Implementación de criptografía post-cuántica NTRU en servicios HTTPS

Diego Cordoba [1], Miguel Méndez-Garabetti[2,3,], Jorge García Guibout[2]
[1]Universidad de Mendoza, Facultad de Ingeniería, Subsede San Rafael
[2] Univerisdad de Mendoza, Dirección de Posgrado, Facultad de Ingeniería
[3] Universidad Atlántida Argentina, Facultad de Ingeniería
diego.cordoba@um.edu.ar, miguel.mendez@um.edu.ar, jgarcia@itu.uncu.edu.ar

Resumen

Se estima que en pocas décadas podrían desarrollarse computadoras cuánticas con la cantidad suficiente de qubits necesaria para vulnerar los principales algoritmos criptográficos asimétricos actuales, y los mecanismos de intercambio de claves basados en Diffie-Hellman. Esto quiere decir que tecnologías como RSA, DSA o ECDH quedarían obsoletas.
En este escenario, casi la totalidad de las comunicaciones basadas en SSL/TLS serían vulnerables, y esto incluye a protocolos tan comunes como HTTPS.
Afortunadamente se están desarrollando alternativas resistentes al criptoanálisis cuántico, nuevos algoritmos que dan nombre a la criptografía post-cuántica.
Si bien estos algoritmos se encuentran en etapas experimentales de desarrollo, ya existen bibliotecas de código que pueden ser integradas junto a implementaciones de protocolos que hacen uso de SSL/TLS.
El presente trabajo de investigación analiza la integración de Apache2 con WolfSSL como capa de seguridad TLS con la intención de brindar un servicio HTTPS que utilice algoritmos post-cuánticos tanto en el intercambio de claves como en la ciphersuite de cifrado de tráfico.

Introducción

La seguridad y la privacidad en Internet son motivo de gran preocupación por muchos usuarios de la red. Actualmente casi la totalidad del tráfico en Internet es intercambiado por aplicaciones que hacen uso de mecanismos criptográficos para garantizar la seguridad, entendida ésta como la verificación de la privacidad, de la autenticidad, y de la integridad de los datos.

La privacidad se entiende como la manera de permitir al receptor de un mensaje su lectura, y evitar que un tercero que reciba o capture dicho mensaje pueda visualizar su contenido. En el concepto de autenticación se engloban aquellos mecanismos que permiten al receptor de un mensaje verificar que el emisor sea quien dice ser. Finalmente, la integridad incluye las técnicas que permiten al receptor verificar que el mensaje no se haya modificado desde que fue enviado por el emisor. Si se cumplen estos tres requisitos se dice que la comunicación es segura.

En Internet estos requisitos se logran generalmente haciendo uso de la suite de protocolos SSL/TLS (Secure Socket Layer / Transport Layer Security) [1], [2].

Las versiones más actualizadas de esta suite de protocolos, y por consiguiente, las mayormente utilizadas, son TLS v1.2 y v1.3, ambas estandarizadas. Internamente TLS hace uso de algoritmos simétricos y asimétricos para lograr su propósito de brindar una capa de seguridad para protocolos de aplicación, y utiliza algoritmos basados en Diffie-Hellman (DH) [3] para lograr el intercambio de claves seguros entre dos nodos de la red sin ningún tipo de intercambio previo. Particularmente en la mayoría de las implementaciones se hace uso de la variante de curva elíptica de este algoritmo, denominada ECDH (Elliptic-curve Diffie-Hellman) [4]

Los principales algoritmos asimétricos utilizados por TLS son RSA (Rivest Shamir Adleman)[5], DSA (Digital Signature Algorithm)[6] y algunas variantes de este último, como por ejemplo ECDSA (Elliptic-curve DSA) [7]. En cuanto al cifrado simétrico del tráfico en general se utiliza AES (Advanced Encryption Standard) [8].

La amenaza cuántica

Si bien el desarrollo de la computación cuántica es todavía experimental, y se han logrado pruebas de concepto mínimamente funcionales, tales como la computadora cuántica de IBM[9], que por el momento dispone de 53 qubits, el campo es prometedor y se espera que en los próximos años se incremente la cantidad de qubits. Este incremento supone un aumento de la capacidad de procesamiento, y por ende, una reducción del tiempo necesario para resolver problemas complejos.

El primer algoritmo cuántico no trivial que demostró un potencial de crecimiento exponencial de velocidad sobre los algoritmos clásicos es el algoritmo de Shor [10]. Este algoritmo permite descifrar un mensaje encriptado mediante RSA descomponiendo en factores la clave pública, que es producto de dos números primos grandes, en un tiempo $O((\log N)^3)$, siendo N el numero primo que representa dicha clave pública. Los algoritmos clásicos en computadoras clásicas no pueden llevar a cabo esta factorización en un tiempo menor, por lo que RSA sigue siendo considerado, en la actualidad, un algoritmo seguro [11], [12].

Esto ha llevado a desarrollar algoritmos criptográficos que no fundamenten su seguridad en los principios matemáticos de complejidad computacional, es decir, en operaciones matemáticas de una sola vía, cálculos relativamente simples de realizar, pero extremadamente difíciles de revertir. Estos desarrollos dan origen a una serie de algoritmos que son resistentes al criptoanálisis cuántico: los algoritmos post-cuánticos [11].

Criptografía post-cuántica

La criptografía post-cuántica engloba todos aquellos sistemas criptográficos se cree que son resistentes tanto a computadoras clásicas como cuánticas. En otras palabras, nadie ha encontrado la forma de utilizar el algoritmo de Shor, que sí puede romper efectivamente a RSA, DSA y ECDSA, para vulnerar estos nuevos sistemas. Tanto los algoritmos como las implementaciones son todavía experimentales. El NIST (National Institute of Standards and Technology) se encuentra en etapa de selección de sistemas post-cuánticos para su posterior análisis y estandarización[13].

En la actualidad existen varios sistemas criptográficos post-cuánticos, entre los que se encuentran:

- Criptografía basada en hash.
- Criptografía basada en código.
- Criptografía basada en rejilla.
- Criptografía basada en ecuaciones cuadráticas multivariadas.
- Criptografía de clave secreta.
- Criptografía basada en isogenias supersingulares.

Todos estos sistemas disponen de una serie de algoritmos implementados en código que representan alternativas fiables a sus pares clásicos. Debido a los límites de extensión del presente trabajo, se centrará la atención en los algoritmos basados en rejilla, particularmente en uno de ellos: el cifrado de clave pública de Hoffstein-Pipher-Silverman "NTRU" [14]. Además, y como paso intermedio, se analizará el uso de cihpersuites tradicionales basadas en RSA y su combinación con un intercambio de claves resistente, logrando una comunicación basada en cifrado híbrido QSH (Quantum Safe Hybrid) [15].

NTRU

NTRU es un sistema criptográfico que consiste en dos algoritmos, NTRUEncrypt y NTRUSign, diseñados para cifrado/descifrado y firma digital respectivamente. Si bien esta suite fue patentada, en 2017 se liberó bajo dominio público y puede ser utilizada en aplicaciones liberadas bajo licencia GPL.

