Escuela de Doctorado
y Estudios de Posgrado
Universidad de La Laguna

Máster Universitario en Ciberseguridad e Inteligencia de Datos

ESTUDIO PRELIMINAR SOBRE TLS POST-CUÁNTICO

Preliminary study on Post-Quantum TLS

Carlos Barreda Falciano

La Laguna, 7 de septiembre de 2022

D. **Pino Caballero Gil**, con N.I.F. 45.534.310-Z Catedrática de Universidad adscrita al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutora

D. **Jezabel Molina Gil**, con N.I.F. 78.507.682-B profesora Ayudante Doctora adscrita al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutora

C E R T I F I C A (N)

Que la presente memoria titulada: “Estudio preliminar sobre TLS Post-Cuántico” ha sido realizada bajo su dirección por D. **Carlos Barreda Falciano**, con N.I.F. 78.635.428-S.

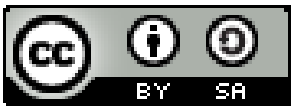
Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 7 de septiembre de 2022.

Agradecimientos

En primer lugar, agradecer a mi tutora, Pino Caballero Gil, y a mi cotutora, Jezabel Molina Gil, por su orientación, apoyo y profesionalidad durante la realización de este proyecto.

También agradecer a mi pareja, familia y amigos porque han sido un apoyo fundamental durante la realización de este trabajo.

Licencia



Resumen

La protección de la ciberseguridad siempre ha estado estrechamente relacionada con los avances tecnológicos. El inminente desarrollo de los ordenadores cuánticos de uso personal constituye una grave amenaza contra la seguridad de todos los protocolos criptográficos, que actualmente usamos en nuestras redes de comunicaciones seguras. Al incluir como vector de ataque la computación cuántica, todos los algoritmos de intercambio de claves, de cifrado y de firma digital actuales usados en el protocolo TLS tendrán que ser actualizados para convertirlos en sistemas seguros en la era post-cuántica. Aunque la tecnología cuántica aún no ha llegado a nuestras casas, debemos prepararnos cuanto antes porque la amenaza ya es real. En criptografía, detalles pequeños pueden tener graves impactos, así que se hace necesario conocer en detalle todas las bases de los esquemas actuales que componen el omnipresente protocolo TLS para poder saber qué aspectos podrían ser vulnerables en la era post-cuántica, y así ser capaces de proponer soluciones. Ese es precisamente el enfoque del presente trabajo.

Palabras clave: TLS 1.3, criptografía post-cuántica, ciberseguridad

Abstract

Cybersecurity protection has always been closely linked to technological advances. The apparently imminent development of quantum computers for personal use constitutes a serious threat to the security of all the cryptographic protocols that are currently used in secure network communications. By including quantum computing as an attack vector, all current key exchange, encryption and digital signature algorithms used in the TLS protocol will have to be updated to make them secure systems in the post-quantum era. Despite the fact that quantum technology has not yet reached our homes, the change in the cryptographic systems must be carried out as soon as possible because the threat is already real. In cryptography, small details can have serious impacts, so it is vital to know in detail all the bases of the current schemes that make up the ubiquitous TLS protocol in order to identify which aspects might be vulnerable in the post-quantum era, and thus be able to propose solutions. That is precisely the focus of this work.

Key words: TLS1.3, post-quantum cryptography, cybersecurity

Índice general

Índice de figuras

Índice de tablas

1. Introducción

Resulta difícil imaginarse un mundo sin Internet. Navegar por la web, compartir contenidos, usar el correo electrónico, o realizar compras online son acciones imprescindibles que hacemos cotidianamente en nuestra vida personal y profesional. Indudablemente Internet nos proporciona muchísimos beneficios, pero también es importante saber que muchas de las acciones que realizamos cada día implican el intercambio de información sensible que debe ser protegida. Es por ello que en Internet se emplea el protocolo TLS (Transport Layer Security) que blinda las comunicaciones entre dispositivos.

Si bien todavía no se han desarrollado ordenadores cuánticos de uso personal, la amenaza cuántica va tomando forma, ya que, se ha podido confirmar la intercepción y almacenamiento de grandes cantidades de información cifrada, que se verá comprometida en el futuro cuando los ordenadores cuánticos estén disponibles.

El protocolo TLS es uno de los esquemas criptográficos que más peligro corren en la era post-cuántica dada su gran dependencia de criptografía de clave pública. Por tanto, es necesario tomar medidas de carácter urgente, para la sustitución en TLS, de todos los componentes criptográficos que no sean resistentes a la computación cuántica.

En este documento se analiza en detalle el protocolo TLS, incluyendo su arquitectura y componentes, así como algunas propuestas de modificación para que las comunicaciones que actualmente se protegen mediante TLS sigan siendo seguras cuando lleguen los ordenadores cuánticos.

2. Antecedentes

La importancia de realizar un estudio sobre el protocolo TLS (Transport Layer Security) se debe a que es el encargado de ofrecer seguridad a la mayoría de las conexiones que realizamos a diario, como navegación web, envíos de correos electrónicos, conexiones FTP y conexiones VPNs, mensajería instantánea y VoIP. De hecho, el protocolo TLS está implicado en la mayoría de las comunicaciones donde se transmiten y transfieren datos sensibles, como la banca online o los controles de acceso [1].

2.1. Objetivos del protocolo TLS

El objetivo del protocolo TLS es brindar seguridad, protegiendo los tres conceptos fundamentales de la ciberseguridad:

- **Confidencialidad**: Para proporcionar confidencialidad se emplean algoritmos de cifrado para proteger la información frente al acceso de terceros.
- **Autenticidad**: Para disponer de una comunicación segura, es necesario identificar a las partes implicadas.
- **Integridad**: Permite que ningún mensaje o información pueda ser manipulado a posteriori.

2.2. Evolución del protocolo TLS

El protocolo TLS tuvo un predecesor, conocido como protocolo SSL, desarrollado por Netscape en 1995. Con el paso de los años, las vulnerabilidades encontradas en SSL determinaron la prohibición de su uso, dando paso al actualmente conocido como TLS. Del protocolo TLS existen 4 estándares publicados por el IETF (Internet Engineering Task Force). Como se observa en la Figura 1, dos de ellos ya están obsoletos (TLS 1.0 y TLS 1.1), pues actualmente sólo se trabaja con las dos últimas versiones (TLS 1.2 y TLS 1.3). La última versión, TLS 1.3, aprobada en el año 2018 está siendo fundamental para el desarrollo de futuras modificaciones que permitan crear una versión post-cuántica del protocolo [14].

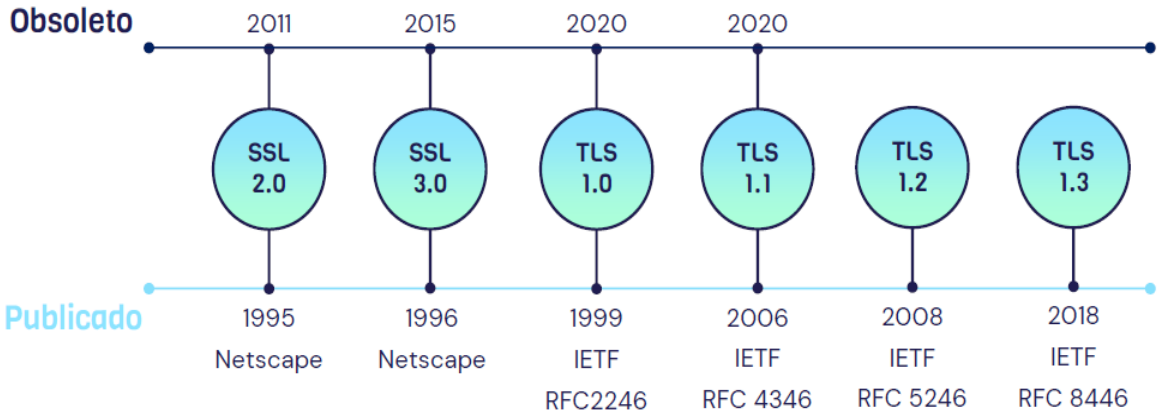


Figura 1. Evolución del protocolo TLS.

3. Conceptos previos

3.1. Cifrados simétricos en TLS 1.3

En la versión de TLS 1.3 se simplifica el conjunto de algoritmos simétricos que se emplean para cifrar la información, los únicos empleados son: AES-GCM y ChaCha20-Poly1365 [2] [3].

3.1.1. AES

AES (Advanced Encryption Standard) es un algoritmo de cifrado en bloques de tamaño fijo (128 bits o 16 bytes). La longitud de clave puede variar. En la actualidad se trabaja con 256 bits, pero se estima que deba aumentarse con la llegada de la computación cuántica.

El procedimiento de cifrado de un bloque requiere de aplicar varias funciones que serán fácilmente invertibles siguiendo un orden determinado. Además, para cada bloque deben aplicarse estas funciones un número determinado de vueltas que depende del tipo de algoritmo elegido.

Combinaciones posibles de estados de AES	Long. del bloque (Nb palabras)	Long. del bloque (Nk palabras)	Num. de Rondas (Nr)
AES-128	**4**	**4**	**10**
AES-192	**4**	**6**	**12**
AES-256	**4**	**8**	**14**

Tabla 1. Número de rondas necesarias para cada versión de AES.

Las 4 funciones fácilmente invertibles que emplea AES son conocidas como:

- **Addroundkey**: Es una función XOR, que se hace con la matriz de texto en claro (128 bits) y la K_0 de 128 bits. Si el algoritmo de cifrado es AES 128, entonces K_0 es exactamente igual al tamaño de la clave. Si la clave fuese 256 bits, entonces K_0 serían los primeros 128 bits de la clave, porque K_0 debe tener el mismo tamaño que la matriz de texto en claro.
- **SubBytes**: Es una función que permite cambiar un byte por otro según la tabla de inversos en GF(2^8).
- **ShiftRow**: Esta función permite realizar un desplazamiento de los bytes de la matriz 4x4, es decir, es una operación de permutación. Para cifrar

se desplaza a la izquierda, y para descifrar a la derecha. Cada una de las filas realiza un desplazamiento diferente; la fila 1 no se desplaza, la fila 2 desplaza un byte, la fila 3 desplaza 2 bytes y la fila 4 desplaza 3 bytes.

- **MixColumns**: El objetivo de esta función es maximizar la difusión. Para ello realiza una mezcla de datos en cada una de las 4 columnas de la matriz de estado.

3.1.1.1. AES-GCM

Un modo de cifrado de un cifrado en bloque es una forma de aplicar repetidamente la operación de cifrado para transformar de forma segura cantidades de datos mayores que un bloque. El Instituto Nacional de Estándares y Tecnología (NIST) ha aprobado un total de 14 modos de cifra diferentes para cifrados simétricos en bloque, que proporcionan características diferentes [4]:

- Modos que aportan sólo confidencialidad (8): ECB, CBC, OFB, CFB, CTR, XTS-AES, FF1 y FF3
- Modo que aporta sólo autenticación (1): CMAC
- Modos combinados, que aportan tanto confidencialidad como autenticación (5): CCM, GCM, KW, KWP y TKW

Por motivos de seguridad o eficiencia, muchos de estos modos ya no están permitidos, y en TLS 1.3 sólo se admite AES-GCM. El modo GCM (Galois Counter Mode), se trata de una “versión” del modo CTR (contador), y por tanto aprovecha todas las ventajas de este modo:

- Proporciona un nivel fuerte seguridad
- Permite la longitud de datos arbitraria sin necesidad de usar relleno
- Mayor eficiencia debido a que permite el procesamiento paralelo.

Además de las características propias del modo CTR, el modo GCM agrega autenticación de mensajes generando una etiqueta de autenticación de mensajes criptográficos.

GCM requiere de un vector de inicialización (IV), conocido también como nonce, y que por lo general suele generarse de forma aleatoria. El vector de inicialización (IV) es un bloque de bits que permite aleatorizar el cifrado y, por lo tanto, producir textos cifrados distintos incluso si el mismo texto plano se cifra varias veces, sin la necesidad de realizar un proceso de cambio de clave.

3.1.1.2. Funcionamiento de AES-GCM

El funcionamiento del modo GSM se describe en los siguientes pasos:

1. Al igual que en CTR, en el modo GSM los bloques se numeran secuencialmente. Este número de bloque no proporciona suficiente aleatoriedad por sí sólo, y por tanto se debe combinarse con un IV. Ambos se cifran con un cifrado de bloque (E), en nuestro caso el cifrado que se emplea es AES. El IV suele ser de 96 bits y el contador de 32 bits, creando un bloque deseado de 128 bits para el cifrado AES.
2. 2. Al resultado de esta encriptación se le aplica un XOR con el texto sin formato (plaintext), que genera los bloques de texto cifrado. Hasta este punto, únicamente se ha podido garantizar la confidencialidad de la información, exactamente igual que ocurre en el modo CTR. Sin embargo, también queremos garantizar la integridad de los datos.
3. Para garantizar la integridad de los datos el algoritmo realiza un XOR entre el bloque de texto cifrado y la salida de una función hash. El resultado de una etapa (bloque cifrado y autenticado) alimenta a la siguiente.
4. El último paso es añadirle a la concatenación, mediante un XOR, la longitud de los bloques (en el esquema: len(A) || len(C)). A este valor resultante se le aplica otro XOR con el valor del nonce inicial cifrado (IV) y se obtiene la etiqueta final.

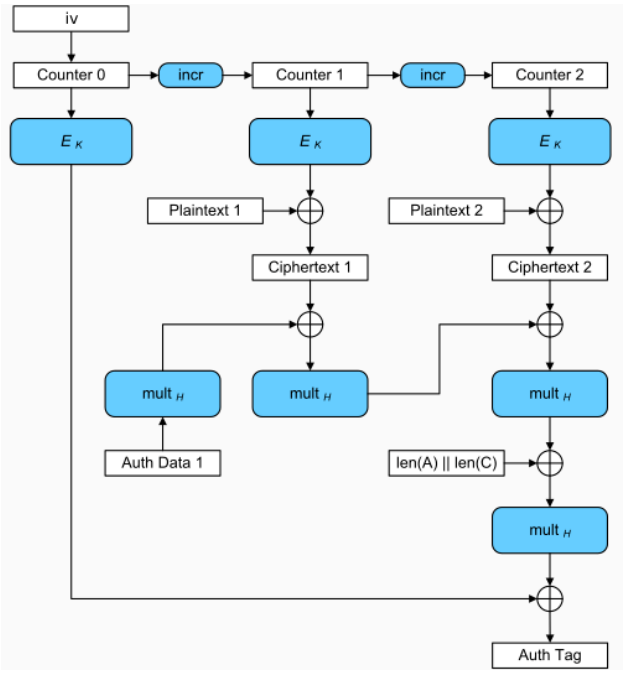


Figura 2. Esquema de funcionamiento de AES-GCM.

El resultado es un texto cifrado que contiene el IV, el texto cifrado y una etiqueta de autenticación de mensaje.

3.1.2.Chacha20

El algoritmo Chacha20 es un sucesor del algoritmo Salsa20. Ambos algoritmos operan de manera similar, pero Chacha20 ofrece mayor difusión por ronda y un mejor rendimiento. Se trata de un algoritmo de cifrado en flujo, desarrollado por Daniel Bernstein en el año 2008. Destaca porque no requiere de demasiada carga computacional, siendo especialmente útil en dispositivos de baja potencia, y junto con AES, uno de los algoritmos seleccionados para realizar el cifrado en el protocolo TLS 1.3.

3.1.2.1. Funcionamiento de Chacha20

El estado inicial de este algoritmo toma como entrada:

- Una clave secreta de 256 bits (8 palabras de 32 bits)
- Un nonce pseudoaleatorio de 96 bits (3 palabras de 32 bits)
- Un contador de 32 bits (1 palabra de 32 bits)
- Una constante de 128 bits (4 palabras de 32 bits; “expa”, “nd 3”, “2-by” y “te k”). Las palabras constantes son un conjunto de letras en ASCII, concretamente “Expand 32 byte k”, es decir, expandir 32 bytes k, que luego toman sus valores en hexadecimal.

