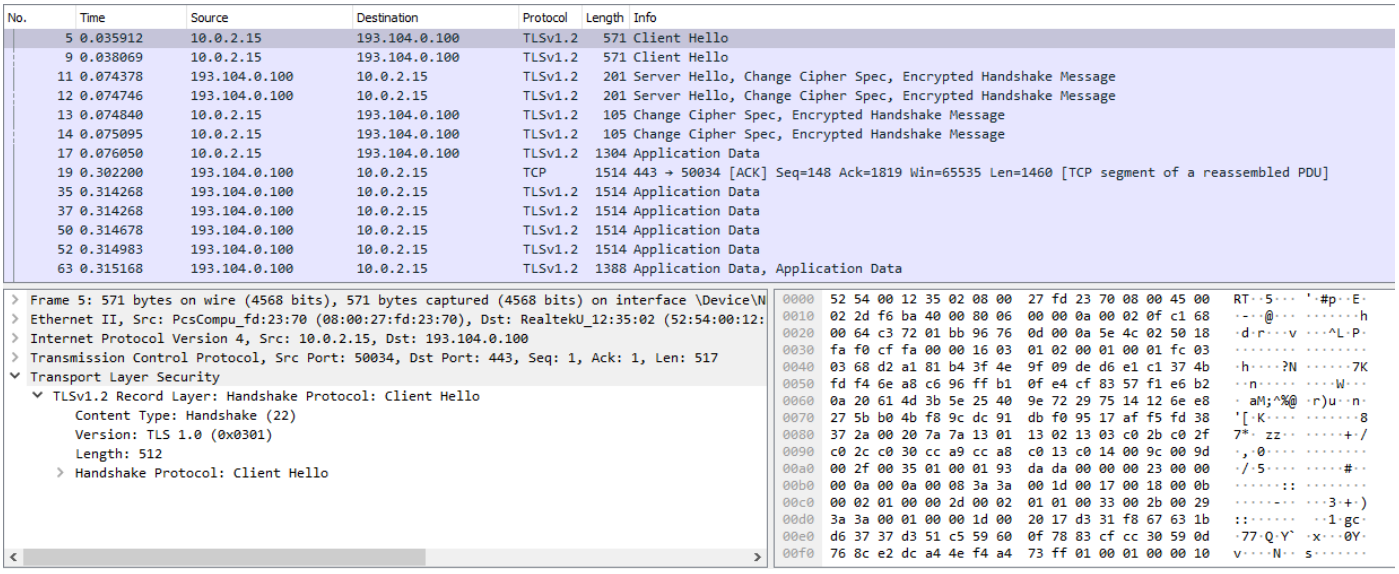
5. Podemos exportar el certificado.



Descargar certificado:

www.fnmt.es.crt

Visualización de TLS en Wireshark



No.	Time	Source	Destination	Protocol	Length	Info
5	0.035912	10.0.2.15	193.104.0.100	TLSv1.2	571	Client Hello
9	0.038069	10.0.2.15	193.104.0.100	TLSv1.2	571	Client Hello
11	0.074378	193.104.0.100	10.0.2.15	TLSv1.2	201	Server Hello, Change Cipher Spec, Encrypted Handshake Message
12	0.074746	193.104.0.100	10.0.2.15	TLSv1.2	201	Server Hello, Change Cipher Spec, Encrypted Handshake Message
13	0.074840	10.0.2.15	193.104.0.100	TLSv1.2	105	Change Cipher Spec, Encrypted Handshake Message
14	0.075095	10.0.2.15	193.104.0.100	TLSv1.2	105	Change Cipher Spec, Encrypted Handshake Message
17	0.076050	10.0.2.15	193.104.0.100	TLSv1.2	1304	Application Data
19	0.302200	193.104.0.100	10.0.2.15	TCP	1514	443 → 50034 [ACK] Seq=148 Ack=1819 Win=65535 Len=1460 [TCP segment of a reassembled PDU]
35	0.314268	193.104.0.100	10.0.2.15	TLSv1.2	1514	Application Data
37	0.314268	193.104.0.100	10.0.2.15	TLSv1.2	1514	Application Data
50	0.314678	193.104.0.100	10.0.2.15	TLSv1.2	1514	Application Data
52	0.314983	193.104.0.100	10.0.2.15	TLSv1.2	1514	Application Data
63	0.315168	193.104.0.100	10.0.2.15	TLSv1.2	1388	Application Data, Application Data

```
> Frame 5: 571 bytes on wire (4568 bits), 571 bytes captured (4568 bits) on interface \Device\N
> Ethernet II, Src: PcsCompu_fd:23:70 (08:00:27:fd:23:70), Dst: RealtekU_12:35:02 (52:54:00:12:
> Internet Protocol Version 4, Src: 10.0.2.15, Dst: 193.104.0.100
> Transmission Control Protocol, Src Port: 50034, Dst Port: 443, Seq: 1, Ack: 1, Len: 517
v Transport Layer Security
  v TLSv1.2 Record Layer: Handshake Protocol: Client Hello
      Content Type: Handshake (22)
      Version: TLS 1.0 (0x0301)
      Length: 512
    > Handshake Protocol: Client Hello
```

```
0000  52 54 00 12 35 02 08 00  27 fd 23 70 08 00 45 00   RT··5··· '·#p··E·
0010  02 2d f6 ba 40 00 80 06  00 00 0a 00 02 0f c1 68   ·-··@··· ·······h
0020  00 64 c3 72 01 bb 96 76  0d 00 0a 5e 4c 02 50 18   ·d·r···v ···^L·P·
0030  fa f0 cf fa 00 00 16 03  01 02 00 01 00 01 fc 03   ········ ········
0040  03 68 d2 a1 81 b4 3f 4e  9f 09 de d6 e1 c1 37 4b   ·h····?N ······7K
0050  fd f4 6e a8 c6 96 ff b1  0f e4 cf 83 57 f1 e6 b2   ··n····· ····W···
0060  0a 20 61 4d 3b 5e 25 40  9e 72 29 75 14 12 6e e8   · aM;^%@ ·r)u··n·
0070  27 5b b0 4b f8 9c dc 91  db f0 95 17 af f5 fd 38   '[·K···· ·······8
0080  37 2a 00 20 7a 7a 13 01  13 02 13 03 c0 2b c0 2f   7*· zz·· ·····+·/
0090  c0 2c c0 30 cc a9 cc a8  c0 13 c0 14 00 9c 00 9d   ·,·0···· ········
00a0  00 2f 00 35 01 00 01 93  da da 00 00 00 23 00 00   ·/·5···· ·····#··
00b0  00 0a 00 0a 00 08 3a 3a  00 1d 00 17 00 18 00 0b   ·····:: ········
00c0  00 02 01 00 00 2d 00 02  01 01 00 33 00 2b 00 29   ·····-·· ···3·+·)
00d0  3a 3a 00 01 00 00 1d 00  20 17 d3 31 f8 67 63 1b   ::····· ··1·gc·
00e0  d6 37 37 d3 51 c5 59 60  0f 78 83 cf cc 30 59 0d   ·77·Q·Y` ·x···0Y·
00f0  76 8c e2 dc a4 4e f4 a4  73 ff 01 00 01 00 00 10   v····N·· s·······
```