NTRU puede ejecutar operaciones de clave privada con un nivel de seguridad similar al de RSA, pero mucho más rápido [16]. Esto se debe a que el tiempo que demora RSA en realizar este tipo de operaciones se incrementa con el cubo del tamaño de la clave, mientras que en NTRU este incremento es cuadrático. Según Hermans, Vercauteren y Preneel [17] con una GPU GTX280 se pudo procesar NTRU con un nivel de seguridad de 256 bits con una tasa de sólo 20 veces más lenta que la de un cifrador simétrico AES.

A diferencia de RSA y ECDSA, NTRU es resistente a todos los ataques cuánticos, o al menos a los ataques conocidos hasta el momento. En 2009 Perlner y Cooper [18] mencionaron a NTRU como una alternativa post-cuántica para cifrado y firma digital resistentes al algoritmo de Shor, e hicieron alusión a que, de todos los algoritmos criptográficos basados en el esquema de rejilla, NTRU parecía ser el más práctico, incluso más que versiones alternativas como Stehle-Steinfeld NTRU [19].

Implementaciones utilizadas

En el presente trabajo se utilizó WolfSSL [20] como implementación de SSL/TLS para llevar a cabo las pruebas de NTRU.

WolfSSL (anteriormente CyaSSL) es una biblioteca de cifrado SSL de código abierto, ligera, portable y escrita en lenguaje C, cuyo objetivo principal es la implementación de protocolos de cifrado SSL/TLS en dispositivos de Internet de las cosas (Internet of Things, IoT), dispositivos embebidos, y entornos RTOS (Real Time Operating System - Sistemas operativos de tiempo real) debido principalmente al reducido tamaño de esta suite criptográfica, a su velocidad y al conjunto de características que ofrece.

WolfSLL puede correr tanto sobre computadoras de escritorio, como en entornos empresariales y en computación de nube. Soporta estándares de la industria tales como TLS v1.3 y DTLS v1.2, y su tamaño es hasta 20 veces menor que el de otras suites alternativas como OpenSSL. WolfSSL ofrece una *API* y una capa de compatibilidad con OpenSSL, y soporta los mecanismos de verificación y revocación de certificados digitales x509, OCSP (Online Certificate Status Protocol - Protocolo de comprobación de estado de certificados) y CRL (Certificate Revocation List - Lista de revocación de certificados) respectivamente.

Como *backend* criptográfico WolfSSL utiliza la biblioteca de cifrado wolfCrypt, un motor criptográfico ligero escrito en ANSI C cuyo objetivo también es el software embebido y su uso en sistemas con recursos de hardware limitados. WolfCrypt soporta los algoritmos de cifrado más comunes, a saber, RSA, ECC, DSS, Diffie–Hellman, EDH, NTRU, DES, Triple DES, AES (CBC, CTR, CCM, GCM), Camellia, IDEA, ARC4, HC-128, ChaCha20, MD2, MD4, MD5, SHA-1, SHA-2, SHA-3, BLAKE2, RIPEMD-160 y Poly1305, así como también los algoritmos experimentales como RABBIT[21], un software de transmisión de códigos de dominio público del Proyecto eSTREAM de la Unión Europea, y sumamente útil para cifrado de transmisiones en entornos de alto rendimiento.

WolfSSL también incluye los algoritmos recientes Curve25519 y Ed25519, y el algoritmo de criptografía de clave pública NTRU, objetivo de este trabajo.

Al utilizar la mínima cantidad de bits necesaria para proveer un nivel de seguridad equivalente a otros algoritmos asimétricos, NTRU es especialmente útil en dispositivos móviles y sistemas embebidos. Además, como se ha adelantado, es un algoritmo resistente a ataques cuánticos. Por otro lado, WolfSSL incluye varias suites de cifrado simétrico que hacen uso de NTRU internamente, tales como AES-256, RC4 y HC-128.

Entorno de pruebas: especificaciones

El montaje del entorno de pruebas consistió en los siguientes pasos:

1. Descarga, compilación e instalación de la biblioteca NTRU en el sistema operativo.

2. Descarga y compilación de WolfSSL integrado con la biblioteca NTRU.

3. Prueba de concepto de la negociación TLSv1.2.

4. Prueba de concepto de negociación QSH.

5. Prueba de concepto de negociación NTRU.

6. Descarga, adaptación de código y compilación de la versión 2.4.x de Apache2 httpd [22] desde sus repositorios oficiales SVN.

7. Prueba de concepto de negociación TLS con cifrado post-cuántico en un servidor Apache2 integrado con WolfSSL.

En las siguientes secciones del presente documento se exponen algunos detalles propios del montaje de las pruebas, y los resultados obtenidos. Como información adicional cabe señalar que los experimentos se realizaron sobre un sistema operativo GNU/Linux Lubuntu 19.04 con kernel 5.0.0-27-generic.

Entorno de pruebas: libntru y wolfssl

Para compilar WolfSSL de manera integrada con NTRU, primero se instaló la biblioteca NTRU en el sistema operativo. Para ello, y con la intención de mantener el orden, se creó el directorio de trabajo `/opt/wolfssl`.

Dentro de este directorio se descargó, desde los repositorios oficiales, la librería NTRU, se clonó el repositorio oficial, se generaron los archivos de configuración, se compiló y se instaló la librería en el sistema de la siguiente manera:

```
sudo git clone
https://github.com/NTRUOpenSourceProject/
NTRUEncrypt.git

cd NTRUEncrypt/

./autogen.sh

./configure

make

make install
```

Por defecto, la librería NTRU se instaló en el directorio `/usr/local/lib/` tal y como puede verse en la Figura 1.

```
root@juncotic-lubuntu:/opt/wolfssl# ls -l /usr/local/lib/ --color=no
total 3536
-rw-r--r-- 1 root root   352930 feb 11 11:32 libntruencrypt.a
-rwxr-xr-x 1 root root      981 feb 11 11:32 libntruencrypt.la
lrwxrwxrwx 1 root root       23 feb 11 11:32 libntruencrypt.so -> libntruencrypt.so.0.1.0
lrwxrwxrwx 1 root root       23 feb 11 11:32 libntruencrypt.so.0 -> libntruencrypt.so.0.1.0
-rwxr-xr-x 1 root root   233120 feb 11 11:32 libntruencrypt.so.0.1.0
-rwxr-xr-x 1 root root      956 feb 26 19:36 libwolfssl.la
lrwxrwxrwx 1 root root       20 feb 26 19:36 libwolfssl.so -> libwolfssl.so.24.0.0
lrwxrwxrwx 1 root root       20 feb  4 16:57 libwolfssl.so.23 -> libwolfssl.so.23.0.0
-rwxr-xr-x 1 root root  1486376 feb  4 16:57 libwolfssl.so.23.0.0
lrwxrwxrwx 1 root root       20 feb 26 19:36 libwolfssl.so.24 -> libwolfssl.so.24.0.0
-rwxr-xr-x 1 root root  1522024 feb 26 19:36 libwolfssl.so.24.0.0
drwxr-xr-x 2 root root     4096 feb 26 19:36 pkgconfig
drwxrwsr-x 4 root staff    4096 oct 28  2019 python2.7
drwxrwsr-x 3 root staff    4096 abr 16  2019 python3.7
root@juncotic-lubuntu:/opt/wolfssl#
```