Estas 16 palabras de 32 bits (enteros sin signo de 32 bits) están dispuestas en una matriz 4x4.

"expa"	"nd 3"	"2-by"	"te k"
Key	Key	Key	Key
Key	Key	Key	Key
Counter	Nonce	Nonce	Nonce

Figura 3. Disposición de palabras en Chacha20.

ChaCha20 se usa en modo de contador para derivar un flujo de clave que se somete a XOR con el texto sin formato. Cada uno genera un bloque cifrado.

Luego, el texto cifrado y los datos asociados se autentican utilizando la autenticación de mensajes de Poly1305.

Chacha20 aplica una función llamada Quarter-Round (QR), que realiza operaciones del tipo ARX (Add – Rotate - Xor), sobre una entrada de 4 palabras de 32 bits, y genera como salida otras 4 palabras distintas.

QR(a,b,c,d) = z_0 , z_1 , z_2 , z_3

La operación de cifrado se define como:

a=a+b; d=d⊕a; d<<16;

c=c+d; b=b⊕c; b<<12;

a=a+b; d=d⊕a; d<<8;

c=c+d; b=b⊕c; b<<7;

Notación operacional	
Símbolo	**Operación**
+	Suma de enteros modulo 2^{32}
⊕	OR exclusivo bit a bit
<<	Desplazamiento a izquierda. de n bits

También puede ser representada mediante el siguiente esquema, donde se aprecia de forma visual el flujo de operaciones que sigue el algoritmo.

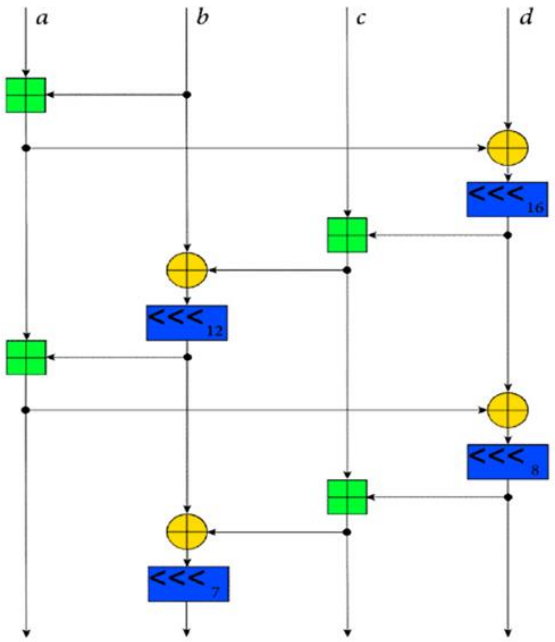


Figura 4. Esquema de operación de cifrado de Chacha20.

3.2. Criptografía asimétrica

La criptografía de clave asimétrica o criptografía de clave pública, se define como criptosistema utiliza dos claves, una clave pública y otra privada, para el enviar

la información de forma segura a través de un canal inseguro. La clave pública es la responsable del cifrado y la clave privada del descifrado.

Para que las claves sean seguras deben obtenerse mediante un problema matemático unidireccional, es decir, cuya solución es fácil en un sentido, pero no en el otro, aunque se conozcan algunos de sus parámetros. Los métodos más utilizados son:

- **RSA:** Se basa en la factorización de números primos grandes. Actualmente no se usa para el intercambio de claves por problemas de diseño. Su uso actual es únicamente para firmas digitales.
- **Diffie-Hellman:** Se basa a la ineficiencia computacional para resolver el Problema del Logaritmo Discreto en un campo finito cuando los números son de un tamaño considerable. Este problema se ha trabajado sobre cuerpos finitos, dando lugar a la versión de Diffie Hellman efímero sobre cuerpos finitos (FFDHE); y sobre curvas elípticas, dando lugar a versión de Diffie Hellman Efímero sobre Curva Elíptica (ECDHE). El protocolo de Diffie Hellman es uno de los más usados en la actualidad y es el empleado en la versión de TLS 1.3. Debido a su largo recorrido, y a la necesidad de aumentar los niveles de seguridad, ha ido sufriendo leves modificaciones, y es por ello encontramos adaptaciones respecto al protocolo original.

3.2.1. Versiones de Diffie Hellman

Diffie Hellman puede implementarse de diferentes formas, sin embargo, por motivos de seguridad, en la práctica únicamente se emplea Diffie Hellman Efímero (DHE) o Diffie Hellman Efímero sobre curvas elípticas (ECDHE).

- **Diffie Hellman anónimo:** Esta versión de DH no utiliza ningún método de autenticación de las partes, siendo vulnerables porque una tercera parte podría hacerse pasar por quien no es (ataque MITM). No debe implementarse.
- **Diffie Hellman estático:** Se le añade autenticación utilizando parámetros estáticos que se obtiene del certificado del cliente y del servidor para la creación de la clave compartida. Si se compromete alguno de los certificados se podría tener acceso a la información de todo el tráfico capturado en el pasado. No se emplea en la práctica.

- **Diffie Hellman efímero:** Se añaden claves efímeras **(DHE)**, que permite llegar a un acuerdo de clave, pero sin utilizar los parámetros estáticos de los certificados. Se consigue Forward Secrecy porque se evitan los ataques basados en la obtención de certificados, y una tercera parte malintencionada no puede descifrar la información de los datos capturados anteriormente. Esta versión puede considerarse segura.

3.3. Definición del PLD

El Problema del Logaritmo Discreto (PLD) en el que se basa Diffie Hellman se define en criptografía como:

Sea **p** un primo, y **a** un generador de ese primo, ambos números públicos, tenemos las siguientes funciones:

- La exponenciación modular se define en la formula: $\mathbf{y = a^x \bmod p}$

Donde **y**, **p** y **a** pueden ser valores públicos, siendo **x** el único valor privado (secreto). Independientemente de los valores de a, p y x, el cálculo será computacionalmente rápido. Sin embargo, conociendo los valores públicos (a, p, y), encontrar el valor privado **x** significaría ser capaces de resolver la ecuación $\mathbf{x=\log_a y \bmod p}$

Este problema se conoce como Problema del Logaritmo discreto, y se considera inabordable computacionalmente cuando p es un número primo muy grande.

3.3.1. Aplicación del PLD en Diffie Hellman

Para dos partes A y B que intentan establecer una clave secreta (K), y un posible adversario E a la escucha, el algoritmo consiste en los siguientes pasos:

Se establecen un primo **p** y un generador $\mathbf{g \in Z^*_p}$. Siendo Z* el conjunto de los enteros menores que “p”, que son primos relativos de éste y además es un grupo bajo la multiplicación módulo “p”. Además, los valores (“**g**” y “**p**”) son públicos.

1. A escoge $\mathbf{a \in Z_{p-1}}$ al azar, calcula $\mathbf{X = g^a \pmod p}$, y le envía el resultado X a B.
2. B escoge $\mathbf{b \in Z_{p-1}}$ al azar, calcula $\mathbf{Y = g^b \pmod p}$, y le envía el resultado Y a A.
3. A calcula $\mathbf{K = (g^b \bmod p)^a \bmod p \rightarrow K = (Y)^a \bmod p}$
4. B calcula $\mathbf{K = (g^a \bmod p)^b \bmod p \rightarrow K = (X)^b \bmod p}$

El resultado de K, es exactamente el mismo, y se trata de la clave compartida simétrica de ambas partes (A y B). El valor de K será imposible de calcular para un atacante porque no conoce ni “a” ni “b”, ya que, si un atacante llegase a capturar X e Y (valores que se transmiten por el canal inseguro entre A y B), y además conoce los parámetros públicos p y g, seguiría sin poder obtener el secreto compartido (K) porque no conoce ni “a” ni “b”, ya que es intratable computacionalmente obtener “a” dado X (problema del Logaritmo discreto en Z_p).

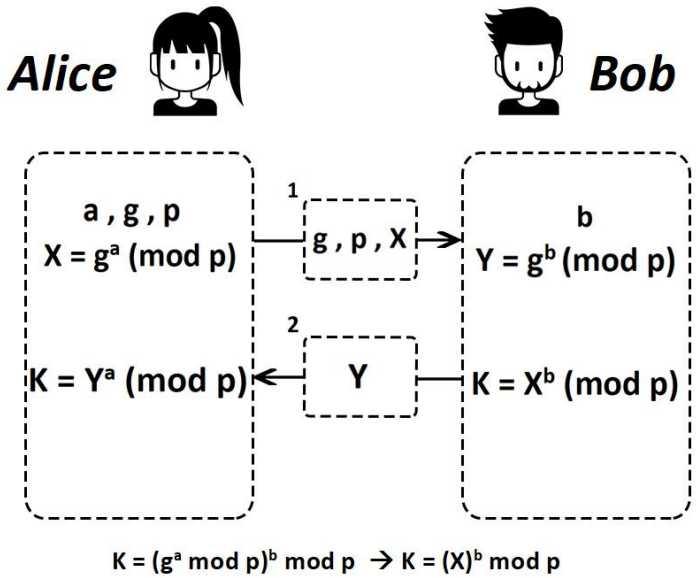


Figura 5. Intercambio de claves Diffie Hellman.

3.3.2.PLD aplicado a curvas elípticas

Diffie Hellman puede llevarse a cabo en cualquier grupo donde la exponenciación sea simple y el logaritmo discreto difícil, como por ejemplo en el grupo de puntos definidos por una curva elíptica sobre un campo finito.

En el caso anterior al trabajar con $\mathbf{Z_p}$, existen algunas desventajas de eficiencia sobre el protocolo Diffie-Hellman, ya que, el número **p** y las potencias **a** y **b** deben ser números de gran tamaño para que el protocolo sea útil (seguro); y esto hace que los cálculos necesarios no sean eficientes. El hecho de introducir curvas elípticas en criptosistemas se debe a la fácil construcción de un grupo y a la rapidez en el cálculo de su operación.

El Problema del Logaritmo Discreto Elíptico para una curva consiste en encontrar el entero positivo **x** tal que **[x]P=Q,** siendo **E** una curva elíptica sobre $\mathbf{Z_p}$, y dados dos puntos **P, Q ∈ E.**

La definición formal de una curva elíptica responde a la siguiente ecuación: **$y^2+axy+by=x^3+cx^2+dx+e$**, con a,b,c,d,e enteros.

Además, se define la suma de puntos **P+Q=R** de la forma:

- Si **P≠Q** y **P≠-Q** (donde si P=(x,y), -P=(x,-y)) y se traza la recta entre P y Q, se obtiene –R, que es el tercer punto de intersección de la recta con la curva
- Si **P=Q**, entonces -R es el punto de intersección de la recta tangente en P si hay, y si no hay es R=P
- Si **P=-Q**, R es el infinito

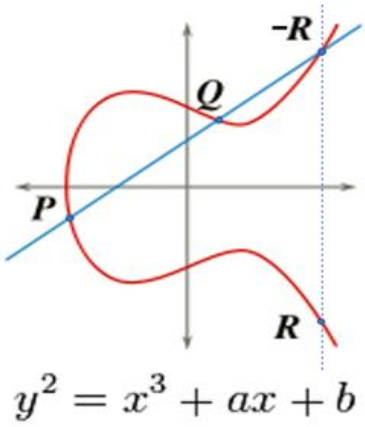


Figura 6. Ejemplo de curva elíptica.

En criptografía se utilizan curvas elípticas definidas sobre cuerpos finitos denominados como **F_p** (donde p es primo y p>3). Al estar definidas sobre cuerpos finitos las curvas tienen un número finito de puntos (matriz cuadrada de tamaño p x p), y todos los puntos en la curva tienen coordenadas de números enteros.

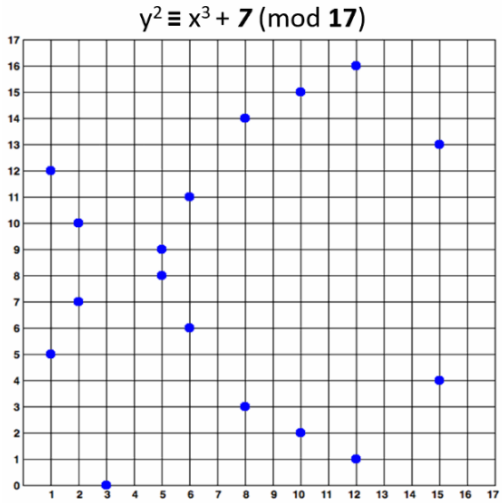


Figura 7. Curva elíptica sobre el campo finito F_{17}

Los algoritmos en Criptografía de Curva Elíptica (ECC, Elliptic Curve Cryptography) utilizan un par de claves (públicas/privadas), donde la clave privada es un número entero generado de forma aleatoria, y la clave pública es un punto en la curva elíptica (punto EC).

Diffie Hellman sobre curva elíptica se utiliza para crear una clave compartida de la siguiente manera:

1. Bob generará una clave pública y una clave privada tomando un punto en la curva. La clave privada es un número aleatorio ($\mathbf{d_B}$) y la clave pública de Bob ($\mathbf{Q_B}$) será: $\mathbf{Q_B = d_B * G}$, siendo G un punto base generador de la curva.
2. Alice hará lo mismo, generará su clave pública ($\mathbf{Q_A}$) a partir de su clave privada (d_A): $\mathbf{Q_A = d_A * G}$
3. Luego intercambian sus claves públicas.
4. Alice usará la clave pública de Bob y su clave privada para calcular: $\mathbf{ClaveCompartida_{Alice} = d_A * Q_B}$ → Es lo mismo que $ClaveCompartida_{Alice} = d_A * d_B * G$
5. Bob usará la clave pública de Alice y su clave privada para calcular: $\mathbf{ClaveCompartida_{Bob} = d_B * Q_A}$ → Es lo mismo que $ClaveCompartida_{Bob} = d_B * d_A * G$
6. Podemos ver que ambos alcanzan una clave común.

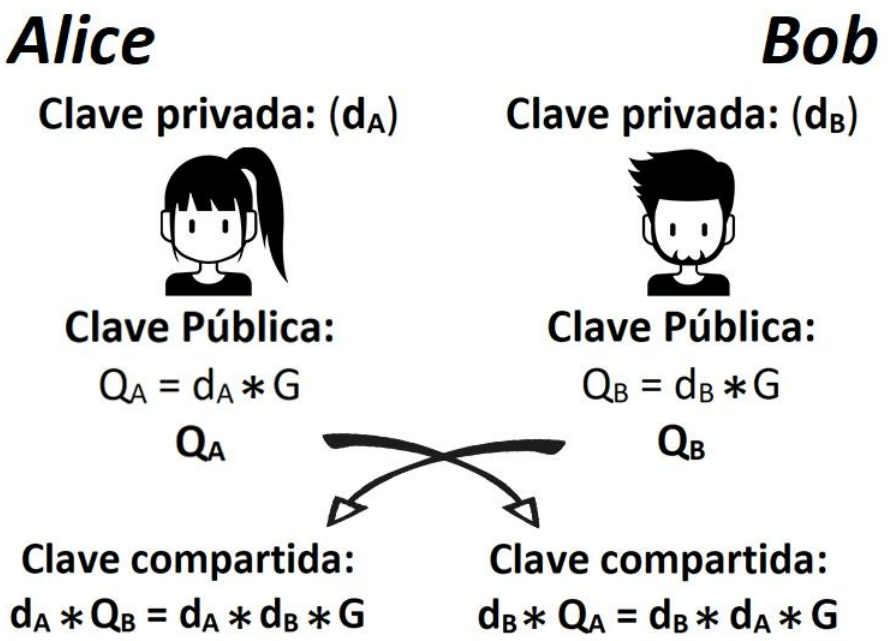


Figura 8. Intercambio de clave Diffie Hellman con curva elíptica.

La ventaja de utilizar Diffie Hellman aplicado a Curvas Elípticas es que hace posible que para un nivel de seguridad similar se utilicen operandos más pequeños. El cambio sobre las operaciones entre las dos versiones de Diffie

Hellman se traduce en una mayor eficiencia para el sistema de curva elíptica, ya que, el logaritmo elíptico es la formulación aditiva del logaritmo discreto, es decir, en el PLD nos enfrentamos a operaciones de exponenciación y productos, mientras que en el ECPLD estas operaciones se transforman en sumas y productos (requiere menos carga computacional).

