Introducción a la criptografía Post-Cuántica

- -

Introduction to Post-Quantum Cryptography

TRABAJO FIN DE GRADO
GRADO EN INGENIERÍA INFORMÁTICA
CURSO 2021–2022

David de Santiago Lafuente

Directores
Luis Javier García Villalba
Elena Salomé Almaraz Luengo

Departamento de Ingeniería del Software e Inteligencia Artificial
Facultad de Informática
Universidad Complutense de Madrid

Madrid, Junio de 2022

Agradecimientos

Quiero agradecer a mis directores, D. Luis Javier Garcıa Villalba y Dª. Elena Salomé Almaraz Luengo, su esfuerzo y dedicación, ya que me han guiado y aconsejado durante este proyecto y en especial al profesor Robson de Oliveira de la Universidad de Brasilia que ha resuelto con diligencia todas las dificultades que se han presentado en este trabajo.

A la Facultad de Informática de la UCM y a todos los profesores que me han transmitido los conocimientos necesarios para poder afrontar este reto.

Tampoco quiero olvidarme de mi familia, pareja y amigos que me han apoyado en todo momento.

Gracias a todos.

Índice General

Índice de Figuras

Índice de Tablas

Índice de Algoritmos

Lista de Acrónimos

AAD	*Additional Authentication Data*
AEAD	*Authenticated Encryption with Associated Data*
AES	*Advanced Encryption Standard*
AJAX	*Asynchronous JavaScript And XML*
BB84	*Bennett and Brassard 1984*
BSD	*Berkeley Software Distribution*
CBC	*Cipher-Block Chaining*
CCA	*Chosen Ciphertext Attack*
CPA	*Chosen Plaintext Attack*
CPU	*Central Processing Unit*
DoS	*Denial of Service*
DSA	*Digital Signature Algorithm*
ECC	*Elliptic Curve cryptography*
ECDH	*Elliptic Curve Diffie-Hellman*
ECDSA	*Elliptic Curve Digital Signature Algorithm*

GHz	Gigahercio
HKDF	*Hash based Key Derivation Function*
HMAC	*Hash-Based Message Authentication Code*
HSM	*Hardware Secure Modules*
HTML	*HyperText Markup Language*
HTTPS	*HyperText Transfer Protocol Secure*
IDE	*Integrated Development Environment*
IOS	*iPhone operating system*
IoT	*Internet of Things*
IP	*Internet Protocol*
IV	*Initialization Vector*
J2EE	*Java 2nd Platform Enterprise Edition*
JDK	*Java Development Kit*
JSF	*Java Server Faces*
KDF	*Key Derivation Function*
LAN	*Local Area Network*
MAC	*Macintosh*
macOS	*Macintosh Operating System*
MB	*MegaByte*
MVC	Modelo Vista Controlador

NIST	*National Institute of Standards and Technology*
NTRU	*N-th degree Truncated polynomial Ring Units*
OpenSSL	*Open Secure Sockets Layer*
PC	*Personal Computer*
PKI	*Public Key Infrastructure*
PRF	*Pseudo-Random Functions*
PRNG	*Pseudo-Random Number Generator*
QKD	*Quantum Key Distribution*
RAM	*Random Access Memory*
RFC	*Remote Function Call*
RFID	*Radio Frequency IDentification*
RSA	*River-Shamir-Adleman*
SIDH	*Supersingular isogeny Diffie-Hellman*
SIKE	*Supersingular Isogeny Key Encapsulation*
SIS	*Short Integer Solution*
SLP	*Sequential Linear Programming*
SSH	*Secure SHell*
TFG	Trabajo de Fin de Grado
TLS	*Transport Layer Security*

UI	*User Interface*
UML	*Unified Modeling Language*
VPN	*Virtual Private Network*
web	*World Wide Web*
XOR	*Exclusive Or*

Abstract

In this work we will deal with the importance of quantum computing in computing, and more specifically in cybersecurity. To do this, first, a detailed introduction of what cryptography is as well as the main cryptographic algorithms is made. Secondly, quantum computing is exposed, commenting on its origins, explaining its operation and focusing on cybersecurity aspects as well as the main algorithms it uses or the most well-known attacks. Once these concepts are detailed, it focuses on VPNs, more specifically on WireGuard, of which its different algorithms are analyzed in detail and changes are proposed to adapt it to the post-quantum future. Finally, the digital signature analysis laboratory web application is also explained and tested, with which tests are carried out between different hash algorithms to see their different characteristics and performance.

Keywords: Analysis, Cryptography, Hash, Post-Quantum Cryptography, Quantum Computing, VPN, WireGuard.

Resumen

En este trabajo se trata la importancia de la computación cuántica en la informática, y más en concreto en la ciberseguridad. Para ello, en primer lugar, se realiza una introducción detallada de qué es la criptografía así como los principales algoritmos criptográficos. En segundo lugar, se expone la computación cuántica, comentando sus orígenes, explicando su funcionamiento y enfocándose en aspectos de ciberseguridad así como los principales algoritmos que usa o los ataques más conocidos. Una vez detallados estos conceptos, se centra en las VPNs, más en concreto en WireGuard, del cual se analizan detalladamente sus distintos algoritmos y se proponen cambios para adecuarlo al futuro post-cuántico. Por último también se explica y prueba la aplicación web de laboratorio de análisis de firmas digitales, con la que se realizan pruebas entre distintos algoritmos de *hash* para ver sus distintas características y rendimientos.

Palabras clave: Análisis, Computación cuántica, Criptografía, Criptografía Post-Cuántica, *Hash*, VPN, WireGuard.