Figura 1

Una vez que se tuvo la librería NTRU compilada e instalada, el siguiente paso fue compilar Wolfssl. Si bien al día de la fecha se encuentra disponible la versión v4.4.0-stable, la misma no se encuentra aún integrada a NTRU, por lo que se generan errores de compilación. Debido a esto se optó por utilizar la versión estable anterior, v4.3.0-stable. El directorio de trabajo para Wolfssl fue `/opt/wolfssl/wolfssl-git/`. Los siguientes comandos muestran cómo clonar el repositorio, cambiar la rama a la v4.3.0-stable, generar la configuración, configurar la implementación, compilarla e instalarla.

```
git clone
https://github.com/wolfSSL/wolfssl.git &&
cd wolfssl/

git checkout v4.3.0-stable

./autogen.sh

./configure --enable-qsh
--with-ntru=/usr/local/lib -enable-
opensslextra --enable-supportedcurves --
enable-apachehttpd --enable-tlsx --
enable-all --enable-ecc --enable-psk --
enable-aesccm C_EXTRA_FLAGS="-
DWOLFSSL_STATIC_RSA -DHAVE_SNI"

make && make install
```

Como puede observarse, el script de configuración, `./configure`, recibió por argumento una serie de opciones que incluyen las siguientes:

- `--enable-qsh`: Habilita QSH.
- `--with-ntru=/usr/local/lib`: Especifica el directorio donde se instaló libNTRU.
- `--enable-apachehttpd`: Habilita a wolfssl su integración con Apache2.

Debe notarse también que se habilitó un *flag* especial, `WOLFSSL_STATIC_RSA`, que obliga a la suite WolfSSL a utilizar claves RSA estáticas en lugar de efímeras. Si bien esto es altamente desaconsejado en servicios de producción, existen sistemas criptográficos como NTRU que requieren de claves estáticas para funcionar [23].

Negociación TLS

Con WolfSSL compilado y funcional, se pudo realizar una prueba de concepto utilizando los algoritmos por defecto provistos por la suite. WolfSSL incluye los binarios de ejemplo de servidor y cliente para realizar la verificación de conexión y funcionamiento, y además sus códigos fuente sirven de ejemplo para desarrollar aplicaciones que hagan uso de SSL/TLS o, como es el caso del presente trabajo, adaptar el código de Apache2 para que utilice esta suite en la negociación del intercambio seguro.

Para correr el servidor se utilizó el siguiente binario ejemplo:

```
cd /opt/wolfssl/wolfssl-git/

./examples/server/server
```

El servidor por defecto atiende en el puerto `TCP 11111`. La respuesta de servidor, sin ningún tipo de configuración adicional, es la que se ve en la Figura 2.

```
root@juncotic-lubuntu:/opt/wolfssl/wolfssl-git# ./examples/server/server
peer's cert info:
 issuer : /C=US/ST=Montana/L=Bozeman/O=wolfSSL_2048/OU=Programming-2048/CN=www.wolfs
sl.com/emailAddress=info@wolfssl.com
 subject: /C=US/ST=Montana/L=Bozeman/O=wolfSSL_2048/OU=Programming-2048/CN=www.wolfs
sl.com/emailAddress=info@wolfssl.com
 serial number:aa:c4:bf:4c:50:bd:55:77
SSL version is TLSv1.2
SSL cipher suite is QSH:TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
SSL curve name is FFDHE_3072
Server Random : 18CE52B5754C26DD14ED3B707F769EAF99988F9DB5689A5D9364B5D39CBF7E09
Client message: hello wolfssl!
root@juncotic-lubuntu:/opt/wolfssl/wolfssl-git#
```

Figura 2

La conexión del cliente pudo realizarse de la siguiente manera. En este caso la conexión fue local en el mismo equipo en el que corría el servidor.

```
cd /opt/wolfssl/wolfssl-git/

./examples/client/client
```

La respuesta de la negociación estándar se ve en la Figura 3. Como puede apreciarse en ambas capturas, la negociación SSL/TLS estándar utiliza TLS1.2, la ciphersuite `TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384`, y hace uso de `QSH`.

El cliente provee algunos comandos y opciones adicionales que pueden listarse con la opción `-h`. Entre ellos se encuentra la opción `-e`, que permite listar las ciphersuites disponibles que soporta esta versión del protocolo. Como se ha realizado la compilación utilizando los *flags* de NTRU y de QSH, se verán estos algoritmos en la salida, y también se verán los algoritmos mencionados al principio de esta sección.

```
root@juncotic-lubuntu:/opt/wolfssl/wolfssl-git# ./examples/client/client
Session Ticket CB: ticketSz = 142, ctx = initial session
peer's cert info:
 issuer : /C=US/ST=Montana/L=Bozeman/O=Sawtooth/OU=Consulting/CN=www.wolfssl.com/ema
ilAddress=info@wolfssl.com
 subject: /C=US/ST=Montana/L=Bozeman/O=wolfSSL/OU=Support/CN=www.wolfssl.com/emailAd
dress=info@wolfssl.com
 serial number:01
SSL version is TLSv1.2
SSL cipher suite is QSH:TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
SSL curve name is SECP256R1
Session timeout set to 300 seconds
Client Random : B130C9062F68C9993DE18F7E4D53EFAB18BE8E49D01A5757CFD21B3B5BECCD21
SSL-Session:
    Protocol  : TLSv1.2
    Cipher    : TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
    Session-ID:
    Session-ID-ctx:
    Master-Key: BD3C16C9D94A85909F49D8A311F1B9D7C6F8D7F8ECFFC0CAC82C1A3591E65BF176A4
53C1B1B7BD4004C81C91800545AB
    TLS session ticket:
    0000 - 7F C7 62 21 A1 DF 8A 84-20 B1 D1 4B 21 17 D1 EC   oGb..O....AK..Al
    0010 - 11 03 0D 0C 64 C7 4F 0A-0C 27 5F 65 B5 09 8A BE   ....dGO...Oe....
    0020 - 00 4C 1F CD C5 A8 DC 27-90 BB F3 F0 BB D0 80 60   .L.ME.L...c`...`
    0030 - 07 A6 E0 0D 8C C8 79 5D-21 C9 BF 5F 27 7E 6F B3   ..`..HiM.I.O.no.
    0040 - B2 DC 78 78 6F B0 D9 93-9D C8 BD 3C 76 C6 52 98   .Lhho.I..H..fFB.
    0050 - 37 60 64 0A C9 67 DD 5E-78 F4 BE 8F 65 D8 53 B4   .`d.IgMNhd..eHC.
    0060 - B0 B3 DC DC 84 E5 84 75-F6 0B FC 6E 62 64 E3 9C   ..LL.e.ef.lnbdc.
    0070 - 7E 21 D6 FA 14 75 77 29-A6 D6 C1 93 FE 72 00 00   n.Fj.eg..FA.nb..
    008E - 00 00 00 00 00 00 00 00-00 00 00 00 00 00         ..............
    Start Time: 1592942590
    Timeout   : 300 (sec)
    Extended master secret: yes
I hear you fa shizzle!
root@juncotic-lubuntu:/opt/wolfssl/wolfssl-git#
```