3.4. DHE y ECDHE en TLS 1.3

En TLS 1.3, cuando un cliente envía la extensión “supported_groups” (que será explicada en los siguientes apartados de este documento), está indicando los nombres de los grupos Diffie Hellman de Campos Finitos y las Curvas Elípticas que admite para el intercambio de claves [8] [9].

```
/* Elliptic Curve Groups (ECDHE) */
secp256r1(0x0017), secp384r1(0x0018), secp521r1(0x0019),
x25519(0x001D), x448(0x001E),

/* Finite Field Groups (DHE) */
ffdhe2048(0x0100), ffdhe3072(0x0101), ffdhe4096(0x0102),
ffdhe6144(0x0103), ffdhe8192(0x0104),
```

Figura 9. ECDHE y DHE soportados en TLS 1.3. [Apartado 4.2.7 del RFC8446]

3.4.1. Cuerpos finitos DH admitidos en TLS 1.3

Las implementaciones de TLS que utilicen un intercambio de claves con cuerpos finitos (FFDHE), deben considerar la fuerza del grupo que negocian, puesto que al seleccionar un grupo se está eligiendo la fortaleza y resistencia de la conexión frente a ataques de confidencialidad e integridad de la sesión, ya que, las claves de sesión derivan del protocolo DHE, y por tanto del grupo seleccionado.
Los grupos propuestos responden a la siguiente descripción [9] [10] [11]:

p = 2^b - 2^{b-64} + {[2^{b-130} e] + X} * 2^64 - 1

- **p** = en estos grupos de campos finitos los primos (**p**) son todos primos seguros; es decir, se cumple que **q = (p-1)/2** también es primo.
- **e** = base del logaritmo natural
- **b** = longitud de bit asignada

- **X** = entero positivo más bajo que crea un primo seguro que cumple la ecuación

Por ejemplo, para el grupo **ffdhe2048** cuya longitud es de 2048 bits, se calcula a partir de la fórmula:

p = 2^2048 - 2^1984 + {[2^1918 * e] + 560316} * 2^64 – 1

- El generador es: **g = 2**
- El tamaño del grupo es: **q = (p-1)/2**
- La fuerza equivalente simétrica estimada de este grupo es de 103 bits.
- Los pares que usan ffdhe2048 y desean optimizar su intercambio de claves con un exponente corto deben elegir una clave secreta de al menos 225 bits.

El resto de grupos admitidos se corresponde con:

Grupo ffdhe3072: p = 2^3072 - 2^3008 + {[2^2942 * e] + 2625351} * 2^64 - 1

Grupo ffdhe4096: p = 2^4096 - 2^4032 + {[2^3966 * e] + 5736041} * 2^64 - 1

Grupo ffdhe6144: p = 2^6144 - 2^6080 + {[2^6014 * e] + 15705020} * 2^64 - 1

Grupo ffdhe8192: p = 2^8192 - 2^8128 + {[2^8062 * e] + 10965728} * 2^64 - 1

En la Tabla 2 se representan, para cada uno de los cuerpos finitos admitidos en TLS 1.3, la longitud de los parámetros, es decir, de p. Su fortaleza simétrica estimada y el tamaño de clave necesario. Las especificaciones de los cuerpos finitos admitidos han sido obtenidas del RFC7919.

Grupo	Longitud (bits) de los parámetros (p)	Fortaleza (bits)	Clave superior a: (bits)
ffdhe2048	**2048**	**103**	**225**
ffdhe3072	**3072**	**125**	**275**
ffdhe4096	**4096**	**150**	**325**
ffdhe6144	**6144**	**175**	**375**
ffdhe8192	**8192**	**192**	**400**

Tabla 2. Grupos de cuerpos finitos admitidos en TLS 1.3.

3.4.2.Curvas elípticas admitidas en TLS 1.3

Para utilizar criptografía de curva elíptica (ECC), todas las partes deben estar de acuerdo en los parámetros de dominio del esquema, es decir, en todos los elementos que definen la curva elíptica que normalmente viene descrita por una tupla como (p,a,b,G,n,h), donde:

- **p:** Número primo.
- **a:** coeficiente a de la ecuación de la curva en particular.
- **b:** coeficiente b de la ecuación de la curva en particular.
- **G:** es el punto (constante) generador de la curva que permite generar cualquier otro punto sobre la curva elíptica al ser multiplicado por un entero en el rango **[0-r]**, siendo r el orden del subgrupo cíclico (número de puntos en el subgrupo). **r** se define como: **h = n/r**
- **n:** el orden (tamaño) de la curva, es decir, el número de todos sus puntos.
- **h:** es el cofactor de la curva. Los puntos sobre una curva elíptica permanecen en uno o varios subconjuntos no superpuestos, llamados subgrupos cíclicos. El número de subgrupos se llama "cofactor".

Los algoritmos criptográficos de curva elíptica pueden utilizar diferentes curvas, que a su vez proporcionan diferentes niveles de seguridad (fuerza criptográfica), diferente rendimiento y diferentes longitudes de clave (velocidad).

Para disponer de una solidez criptográfica, los investigadores seleccionan cuidadosamente los parámetros del dominio de la curva elíptica (ecuación de la curva, punto generador, cofactor, etc.). Las curvas populares y estandarizadas son conocidas por su nombre.

El NIST definió 25 curvas para usar en TLS. Las 22 primeras no se usan y únicamente se utilizan las curvas 23, 24 y 25, que corresponden a las Curvas **secp256r1, secp384r1, secp521r1** respectivamente. Las curvas **x25519** y **x448**, correspondientes en TLS 1.3 a los identificadores 29 y 30 respectivamente, se definen en el RFC7748. Se pueden encontrar los parámetros recomendados para cada una de estas curvas en el Grupo de Estándares Eficientes para Criptografía (SECG) [9].

Para visualizar un ejemplo de cómo vienen definidas estas curvas, se procede a visualizar los parámetros del dominio de la curva elíptica secp256r1 sobre F_p, que se especifican mediante (p,a,b,G,n,h) donde el campo finito F_p se define por:

- **p** = FFFFFFFF 00000001 00000000 00000000 00000000 FFFFFFFF FFFFFFFF FFFFFFFF → $p = 2^{224}(2^{32} - 1) + 2^{192} + 2^{96} - 1$

La curva E: $\mathbf{y^2 = x^3 + ax + b}$ sobre $\mathbf{F_p}$

- **a** = FFFFFFFF 00000001 00000000 00000000 00000000 FFFFFFFF FFFFFFFF FFFFFFFC

- **b** = 5AC635D8 AA3A93E7 B3EBBD55 769886BC 651D06B0 CC53B0F6 3BCE3C3E 27D2604B

La elección de sus parámetros aleatorios (a, b) viene especificada en el estándar ANSI X9.62, publicado por el American National Standards Institute donde se proporciona una forma de generar curvas elípticas aleatorias verificables. En este caso se utiliza la semilla:

S = C49D360886E704936A6678E1139D26B7819F7E90

El punto base o generador:

En formato comprimido:

- **G** = 03 6B17D1F2 E12C4247 F8BCE6E5 63A440F2 77037D81 2DEB33A0 F4A13945 D898C296

En formato sin comprimir:

- **G** = 04 6B17D1F2 E12C4247 F8BCE6E5 63A440F2 77037D81 2DEB33A0 F4A13945 D898C296 4FE342E2 FE1A7F9B 8EE7EB4A 7C0F9E16 2BCE3357 6B315ECE CBB64068 37BF51F5

Y por último el orden (n) y el cofactor (h):

- **n** = FFFFFFFF 00000000 FFFFFFFF FFFFFFFF BCE6FAAD A7179E84 F3B9CAC2 FC632551
- **h** = 01

3.4.2.1. Curvas elípticas definidas por el NIST

El NIST define tres tipos de curvas, sobre cuerpos primos, sobre cuerpos binarios y de tipo Koblitz. Las ecuaciones para cada uno de los grupos definidos anteriormente, y los nombres que han sido asignados a cada una de ellas son:

- Curvas NIST sobre cuerpos primos son curvas pseudoaleatorias de la forma: $\mathbf{y^2 = x^3 - 3x + b}$. Son denominadas con los nombres P-192, P-224, P-256, P-384 y P-521.
- Curvas NIST sobre cuerpos binarios son curvas pseudoaleatorias de la forma $\mathbf{y^2 + xy = x^3 + x^2 + b}$. Se conocen con los nombres B-163, B-233, B-283, B-409 y B-571.
- Curvas NIST tipo Koblitz son de la forma $\mathbf{y^2 + xy = x^3 + ax^2 + 1}$, donde a=1 o a=0. Las curvas Koblitz más conocidas son K-163, K-233, K-283, K-409 y K-571.

3.4.2.2. Curva elíptica x25519

Es importante conocer la curva 25519, porque es una de las más utilizadas y eficientes, siendo considerada una de las curvas más seguras. Fue propuesta por Daniel J. Bernstein en el año 2005, pero su uso se popularizó a partir del año 2013 debido a las sospechas de la NSA, de haber podido implementar una puerta trasera sobre la familia de curvas que aconsejaban utilizar hasta el momento. Con el tiempo se ha convertido en una alternativa a la curva P-256 y ha sido implantada en numerosos sistemas informáticos. Pertenece a la familia llamada curvas de Montgomery y viene definida por la ecuación:
$y^2 = x^3 + 486662x^2 + x$, sobre el cuerpo finito definido por el primo **$p=2^{255} -19$**, y tomando como punto base **x=9**.

3.4.2.3. Fuerza de seguridad de las curvas elípticas

El algoritmo más rápido conocido permite resolver el Problema del Logaritmo Elíptico, para una clave de tamaño k, en $\sqrt{k}$ pasos, esto significa que para lograr una fuerza de seguridad de k bits, se necesita al menos una curva de 2*k bits. La Tabla 3 indica las 5 curvas elípticas que son soportadas en TLS 1.3 y las versiones anteriores del protocolo. Además, también indica la fuerza de seguridad para cada una de estas curvas.

Curva Elíptica	Identificador de Curva Elíptica	Fortaleza (bits)	Versiones de TLS que la soportan
secp256r1	**23**	**128**	**TLS V1.0, V1.1, V1.2, V1.3**
secp384r1	**24**	**192**	**TLS V1.0, V1.1, V1.2, V1.3**
secp521r1	**25**	**256**	**TLS V1.0, V1.1, V1.2, V1.3**
x25519	**29**	**128**	**TLS V1.3**
x448	**30**	**224**	**TLS V1.3**

Tabla 3. Curvas Elípticas admitidas en TLS.

3.5. Comparación de tamaños de claves ECC-RSA

Como se ha comentado anteriormente, al utilizar criptografía de curva elíptica mejora muchísimo la eficiencia con respecto a otros métodos, como por ejemplo RSA. En la Tabla 4, se efectúa una comparación de los tamaños de clave necesarios para un mismo nivel de seguridad, comparando RSA con ECC. Se

puede apreciar que, en Curva Elíptica, para un nivel de seguridad alto, sólo se necesitan 283 bits frente a 3072 bits en RSA.

Tamaño de clave RSA (bits)	Tamaño de clave en ECC (bits)
1024	163
2240	233
3072	283

Tabla 4. Comparación de tamaños de claves ECC-RSA.

4. Arquitectura del protocolo TLS

Según el esquema tradicional de redes, utilizando los modelos OSI o TCP/IP, el protocolo TLS actúa siempre en la capa de transporte, dando seguridad a los protocolos de los niveles superiores, como, por ejemplo, al protocolo HTTP situado en la capa de aplicación.

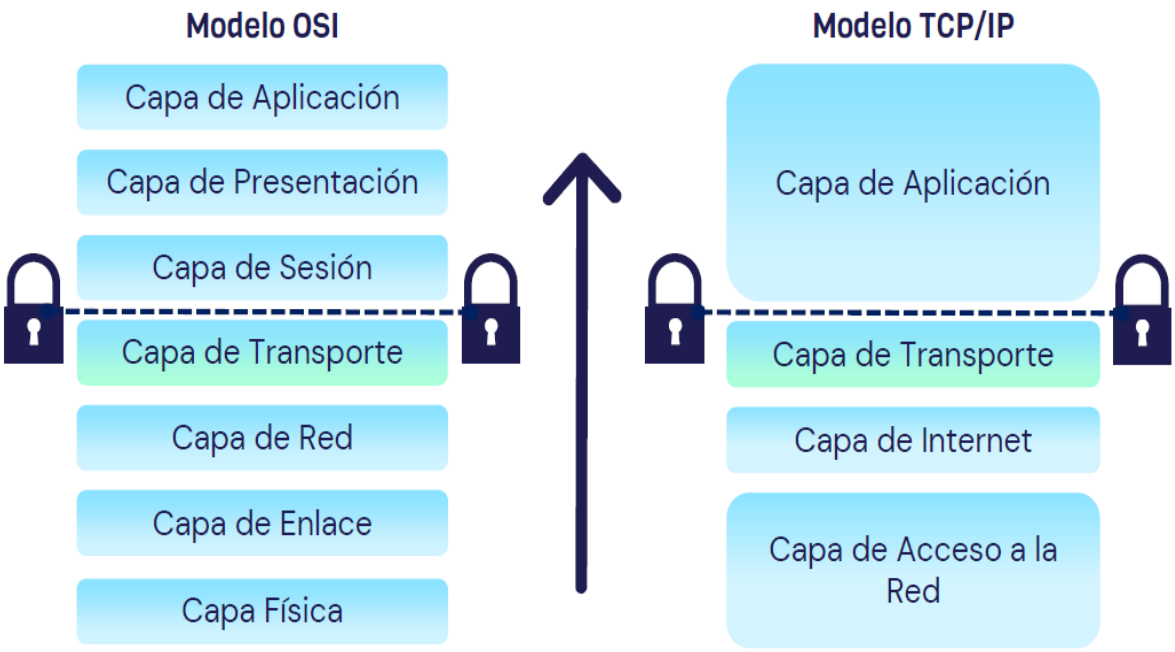


Figura 10. Nivel de actuación del protocolo TLS sobre los esquemas de redes.

El protocolo TLS se divide en dos fases: el protocolo de handshake y el protocolo de registro.

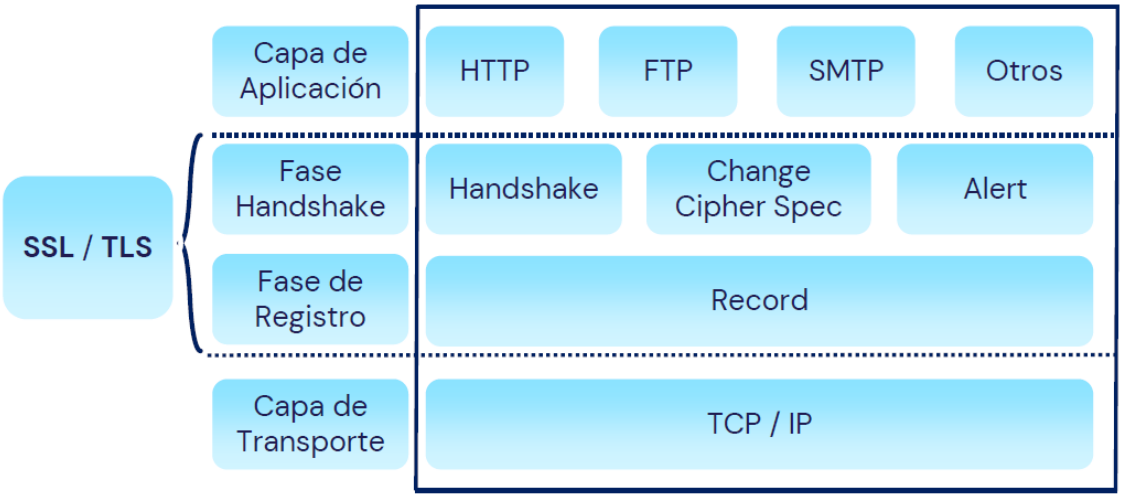


Figura 11. Arquitectura del protocolo TLS.