Capítulo 1

Introducción

1.1. Motivación

La criptografía es la técnica de escribir con clave secreta, se encarga de la escritura y el cifrado de mensajes para que resulten difíciles de leer a cualquier tercero que intercepte nuestra información. Etimologicamente viene del griego *cripto*, que significa oculto o secreto y *grafía*, que es su representación gráfica. El criptoanálisis se ocupa de los métodos de descifrado sin el conocimiento de la clave que se ha empleado para cifrar.

En la Grecia clásica usaban antorchas para deletrear palabras y enviar mensajes. En el Imperio Romano, se enviaban mensajes ininteligibles gracias a la colocación de las letras de modo que no podía formarse ninguna palabra y para descifrarlas se debía cambiar el orden de las letras. En tiempos más modernos se han empleado otros métodos como por ejemplo espejos que reflejaban la luz del Sol usando el Código Morse. Hasta este momento se usaban cifrados por sustitución monoalfabéticos.

León Battista Alberti escribió en 1466 *De Componendis Cyphris*, el libro sobre criptografía más antiguo que se conoce en el mundo occidental donde se analiza este tipo de cifrado. Históricamente es el primero en proponer los cifrados polialfabéticos.

Durante las Guerras Mundiales, la mayoría de los mensajes se transmitían por radio, pero tenía el inconveniente de que cualquier persona que dispusiera de un receptor sintonizado en la frecuencia adecuada podía escucharlos, por lo que había que encriptarlos. El ejército alemán utilizó en la Segunda Guerra Mundial la máquina de encriptación más famosa de la historia: Enigma. Alan Turing fue el principal responsable de descifrar Enigma, contribuyendo con ello a acortar la guerra. A partir de este momento se desarrolló la criptografía moderna.

El medio de telecomunicaciones más utilizado en la actualidad es Internet, que tiene como pilar básico el uso de la criptografía como forma de garantizar la seguridad y operatividad de sus recursos con respecto a la confidencialidad, integridad, autenticidad y no repudio de los mensajes intercambiados. Lo que garantiza este apoyo es la imposibilidad práctica de romper la criptografía en los sistemas y aplicaciones con los recursos computacionales que existen hoy en día. Sin embargo, con la aparición de los primeros computadores cuánticos muchas de las claves de cifrado, actualmente seguras, en un futuro podrían ser vulnerables dejando al descubierto todo el sistema de cifrado actual, por ello es necesario preparar nuestra criptografía para hacerla resistente a la computación cuántica.

En este Trabajo de Fin de Grado (TFG) se tratarán los principios básicos de la criptografía así como sus distintos tipos y utilidades comentando los principales ataques, también se tratará la computación cuántica teniendo en cuenta sus ataques teóricos y los posibles problemas que esta generaría en la seguridad clásica actual, para centrarse en *Virtual Private Network* (VPN) WireGuard y su adaptación para un futuro cuántico. En este trabajo introductorio se propondrán una serie de indicaciones para la sustitución de los algoritmos de WireGuard por unos post-cuánticos, centrándonos en el cambio de blake2s-256 por blake2b-512, para lo cual se detallarán las acciones teóricas necesarias para llevar a cabo esta tarea. Como parte práctica surgió la idea de la creación de una herramienta de laboratorio de firmas digitales, que permitiera analizar de forma fácil y comparativa distintos parámetros de algoritmos de *hash* y que, gracias a su análisis de datos y comparación, sirvió para apoyar la decisión del cambio de *hash* propuesto para WireGuard.

1.2. Objeto de la Investigación

Comprender qué es y cómo funciona la computación cuántica, más en concreto en su aplicación en ciberseguridad y en cómo afectaría este escenario al cifrado y criptografía clásicos actuales.

Elegir una VPN muy usada actualmente y de código libre y afrontar el estudio de los cambios necesarios para adecuarla a un comportamiento post-cuántico.

Implementación de una aplicación Java Web de análisis de algoritmos de *hash* que, además de realizar resúmenes de preimagenes introducidas, posibilita la comparación dinámica y visual entre distintos algoritmos así como la obtención de medidas relevantes a la hora de elegir una función de resumen u otra.

1.3. Plan de Trabajo

Se han completado cuatro fases:

1. **Documentación**: Esta primera fase ha consistido principalmente en la búsqueda y la familiarización con el contenido del proyecto, en especial con la computación cuántica y con la herramienta de VPN WireGuard. Tras las primeras reuniones con los profesores, se orientaron las tareas hacia la búsqueda de información sobre criptografía, incluyendo los algoritmos *hash*, los conceptos básicos sobre la criptografía post-cuántica, los conceptos matemáticos asociados y lo que implica este tipo de computación de cara a un futuro. Se recopila información de distintas fuentes y bibliografía así como se estudia y prueba la aplicación a modificar para entender su funcionamiento.

2. **Desarrollo**: Una vez entendidos y presentados los conceptos, se pasa a una fase de desarrollo práctico. Se toma una VPN reconocida, como es WireGuard, y se plantea como serían los cambios a realizar para que uno de sus algoritmos pase a ser post-cuántico. Para ello se accede al código y se analizan los distintos ficheros en donde aparece el algoritmo a modificar. Se anotan todas las indicaciones y se recogen en la memoria de este proyecto. Para comprobar la validez de este

cambio, y que ventajas proporciona la integración de un algoritmo post-cuántico, surge la idea de implementar una Aplicación Web que nos facilite este paso. Se decide su implementación con el objeto de servir como comparador y analizador de distintos algoritmos de *hash*, donde en función de una preimagen de entrada nos proporciona el resumen calculado acompañado de una serie de medidas que se han considerado de utilidad, todo ello en función del tamaño de palabra introducida. La salida se acompaña de gráficos comparativos, tabla con los datos y exportaciones a Excel y gráficas. Además se incorpora una breve información sobre los algoritmos elegidos para el estudio. En el caso concreto que nos ocupa se necesitaba comparar blake2s-256 con blake2b-512. Además se deciden incluir MD5, SHA256 y SHA512 como representación de algoritmos más utilizados. La aplicación es escalable por lo que, en caso de necesidad, podrían incorporarse más algoritmos y medidas.