Figura 3

Para listar las ciphersuites de una manera entendible puede usarse este comando:

```
./examples/client/client -e|sed
's/:/\n/g'
```

Como el interés de este trabajo se centra en criptografía post-cuántica, a continuación se muestra el listado de las ciphersuites que son de interés:

- NTRU-RC4-SHA
- NTRU-DES-CBC3-SHA
- NTRU-AES128-SHA
- NTRU-AES256-SHA
- QSH

El cliente puede especificar una ciphersuite particular utilizando la opción `-l`. El siguiente ejemplo ilustra este caso:

```
./examples/client/client -l ECDHE-RSA-
AES256-SHA:ECDHE-ECDSA-AES128-SHA:ECDHE-
ECDSA-AES256-SHA
```

La salida mostró, en la sección `SSL-Session`, la ciphersuite elegida, ya que en este ejemplo particular se especificó más de una:

```
SSL-Session:

        Protocol  : TLSv1.2

        Cipher    :
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
```

Negociación QSH

En este caso particular no se habilitó al cliente a negociar QSH, por lo que la ciphersuite asociada no lo muestra tampoco. Con la intención de entender la implementación de WolfSSL en Cliente y Servidor, y para facilitar la identificación del uso de QSH durante la negociación SSL/TLS, se modificó el código fuente del servidor agregando la siguiente instrucción, aproximadamente en la línea `469` (ver Figura 4).

```
printf("Quantum-Safe Handshake
Ciphersuite?? -> %s\n",

  (wolfSSL_isQSH(ssl))?"Yes":"No");
```

```
server.c                                                                buffers
455         #endif
456             if (err != WOLFSSL_ERROR_WANT_READ) {
457                 printf("SSL_read input error %d, %s\n", err,
458                                                     ERR_error_string(err, buffer))
    ;
459                 err_sys_ex(runWithErrors, "SSL_read failed");
460             }
461         }
462         else if (SSL_get_error(ssl, 0) == 0 &&
463                                 tcp_select(SSL_get_fd(ssl), 0) == TEST_RECV_READY)
    {
464                 err = WOLFSSL_ERROR_WANT_READ;
465         }
466     } while (err == WC_PENDING_E || err == WOLFSSL_ERROR_WANT_READ);
467     if (ret > 0) {
468         input[ret] = 0; /* null terminate message */
469         printf("Quantum-Safe Handshake Ciphersuite?? -> %s\n",
470                 (wolfSSL_isQSH(ssl))?"Yes":"No");
471         printf("Client message: %s\n", input);
472     }
473 }
474
475 static void ServerWrite(WOLFSSL* ssl, const char* output, int outputLen)
476 {
477     int ret, err;
478     char buffer[WOLFSSL_MAX_ERROR_SZ];
479     int len;
NORMAL  examples/server/server.c        c   18% ☰  458/2503 ㏑ : 13  ☰ [469]tr…
```

Figura 4

De esta manera, la salida del servidor para el mismo ejemplo de conexión mencionado arriba es la que muestra la Figura 5.

```
root@juncotic-lubuntu:/opt/wolfssl/wolfssl-git# ./examples/server/server
peer's cert info:
 issuer : /C=US/ST=Montana/L=Bozeman/O=wolfSSL_2048/OU=Programming-2048/CN=www.wolfs
sl.com/emailAddress=info@wolfssl.com
 subject: /C=US/ST=Montana/L=Bozeman/O=wolfSSL_2048/OU=Programming-2048/CN=www.wolfs
sl.com/emailAddress=info@wolfssl.com
 serial number:aa:c4:bf:4c:50:bd:55:77
SSL version is TLSv1.2
SSL cipher suite is TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
SSL curve name is FFDHE_3072
Server Random : AC90223F051262CD53DBE916D9B131C6D877675A39AF2D98D6FF099F967DF54B
Quantum-Safe Handshake Ciphersuite?? -> No
Client message: hello wolfssl!
```

Figura 5

Ahora, si el cliente ofrece la ciphersuite QSH, el servidor la utiliza de manera adicional a una ciphersuite tradicional, y muestra el mensaje que se agregó al código (Figura 6).

```
./examples/client/client -l ECDHE-RSA-
AES256-SHA:ECDHE-ECDSA-AES128-SHA:ECDHE-
ECDSA-AES256-SHA:QSH
```

```
root@juncotic-lubuntu:/opt/wolfssl/wolfssl-git# ./examples/server/server
peer's cert info:
 issuer : /C=US/ST=Montana/L=Bozeman/O=wolfSSL_2048/OU=Programming-2048/CN=www.wolfs
sl.com/emailAddress=info@wolfssl.com
 subject: /C=US/ST=Montana/L=Bozeman/O=wolfSSL_2048/OU=Programming-2048/CN=www.wolfs
sl.com/emailAddress=info@wolfssl.com
 serial number:aa:c4:bf:4c:50:bd:55:77
SSL version is TLSv1.2
SSL cipher suite is QSH:TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
SSL curve name is FFDHE_3072
Server Random : F0D09A10846E1305A7F9383DE78C786CAEC030C83B1F97D1418833B66439E91D
Quantum-Safe Handshake Ciphersuite?? -> Yes
Client message: hello wolfssl!
```

Figura 6

Otras opciones interesantes y útiles que pueden utilizarse en el servidor para realizar pruebas son las siguientes:

- `-x`: Permite imprimir los mensajes de error sin cerrar la conexión.
- `-b`: Levanta el servicio en todas las interfaces de red, no solamente en localhost.
- `-i`: Permite mantener el servidor activo para realizar múltiples conexiones desde los clientes.
- `-n`: Habilita el cifrado NTRU en la negociación SSL/TLS (ver más adelante).
- `-c`: Permite especificar la ruta al certificado digital a utilizar en la conexión.
- `-k`: Permite especificar la ruta a la clave privada a utilizar en la conexión.

El cliente también acepta otras opciones que serán de utilidad más adelante:

- `-l`: Ya mencionada, permite especificar una ciphersuite determinada.
- `-h`: Permite conectar a un host particular, por IP o nombre, no necesariamente localhost.
- `-p`: Permite especificar un puerto destino distinto de `11111`.
- `-d`: Desactiva la verificación de par, lo que facilita las pruebas con certificados digitales autofirmados.
- `-g`: Envía un mensaje `HTTP GET` al servidor, útil si el servidor está interpretando https.