4.1. Protocolo de handshake

En cualquier comunicación cifrada, el primer punto a tener en cuenta es el intercambio de claves entre las dos partes. El protocolo TLS resuelve este problema mediante el protocolo de handshake, que se encarga de realizar la configuración de la conexión de la siguiente manera:

- Negocia la versión de protocolo TLS y la suite de cifrado que se van a emplear.
- Realiza un intercambio de claves entre emisor y receptor.
- Autentica a los participantes de la comunicación utilizando una infraestructura de clave pública para la verificación de certificados.

En el protocolo de handshake también intervienen otros dos protocolos, el protocolo de alerta y el protocolo de especificación de cifrado.

4.1.1. Protocolo de alerta

Se encarga de informar e identificar los diferentes tipos de mensajes de alerta. Incluye un mensaje indicando la severidad de la alerta (Warning o Fatal) y una descripción de la misma. Si la severidad de la alerta es Fatal, la conexión es interrumpida.

4.1.2. Protocolo de especificación de cifrado

Se compone únicamente de un mensaje que el cliente envía al servidor para indicar el cambio en la estrategia de cifrado, no es utilizado en la última versión de TLS.

4.2. Protocolo de registro

En esta fase, el protocolo de registro se encarga de transmitir los datos de forma segura. Para ello, los datos y la clave intercambiada entre cliente y servidor son cifrados mediante algoritmos simétricos, como, por ejemplo, AES.
Además, para comprobar si se han manipulado los flujos de datos, usando una función hash se añade un MAC (Message Authentication Code) que sólo puede ser interpretado por el remitente y el destinatario que tienen la clave usada. Cabe destacar, que la compresión no se utiliza en la versión de TLS 1.3.

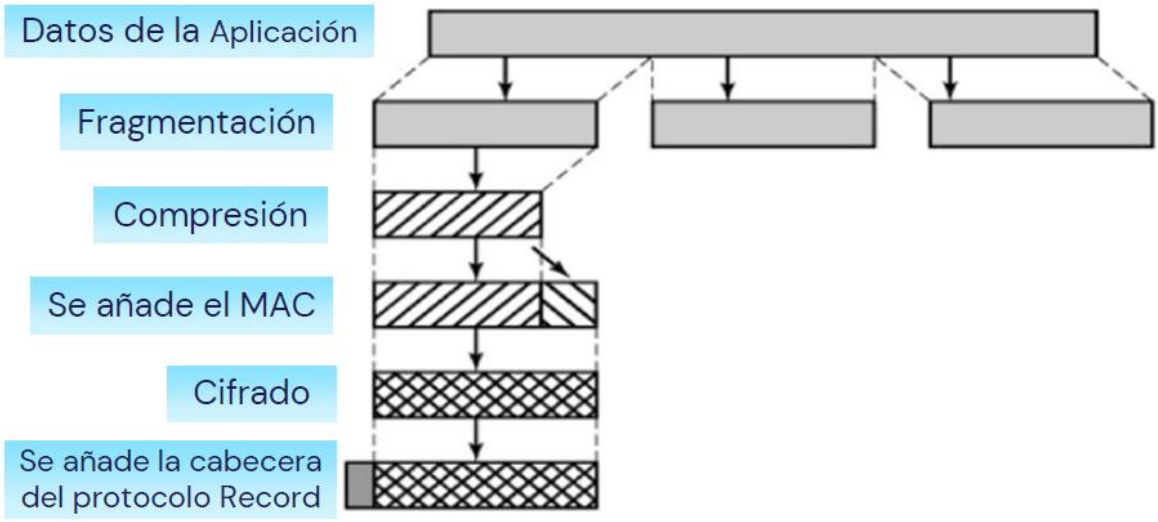


Figura 12. Procedimiento con el que opera el protocolo de registro.

5. Mejoras de la versión TLS 1.3

5.1. Mejoras de eficiencia

La versión de TLS 1.3 es mucho más eficiente que la versión anterior, debido a simplificaciones en la máquina de estados del handshake y a la eliminación del envío de mensajes superfluos, como, por ejemplo, el “ChangeCipherSpec”.

Además, TLS 1.3 permite un nuevo método de transmisión de información confidencial, conocido como modo 0-RTT (Zero Round Trip Time). Este método permite a un cliente que ha visitado un servidor web, reciclar las claves previamente compartidas de una sesión anterior, reduciendo enormemente los tiempos de latencia en las redes porque el cliente puede enviar datos cifrados desde un primer instante, sin necesidad de renegociar las claves nuevamente.
En la Figura 13, se puede ver que, para finalizar el protocolo, TLS 1.2 necesita el doble de interacciones que TLS 1.3.

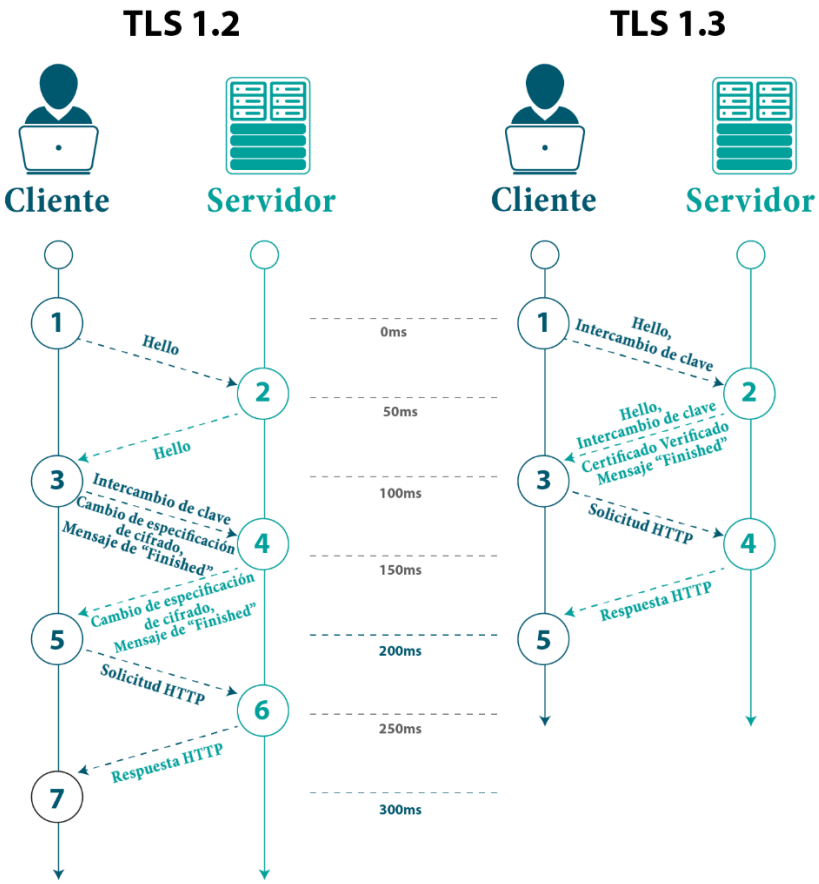


Figura 13. Mejoras de eficiencia en TLS 1.3.

5.2. Mejoras de seguridad

En TLS 1.3, a diferencia de las versiones anteriores, todos los mensajes tras el Server Hello viajan cifrados, lo que aumenta la seguridad. De hecho, la versión de TLS 1.3 destaca por ser la más segura de todas las versiones del protocolo, debido principalmente a la eliminación de todos los algoritmos simétricos

heredados considerados obsoletos, como, por ejemplo, MD5, DES, RC4, AES-CBC, etc. De hecho, los únicos algoritmos de cifrado que se utilizan son los de tipo AEAD (Authenticated Encryption with Associated Data), que son capaces de proporcionar tanto confidencialidad como integridad en la misma operación.

Cuando se establece una comunicación TLS, el ofrece una lista ordenada por preferencia de los conjuntos de cifrados compatibles. La función de una suite de cifrado es establecer un conjunto específico de algoritmos criptográficos a utilizar para el establecimiento de una sesión TLS. En las versiones anteriores de TLS la suite de cifrado incluía intercambio de claves, cifrado y algoritmos MAC. Sin embargo, en TLS 1.3 se separa el mecanismo de intercambio de claves del algoritmo de cifrado.

La suite de cifrado de TLS 1.3 sólo utiliza el algoritmo de cifrado en bloque AES en modo GCM, y el algoritmo de cifrado en flujo Chacha20, que aunque inicialmente surge como una alternativa a AES, en los últimos años ha tomado fuerza porque presenta un mejor rendimiento.

La suite de cifrados admitidos en TLS 1.3 se resume en la Tabla 5.

Cipher Suites en TLS 1.3	
Valor	**Cipher Suite**
0x01	**TLS_AES_128_GCM_SHA256**
0x02	**TLS_AES_256_GCM_SHA384**
0x03	**TLS_CHACHA20_POLY1305_SHA256**
0x04	**TLS_AES_128_CCM_SHA256**
0x05	**TLS_AES_128_CCM_8_SHA256**

Tabla 5. Cipher Suite TLS1.3

Un aspecto relevante es que en la versión de TLS 1.3 toda la criptografía de clave pública debe proporcionar secreto perfecto hacia delante (forward secrecy), por lo que también han sido eliminadas las suites de cifrados estáticos RSA y Diffie-Hellman. Los únicos mecanismos de intercambio de claves con los que se trabaja en TLS 1.3 son Diffie-Hellman Efímero sobre cuerpos finitos (DHE) y Diffie-Hellman Efímero con Curvas Elípticas (ECDHE), siendo este último más utilizado debido a la eficiencia de trabajar con curvas elípticas. Por este motivo, la criptografía de curva elíptica se encuentra en la especificación de TLS 1.3, donde se incluyen nuevos algoritmos de firma digital como el de la curva de Edwards EdDSA (Edwards-curve Digital Signature Algorithm) [5].

Otro cambio importante es que en la versión de TLS 1.3 no se admite compresión debido a que se ha probado que las funciones empleadas en versiones anteriores para comprimir presentan vulnerabilidades.

Todas las modificaciones descritas hacen de TLS 1.3 un protocolo más robusto y rápido que las versiones anteriores.

Intercambio de claves y autenticación				
Algoritmo	**TLS 1.0**	**TLS 1.1**	**TLS 1.2**	**TLS 1.3**
RSA	Sí	Sí	Sí	No
DH-RSA	Sí	Sí	Sí	No
DHE-RSA (forward secrecy)	Sí	Sí	Sí	Sí
ECDH-RSA	Sí	Sí	Sí	No
ECDHE-RSA (forward secrecy)	Sí	Sí	Sí	Sí
DH-DSS	Sí	Sí	Sí	No
DHE-DSS (forward secrecy)	Sí	Sí	Sí	No
ECDH-ECDSA	Sí	Sí	Sí	No
ECDHE-ECDSA (forward secrecy)	Sí	Sí	Sí	Sí
ECDH-EdDSA	Sí	Sí	Sí	No
ECDHE-EdDSA (forward secrecy)	Sí	Sí	Sí	Sí
PSK	Sí	Sí	Sí	No
PSK-RSA	Sí	Sí	Sí	No
DHE-PSK (forward secrecy)	Sí	Sí	Sí	Sí
ECDHE-PSK (forward secrecy)	Sí	Sí	Sí	Sí
SRP	Sí	Sí	Sí	No
SRP-DSS	Sí	Sí	Sí	No
SRP-RSA	Sí	Sí	Sí	No
Kerberos	Sí	Sí	Sí	No

Tabla 6. Algoritmos de intercambio de claves y autenticación admitidos en las diferentes versiones de TLS.

Integridad de los datos				
Algoritmo	**TLS 1.0**	**TLS 1.1**	**TLS 1.2**	**TLS 1.3**
HMAC - MD5	Sí	Sí	Sí	No
HMAC - SHA1	Sí	Sí	Sí	No
HMAC - SHA256/384	No	No	Sí	No
AEAD	No	No	Sí	Sí

Tabla 7. Algoritmos de integridad de datos admitidos en las diferentes versiones de TLS.

6. Funcionamiento del handshake en TLS

Como se puede apreciar en la Figura 13, en la versión de TLS 1.3 el handshake se realiza en menos pasos que en versiones anteriores del protocolo. Esto se debe a que se han unificado las fases donde se envían los mensajes de “Hello” con el intercambio de clave. Durante el desarrollo de este trabajo fue necesario realizar un estudio exhaustivo para comprender el funcionamiento del protocolo, utilizado el software de Wireshark se efectúa un estudio profundo del contenido y trazabilidad de los mensajes de Client Hello y Server Hello.

6.1. Client Hello

El mensaje de Client Hello contiene:

- **Client Version**: su valor (0x0303) es siempre el mismo porque corresponde a la versión de TSL 1.2. Este campo en TLS 1.3 ya no se utiliza, pero debe indicarse por compatibilidad. En su lugar, la versión es especificada en la extensión que se comenta más adelante. TLS 1.3 se enmascara porque en la práctica no estaba incluido en la mayoría de middlebox desplegadas, que tienen reglas muy estrictas de cómo debe funcionar TLS 1.2.
- **Cipher Suites**: el cliente ofrece una lista ordenada por preferencia de los conjuntos de cifrados compatibles.
- **Session ID**: este campo tampoco se utiliza en TLS 1.3 porque el resumen de la reanudación de la sesión se realiza a través del mecanismo de claves pre-compartidas. Sin embargo, por compatibilidad se debe rellenar de manera aleatoria para no ocasionar problemas con versiones anteriores. Por su parte, el servidor responde con el mismo valor.
- **Client Random:** un número aleatorio de 32 bytes que se utiliza para derivar el material de claves.
- **Compression Methods:** TLS 1.3 no permite compresión y por eso se rellena el valor del campo como nulo (0x00). La compresión ocasionó varios ciberataques en versiones anteriores del protocolo, alguno de ellos muy conocido, como CRIME [6].

El resto del mensaje está compuesto por un grupo de extensiones que se introducen de forma consecutiva al final de los mensajes de Client Hello y Server Hello. Estas extensiones añaden funcionalidad al protocolo sin necesidad de tener que modificarlo. Las extensiones son utilizadas por protocolos de capas superiores, por ejemplo, por HTTP. A continuación, se nombran algunas de las más relevantes [7]:

- **Extension Server Name:** el cliente debe indicar el nombre del servidor con el que quiere conectarse para garantizar que se pueda descargar el certificado SSL correcto. Esta extensión permite la virtualización de hosts para una misma IP.
- **Extension Master Secret**: ofrece seguridad adicional a las sesiones TLS. Permite a ambas partes ponerse de acuerdo y usar el cálculo de secreto maestro extendido.
- **Extension Renegotiation info:** No se utiliza en el protocolo TLS 1.3 porque se reemplaza la renegociación y sus deficiencias por un mecanismo simple de actualización de claves para una sesión segura existente.
- **Extension Supported Groups:** indica una lista por orden de preferencia con los nombres de los grupos finitos y curvas elípticas soportados para el intercambio de claves Diffie-Hellman.
- **Extension EC Point Formats:** indica cómo recibir el formato de los puntos de curva elíptica.
- **Extension Session Ticket:** esta extensión permite al servidor guardar la información de la sesión, cifrarla y enviarla al cliente en forma de ticket. En TLS 1.3 los tickets son rotados con frecuencia y solo se utilizan para reanudar una conexión. Los clientes indican si soportan este mecanismo enviando la extensión Session Ticket vacía en el Client Hello, y un servidor que soporte esta extensión introducirá el mismo campo vacío en el Server Hello.
- **Extension ALPN:** esta extensión permite la negociación de diferentes protocolos de capa de aplicación sobre una conexión TLS. Si un cliente soporta la extensión ALPN la usa en el mensaje de Client Hello para listar los protocolos de capa de aplicación, informando al servidor de los

protocolos que soporta. Si el servidor lo soporta, elegirá el protocolo que desee el cliente. En caso contrario, simplemente ignorará esta extensión.