3. **Realización de pruebas**: Se desarrollan las pruebas que toda la aplicación debe incluir para comprobar su correcto funcionamiento. Se verifica con la implementación de pruebas unitarias para todos los cálculos de los algoritmos y la entropía de Shannon, donde para diversas entradas se comprueba que devuelven los valores esperados. También se realizan las pruebas funcionales para complementar a las pruebas unitarias. Dentro de esta fase se destaca el estudio del comportamiento de los algoritmos en función de los resultados y cómo esto se puede aplicar al estudio de los cambios propuestos en WireGuard. Se llegan a conclusiones reflejadas en la memoria y se da un avance del trabajo de sustitución en WireGuard que queda abierto como continuación.

4. **Memoria**: Implica recoger todo lo desarrollado, catalogarlo e incluirlo en sus apartados correspondientes para que la información se presente estructurada y quede reflejado todo el proceso realizado. Se añaden las conclusiones y el trabajo futuro que se abre a partir de este proyecto introductorio.

1.4. Estructura del Trabajo

El trabajo se organiza en 10 capítulos, antes de comenzar con los capítulos se catalogan las imágenes y las tablas y se identifican los acrónimos.

El Capítulo 1 comprende la introducción y motivación del TFG. En él se detallan los aspectos de por qué y cómo se ha llevado a cabo este trabajo, explicando el objeto y el plan de trabajo seguidos en su realización.

El Capítulo 2 trata sobre la computación cuántica. En él se introduce este nuevo tipo de computación, se explica su funcionamiento y utilidad y se comentan sus principales algoritmos. También se realiza una comparación con la actual computación clásica y se detallan los posibles inconvenientes que esta provocaría en el campo de la ciberseguridad.

El Capítulo 3 muestra la importancia de la criptografía. En él se comenta el interés en este campo hoy en día así y realiza una explicación de los distintos tipos de criptografía clásica.

En el Capítulo 4 se trata la criptografía resistente a la computación cuántica, también llamada criptografía post-cuántica. Se explican los nuevos tipos de técnicas de cifrado que han surgido para hacer frente al escenario de la computación cuántica.

El Capítulo 5 presenta las características VPN resistentes a la computación cuántica. Se

define el concepto de VPN y se explica el concurso de algoritmos de carácter post-cuántico realizado por el *National Institute of Standards and Technology* (NIST).

En el Capítulo 6 se introduce la herramienta de VPN WireGuard así como todo el abanico de algoritmos criptográficos que esta usa. Tras la explicación se realiza un análisis de seguridad de cada uno de ellos proponiendo soluciones post-cuánticas. En el caso de la sustitución de blake2s-256 por blake2b-512 se ofrece una propuesta teórica de cambio detallada.

El Capítulo 7 comprende la Aplicación Web de laboratorio de firmas digitales, dónde, una vez comentada la estructura y herramientas utilizadas, se presentan sus funcionalidades, así como pruebas de buen funcionamiento y diagramas *Unified Modeling Language* (UML) correspondientes. En último lugar se muestra un ejemplo de análisis real haciendo uso de ella.

El Capítulo 8 indica las principales conclusiones de este trabajo, las líneas futuras de investigación y los posibles trabajos derivables.

Los Capítulos 9 y 10 son las traducciones al inglés de la Introducción y de las Conclusiones.

Por último se puede encontrar la bibliografía utilizada.

Capítulo 2

Computación cuántica

Fué desarrollada por Paul Benioff en 1981 y se baso en las leyes cuánticas y principalmente en el "*cuanto*" como concepto aplicado a la informática. Supo prever la enorme potencia que implicaría en el proceso informático y como podría ser una revolución cuantitativa en este aspecto.

Una computadora cuántica es un dispositivo de computación que hace uso de fenómenos mecánicos cuánticos. [Webd]. Las supercomputadoras más potentes actualmente se verán superadas por las máquinas de caracter cuántico y todo ello con un gasto energético menor. Es importante aclarar que aunque los ordenadores cuánticos se presenten como sustitutos de los ordenadores actuales, para muchas tareas estos seguiran siendo más eficientes. Este tipo de computación nos ofrece grandes aplicaciones que podrán desarrollarse exponencialmente en una multitud de campos.

A continuación vamos a detallar las principales características de este tipo de computación así como sus algoritmos más importantes y ataques relevantes.

2.1. Computación cuántica VS Computación clásica

Para comenzar, se introducirá qué es la computación cuántica y que diferencias tiene respecto de la clásica. En primer lugar, los ordenadores clásicos usan *bits* con dos estados: 0 y 1. Por lo que en un ordenador con N bits la cantidad de información que contiene un estado concreto de la máquina es N. Mientras que por otro lado, los ordenadores cuánticos usan *qubit*s, los cuales no tienen dos estados 0 y 1 sino una combinación de ambos. Con N *qubits*, la cantidad de información que contiene un estado concreto de la maquina es 2^N, siendo esta mucho mayor que la comentada anteriormente con *bits*. Cabe mencionar que si se aumenta en 1 el número de *bits*, la información que almacena el estado de un ordenador clásico aumenta de N a $N+1$, mientras que si se aumenta en 1 el número de *qubits* en un ordenador cúantico, la cantidad de información de cada estado se duplica, es decir pasa de 2^N a 2^{N+1} brindando una mayor potencia de cálculo [Webe].Otra diferencia es el uso de puertas cuánticas en vez de las puertas lógicas de los ordenadores clásicos, siendo estas mucho más eficientes para la realización de operaciones. Hoy en día existen algunos ordenadores cuánticos pero de muy pocos *qubits*, ya que a medida que su número aumenta se hace más complicado mantener el entrelazamiento entre ellos, siendo un requisito indispensable en este tipo de computación.