Negociación NTRU

La suite WolfSSL provee, junto con sus archivos de código fuente y configuraciones, una serie de claves y certificados digitales almacenados en el directorio `certs/` del raíz del repositorio. Dentro de este directorio se encuentran una clave y un certificado digital generados utilizando el algoritmo NTRU, a saber:

```
certs/ntru-cert.pem  #Certificado digital
certs/ntru-key.raw   #Clave privada
```

Con esta información, se puede lanzar el servidor utilizando el certificado y clave NTRU, y la opción `-n` que es requerida para activar este algoritmo, como se ve en la Figura 7. La línea de ejecución fue la siguiente:

```
./examples/server/server -c ./certs/ntru-cert.pem -k ./certs/ntru-key.raw -n
```

El cliente por defecto, y sin especificar ningún argumento, intentó establecer la comunicación utilizando NTRU también, como muestra la Figura 8.

```
root@juncotic-lubuntu:/opt/wolfssl/wolfssl-git# ./examples/server/server -c ./certs/
ntru-cert.pem -k ./certs/ntru-key.raw -n
peer's cert info:
 issuer : /C=US/ST=Montana/L=Bozeman/O=wolfSSL_2048/OU=Programming-2048/CN=www.wolfs
sl.com/emailAddress=info@wolfssl.com
 subject: /C=US/ST=Montana/L=Bozeman/O=wolfSSL_2048/OU=Programming-2048/CN=www.wolfs
sl.com/emailAddress=info@wolfssl.com
 serial number:aa:c4:bf:4c:50:bd:55:77
SSL version is TLSv1.2
SSL cipher suite is TLS_NTRU_RSA_WITH_AES_256_CBC_SHA
SSL curve name is FFDHE_3072
Server Random : E781A60ABAC53769F17FAAD1D75F6BDA596852816C4A7DB29E44A26F750AB2A7
Quantum-Safe Handshake Ciphersuite?? -> No
Client message: hello wolfssl!
root@juncotic-lubuntu:/opt/wolfssl/wolfssl-git#
```

Figura 7

```
root@juncotic-lubuntu:/opt/wolfssl/wolfssl-git# ./examples/client/client
Session Ticket CB: ticketSz = 142, ctx = initial session
peer's cert info:
 issuer : /C=US/ST=Montana/L=Bozeman/O=Sawtooth/OU=Consulting/CN=www.wolfssl.com/ema
ilAddress=info@wolfssl.com
 subject: /C=US/ST=Orgeon/L=Portland/SN=Test/O=wolfSSL/OU=Development/CN=www.wolfssl
.com/emailAddress=info@wolfssl.com
 serial number:94:68:8b:78:9e:1e:8b:e9:97:91:cf:80:21:bf:2d
SSL version is TLSv1.2
SSL cipher suite is TLS_NTRU_RSA_WITH_AES_256_CBC_SHA
Session timeout set to 300 seconds
Client Random : 5E2EE5200C69074549975D7F233200BB09A996208A3633706AF6A06CAFE635FC
SSL-Session:
    Protocol  : TLSv1.2
    Cipher    : TLS_NTRU_RSA_WITH_AES_256_CBC_SHA
    Session-ID:
    Session-ID-ctx:
    Master-Key: 0177BB9B7F8557AB1AC034871774AEFB4C393A5BD8D5EF9949510C6A812C6F7C748E
C8554FD80E4295F093B338FE329A
    TLS session ticket:
    0000 - 53 54 74 19 3D 3E B1 40-F5 FD 10 A9 9E 3D 82 3A   CDd.....em......
    0010 - 7F 9C 6F 93 78 68 BF 5F-C2 7F 29 EA 91 FA CB 33   o.o.hh.OBo.j.jK.
    0020 - 00 4C B2 D6 38 1E C4 41-D8 D4 24 20 1C CE 42 6B   .L.F..DAHD...NBk
    0030 - D6 4B 61 7A 77 77 D2 D7-02 EF 69 F7 5E E7 DF A2   FKajggBG.oigNgO.
    0040 - 4B D2 FC 52 16 29 0A EF-E3 BD BF 48 DE 50 58 70   KBlB...oc..HN.H`
    0050 - 53 FA BE E9 C0 84 78 98-E6 80 19 B2 DD 10 63 C7   Cj.i..h.f...M.cG
    0060 - EC 57 35 A0 AC C0 D3 6E-AF 6B 17 12 07 45 64 CB   lG....Cn.k...EdK
    0070 - 39 C8 09 F1 F5 D8 65 8E-20 C4 9D DA 50 E8 00 00   .H.aeHe..D.J.h..
    008E - 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
    Start Time: 1592958006
    Timeout   : 300 (sec)
    Extended master secret: yes
I hear you fa shizzle!
root@juncotic-lubuntu:/opt/wolfssl/wolfssl-git#
```

Figura 8

Como puede verse en la salida del servidor, al negociar una ciphersuite NTRU en SSL/TLS, no hace uso de QSH, un algoritmo que se habilitaba de manera predeterminada como se apreció en las Figuras 2 y 3.

Surge en este punto la duda de si WolfSSL, al habilitar NTRU, desactiva QSH, o desactiva cualquier otra ciphersuite distinta de NTRU. Si se intenta conectar un cliente especificando una ciphersuite estándar no NTRU, el servidor rechaza la negociación SSL/TLS porque no puede verificar la ciphersuite, como puede verse en la Figura 9.

```
root@juncotic-lubuntu:/opt/wolfssl/wolfssl-git# ./examples/server/server -c ./certs/
ntru-cert.pem -k ./certs/ntru-key.raw -n -x

SSL_accept error -501, can't match cipher suite
wolfSSL error: SSL_accept failed
Continuing server execution...

peer has no cert!
SSL version is TLSv1.2
SSL cipher suite is NONE
SSL curve name is FFDHE_3072
wolfSSL error: SSL in error state
Continuing server execution...

Server Random : 0000000000000000000000000000000000000000000000000000000000000000
SSL_read input error -501, can't match cipher suite
wolfSSL error: SSL_read failed
Continuing server execution...
```

Figura 9

Al invocarse al cliente sin utilizar ninguna ciphersuite, se negocia NTRU con el servidor. Sin embargo, incluso si se fuerza la negociación de QSH junto con ciphersuites NTRU, QSH no se habilita.

Por ejemplo, si el cliente especifica la ciphersuite `NTRU-AES256-SHA:QSH`, el cifrador resultante de la negociación SSL/TLS será | `TLS_NTRU_RSA_WITH_AES_256_CBC_SHA`, correspondiente a la ciphersuite ofrecida por el cliente, pero sin habilitar QSH, tal y como ocurrió con la salida mostrada en la Figura 5.

Esto se debe a que WolfSSL activa QSH a modo de ciphersuite adicional como complemento resistente a ataques cuánticos en algoritmos comunes, pero NTRU ya dispone de un intercambio QSH integrado en la librería, que se activa automáticamente sin necesidad de interacciones adicionales entre cliente y servidor.

La ciphersuite QSH tiene mayor preferencia que otros protocolos comunes a la hora de que cliente y servidor WolfSSL negocien los algoritmos. Básicamente, si un usuario integra NTRU dentro de WolfSSL y ambos extremos de conexión soportan NTRU, entonces la ciphersuite NTRU, incluido su intercambio de claves seguro, será seleccionada de manera predeterminada, salvo el caso de que el usuario excluya NTRU explícitamente permitiendo únicamente otras ciphersuites comunes [23].