- **Extension Status Request:** esta extensión permite verificar la validez de los certificados del servidor en el protocolo de enlace TLS, evitando así la transmisión de Listas de Revocación de Certificados CRL (Certificate Revocation Lists), y ahorrando ancho de banda y recursos.
- **Extension Delegated Credentials:** permite crear certificados de corta duración, cuyo objetivo es prevenir el uso indebido de certificados comprometidos al reducir su periodo máximo de validez a un periodo de tiempo muy corto, que incluso puede ser de sólo unas pocas horas de uso.
- **Extension Key Share:** en la versión de TLS 1.3, el cliente selecciona un método de intercambio de clave y supone que el servidor puede aceptarlo. Esto permite que el resto de mensajes del handshake en TLS 1.3 vayan cifrados. Si el servidor acepta el método seleccionado por el cliente, se efectúa el intercambio. En caso contrario, el servidor envía un mensaje de reintento.
- **Extension Supported Versions:** es con esta extensión donde el cliente confirma qué versión TLS soporta. El valor 0x0303 es el valor asignado para la versión TLS1.2 y el valor 0x0304 es para la versión de TLS 1.3.
- **Extension Signature Algorithms:** especifica los algoritmos de firma digital e integridad que admite el cliente. Influye en pasos posteriores relacionados con los certificados. La lista de algoritmos se presenta de forma ordenada por preferencia, siendo los más utilizados los algoritmos basados en ECC. Los algoritmos de firma utilizados en TLS 1.3 son RSA, ECDSA y EdDSA [12].
- **Extension PSK Key Exchange Modes:** con esta extensión el cliente informa sobre los modos disponibles para establecer claves precompartidas PSK (PreShared Keys).
- **Extension Record Size Limit:** se utiliza para limitar el tamaño de los registros que se crean al codificar los datos de la aplicación. Los mensajes desprotegidos no están sujetos al límite establecido.

6.2. Server Hello

El mensaje de Server Hello es la respuesta por parte del servidor al mensaje de Client Hello. El contenido en TLS 1.3 del Server Hello es el siguiente:

- **Server Version:** igual que como ocurre en el Client Hello, este campo en TLS 1.3 ya no se utiliza, pero debe indicarse por compatibilidad.
- **Cipher Suites:** en este campo el servidor selecciona el conjunto de cifrado en función de las propuestas ofertadas por el cliente en su mensaje de Hello.
- **Session ID:** es un campo heredado de versiones anteriores, y en TLS 1.3 el servidor simplemente repite el ID de sesión enviado por el cliente.
- **Server Random:** el servidor genera un número aleatorio de 32 bytes, que junto con el Client Random se utiliza para derivar el material de claves.
- **Compression Methods:** el campo correspondiente a la compresión no realiza ningún cambio porque en TLS 1.3 no es admitida.

Con respecto a las extensiones enviadas en el Server Hello, el servidor se asegura de que el cliente comprende y admite todas las extensiones porque no puede responder con una extensión que el cliente no haya enviado en su mensaje de Client Hello.

- **Extension Supported Version:** es aquí donde el servidor indica la versión de TLS negociada.
- **Extension Key Share:** La otra extensión que envía el servidor es para la clave compartida. El servidor envía la clave pública utilizando el algoritmo de intercambio de clave seleccionado por el cliente, normalmente ECDHE con curva x25519. Tras haber sido enviada se puede calcular las claves de cifrado y cifrar el resto del protocolo.

En la versión de TLS1.3 ocurre una diferencia significativa con respecto a las versiones anteriores, ya que tras recibir el Client Hello, el servidor dispone de todos los parámetros requeridos para generar el master secret.

6.3. Intercambio y derivación de claves

6.3.1. Intercambios de clave

El intercambio de claves se efectúa siguiendo los pasos que se describen a continuación:

1. El intercambio de claves en una comunicación TLS empieza con la generación de un par de claves (pública/privada) por parte del cliente. El cliente elige la clave privada seleccionando un número entero aleatorio entre 0 y 2^{256}-1, es decir, generando 32 bytes (256 bits) de datos aleatorios.
    - Clave privada del cliente: $\mathbf{k_{cliente}}$= número aleatorio del cliente

    La clave pública del cliente equivale a la multiplicación de la clave privada por un punto de la curva (P).
    - Curva: x25519 → $y^2 = x^3 + 486662x^2 + x$
    - P: punto de la curva, donde x=9
    - Clave pública del cliente: $\mathbf{k_{cliente}*P}$
2. Acto seguido, el cliente proporciona, en el mensaje de Client Hello, toda la información que necesita conocer el servidor para calcular las claves que permiten cifrar el resto del protocolo de enlace.
3. El servidor también genera un par de claves (pública/privada). El servidor elige la clave privada seleccionando un número entero aleatorio entre 0 y 2^{256}-1, es decir, generando 32 bytes (256 bits) de datos aleatorios.
    - Clave privada del servidor: $\mathbf{k_{servidor}}$ = número aleatorio del servidor

    La clave pública del servidor se calcula de la misma forma que la clave pública del cliente, es decir, multiplicando la clave privada por el punto P.
    - Clave pública del servidor: $\mathbf{k_{servidor}*P}$
4. Con la información anterior, el servidor calcula el secreto compartido, que sólo es el resultado del intercambio de claves que permite que el cliente y el servidor acuerden un mismo número. Para ello, el servidor multiplica la clave pública del cliente por la clave privada del servidor utilizando la curva elegida (curve25519), y se obtiene como resultado el secreto compartido.
    - El secreto compartido será: $\mathbf{k_{cliente} * P * k_{servido}}$

6.3.2. Derivación de clave

Una vez disponemos de la clave de origen, ya sea, el secreto compartido obtenido a través del protocolo de intercambio de clave DH, o una clave precompartida (PSK) correspondiente a una sesión anteriormente establecida, se debe ejecutar dicha clave a través de una función de derivación de clave (KDF, Key Derivation Function) para generar todas las claves de sesión necesarias. El KDF típico es HKDF, que está basado en HMAC.

HKDF está compuesto por dos fases, una fase de extracción (HKDF-Extract) y una fase de expansión (HKDF-Expand). La fase de expansión debe ir siempre precedida de una fase de extracción, por lo que la primera fase será siempre una fase de extracción.

- **HKDF-Extract:** La primera etapa toma el material de clave de entrada y "extrae" de él una clave pseudoaleatoria K de longitud fija. La fase de "Extracción", implica generar un hash HMAC sobre un valor de Salt (que se trata de un valor público que se generó al azar al principio) y la clave de origen. La fase de extracción puede verse como una función, cuya entrada es un Salt y el material de clave de origen (IKM), y genera como salida otra clave secreta aleatoria (PRK). Se puede usar cualquier algoritmo de hash criptográfico estándar, pero los más comunes son la familia de hash SHA, siendo SHA256 la opción más frecuente.

```
HKDF-Extract(salt, IKM) -> PRK
PRK = HMAC-Hash(salt, IKM)
```

- **HKDF-Expand:** La segunda etapa "expande" la clave K en varias claves pseudoaleatorias adicionales (la salida del KDF), el número y la longitud de las claves de salida dependen de los algoritmos criptográficos específicos para los que se necesitan las claves. La fase de "Expansión" puede verse como una función que recibe la PRK generada en la fase de extracción; info es un campo opcional y depende del contexto y aplicación; y L que indica la longitud de salida en octetos del material de clave. La salida de la función HKDF-Expand produce todo el material de clave necesario (OKM).

```
HKDF-Expand(PRK, info, L) -> OKM
```

6.3.2.1. HKDF en TLS 1.3

En TLS 1.3 hay tres rondas de las funciones de extracción y expansión de HKDF. En cada una de estas rondas se forman claves para diferentes etapas de la conexión.

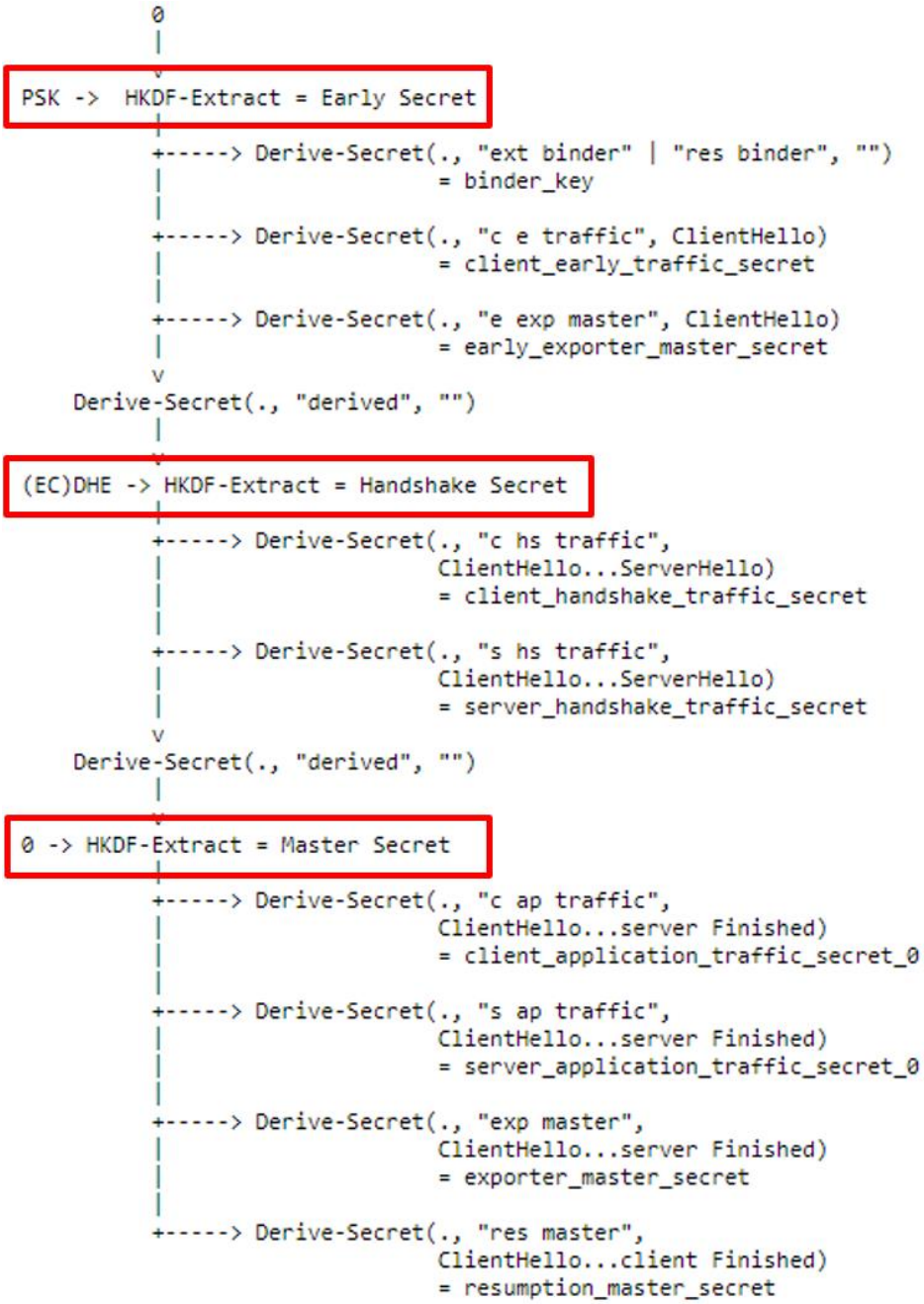


Figura 14. HKDF en TLS 1.3.

Primera ronda: Early Secret

Si el cliente y el servidor ya han establecido una clave precompartida en una conexión anterior, esta clave actúa como material de clave inicial. Cuando no

hay una clave precompartida, su entrada se reemplaza con una cadena de ceros que tiene la misma longitud que el hash. El 0 que actúa como el primer salt también es una cadena de ceros de longitud hash. (Para una primera conexión "estándar" entre ambas partes, ambas entradas HKDF-extract son cero). La salida que produce HKDF-extract se conoce como Early Secret, y es una de las entradas de la función HKDF Expand de la primera ronda, que junto con hash del mensaje de Client Hello (único mensaje conocido hasta este punto en el hadshake) producen los primeros secretos resultantes (Binder Key, Client Early Traffic Secret, Early Exporter Master Secret y Derive Secret). El único contenido que protegen estas claves sería cualquier dato temprano incluido en el saludo del cliente si el cliente estuviera iniciando un protocolo de enlace de reanudación 0-RTT.

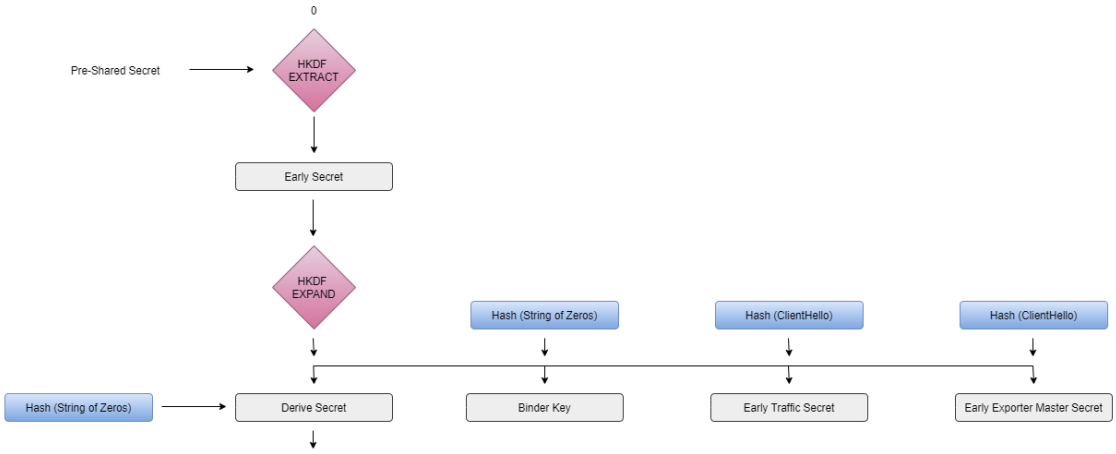


Figura 15. Primera ronda HKDF - Early Secret.

Segunda ronda: Handshake Secret

En la segunda ronda, los datos de entrada de la función HKDF-Extract es el secreto derivado de la ronda anterior (Derive Secret) como Salt, y el Secreto Compartido entre el cliente y el servidor durante el intercambio de clave ECDHE (en la extensión Key_share) que actúa como IKM. El resultado de la fase de extracción anterior da como resultado el Handshake Secret.

```
HKDF-Extract(derive_secret, shared_secret) -> handshake_secret
```

El handshake_secret y varios mensajes hash diferentes son la entrada de HKDF_Expand en esta segunda ronda, ya que se ha intercambiado más información, y por tanto, se dispone de más material para "mezclar" (hash de: Client Hello y Server Hello). Las salidas de la función hash generan los siguientes

secretos: (Client Handshake Traffic Secret, Server Handshake Traffic Secret, Early Exporter Master Secret y Derive Secret). El único contenido que protegen estas claves sería el Certificado del servidor, las Extensiones cifradas del servidor y el Finalizado del servidor. (y en los casos en que se utilice autenticación mutua, solicitud de certificado y verificación de certificado).