2.2. Consecuencias de la computación cuántica en la seguridad

Hoy en día la mayoría del cifrado de Internet viene dado por el sistema de clave pública y privada. Dicho sistema es esencial para la seguridad debido a su capacidad para resolver el problema de la distribución de claves y brindar alta seguridad en canales de comunicación no seguros que nos permiten acceder a sitios *World Wide Web* (web), realizar intercambios de correos electrónicos, transacciones financieras, firmar documentos digitalmente, realizar comunicaciones militares o almacenar datos médicos. También cabe mencionar que este sistema está integrado en muchas de las capas de Internet, como *Transport Layer Security* (TLS), y es utilizado tanto en ordenadores tradicionales como en *Internet of Things* (IoT).

Los ordenadores cuánticos son teóricamente capaces de romper este tipo de cifrado y, aunque no pueden vulnerar claves muy extensas debido a los pocos *qubits* que tienen, se prevé que podrían hacerlo a medida que la tecnología aumente. De ahí surge la necesidad de la creación de unos algoritmos post-cuánticos resistentes a este tipo de ataques [Ket]. Por lo tanto, apuntando a un posible escenario inminente en el que la criptografía clásica se vería amenazada por el desarrollo de técnicas de computación cuántica en un futuro aún incierto, o en el que es posible que los datos cifrados actualmente ya se estén capturando para futuras rupturas de cifrado, el criptoanálisis toma una gran importancia, estableciendo un esquema de desarrollo de algoritmos criptográficos con la finalidad de crear algoritmos resistentes a este tipo de computación a la vez que siguen siendo eficientes y resistentes a los ataques ya conocidos [Ket].

El esquema de desarrollo de algoritmos es el siguiente:

- Los criptógrafos diseñan los sistemas para codificar y descifrar.
- Los criptoanalistas ponen a prueba estos sistemas.
- Los diseñadores implementan los algoritmos resistentes más rápidos.

La defensa contra estos ataques actualmente se enfoca en el tamaño de la clave como solución temporal ya que, como hemos dicho, no son capaces de descifrar claves de gran tamaño por lo que se ha decidido aumentar el tamaño mínimo de clave recomendado [Ket]. Se supone que los algoritmos simétricos y las funciones *hash* solo requerirán aumentar su tamaño de la clave de salida. Por ejemplo, las claves con una longitud de 256 *bits* se considera que tienen un alto grado de seguridad y son teóricamente resistentes a los ataques de fuerza bruta de una computadora cuántica [FC]. El tamaño de clave *River-Shamir-Adleman* (RSA) mínimo recomendado es actualmente entre 2048 y 4096 *bits* (según el tipo de información a proteger), ya que para RSA de 768 y 1024 *bits* las implementaciones se rompieron alrededor de 2010. [jea10] Pero con el tiempo y los avances esto podría cambiar, más en concreto, se estima que en los próximos 20 años, las computadoras cuánticas serán suficientemente funcionales para romper los fuertes criptosistemas de clave pública actuales fácilmente, por lo que son necesarios algoritmos de cifrado verdaderamente post-cuánticos [Mos18].

Existen ya algunos algoritmos de cifrado resistentes teóricamente a los ordenadores cuánticos pero aún queda perfeccionar algunos aspectos tales como:

- La eficiencia: actualmente no son viables a gran escala debido al tiempo que tardan y al espacio que ocupan.

- La confianza: es un campo muy novedoso y es necesario generar confianza real en ellos por parte de los usuarios.
- La usabilidad: necesitamos unas implementaciones y pruebas adecuadas para una verdadera integración sin contratiempos inesperados.

2.3. Principales algoritmos cuánticos

Como ya hemos dicho, los ordenadores cuánticos son capaces de romper la criptografía actual, para ello hacen uso de algoritmos. Un algoritmo computacional cuántico es una secuencia finita y ordenada de pasos a ser realizados por una máquina de computación cuántica para realizar una acción o resolver un problema. Son algoritmos teóricos para la ejecución en un modelo computacional cuántico realista. La mayoría de ellos fueron desarrollados bajo modelos discretos de computación, es decir, que se basan en un espacio de estados y pasos de tiempo discretos [Ket].

Los principales algoritmos de este tipo referentes a la ciberseguridad son el algoritmo de Shor y el de Grover.

2.3.1. Algoritmo de Shor

Este algoritmo fue desarrollado en 1996 por el matemático Peter Shor y es capaz de descomponer un número en factores primos [Webg]. Consta de dos partes: una cuántica donde se determina el periodo de una función y una clásica donde se hace uso de ese resultado para determinar cuáles son los factores primos de un número. Puede realizar dicha tarea en un tiempo polinómico, en el cual el tiempo de ejecución es calculado a partir del número de variables involucradas. La seguridad actual se basa en que eso no es posible, por lo que es capaz, en principio, de romper cualquier algoritmo asimétrico tradicional de distribución de claves [Webf].

La información se cifra mediante clave privada y clave pública haciendo uso de algoritmos como RSA, *Elliptic Curve Digital Signature Algorithm* (ECDSA), *Elliptic Curve cryptography* (ECC) o *Digital Signature Algorithm* (DSA). Este tipo de cifrado está basado en números primos y en la premisa de que la descomposición de un número en factores primos no se puede llevar a cabo en un tiempo polinomial, premisa que no se cumpliría con la aparición de este algoritmo cuántico. Aunque nuevamente, debido a las limitaciones actuales, no se puede sacar el máximo rendimiento del algoritmo ya que se dispone de *qubits* limitados, por lo que actualmente bastaría con usar claves lo suficientemente extensas para no poder descifrarlas [Ket].