El mismo manual de usuario de WolfSSL menciona que NTRU acelera el proceso de conexión SSL/TLS de 20 a 200 veces sobre un intercambio común RSA, y la mejora aumenta en la medida en que aumente el tamaño de la clave utilizada, es decir, el proceso de negociación de SSL/TLS será más rápido con claves de 8192 bits que con claves chicas de 1024 bits.

Ahora bien, debido a que NTRU no es ampliamente adoptado como ocurre con RSA, WolfSSL lo implementa en combinación con otras ciphersuites en cifradores híbridos. Los desarrolladores de WolfSSL propusieron la inclusión de NTRU en modo híbrido en la definición de QSH. De esta manera SSL/TLS se beneficia de las ventajas de NTRU en seguridad cuántica, y permite utilizar pares de claves resistentes a ataques cuánticos por única vez, además de los pares de claves de algoritmos tradicionales [15], [20]

Entorno de pruebas: WolfSSL y Apache

Para realizar la prueba de concepto de WolfSSL integrado en un servidor web Apache2, es necesario disponer del código fuente de WolfSSL compilado convenientemente para permitir su integración, y una versión moderna de Apache2. Para este caso se utilizó la versión 2.4.42-dev (Unix) descargada directamente desde el repositorio SVN del proyecto, y por recomendación de los desarrolladores de WolfSSL. Cabe aclarar que WolfSSL tiene soporte para su integración con versiones de Apache2 posteriores a la v2.4.0 [24].

Primero se instalaron las dependencias necesarias para realizar la compilación y ejecución del servidor web integrado con WolfSSL.

```
sudo apt install libxml2 libxml2-dev
```

Luego fue necesario descargar y compilar la versión desde el repositorio SVN de Apache2. Se realizaron los procedimientos dentro del directorio de trabajo de wolfssl:

```
cd /opt/wolfssl/

svn checkout
https://svn.apache.org/repos/asf/httpd/ht
tpd/branches/2.4.x httpd-2.4.x/

cd httpd-2.4.x/
```

Dado que Apache2 está desarrollado pensando en su integración con OpenSSL y variantes, para lograr correr el motor SSL/TLS utilizando WolfSSL se debió modificar el código fuente de dicho motor, particularmente el archivo `modules/ssl/ssl_engine_init.c`.

Los cambios incluyeron modificaciones de los nombres de las funciones `SSL_CTX_*` por su equivalente en WolfSSL, `wolfSSL_CTX*`.

WolfSSL implementa gran cantidad de funcionalidades de SSL, de las cuales solo algunas fue necesario modificar en Apache2 para lograr la integración. Para facilitar la identificación de estas funciones y su secuencia, se utilizó como base el código del ejemplo `server.c`. Se realizó la compilación utilizando el modificador `-E` en `gcc`, lo que permitió interpretar las directivas de preprocesamiento y obtener una versión final del código que realmente implementa NTRU, a la que se llamó `server.i`.

```
cd /opt/wolfssl/wolfssl-git

gcc examples/server/server.c -o
/tmp/server.i -I. -E
```

Además, se agregaron algunas líneas de depuración para verificar el resultado de la lectura de la clave NTRU del servidor:

```
if
(wolfSSL_CTX_use_NTRUPrivateKey_file(mctx
->ssl_ctx, keyfile) == WOLFSSL_SUCCESS)

       printf("DEBUG
wolfSSL_CTX_use_NTRUPrivateKey_file
SUCCESS!!!\n");
```

Luego de realizados estos cambios, se procedió a generar los scripts de soporte para la compilación, y la

configuración del código para su posterior compilación e instalación.

```
./buildconf
./configure \
        --enable-access_compat=shared \
        --enable-actions=shared \
        --enable-alias=shared \
        --enable-allowmethods=shared \
        --enable-auth_basic=shared \
        --enable-authn_core=shared \
        --enable-authn_file=shared \
        --enable-authz_core=shared \
        --enable-authz_groupfile=shared \
        --enable-authz_host=shared \
        --enable-authz_user=shared \
        --enable-autoindex=shared \
        --enable-dir=shared \
        --enable-env=shared \
        --enable-headers=shared \
        --enable-include=shared \
        --enable-log_config=shared \
        --enable-mime=shared \
        --enable-negotiation=shared \
        --enable-proxy=shared \
        --enable-proxy_http=shared \
        --enable-rewrite=shared \
        --enable-setenvif=shared \
        --enable-ssl=shared \
        --enable-unixd=shared \
        --enable-ssl \
--with-wolfssl=/opt/wolfssl/wolfssl-git \
        --enable-ssl-staticlib-deps \
        --enable-mods-static=all \
        --enable-static-ab \
        --with-included-apr \
        --with-libxml2
```

Una nota importante respecto de estas líneas de configuración es la referencia de la opción `--with-wolfssl`, que indica el directorio de los fuentes de wolfssl que deben utilizarse durante la compilación de Apache2. Debe recordarse que esta integración es bidireccional, durante la compilación de WolfSSL también debió especificarse el soporte para su integración con https como se mencionó cuando se configuró Wolfssl anteriormente.

La compilación e instalación de Apache2 se realizó con los siguiente comandos:

```
make
make install
```

Apache2 quedó correctamente instalado en el directorio predeterminado `/usr/local/apache2` y listo para ser utilizado.

En la Figura 10 puede verse el uso del comando `ldd` para verificar que Apache2 quedó efectivamente compilado con las librerías de wolfssl y NTRU.

```
root@juncotic-lubuntu:/usr/local/apache2# ldd bin/httpd
        linux-vdso.so.1 (0x00007fffc57d0000)
        libpcre.so.3 => /lib/x86_64-linux-gnu/libpcre.so.3 (0x00007fe8581d6000)
        libz.so.1 => /lib/x86_64-linux-gnu/libz.so.1 (0x00007fe8581ba000)
        libapr-2.so.0 => /usr/local/apache2/lib/libapr-2.so.0 (0x00007fe85814d000)
        libpthread.so.0 => /lib/x86_64-linux-gnu/libpthread.so.0 (0x00007fe85812c000)
        libwolfssl.so.24 => /usr/local/lib/libwolfssl.so.24 (0x00007fe857f45000)
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007fe857d5a000)
        libuuid.so.1 => /lib/x86_64-linux-gnu/libuuid.so.1 (0x00007fe857d4f000)
        libcrypt.so.1 => /lib/x86_64-linux-gnu/libcrypt.so.1 (0x00007fe857d15000)
        libdl.so.2 => /lib/x86_64-linux-gnu/libdl.so.2 (0x00007fe857d0f000)
        libxml2.so.2 => /lib/x86_64-linux-gnu/libxml2.so.2 (0x00007fe857b67000)
        /lib64/ld-linux-x86-64.so.2 (0x00007fe8583f3000)
        libm.so.6 => /lib/x86_64-linux-gnu/libm.so.6 (0x00007fe857a19000)
        libntruencrypt.so.0 => /usr/local/lib/libntruencrypt.so.0 (0x00007fe857a0a000)
        libicuuc.so.63 => /lib/x86_64-linux-gnu/libicuuc.so.63 (0x00007fe857839000)
        liblzma.so.5 => /lib/x86_64-linux-gnu/liblzma.so.5 (0x00007fe857812000)
        libicudata.so.63 => /lib/x86_64-linux-gnu/libicudata.so.63 (0x00007fe855e22000)
        libstdc++.so.6 => /lib/x86_64-linux-gnu/libstdc++.so.6 (0x00007fe855c40000)
        libgcc_s.so.1 => /lib/x86_64-linux-gnu/libgcc_s.so.1 (0x00007fe855c25000)
```