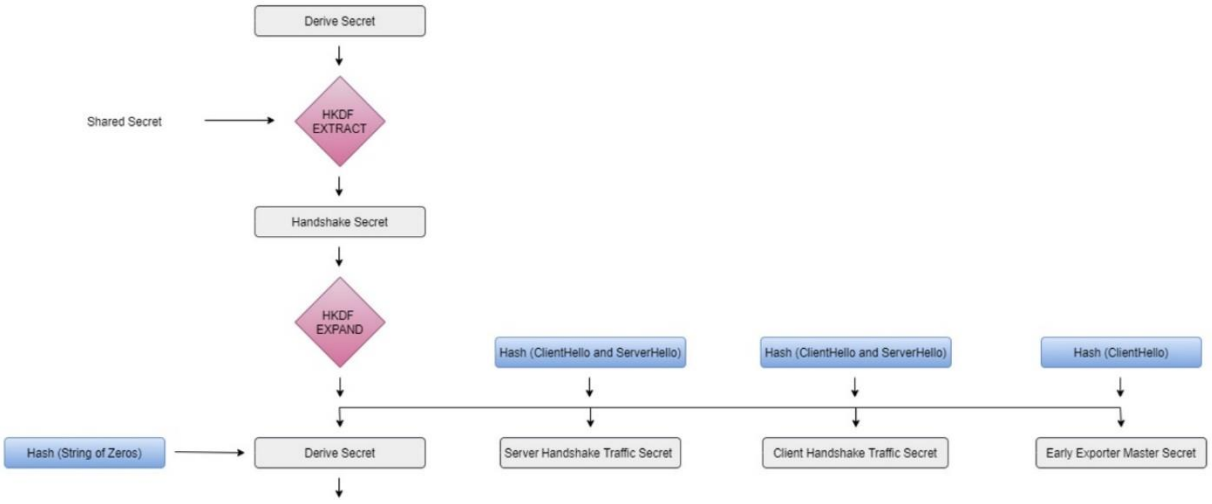


Figura 16. Segunda ronda HKDF - Handshake Secret.

Tercera ronda: Master Secret

Con la respuesta del Cliente al Server Hello, se dispone de más registros del protocolo de enlace que permiten crear nuevas claves derivadas usando HKDF. El salt correspondiente al HKDF-Extract de la tercera ronda pertenece al Derive Secret de la ronda anterior, y el material de clave es una cadena de ceros. El resultado obtenido de pasar los valores anteriores por HKDF -Extract es llamado Secreto Maestro.

```
HKDF-Extract(derive_secret2, zero-string) -> master_secret
```

Se ejecuta una vez más HKDF_Expand, cuya entrada será el Secreto Maestro y el hash de todos los mensajes recibidos (desde el Client Hello hasta Server Finished). La salida genera todas las claves necesarias para asegurar la conexión. Estas claves son las que realmente protegerán los datos de la aplicación enviados por ejemplo a través de HTTPS (Client Application Traffic Secret, Server Application Traffic Secret, Exporter Master Secret, Resumption Master Secret).

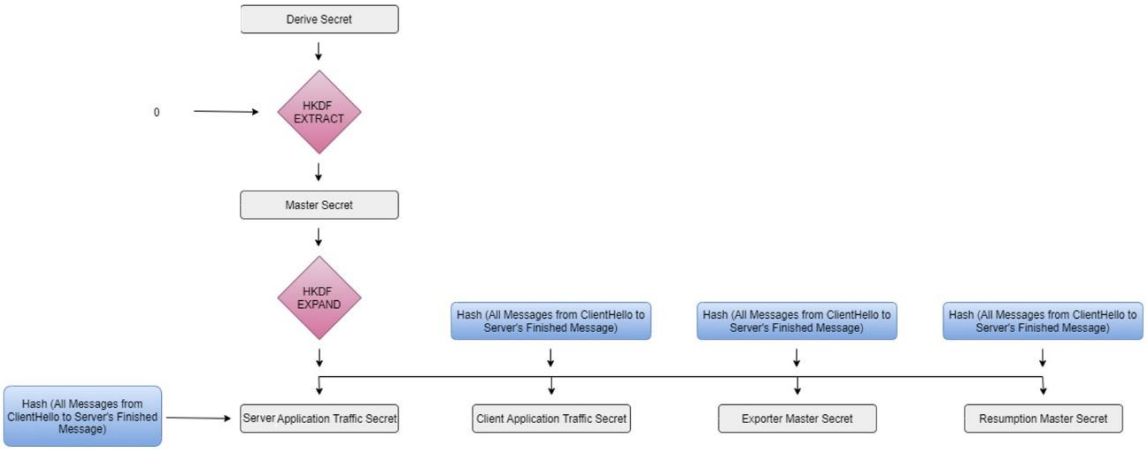


Figura 17. Tercera ronda HKDF - Master Secret.

Cada una de las claves generadas cumple su función durante el handshake. Viendo cómo se deriva el material de clave, se puede comprender la seguridad que aporta TLS a una comunicación, ya que, Debido a la operación de extracción repetida en cada una de las tres rondas anteriores, las claves de sesión derivadas de un paso no proporcionan información sobre qué claves de sesión se derivarán en otros pasos.

7. Criptografía post-cuántica

Como se ha comentado en apartados anteriores, La criptografía de clave pública se basa en el cálculo de un par de claves privada/pública que se obtienen mediante un problema matemático unidireccional en el que la solución es fácil en un sentido, pero no en el otro. El problema matemático en el que se basa el cifrado RSA es la factorización del producto de dos números primos de gran tamaño. Por otra parte, el algoritmo de Diffie-Hellman se basa en el problema del logaritmo discreto y en su versión elíptica ECDH, en el problema del logaritmo discreto sobre curva elíptica.

En el año 1996 Peter Shor propuso un algoritmo que permite a un ordenador cuántico resolver los problemas de la factorización y del logaritmo discreto en un tiempo polinomial, poniendo en riesgo toda la criptografía de clave pública actual. Con los ordenadores cuánticos la criptografía de clave secreta, así como las funciones hash, también se verán afectadas, pero se estima que en esos casos

simplemente aumentando el tamaño de las claves y de las salidas hash, el problema pueda solucionarse.

Algunas investigaciones y experimentos realizados en estos últimos años, confirman que el protocolo TLS puede convertirse en un protocolo post-cuántico, sustituyendo los algoritmos de intercambio de claves (ECDHE) y firmas digitales (ECDSA, EdDSA y RSA) actuales, por algoritmos y firmas post-cuánticas.

7.1. Proceso de estandarización de criptografía post-cuántica del NIST

El proceso de estandarización de criptografía post-cuántica del NIST, conocido como Post-Quantum Cryptography Standardization, es un proyecto abierto para intentar estandarizar los algoritmos post-cuánticos de establecimiento de claves y cifrado de clave pública, y de firma digital [13].

El proceso necesitó de la ejecución de varias rondas para decidir que algoritmos post-cuánticos seleccionar. En la primera ronda se presentaron un total de 69 algoritmos candidatos. Al finalizar la tercera ronda, que ha ocurrido durante la realización de este trabajo (julio 2022), los algoritmos finalistas que han sido considerados para estandarización son:

- **Crystals Kyber:** es el algoritmo seleccionado para cifrado de clave pública y establecimiento de clave.
- **Crystals-Dilithium, Falcon y SPHINCS+:** son los tres algoritmos seleccionados para firma digital.

7.2. TLS post-cuántico

La transición a la criptografía post-cuántica ha dado lugar a un supuesto que no había ocurrido antes en criptografía, y se trata del uso de dos o más algoritmos simultáneamente. El IETF propone una construcción donde se utilice un algoritmo tradicional (por ejemplo, ECDHE) y alguno de los algoritmos post-cuánticos propuestos por el NIST, dando lugar a un intercambio de claves híbridas en TLS 1.3. Esta propuesta surge principalmente para y mantener la seguridad actual en caso de una vulnerabilidad en el componente post-cuántico, puesto que son algoritmos menos estudiados y recientemente implementados. Además, para respetar el marco legal, es necesario mantener el algoritmo

tradicional, ya que los sistemas post-cuánticos aún no han sido incorporados en las normativas de seguridad.

Sin embargo, TLS está diseñado para elegir en la negociación un solo algoritmo que utilizará luego para cada propósito (intercambio de claves, autenticación, cifrado simétrico, función hash, etc.). Por tanto, TLS requiere de una modificación en el protocolo que indique como negociar una combinación de algoritmos y como aplicarlos criptográficamente.

El objetivo principal de trabajar con un TLS híbrido es garantizar que la seguridad se mantenga intacta, al menos mientras uno de los componentes (tradicional o post-cuántico) permanece invulnerable. En la práctica, no sólo debe tenerse en cuenta la seguridad del sistema, sino también otros aspectos que serán críticos para la implantación.

- **Compatibilidad:** Los clientes y servidores que añadan los sistemas híbridos, deberán seguir siendo compatibles con el resto de midlebox antiguos que no han incorporado el sistema híbrido. Se pueden dar tres escenarios:
    - **Cliente y servidor híbridos:** Deben negociar modos híbridos.
    - **Cliente híbrido y servidor no-híbrido:** En este caso se negocia un sistema tradicional, siempre que el cliente esté dispuesto a utilizar un sistema no post-cuántico (tradicional).
    - **Cliente no-híbrido y servidor híbrido:** En este caso se negocia un sistema tradicional, siempre que el servidor esté dispuesto a utilizar un sistema no post-cuántico (tradicional).
- **Rendimiento:** El costo computacional depende directamente de los algoritmos criptográficos utilizados. La incorporación de las claves híbridas no debería suponer un aumento excesivo en el rendimiento computacional. El uso de los modos híbridos no debería aumentar la latencia en el establecimiento de una conexión. Los factores que pueden afectar al incremento de la latencia son:
    - Características de rendimiento de los algoritmos utilizados.
    - El tamaño de los mensajes a transmitir, ya que los algoritmos de criptografía post-cuántica utilizan un tamaño de clave pública y texto cifrado mayor.

- La incorporación de los sistemas híbridos no debería dar lugar a un aumento en las interacciones para negociar el intercambio, es decir, debería mantenerse la cantidad de iteraciones de la máquina de estados del handshake en los 3 escenarios nombrados anteriormente.

7.2.1. Negociación híbrida en TLS 1.3

En TLS 1.3 la negociación del cipher suite se produce cuando el cliente propone una lista ordenada por orden de preferencia de algoritmos soportados, y el servidor responde con la elección de uno de los algoritmos de la lista. Si no se encuentra un algoritmo mutuo compatible, se comunica un error.

En versiones anteriores del protocolo TLS, todas las opciones criptográficas se negocian a la vez. Sin embargo, en la versión de TLS 1.3 cada componente se elige por separado (cifrado simétrico, esquema de firma digital y métodos de intercambio de claves), y es precisamente el diseño modular de la última versión de TLS, lo que ha facilitado la negociación híbrida.

En la modalidad híbrida, el objetivo es negociar dos o más algoritmos y utilizar ambos, y es aquí donde surge la primera decisión de cómo debe negociarse, es decir, ambos algoritmos deben negociarse por separado, o como un único algoritmo combinado. Otra segunda elección es si los parámetros deben negociarse conjuntamente o independiente al algoritmo criptográfico (es decir, ECDH o ECDH+P256). El diseño elegido en la negociación va a ser importante porque:

- Si cada algoritmo se negocia por separado, debe modificarse la lógica y formato de los mensajes del protocolo TLS, por ejemplo, añadiendo otra extensión similar a supported_group que incluya los algoritmos post-cuánticos.
- Si la negociación se realiza combinando ambos tipos de algoritmos (tradicionales y post-cuánticos), no será necesario modificar el funcionamiento de TLS, sino que simplemente deberán añadirse a la extensión supported_group nuevos identificadores que representen a un par de algoritmos.

7.2.2. Mecanismo de encapsulamiento de clave

El borrador del IETF también incluye una explicación para definir un mecanismo de encapsulamiento de claves (KEM, Key Encapsulation Mechanism), según el contexto del proyecto de estandarización de criptografía post-cuántica del NIST. Un KEM es un protocolo para transmitir de forma segura una clave de sesión a través de un canal inseguro, de la misma forma que ocurre con el protocolo de intercambio de claves (Diffie Hellman), pero con una diferencia: cuando se emplea la encapsulación (KEM) la clave es totalmente generada por una de las partes, lo que permite, entre otras cosas, comenzar a transmitir los datos cifrados directamente con la clave encapsulada.

Los KEM constan de tres algoritmos:

- "**KeyGen() -> (pk, sk)":** Un algoritmo de generación de clave probabilística, que genera una clave pública "pk" y una clave secreta "sk“. KeyGen corresponde a seleccionar un exponente "x" como clave secreta y calcular la clave pública g^x
- **"Encaps(pk) -> (ct, ss)":** un algoritmo de encapsulación probabilística, que toma como entrada una clave pública "pk“; y genera un texto cifrado "ct" y un secreto compartido "ss".

  Encaps es la encapsulación que corresponde a seleccionar un exponente "y", calcular el texto cifrado g^y y el secreto compartido g^{xy} .
- "**Decaps(sk, ct) -> ss**": un algoritmo de desencapsulación, que toma como entrada una clave secreta "sk" y un texto cifrado "ct“; y genera un secreto compartido "ss”.

  Decaps es la desencapsulación para obtener el secreto compartido g^{xy} por la otra parte.

La explicación sobre los KEM que ofrece el IETF en el borrador de intercambio de claves híbridas en TLS 1.3, aunque es bastante sencilla, permite entender la Figura 18 que encontramos en varios artículos de la literatura publicada, donde el intercambio de claves DiffieHellman se puede modelar como un KEM, y convierte al protocolo TLS resistente a computación cuántica.

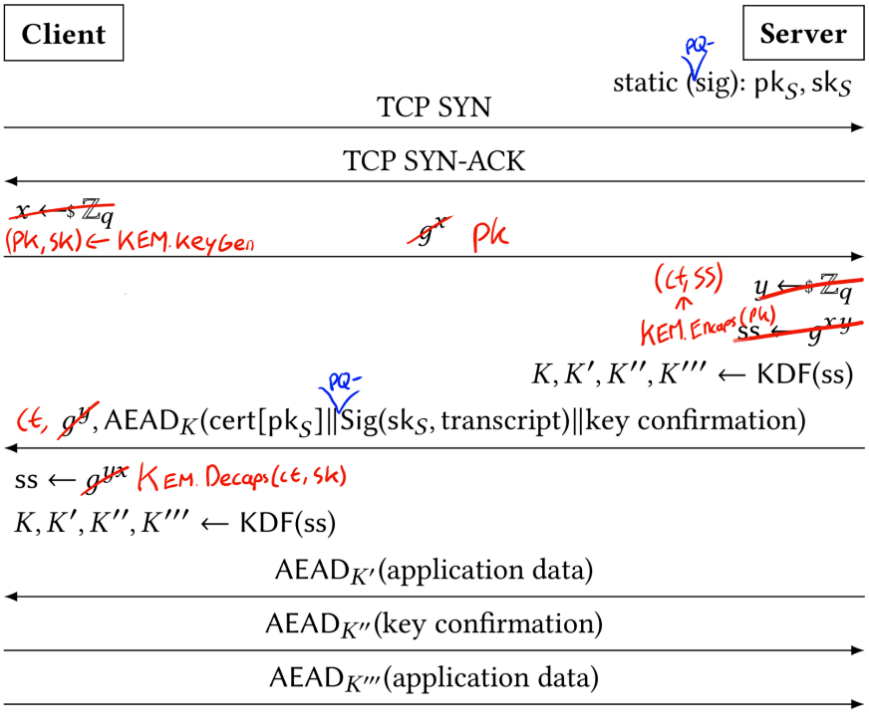


Figura 18. Intercambio de claves Diffie Hellman modelado como KEM.

7.3. Herramientas e investigaciones

7.3.1. Estudios guía para el desarrollo de este trabajo

Debido a las características propias de cada algoritmo post-cuántico, ha sido necesario realizar diversos estudios para conocer cual se ajusta mejor para cada propósito en TLS. Algunas de las investigaciones, que han sido llevadas a cabo por diversas compañías como Cloudflare, Google, Microsoft, IBM, etc. han intentado definir prototipos para el intercambio de claves, la autenticación y el cifrado post-cuánticos en TLS. Algunos ejemplos son [17] [18] [19] [20]:

- **CECPQ1:** se trata de un experimento en Google Canary, llevado a cabo por Matt Braithwaite en el año 2016, en el que se combinó al algoritmo post-cuántico NewHope para establecimiento de clave junto con ECDH (con curva x25519).
- **CECPQ2 y CECPQ2b:** estos dos protocolos son la continuación de CECPQ1 y fueron desarrollados en colaboración entre el navegador de Google y los servidores de Cloudflare. CECPQ2 se compone de NTRU-HRSS + ECDH(x25519) y CECPQ2b está formado por SIKE/p434 +

X25519. El objetivo de este estudio fue medir el rendimiento y eficiencia de ambos en clientes y servidores. Aunque las claves en CECPQ2 eran de mayor tamaño, se obtuvieron mejores resultados que con CECPQ2b debido a la diferencia de los algoritmos post-cuánticos empleados en cada uno.