2.3.2. Algoritmo de Grover

El segundo algoritmo a explicar es el de Grover, propuesto por Lov Grover en 1996, el cual consigue encontrar un elemento concreto dentro de un conjunto desordenado de una forma rápida [Webh]. Proporciona un aumento cuadrático del tiempo medio en encontrarlo. Por ejemplo, en una base de datos genérica no estructurada con N elementos, una búsqueda necesitaría barrer linealmente un promedio de $N/2$ elementos para encontrar el elemento deseado, y, en el peor de los casos, necesitaría escanear los N elementos. Mientras que usando este algoritmo sólo sería necesario un promedio de $\sqrt{N}$ iteraciones.

Pese a que el aumento solo es cuadrático y no exponencial, como nos tiene acostumbrados la computación cuántica, para esta tarea, a efectos prácticos, es el mejor.

Es importante añadir que se trata de un algoritmo probabilístico con una alta probabilidad de éxito, y es posible repetirlo las veces que deseemos aumentando así su probabilidad. Para lograr encontrar el elemento concreto, este algoritmo realiza una simetría de todos los *qubits* respecto de la media, siendo la mayor variación de amplitud con respecto del resto de elementos la del elemento buscado [Ket].

Tiene muchas aplicaciones como en *Machine Learning* o en problemas NP los cuales son muy complejos de resolver para los ordenadores clásicos.

2.4. Principales ataques cuánticos

Una vez explicada la computación cuántica y los principales algoritmos que usa vamos a comentar dos de los ataques cuánticos más relevantes.

2.4.1. Ataque de canal lateral

Se trata de un ataque basado en información obtenida gracias a la propia implementación física o en el entorno de un sistema informático en lugar de basarse en puntos débiles del algoritmo implementado o de las estructuras matemáticas, como sería el caso de recurrir a criptoanálisis o explotar errores en el *software*. Estos ataques se aprovechan de los efectos secundarios del procesamiento de máquinas, como fugas de información, tiempo de ejecución y recursos computacionales utilizados en la ejecución de tareas a través de la irradiación de ondas de campo electromagnéticas, calor disipado o sonido. Esta es una categoría de ataque que se discute con frecuencia en el mundo de la criptografía post-cuántica. Los ataques de canal lateral representan una amenaza grave también para los protocolos tradicionales de *Quantum Key Distribution* (QKD), por ejemplo, esquemas similares a *Bennett and Brassard 1984* (BB84) [HW01]. Se pueden obtener mitigaciones parciales de esta amenaza mediante el llamado QKD independiente del dispositivo [N07].

2.4.2. Ataque de búsqueda de preimagen a múltiples objetivos

Una función *hash* criptográfica debe resistir los ataques a su preimagen. En el contexto del ataque, hay dos tipos de resistencia a la preimagen.

- Resistencia a la preimagen: para todas las salidas preespecificadas es computacionalmente inviable encontrar una entrada que produzca el mismo *hash* que otra.
- Resistencia de la segunda preimagen: para una entrada específica es computacionalmente inviable encontrar otra entrada que produzca la misma salida, es decir, dada x, es difícil encontrar una segunda entrada x' tal que $h(x) = h(x')$. También cabe destacar la resistencia de colisión, en la que no es computacionalmente factible encontrar dos entradas distintas x, x' que tengan la misma salida, es decir, tal que $h(x) = h(x')$.

Este algoritmo tiene como objetivo encontrar las claves *Advanced Encryption Standard* (AES) ejecutando pasos rápidos sobre una malla de procesadores con la finalidad de conseguir al menos una de esas claves.

Se trata de un desafío de seguridad para la criptografía simétrica, especialmente para AES-128.

Capítulo 3

Criptografía

3.1. Importancia de la criptografía

Según su definición, la criptografía se encarga del análisis y el estudio de las técnicas matemáticas aplicadas a la seguridad de la información y está enfocada a garantizar cuatro premisas principales que deben cumplirse [Sou21].

- La confidencialidad es el acceso controlado de la información de tal forma que solo los usuarios autorizados puedan acceder a ella.
- La integridad es la garantía ofrecida a los usuarios de que la información original no ha sido alterada, ya sea por modificación, adición o eliminación.
- La autenticación es la forma que tiene un usuario de presentarse, confirmando por medios comprobables que este es quien dice ser.
- Con la expresión no repudio se define la capacidad de afirmar la autoría de un mensaje o información, evitando que el autor niegue la existencia de su recepción o creación [CC].

El cifrado es el concepto primordial que rige la seguridad de la información moderna y que hoy en día es utilizado en aplicaciones como el protocolo de comunicación de internet *HyperText Transfer Protocol Secure* (HTTPS), el protocolo criptográfico de red de acceso remoto *Secure SHell* (SSH), *Public Key Infrastructure* (PKI), los métodos utilizados en tarjetas inteligentes, *Hardware Secure Modules* (HSM), criptomonedas, certificados digitales, algoritmos criptográficos como los basados en RSA, Diffie-Hellman, criptografía de curva elíptica y las soluciones de VPN, entre otras.

3.2. Tipos de criptografía

Como hemos mencionado anteriormente, la criptografía nos garantiza una comunicación segura entre dos o más partes a través de un canal no seguro. Para lograr esta finalidad se hacen uso de algoritmos criptográficos, que son los encargados de la gestión de las premisas a garantizar en criptografía [FC20].

Los algoritmos criptográficos pueden ser distribuidos en grupos más específicos según sus principios de funcionamiento, tales como: primitivas sin clave, primitivas de clave simétrica y primitivas de clave pública. Pueden evaluarse según su nivel de seguridad, su funcionalidad, sus métodos de operación, su desempeño y su aplicabilidad en cada situación [LMT21].