Figura 10

Con el servidor compilado e instalado, se editó el archivo `conf/httpd.conf` para habilitar el módulo SSL. Esto se logró descomentando la línea que incluye el módulo, y además se habilitó la entrada de `ServerName` para evitar advertencias en tiempo de ejecución:

```
ServerName www.juncotic.com:80
Include conf/extra/httpd-ssl.conf
```

Finalmente se editó en el archivo de configuración de dicho módulo, `conf/extra/httpd-ssl.conf`, las rutas a las claves y certificados NTRU provistos por WolfSSL. Para este caso particular, y por razones de orden, se copiaron todos los certificados y claves provistos por WolfSSL, situados en el directorio `wolfssl/certs` de los fuentes, dentro del directorio `/opt/keys/wolfssl`.

Adicionalmente tuvo que comentarse la opción `SSLHonorCipherOrder` de dicho archivo. Esta opción activada permite que, en el momento de la negociación SSLv3 o TLSv1, se tenga preferencia por la ciphersuite y algoritmos propuestos por el servidor en vez de los propuestos por el cliente. Esta opción no está soportada aún por el motor SSL/TLS provisto por WolfSSL.

```
[...]

#SSLHonorCipherOrder on

[...]

SSLCertificateFile
"/opt/keys/wolfssl/ntru-cert.pem"

[...]

SSLCertificateKeyFile
"/opt/keys/wolfssl/ntru-key.raw"

[...]
```

Una vez que se realizaron estos cambios, pudo iniciarse el servidor Apache2 invocando directamente al binario `bin/httpd`. Puede utilizarse la opción `-X` para que ejecute en modo de depuración, lo que permitirá ver los mensajes de error en el caso de que el servicio o la negociación SSL/TLS fallen. Además se puede especificar un nivel elevado de depuración usando agregando la opción `-e DEBUG` en la ejecución de manera opcional para mostrar mayor información.

```
root@juncotic-lubuntu:/usr/local/apache2#
./bin/httpd -X
```

El servidor quedó atendiendo en el puerto predeterminado TCP 443. Esto pudo verse con el comando `ss -npltu` como lo muestra la Figura 11.

```
root@juncotic-lubuntu:/opt/wolfssl/wolfssl-git# ss -npltu|grep httpd
tcp    LISTEN   0        128                    *:80                  *:*
   users:(("httpd",pid=7037,fd=4))

tcp    LISTEN   0        128                    *:443                 *:*
   users:(("httpd",pid=7037,fd=6))
```

Figura 11

Acto seguido se realizó la prueba de concepto utilizando el mismo cliente de ejemplo provisto por WolfSSL, pero especificando el puerto TCP 443 donde está atendiendo el servidor Apache2.

```
./examples/client/client -p 443 -g -d
```

En la Figura 12 puede verse el servidor Apache2 corriendo en la terminal superior, y el cliente conectado en la inferior. En la terminal del cliente se puede apreciar la ciphersuite NTRU negociada entre ambos, y la versión TLS 1.2 utilizada.

Si bien se ha logrado la conexión cliente-servidor utilizando el cliente provisto por WolfSSL y el servidor Apache2, en algunos casos aleatorios se detectaron fallas en la negociación, y terminaciones abruptas del proceso servidor.

A continuación se aprecia la respuesta del cliente en estos casos de falla:

```
root@juncotic-lubuntu:/opt/wolfssl/
wolfssl-git# ./examples/client/client -p
443 -g -d

CRL callback url =

wolfSSL_connect error -308, error state
on socket

wolfSSL error: wolfSSL_connect failed
```

Mientras que el servidor termina con un fallo de segmentación crítico:

```
Segmentation fault (core dumped)
```

```
root@juncotic-lubuntu:/usr/local/apache2# ./bin/httpd -X
DEBUG wolfSSL_CTX_use_NTRUPrivateKey_file SUCCESS!!!
DEBUG wolfSSL_CTX_use_NTRUPrivateKey_file SUCCESS!!!

root@juncotic-lubuntu:/opt/wolfssl/wolfssl-git# ./examples/client/client -p 443 -g -d
peer's cert info:
 issuer : /C=US/ST=Montana/L=Bozeman/O=Sawtooth/OU=Consulting/CN=www.wolfssl.com/emailAddres
s=info@wolfssl.com
 subject: /C=US/ST=Orgeon/L=Portland/SN=Test/O=wolfSSL/OU=Development/CN=www.wolfssl.com/ema
ilAddress=info@wolfssl.com
 serial number:94:68:8b:78:9e:1e:8b:e9:97:91:cf:80:21:bf:2d
SSL version is TLSv1.2
SSL cipher suite is TLS_NTRU_RSA_WITH_RC4_128_SHA
Session timeout set to 500 seconds
Client Random : 6B12274D6EEA3799C22E75A3749375DD1234D922E01F2F77D7322595089145E4
SSL-Session:
    Protocol  : TLSv1.2
    Cipher    : TLS_NTRU_RSA_WITH_RC4_128_SHA
    Session-ID: A2A8BE9CDDF9C7C242D7C6A782AFB5F25F7B04216B5BDEC0DC394C3D2D16BF64
    Session-ID-ctx:
    Master-Key: 1CFDF5C00B10B4DE49190BE0F91F32376DF49938B566DF4A78D3D97EA36BD9BD71B6BCA2FC3F
425458C4E70EDA6A7EDE
    TLS session ticket: NONE
    Start Time: 1593128787
    Timeout   : 500 (sec)
    Extended master secret: yes
SSL connect ok, sending GET...
HTTP/1.1 200 OK
Date: Thu, 25 Jun 2020 23:46:27 GMT
Server: Apache/2.4.42-dev (Unix) 4.3.0
Last-Modified: Tue, 04 Feb 2020 2
2:11:19 GMT
ETag: "2d-59dc751cc6d93"
Accept-Ranges: bytes
Content-Length: 45
Connection: close
Content-Type: text/html
```

Figura 12

La información que el servidor dejó disponible en el registro `log/error_log` es la siguiente:

```
[Fri Jun 26 21:07:16.448582 2020]
[core:notice] [pid 32013:tid
139640733110464] AH00051: child pid 32484
exit signal Segmentation fault (11),
possible coredump in /usr/local/apache2
```

Se intuye que este error esporádico es por conflictos en la integración de WolfSSL en el módulo `mod_ssl` de Apache2. Esto puede verse en el siguiente mensaje de log:

```
[Fri Jun 26 21:09:44.250532 2020]
[ssl:warn] [pid 1512:tid 139988311048384]
AH01882: Init: this version of mod_ssl
was compiled against a newer library
(4.3.0, version currently loaded is
4.3.0) - may result in undefined or
erroneous behavior
```

Conclusiones parciales y perspectivas

Los experimentos llevados a cabo sirvieron para lograr una primera aproximación al uso de algoritmos de intercambio de claves y de cifrado resistentes a ataques cuánticos en TLS v1.2, y permitieron implementar una prueba de concepto de servicio HTTPS resistente a ataques cuánticos mediante la integración del servidor web Apache2 y la suite de cifrado SSL/TLS WolfSSL.