Podemos observar en wireshark como, antes de que se produzca el intercambio de datos se produce el TLS handshake de la misma forma en que mencionó anteriormente.

En el client hello observamos que se trata de la versión TLS 1.2.

Si presionamos en el TLS handshake observamos todos los pasos y sus detalles, como el random numer o todos los cipher suits compatibles con el cliente.

```
Handshake Protocol: Client Hello
    Handshake Type: Client Hello (1)
    Length: 508
    Version: TLS 1.2 (0x0303)
  > Random: 68d2a181b43f4e9f09ded6e1c1374bfdf46ea8c696ffb10fe4cf8357f1e6b20a
    Session ID Length: 32
    Session ID: 614d3b5e25409e72297514126ee8275bb04bf89cdc91dbf09517aff5fd38372a
    Cipher Suites Length: 32
  v Cipher Suites (16 suites)
      Cipher Suite: Reserved (GREASE) (0x7a7a)
      Cipher Suite: TLS_AES_128_GCM_SHA256 (0x1301)
      Cipher Suite: TLS_AES_256_GCM_SHA384 (0x1302)
      Cipher Suite: TLS_CHACHA20_POLY1305_SHA256 (0x1303)
      Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 (0xc02b)
      Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xc02f)
      Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 (0xc02c)
      Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xc030)
      Cipher Suite: TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256 (0xcca9)
      Cipher Suite: TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256 (0xcca8)
      Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (0xc013)
      Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xc014)
      Cipher Suite: TLS_RSA_WITH_AES_128_GCM_SHA256 (0x009c)
      Cipher Suite: TLS_RSA_WITH_AES_256_GCM_SHA384 (0x009d)
      Cipher Suite: TLS_RSA_WITH_AES_128_CBC_SHA (0x002f)
      Cipher Suite: TLS_RSA_WITH_AES_256_CBC_SHA (0x0035)
    Compression Methods Length: 1
  > Compression Methods (1 method)
    Extensions Length: 403
  > Extension: Reserved (GREASE) (len=0)
  > Extension: session_ticket (len=0)
```

En el hello server observamos la respuesta del servidor, donde se nos muestran todos los detalles ya mencionados, como el copher suit escogido, el otro número random, etc.

```
  v Handshake Protocol: Server Hello
      Handshake Type: Server Hello (2)
      Length: 87
      Version: TLS 1.2 (0x0303)
    > Random: 2d936762709dc9de3949fcf6ba7944edb6f6f62dbd99b9b70b2cb62da137cc07
      Session ID Length: 32
      Session ID: 614d3b5e25409e72297514126ee8275bb04bf89cdc91dbf09517aff5fd38372a
      Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xc02f)
      Compression Method: null (0)
      Extensions Length: 15
    v Extension: renegotiation_info (len=1)
        Type: renegotiation_info (65281)
        Length: 1
      > Renegotiation Info extension
    v Extension: ec_point_formats (len=2)
        Type: ec_point_formats (11)
        Length: 2
        EC point formats Length: 1
      > Elliptic curves point formats (1)
    v Extension: extended_master_secret (len=0)
        Type: extended_master_secret (23)
        Length: 0
      [JA3S Fullstring: 771,49199,65281-11-23]
      [JA3S: 76c691f46143bf86e2d1bb73c6187767]
v TLSv1.2 Record Layer: Change Cipher Spec Protocol: Change Cipher Spec
    Content Type: Change Cipher Spec (20)
    Version: TLS 1.2 (0x0303)
    Length: 1
  > Change Cipher Spec Message
v TLSv1.2 Record Layer: Handshake Protocol: Encrypted Handshake Message
    Content Type: Handshake (22)
    Version: TLS 1.2 (0x0303)
```

Captura completa:

TLS fnmt.pcapng