Podemos ver un esquema con los distintos tipos de criptografía:

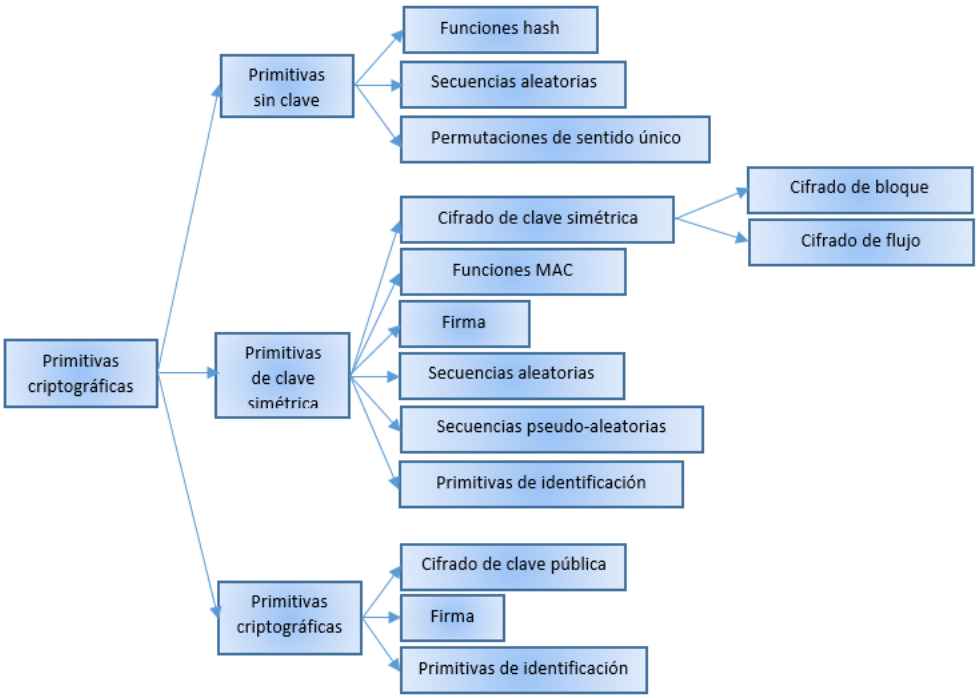

Figura 3.1: Primitivas criptográficas

Un conjunto de algoritmos forman un esquema criptográfico, donde las partes involucradas siguen siendo responsables de la gestión de las claves con las que interactúan los pares, y en consecuencia con los datos bajo encriptación disponibles públicamente. De esta forma, también podemos enumerar los siguientes procesos específicos que intervienen en las relaciones criptográficas.

- Establecimiento de claves: establece y divulga las claves a utilizar.
- Gestión de claves: protege las claves, las mantiene disponibles y las sustituye si fuera necesario.
- Derivación clave: utiliza una clave “ki” para calcular otra clave “kj” usando un token “t” disponible públicamente entre compañeros.

3.3. Funciones *hash*

Es una función matemática que transforma una entrada arbitraria de datos, llamada preimagen, en una nueva serie de caracteres con una longitud fija, llamada resumen.

Independientemente de la longitud de la preimagen, el resumen tendrá siempre la longitud deseada [Sou21]. Ofrece resistencia a la segunda preimagen, es decir, dado un mensaje "x", no es posible encontrar otro mensaje "y" que produzca el mismo resumen, y por tanto resistencia a colisiones, donde no es posible encontrar dos entradas que den lugar al mismo valor *hash*. Este bloque de caracteres es totalmente arbitrario y a partir de él no se puede sacar el contenido del bloque que hemos resumido.

Los algoritmos *hash* son de vital importancia en la firma digital garantizándonos la integridad, ya que si modificas un solo carácter en el texto original y realizas nuevamente la función de resumen, este se verá modificado radicalmente [Webi].

3.4. Criptografía de clave simétrica

Es un método criptográfico en el que ambas partes usan una misma clave para cifrar y descifrar los mensajes intercambiados. Han de acordar y compartir la clave de antemano. Una vez que ambas partes la tienen, el emisor la usa para cifrar un mensaje, lo envía al destinatario y este lo descifra con la misma clave [Taq10].

Figura 3.2: Critografía de clave simétrica

Es usada para cifrar grandes tamaños de datos. La parte más delicada suele encontrarse en el envío de la clave del emisor al receptor que se suele transmitir cifrada con criptografía de clave asimétrica.

3.5. Criptografía de clave asimétrica

Es un método criptográfico que usa un par de claves. Cada usuario tiene una clave pública que puede ser conocida por cualquier persona (de hecho son públicas en internet) y una clave privada que el propietario debe custodiar de modo que nadie tenga acceso a ella. Esta pareja de claves es complementaria, lo que cifra una solo lo puede descifrar la otra y viceversa [Webj]. Los métodos criptográficos garantizan que esta pareja de claves solo se genera vez, de modo que no es posible la obtención de la misma pareja de claves por dos usuarios. Dado que la clave pública y la clave privada están relacionadas matemáticamente, la fuerza de un criptosistema de clave pública depende del esfuerzo computacional (llamado "factor de trabajo" en criptografía) requerido para realizar un ataque de fuerza bruta de búsqueda de clave para encontrar la clave privada de su clave pública emparejada [Webk].

Ambas claves pueden ser usadas para cifrar y descifrar, por lo que este cifrado puede utilizarse con distintas finalidades; como garantizar la confidencialidad o la autenticación.

Para garantizar la confidencialidad el emisor cifra el mensaje con la clave pública del receptor, así solo este puede descifrarlo.