7.3.2. Open Quantum Safe

Uno de los proyectos de código abierto más potentes que existen actualmente para apoyar el desarrollo y la creación de prototipos de criptografía resistente a la computación cuántica es OQS (Open Quantum Safe). Este proyecto se divide en dos líneas de trabajo [15]:

- **Liboqs**: es una librería en C para algoritmos criptográficos post-cuánticos.
- **Aplicaciones y protocolos**: permite trabajar con algoritmos modernos e integrar protocolos y aplicaciones, incluyendo la conocida librería OpenSSL.

Otro proyecto importante externo a OQS es PQClean, que recopila las implementaciones de los esquemas post-cuánticos candidatos en el proceso de estandarización de criptografía post-cuántica del NIST. Las implementaciones desarrolladas en PQClean pueden integrarse fácilmente en librerías como liboqs [16].

8. Ejecución de un entorno de pruebas

Para realizar el análisis teórico/práctico que nos facilite entender en profundidad el funcionamiento de TLS 1.3, así como un primer acercamiento a un TLS post-cuántico, se ha creado un entorno de pruebas a modo de laboratorio. En este entorno se analiza el protocolo TLS que se encuentra actualmente en funcionamiento y las implementaciones creadas dentro del proyecto Open Quantum Safe, que incorporan algoritmos post-cuánticos en las comunicaciones TLS.

8.1. Herramientas

Se elige como sistema operativo Ubuntu, en la versión de 20.04. La mayoría de herramientas utilizadas son multiplataforma, por ejemplo, liboqs puede instalarse en MAC, Windows y Unix, pero Linux, y en concreto el sistema operativo Ubuntu, ha sido seleccionado porque el propio proyecto de OQS utiliza imágenes basadas en este sistema, y algunas de las implementaciones están probadas únicamente sobre Ubuntu.

Figura 19. Sistema operativo elegido.

Además, puesto que se trata de un entorno de pruebas, para evitar modificar ficheros del sistema del propio equipo, se decide y aconseja virtualizar todo el entorno, creando una máquina virtual con el software de VirtualBox.

Figura 20. Software de virtualización.

Otro software que ha sido fundamental para el desarrollo del proyecto ha sido Wireshark. Esta herramienta es utilizada para realizar el análisis de protocolos y solucionar problemas en las redes de comunicaciones, siendo fundamental para ejecutar el análisis de datos en la comunicación TLS.

Figura 21. Analizador de protocolos – Wireshark.

Por último, para facilitar la instalación de las implementaciones del proyecto Open Quantum Safe, se ha instalado Docker. Esta herramienta empaqueta software en unidades estandarizadas que se denominan contenedores, que incluyen todo lo necesario para que el software que deseamos sea ejecutado, incluidas bibliotecas, herramientas de sistema, código, etc. Docker permite trabajar fácilmente con todas las “demos” propuestas en OQS, salvo la de Chromiun, ya que no dispone de una imagen preconstruida de Docker.

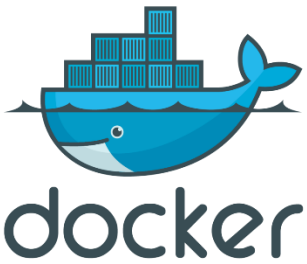


Figura 22. Software Docker.

8.2. Demos de Open Quantum Safe

Dentro del proyecto de OQS, en la línea de trabajo correspondiente a la integración de aplicaciones y protocolos, existen diferentes demos para trabajar con TLS. La marca verde en la Figura 23 representa aquellas integraciones que han sido utilizadas en el desarrollo de este trabajo. El icono de Docker indica que OQS nos ofrece una imagen preconstruida que nos facilita su instalación.

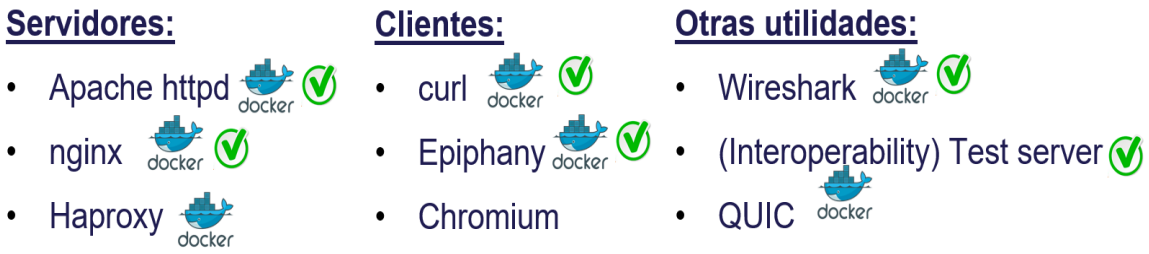


Figura 23. Demos de OQS para trabajar con TLS.

- **Servidores**: la imagen de Docker de Apache y Nginx se basa en la bifurcación OQS y OpenSSL 1.1.1, esto permite que el servidor negocie claves de seguridad y autenticación con algoritmos cuánticos mediante TLS 1.3, ya que los tiene incorporados. Estas imágenes inician el servidor

Apache o Nginx escuchando conexiones TLS 1.3 protegidas con criptografía cuántica.

- **Clientes**: OQS pone a disposición de los usuarios las imágenes preconstruidas del navegador Epiphany y el navegador bajo línea de comandos curl. También puede trabajarse con Chromium, pero se trata de una instalación compleja, ya que requiere de muchas dependencias y es necesario descargar y compilar el binario que encontramos en el enlace de OQS.
- **Wireshark (OQS)**: Para que Wireshark reconozca los algoritmos cuánticos, debe hacerse uso de la imagen de Docker del proyecto de OQS, ya que, en la versión normal del software estos no vienen incluidos.
- **Test Server (Interoperability)**: dentro del proyecto de OQS puede encontrarse una lista que proporciona enlaces a los diferentes puertos de un servidor que se encuentra a disposición de los usuarios. Cada uno de esos puertos representa una combinación concreta de algoritmos de intercambio de claves y firmas digitales. La Tabla 8 representa algunos ejemplos escogidos al alzar, dentro de la gran variedad que ofrece el Servidor de prueba. Por ejemplo, para el puerto 6086 se escoge rsa3072 como algoritmo de firma, y frodo640aes como algoritmo de intercambio de claves.

Algoritmo de firma	Algoritmo de intercambio de claves	Puerto	Enlace
ecdsap256	**p521_kyber1024**	**6051**	ecdsap256/p521_kyber1024
ecdsap256	**p384_sikep610**	**6067**	ecdsap256/p384_sikep610
rsa3072	**frodo640aes**	**6086**	rsa3072/frodo640aes

Tabla 8. Ejemplos de conexiones del Test Server del proyecto OQS

8.3. Pruebas realizadas

Durante el desarrollo de este trabajo fue necesario realizar un estudio exhaustivo para comprender el funcionamiento del protocolo TLS 1.3. Para ello se ha consultado la literatura y los RFC implicados en el funcionamiento de TLS.

Además, para asimilar los conceptos, se ha hecho uso de Wireshark, capturando el tráfico se ha podido establecer una relación entre la teoría y el funcionamiento práctico de TLS, paso fundamental para explicar los apartados anteriores.

8.3.1. TLS 1.3 con primitivas criptográficas actuales

La Figura 24 es un ejemplo del tráfico capturado sobre una comunicación establecida entre nuestro equipo y un servidor de correos considerado seguro (Proton Mail). El paquete representado en la imagen corresponde al mensaje de Server Hello, y en ella se pueden apreciar múltiples campos: versión de TLS, cipher suite elegido entre el cliente y servidor, grupos soportados, etc.

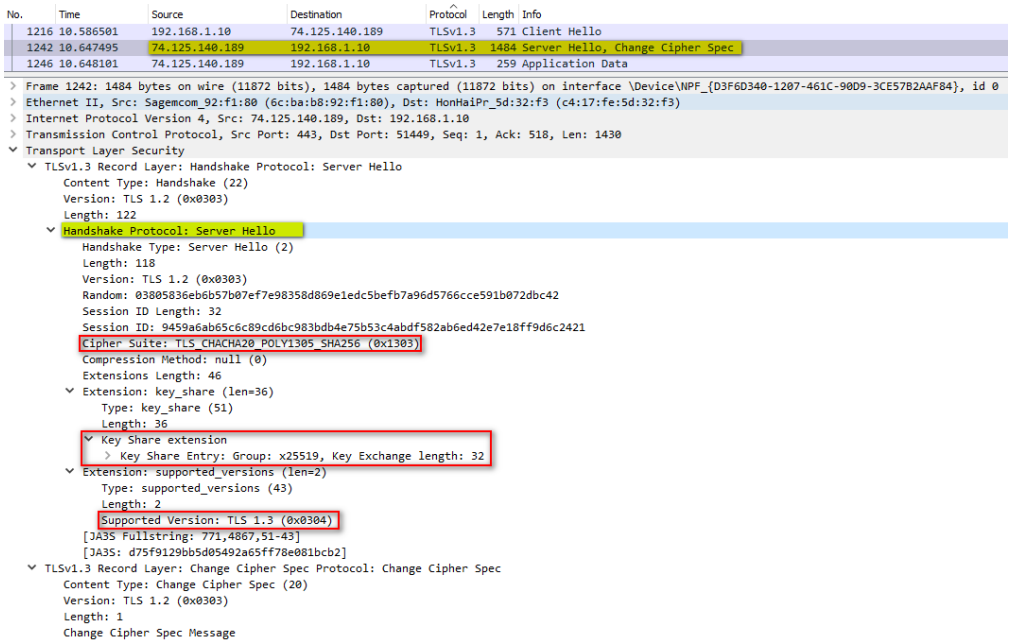


Figura 24. Captura de una comunicación con primitivas criptográficas actuales.

8.3.2. TLS 1.3 con primitivas criptográficas post-cuánticas

La Figura 25 es un ejemplo del tráfico capturado sobre una comunicación establecida en el entorno desarrollado, donde la imagen del navegador curl del proyecto OQS actúa como cliente y solicita una comunicación a la imagen del servidor Apache del proyecto de OQS. Las dos partes, cliente y servidor, disponen de las primitivas criptográficas cuánticas. El paquete representado en la imagen también corresponde al mensaje de Server Hello, y en ella se pueden apreciar múltiples campos: versión de TLS, cipher suite elegido entre el cliente y servidor, grupos soportados, etc. En este caso, se ha forzado para que la negociación entre el cliente y el servidor se produzca mediante una conexión

TLS híbrida, y se utilice como combinación de algoritmos, ECDHE con curva P256 como algoritmo tradicional y Kyber512 como algoritmo post-cuántico.

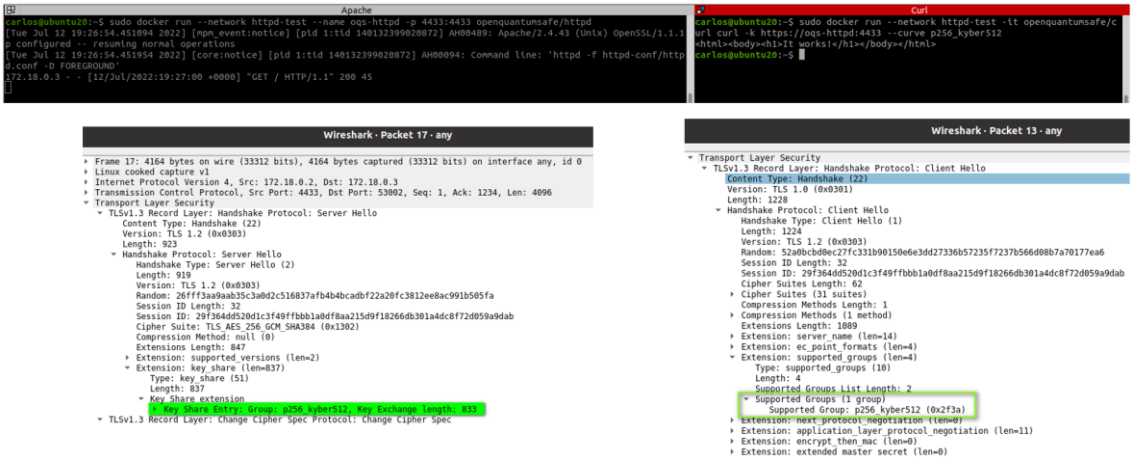


Figura 25. Captura de una comunicación con primitivas criptográficas cuánticas. Curl y Apache.

La Figura 26 muestra una prueba realizada utilizando la imagen del navegador Epiphany del proyecto de OQS y una comunicación sobre el servidor de pruebas de interoperabilidad. En la captura de Wireshark (Client Hello) podemos ver que el navegador tiene habilitados un grupo de protocolos cuánticos, concretamente una la lista: p256_ntru_hps2048509, frodo640aes, p384_sike610, firesaber y p521_kyber1024. El grupo de protocolos que admite Epiphany es modificable en los ficheros de configuración del navegador, o en consola cuando lanzamos el comando para ejecutarlo. Al acceder, desde Epiphany al servidor de pruebas de OQS, e indicarle el puerto correspondiente a uno de los algoritmos negociados, la conexión es satisfactoria. Si se accede un navegador normal, que no reconoce los protocolos cuánticos, o desde Epiphany, pero a otro puerto que no corresponda a un algoritmo soportado (indicados en el Client Hello en la extensión supported_groups) la conexión sería rechazada.

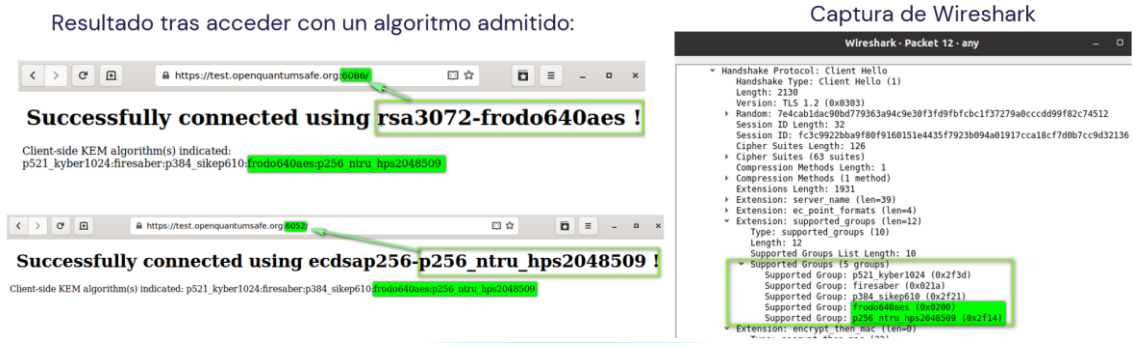


Figura 26. Captura de una comunicación con primitivas criptográficas cuánticas. Epiphany y Test Server OQS.

9. Conclusiones y líneas futuras

A lo largo de la historia el protocolo TLS, se ha visto comprometido numerosas veces por diferentes tipos de ataque, como errores cometidos en el software implicado en las comunicaciones, ataques de canal lateral, errores descubiertos en sistemas criptográficos considerados seguros, etc. La versión actual, TLS 1.3, ha supuesto un importante avance ya que cifra los mensajes desde el handshake, proporciona forward secrecy, y ha eliminado todos los algoritmos dudosos. Sin embargo, esto no será suficiente para la era post-cuántica. El estudio preliminar presentado en este trabajo tiene como objeto conocer el alcance e impacto sobre el protocolo TLS de la llegada de los ordenadores cuánticos, y fundamentar las bases para su necesaria actualización. Para ello ha sido necesario conocer las investigaciones que se están realizando al respecto, así como las diferentes primitivas criptográficas actualmente propuestas para la realidad post-cuántica, con objeto de disponer de un punto de partida para desarrollar una investigación teórica/práctica sobre el futuro post-cuántico del protocolo TLS.

En este sentido, como línea futura se plantea el análisis de las propuestas de esquemas de firma digital post-cuántica, recientemente remitidas a la convocatoria abierta por el NIST en agosto de 2022 [21].