Se logró en primera instancia la negociación TLS resistente a ataques cuánticos en WolfSSL. Primero estableció una comunicación TLS1.2 mediante las implementaciones de ejemplo de cliente y servidor provistas por el proyecto WolfSSL, y haciendo uso de algoritmos asimétricos tradicionales RSA. A dicha comunicación se le incorporó el intercambio de claves híbrido QSH en combinación con los algoritmos tradicionales. Este intercambio híbrido logró brindar una capa adicional de seguridad en el handshake TLS. Por último se utilizaron la clave y el certificado digital NTRU provisto por WolfSSL como ejemplo para lograr negociar ésta ciphersuite en el intercambio de datos entre cliente y servidor.

Con la seguridad de tener una implementación de TLS post-cuántica basada en QSH y NTRU, se procedió a integrar WolfSSL con Apache2. Esto permitió dar una capa de seguridad resistente a ataques cuánticos al protocolo HTTPS servido por Apache2. Las pruebas permitieron dar cuenta del uso de QSH y NTRU en la negociación TLS establecida entre el cliente ejemplo de WolfSSL y el servidor web Apache2.

Durante la realización de los experimentos se detectaron inconsistencias propias de versiones experimentales de las implementaciones de software, y se llegó a la conclusión de que esta integración post-cuántica no es apta para su uso en producción. Se extiende a futuros trabajos un análisis más profundo de los errores, y posibles sugerencias de corrección.

Por otro lado, los resultados obtenidos pueden sentar la base para la realización de pruebas de rendimiento, y su posterior comparación con otras implementaciones de similares características.

Referencias

[1] A. Freier and P. Karlton, “The Secure Sockets Layer (SSL) Protocol Version 3.0.” 2011, [Online]. Available: https://tools.ietf.org/html/rfc6101.

[2] T. Dierks and E. Rescorla, “The Transport Layer Security (TLS) Protocol Version 1.2.” IETF, Aug. 2008, [Online]. Available: https://tools.ietf.org/html/rfc5246.

[3] E. Rescorla, “RFC 2631 - Diffie-Hellman Key Agreement Method.” IETF, 1999, Accessed: Sep. 16, 2019. [Online]. Available: https://tools.ietf.org/html/rfc2631.

[4] R. R. Ahirwal and M. Ahke, “Elliptic curve diffie-hellman key exchange algorithm for securing hypertext information on wide area network,” International Journal of Computer Science and Information Technologies, vol. 4, no. 2, pp. 363–368, 2013.

[5] R. L. Rivest, A. Shamir, and L. Adlemar, A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. EEUU: Communications of the ACM, 1987.

[6] C. Kerry and P. Gallagher, FIPS PUB 186-4: Digital Signature Standard (DSS). FEDERAL INFORMATION PROCESSING STANDARDS PUBLICATION. National Institute of Standards und Technology, 2013.

[7] D. B. Johnson and A. J. Menezes, “Elliptic curve DSA (ECDSA): an enhanced DSA,” in Proceedings of the 7th conference on USENIX Security Symposium, 1998, vol. 7, pp. 13–23.

[8] J. Daemen and V. Rijmen, “AES Proposal: Rijndael.” 1999, Accessed: Sep. 16, 2019. [Online]. Available: http://csrc.nist.gov/CryptoToolkit/aes/rijndael/Rijndael.pdf.

[9] “IBM’s new 53-qubit quantum computer is its biggest yet - CNET,” 2019. https://www.cnet.com/news/ibm-new-53-qubit-quantum-computer-is-its-biggest-yet/ (accessed Aug. 21, 2020).

[10] P. W. Shor, “Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer,” Journal SIAM Journal on Computing, vol. 26, no. 5, pp. 1484–1509, 1997.

[11] D. J. Bernstein, J. Buchmann, and E. Dahmen, Post Quantum Cryptography, 1ra ed. Berlin: Springer, 2009.

[12] E. W. Weisstein, “RSA-640 Factored,” MathWorld Headline News, Nov. 05, 2005.

[13] NIST, “Workshops and Timeline - Post-Quantum Cryptography | CSRC,” 2020. https://csrc.nist.gov/projects/post-quantum-cryptography (accessed May 22, 2020).

[14] OnBoard Security, “NTRUOpenSourceProject/ntru-crypto: Open Source NTRU Public Key Cryptography and Reference Code,” 2017. https://github.com/NTRUOpenSourceProject/ntru-crypto (accessed Aug. 21, 2020).

[15] W. Whyte, Z. Zhang, S. Fluhrer, and O. Garcia-Morchon, “Quantum-safe hybrid (QSH) key exchange for Transport Layer Security (TLS) version 1.3,” Internet-Draft draft-whyte-qsh-tls13-06, Internet Engineering Task Force, 2017. Accessed: Feb. 11, 2020. [Online]. Available: https://tools.ietf.org/html/draft-whyte-qsh-tls13-06.

[16] T. Baktu, “NTRU: Quantum-Resistant High Performance Cryptography,” 2017. https://tbuktu.github.io/ntru/ (accessed Jul. 26, 2020).
[17] J. Hermans, F. Vercauteren, and B. Preneel, “Speed Records for NTRU,” in Topics in Cryptology - CT-RSA 2010, vol. 5985, J. Pieprzyk, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 73–88.
[18] R. A. Perlner and D. A. Cooper, Quantum Resistant Public Key Cryptography: A Survey. Maryland, EEUU: National Institute of Standards and Technology, 2009.
[19] D. Stehlé and R. Steinfeld, Making NTRUEncrypt and NTRUSign as Secure as Standard Worst-Case Problems over Ideal Lattices. 2013.
[20] WolfSSL, “Quantum Safe wolfSSL - wolfSSL,” 2019. https://www.wolfssl.com/quantum-safe-wolfssl/ (accessed Jun. 24, 2020).
[21] M. Boesgaard, M. Vesterager, T. Pedersen, J. Christiansen, and O. Scavenius, “Rabbit: A new high-performance stream cipher,” in International Workshop on Fast Software Encryption, 2003, pp. 307–329.
[22] The Apache Software Foundation, “Apache http server project,” 2018. https://httpd.apache.org/.
[23] WolfSSL, wolfSSL User Manual, V4.1.0. 2019.
[24] WolfSSL, “wolfSSL + Apache httpd - wolfSSL,” 2020. https://www.wolfssl.com/wolfssl-apache-httpd/ (accessed Jun. 25, 2020).