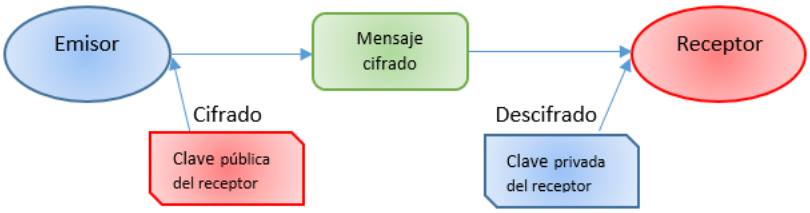

Figura 3.3: Criptografía de clave asimétrica (Confidencialidad)

Para garantizar la autenticación el emisor cifra el mensaje con su clave privada y el receptor lo descifra con la clave pública del emisor, comprobando así que dicho mensaje fue en efecto enviado por este.

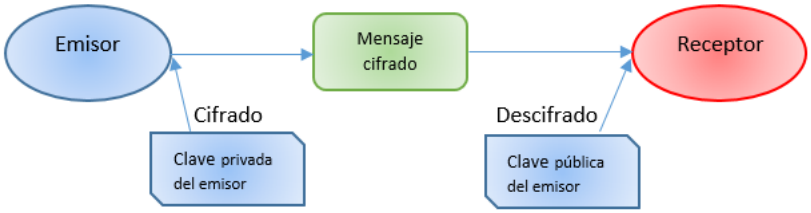

Figura 3.4: Criptografía de clave asimétrica (No repudio)

Estos dos métodos se pueden combinar obteniendo ambos beneficios.

3.6. Cifrado de flujo

Es un cifrado de clave simétrica en el que los dígitos del texto se combinan con un flujo de dígitos de cifrado pseudoaleatorio también llamado flujo de claves.

Cada dígito de texto de entrada se cifra uno a uno con el dígito correspondiente del flujo de claves, danto como resultado un dígito del flujo de texto cifrado. Para descifrarlo se aplica este procedimiento al revés hasta conseguir tener todo el mensaje descifrado [Webl] [FC20].

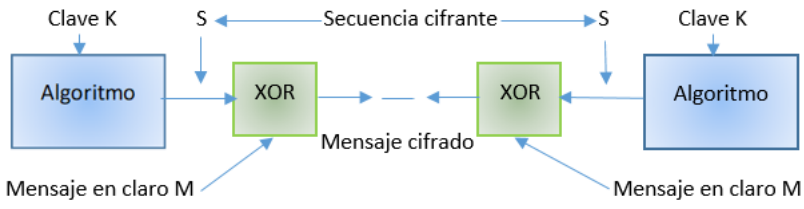

Figura 3.5: Cifrado de flujo

Capítulo 4

Criptografía Post-Cuántica

4.1. Importancia de la criptografía

Las principales técnicas de creación de criptosistemas de las que disponemos para resistir a ataques por ordenador cuántico hoy en día son: basadas en código, basadas en celosía, en isogenia de curvas elípticas supersingulares, basadas en *hash*, esquemas multivariantes e híbridas [Ber09a] A continuación vamos a detallar cada una de ellas.

4.2. Basadas en código

Son sistemas criptográficos que se basan en códigos de corrección de errores.

A grandes rasgos, el problema subyacente a este tipo de construcciones es el de decodificar una palabra codificada a través de un código lineal desconocido. Dicho código se describe a través de 3 parámetros: n y k, que hacen referencia a la longitud y dimensión del código y t, que es el número de errores que es posible corregir en una codificación errónea. Así, una matriz G de dimensión $(n \times k)$ sirve para generar palabras codificadas. Las palabras (vectores binarios de longitud k) se codifican como vectores de longitud n al multiplicarse por G. De la misma manera, si existe un error en la transmisión traducible en un vector acotado por t, existe un algoritmo de decodificacion asociado a G que permite recuperar la palabra original [Ber09b] [LMT21].

Los criptosistemas basados en código requieren de un gran tamaño por lo que se consideran no adecuados para dispositivos con recursos limitados debido a los grandes requisitos de memoria, grandes tamaños de clave pública y textos cifrados largos en comparación con la criptografía basada en celosía. Algunos esquemas de criptografía basados en código son el criptosistema McEliece y el de Niederreiter [FC20] [Buy21].

4.3. Basadas en celosía

Las celosías son estructuras matemáticas que se repiten infinitamente en el espacio. Tienen como característica principal el proceso de generación de claves aleatorias que requieren la intratabilidad de casos promedio o problemas difíciles. No necesita de un gran tamaño de memoria para la realización de operaciones rápidas. Por lo general,

los algoritmos basados en celosía también prefieren usar matrices y vectores en campos pequeños o anillos con parámetros de reducido tamaño, haciéndolos ideales para el campo de la inteligencia artificial donde los dispositivos son de espacio y recursos reducidos [LMT21].

La seguridad de *bit* de estos algoritmos varía de 46 a 128 con la viabilidad de su implementación en *Central Processing Unit* (CPU) de 8 a 32 bits y el tiempo de ejecución varía de 1,9 a 317,4 ms.

Algunos ejemplos de criptografía basada en celosía son: NTRUEncrypt, R-LWEenc, R-BIN-LWEenc y NewHope [FC20] [Buy21].

4.4. Isogenia de curvas elípticas supersingulares

Hace uso de las matemáticas de las curvas elípticas supersingulares para crear un intercambio de claves tipo Diffie-Hellman. Nos proporciona una sustitución post-cuántica para los métodos de intercambio de claves y curva elíptica. Funcionan de forma análoga a las implementaciones existentes de Diffie-Hellman ofreciendo un secreto perfecto hacia adelante evitando así la vigilancia de terceros y el compromiso de claves a largo plazo [Hig13].

El enfoque de intercambio de claves *Supersingular isogeny Diffie-Hellman* (SIDH) se basa en este tipo de gráficos y está protegido contra ataques criptoanalíticos de cualquier adversario. SIDH proporciona la viabilidad de tener un tamaño de clave pequeño y un nivel de seguridad cuántica de 128 *bits* [FC20] [Buy21].