10. Conclusions and future work

Throughout its history, the TLS protocol has been compromised on numerous occasions by different types of attacks, such as errors committed in the software involved in communications, side-channel attacks, errors discovered in cryptographic systems considered secure, etc. The current version of TLS, 1.3, is a major breakthrough as it encrypts messages from the handshake, provides forward secrecy, and has removed all dubious algorithms. However, this will not be enough for the post-quantum era. The preliminary study presented in this paper aims to understand the scope and impact on the TLS protocol of the advent of quantum computers, and lay the foundations for its necessary update. For this, it has been necessary to know the research that is being carried out in this regard, as well as the different cryptographic primitives currently proposed for the post-quantum reality, thus establishing a starting point for the development of a theoretical/practical research on the post-quantum future of the TLS protocol.

In this sense, as a future line, the analysis of the proposals for post-quantum digital signature schemes submitted to the Call recently opened in August 2022 by NIST is proposed [21].

11. ANEXO I: Manual de instalación y ejecución de herramientas

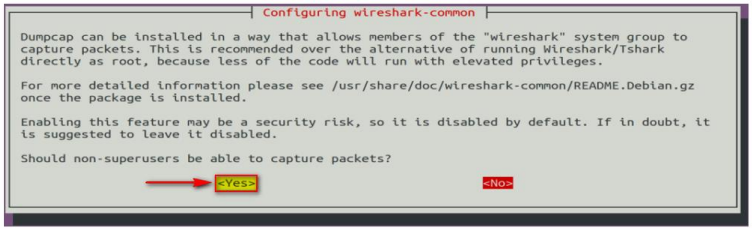
Sistema Operativo:

Es importante trabajar con el sistema actualizado y muy recomendable refrescar la cache del gestor de paquetes en Ubuntu siempre que se proceda a instalar alguna herramienta en Linux. De esta manera nos aseguramos que trabajaremos con paquetes actualizados y con las últimas versiones. Se recomiendo trabajar con la versión de Ubuntu 20.04 o superior.

Wireshark:

Para poder realizar una captura de la comunicación y reconocer los algoritmos post-cuánticos, usaremos el Docker de Wireshark propuesto en el proyecto OQS, pero es necesario tener el software universal instalado.

```
$ sudo apt install wireshark
```



Debe añadirse un nombre de usuario al grupo Wireshark.

```
$ sudo usermod -aG wireshark <<username>>
```

Docker:

Paquetes previos que deben estar instalados:

```
$ sudo apt-get install ca-certificates
$ sudo apt-get install curl
$ sudo apt-get install gnupg
$ sudo apt-get install lsb-release
```

Debe agregarse la clave GPG oficial de Docker. La clave GPG no es una cuestión específica de Docker, sino que se trata de un mecanismo utilizado por

la mayoría de los administradores de paquetes de Linux para validar la integridad de un paquete software antes de ser instalado, verificando su clave GPG.

```
$ sudo mkdir -p /etc/apt/keyrings

$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg |
sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg
```

Configuración del repositorio e instalación de Docker:

```
$ echo "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg]
https://download.docker.com/linux/ubuntu \

$(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null


$ sudo apt-get install docker-ce docker-ce-cli containerd.io docker-compose-plugin
```

Comprobación de instalación de Docker:

```
$ sudo docker run hello-world
```

```
carlos@ubuntu20:~$ sudo docker run hello-world
[sudo] contraseña para carlos:
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
2db29710123e: Pull complete
Digest: sha256:13e367d31ae85359f42d637adf6da428f76d75dc9afeb3c21faea0d976f5c651
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```

Versión de Docker instalada:

```
carlos@ubuntu20:~$ sudo docker version
Client: Docker Engine - Community
 Version:           20.10.17
 API version:       1.41
 Go version:        go1.17.11
 Git commit:        100c701
 Built:             Mon Jun  6 23:02:57 2022
 OS/Arch:           linux/amd64
 Context:           default
 Experimental:      true

Server: Docker Engine - Community
 Engine:
  Version:          20.10.17
  API version:      1.41 (minimum version 1.12)
  Go version:       go1.17.11
  Git commit:       a89b842
  Built:            Mon Jun  6 23:01:03 2022
  OS/Arch:          linux/amd64
  Experimental:     false
 containerd:
  Version:          1.6.6
  GitCommit:        10c12954828e7c7c9b6e0ea9b0c02b01407d3ae1
 runc:
  Version:          1.1.2
  GitCommit:        v1.1.2-0-ga916309
 docker-init:
  Version:          0.19.0
  GitCommit:        de40ad0
```

Liboqs:

Antes de instalar Liboqs, debe comprobarse que las siguientes dependencias están instaladas:

```
$ sudo apt install astyle cmake gcc ninja-build libssl-dev python3-pytest python3-pytest-xdist
unzip xsltproc doxygen graphviz python3-yaml
```

Se descarga Liboqs, desde el proyecto de github

```
$ git clone --branch main https://github.com/open-quantum-safe/liboqs.git
```

Accedemos a la carpeta que se ha creado al descargar el proyecto. Se crea la carpeta (build) donde se compila Liboqs.

```
$ cd liboqs
$ mkdir build && cd build
$ cmake -Gninja ..
$ sudo ninja
$ sudo ninja install
```

Ejecución de test de pruebas

```
$ ninja run_tests
```

```
carlos@ubuntu20:/usr/lib/ssl/liboqs/build$ sudo  cmake -GNinja ..
-- Configuring done
-- Generating done
-- Build files have been written to: /usr/lib/ssl/liboqs/build
carlos@ubuntu20:/usr/lib/ssl/liboqs/build$ sudo ninja
[522/522] Linking C executable tests/test_sig_mem
```

Wireshark (OQS)

La imagen de Docker de Wireshark permite mostrar correctamente las operaciones TLS de criptografía cuántica segura.

```
$ sudo docker run --net=host --privileged --env="DISPLAY" --volume="$HOME/.Xauthority:/root/.Xauthority:rw"
openquantumsafe/wireshark
```

La imagen de Docker de Wireshark de OQS está basada en Ubuntu. Además, al tratarse de una aplicación con interfaz gráfica, requiere que el host ejecute el sistema Unix X-Window.

Apache

Se debe tener Docker instalado. Para descargar y ejecutar la imagen del servidor Apache se ejecuta el siguiente comando, que inicia el servidor en un puerto de escucha indicado (4433):

```
$sudo docker run -p 4433:4433 openquantumsafe/httpd
```

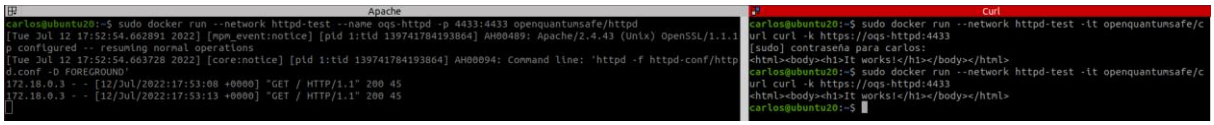
Como se ejecuta todo, Servidor (httpd) y cliente (curl), en un mismo equipo (máquina virtual), se aconseja ejecutar ambas imágenes dentro de una red de Docker. Evita estar trabajando con IPs y reconoce a las máquinas por el nombre asignado, en nuestro caso:

- *Nombre de la red:* ***httpd-test***
- *Nombre del contenedor (coincide con el nombre puesto al servidor Apache):* ***oqs-httpd***

```
$ sudo docker network create httpd-test
$ sudo docker run --network httpd-test --name oqs-httpd -p 4433:4433 openquantumsafe/httpd
```

Para acceder al servidor utilizamos la imagen Docker de curl.

```
$ sudo docker run --network httpd-test -it openquantumsafe/curl curl -k https://oqs-httpd:4433
```



Epiphany

Se descarga la imagen Docker del navegador web GNOME Web/epiphany que permite ejecutar correctamente operaciones TLS de criptografía cuántica segura. La imagen de Epiphany se basa en Ubuntu y requiere que el host ejecute el sistema Unix X-Window.

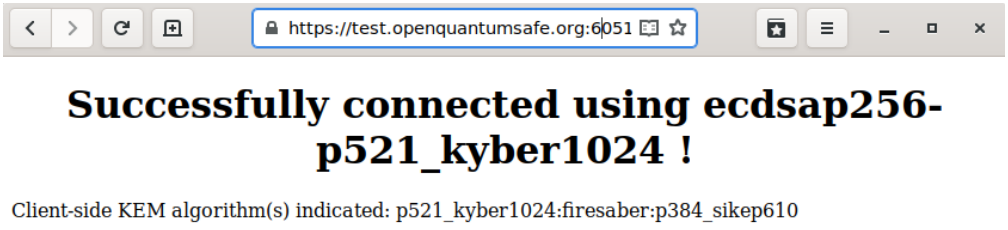
```
$ sudo docker run --net=host --privileged --env="DISPLAY" openquantumsafe/epiphany
```

La imagen de Docker del navegador Epiphany viene configurada de forma predeterminada para aceptar los algoritmos: p521_kyber1024, firesaber y p384_sikep610.

Ejecutamos la imagen de Epiphany por defecto y probamos a realizar la conexión a Test Server en el puerto indicado para el algoritmo: p521_kyber1024, que corresponde a:

Algoritmo de firma	Algoritmo de intercambio de claves	Puerto	Enlace
ecdsap256	p521_kyber1024	6051	ecdsap256/p521_kyber1024

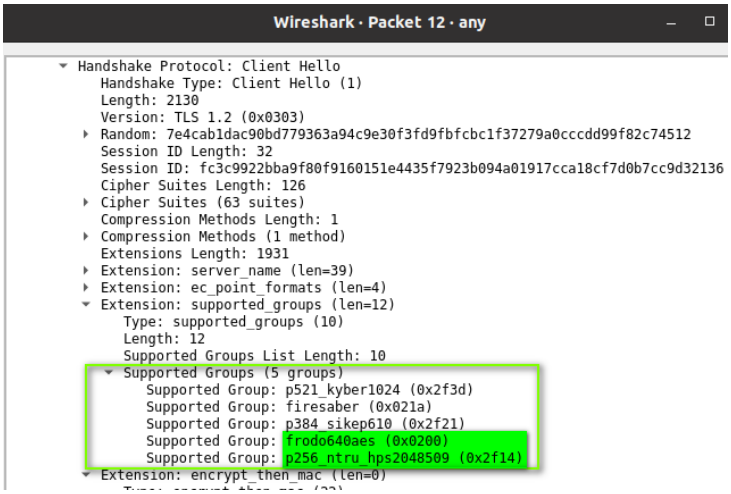
Resultado tras acceder al enlace y puerto correspondiente es:



Durante la ejecución del commando se pude aumentar la lista de algoritmos compatibles que admite el navegador. También puede realizarse en el fichero de configuración.

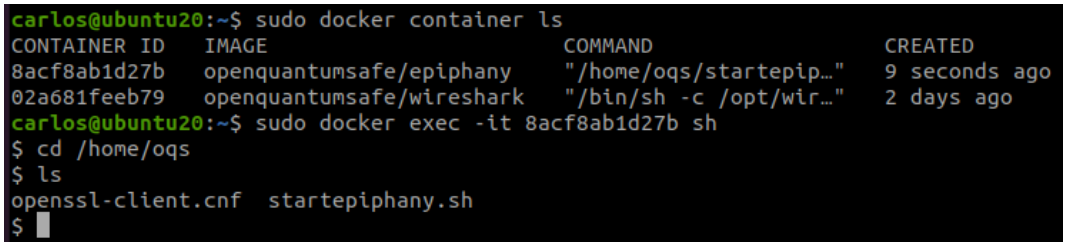
Ejecución de Epiphany aumentando la lista de algoritmos admitidos.

```
$ sudo docker run --net=host --privileged --env="DISPLAY" openquantumsafe/epiphany
frodo640aes:p256_ntru_hps2048509
```



Acceso al fichero de configuración de Epiphany

```
$ sudo docker exec -it <Container ID> sh
$ cd /home/oqs
```



12. ANEXO II: Artículo enviado a congreso

Carlos Barreda Falciano, Pino Caballero Gil and Jezabel Molina Gil

Preliminary study on Post-Quantum TLS

Workshop on Cybersecurity Applications and Intelligent Transportation Systems CAITS

Proceedings of the International Conference on Security and Management SAM

World Congress in Computer Science, Computer Engineering, and Applied Computing CSCE

Las Vegas, USA. July 25-28, 2022

Springer Nature

Indexada en Computing Research and Education (CORE), con ranking C

Indexada en CS Conference Rankings (0.83)

Indexada en GII-GRIN en Class WiP

BIBLIOGRAFÍA

[1] E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3. https://datatracker.ietf.org/doc/html/rfc8446. 2018

[2] S. Gueron and A. Langley and Y. Lindell. AES-GCM-SIV: Nonce Misuse-Resistant Authenticated Encryption. 2019. https://datatracker.ietf.org/doc/html/rfc8452

[3] Y. Nir and A. Langley. ChaCha20 and Poly1305 for IETF Protocols. 2018 https://datatracker.ietf.org/doc/html/rfc8439

[4] National Institute of Standards and Technology (NIST). Block Cipher Techniques. 2020. https://csrc.nist.gov/Projects/block-cipher-techniques/BCM

[5] S. Josefsson, I. Liusvaara, Edwards-curve digital signature algorithm (EdDSA) (No. RFC8032). 2017.

[6] A.E. Eldewahi, T.M. Sharfi, A.A. Mansor, N.A. Mohamed, S.M. Al-wahbani. SSL/TLS attacks: Analysis and evaluation. IEEE International Conference on Computing, Control, Networking, Electronics and Embedded Systems Engineering, pp. 203-208. 2015.

[7] Internet Assigned Numbers Authority. Transport Layer Security (TLS) Extensions. 2022 https://www.iana.org/assignments/tls-extensiontype-values/tls-extensiontype-values.xhtml

[8] D. Gillmor. Negotiated Finite Field Diffie-Hellman Ephemeral Parameters for Transport Layer Security (TLS). 2016. https://datatracker.ietf.org/doc/html/rfc7919

[9] S. Blake-Wilson and N. Bolyard and V. Gupta and C. Hawk and B. Moeller. Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS). 2006. https://datatracker.ietf.org/doc/html/rfc4492

[10] A. Langley and M. Hamburg and S. Turner. Elliptic Curves for Security. 2016.https://datatracker.ietf.org/doc/html/rfc7748

[11] Standards for Efficient Cryptography Group. SEC 2: Recommended Elliptic Curve Domain Parameters. 2014. https://www.secg.org/

[12] T. Pornin. Deterministic Usage of the Digital Signature Algorithm (DSA) and Elliptic Curve Digital Signature Algorithm (ECDSA). 2013. https://datatracker.ietf.org/doc/html/rfc6979

[13] National Institute of Standards and Technology (NIST). Post-Quantum Cryptography.2022. https://csrc.nist.gov/projects/post-quantum-cryptography

[14] National Institute of Standards and Technology (NIST). National Vulnerability Database. 2022. https://nvd.nist.gov/

[15] D. Stebila and M. Mosca. Open Quantum Safe. 2022. https://openquantumsafe.org/

[16] T. Wiggers. PQClean. 2022. https://github.com/PQClean/PQClean.

[17] M. Braithwaite. Experimenting with Post-Quantum Cryptography. 2016. https://security.googleblog.com/2016/07/experimenting-with-post-quantum.html

[18] A. Langley. Post-quantum confidentiality for TLS. 2018 https://www.imperialviolet.org/2018/04/11/pqconftls.html

[19] K. Kwiatkowski and L. Valenta. The TLS Post-Quantum Experiment. 2019. https://blog.cloudflare.com/the-tls-post-quantum-experiment/

[20] Real-world measurements of structured-lattices and supersingular isogenies in TLS. 2019. https://www.imperialviolet.org/2019/10/30/pqsivssl.html

[21] National Institute of Standards and Technology (NIST). Post-Quantum Cryptography: Digital Signature Schemes. 2022. https://csrc.nist.gov/projects/pqc-dig-sig