4.5. Basadas en *hash*

Fueron ideadas por Ralph Merkle durante la década de 1970. Desde entonces se han estudiado como una posible sustitución a las firmas digitales teóricas de números como RSA y DSA [Buc11]. Su mayor defecto es que siempre existe un límite en el número de firmas que se pueden realizar utilizando un conjunto de claves privadas. Pese a ello, este tipo de firmas ofrece grandes alternativas en cuanto a la necesidad de criptografía post-cuántica [BO15].

Podemos destacar las firmas Lamport, el esquema de firma Merkle y los esquemas más nuevos.

4.6. Esquemas multivariantes

El esquema de criptografía polinomial multivariante utiliza operaciones aritméticas simples para protocolos y primitivas de seguridad. Se articula alrededor de problemas asociados a la resolución de sistemas de ecuaciones no lineales en varias variables sobre cuerpos finitos. Estas operaciones incluyen la suma y la multiplicación en pequeños campos finitos, lo que lo hace un candidato importante para garantizar la seguridad en dispositivos con recursos limitados, que incluyen tarjetas *Radio Frequency IDentification* (RFID), sensores, actuadores y tarjetas inteligentes. Por ejemplo, el esquema de firma multivariante da una firma corta, que es de unos pocos cientos *bits* de longitud. En comparación con

otros esquemas de criptografía post-cuántica donde el tamaño de la firma es mucho más grande.

Una opción interesante de esquema de firma multivariante es *Rainbow*, el cual puede proporcionar la base necesaria para una una firma digital post-cuántica segura [Din05].

4.7. Híbridas

Esquema criptográfico que surge de la combinación de los métodos anteriormente mostrados de tal forma que estos se complementen para ofrecer una mayor seguridad y rendimiento.

Es el tipo más usado, ya que como hemos mencionado, unos esquemas suplen los defectos de otros, por lo que es normal que se usen más de uno en conjunto para ofrecer mayores prestaciones.

Capítulo 5

Características VPN resistentes a la computación cuántica

5.1. Características de una VPN

Las VPNs son redes que funcionan bajo arquitectura de túneles criptográficos entre puntos de acceso autorizados, implementadas sobre una red pública o una privada, para garantizar el tráfico seguro de información entre puntos distintos. Pueden servir como mecanismos tanto para acceso remoto a través de Internet, como a redes corporativas a través de un punto ubicación y un proveedor de acceso, o también pueden servir para hacer una conexión de *Local Area Network* (LAN) distantes entre sí, prescindiendo de circuitos de transmisión de larga distancia o como una conexión directa entre computadoras en una red interna [Webm].

Tienen túneles como principio de funcionamiento. El protocolo de tunelización encapsula el paquete con un encabezado que contiene el enrutamiento que permite el tráfico de información entre los puntos del túnel y la ruta lógica a través del cual se transferirán los datos. Esto puede ocurrir en la capa de enlace o de red. En este tipo de redes las partes del túnel deben autenticarse y los mensajes que viajen por dicho túnel en ambas direcciones deben estar debidamente cifrados garantizando así una correcta confidencialidad e integridad de estos [Webq].

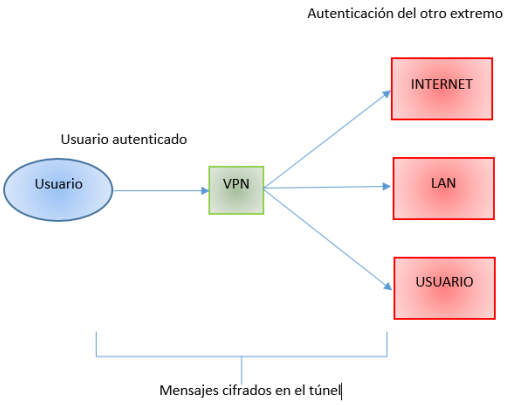

Figura 5.1: Esquema de funcionamiento de VPN

5.2. Algoritmos propuestos

El NIST es un instituto que tiene relevancia internacional en carácter económico y tecnológico. En 2016, enfocándose en el desarrollo de nuevas tecnologías en el campo de la criptografía post-cuántica, inició la evaluación de envíos de implementaciones de algoritmos criptográficos candidatos a ser considerados resistentes a la computación cuántica y estableció como objetivo del proceso de evaluación "desarrollar sistemas criptográficos que sean seguros contra computadoras cuánticas y clásicas y que puedan interoperar con los protocolos y redes de comunicación existentes" [NIS] El proceso de la presentación y evaluación de candidatos para algoritmos criptográficos post-cuánticos fue organizado en rondas, donde los candidatos se evaluaron en 3 rondas.

En este contexto, se establecieron las premisas criptográficas post-cuánticas a seguir por los candidatos, con el fin de garantizar la seguridad deseada en las soluciones de resistencia para aplicaciones basadas en criptografía de clave pública, así como definiciones de seguridad en cifrado simple con el establecimiento de claves utilizando claves efímeras y en firmas digitales. La definición más importante hecha por el NIST para el alcance de este trabajo es la de las categorías de la fuerza de la seguridad para ser objetivo de los algoritmos criptográficos candidatos de acuerdo con los propósitos específicos de cada algoritmo. Tales categorías se definieron en intervalos de fuerza basados en criterios conocidos y establecidos para el cifrado simétrico, considerado significativamente resistente al criptoanálisis cuántico [NIS]. El NIST considera que los ataques cuánticos se pueden medir en términos de la profundidad de circuito o en tiempo de ejecución configurando un parámetro llamado MAXDEPTH, que es considerado relevante en un escenario de ejecución de series computacionales lo suficientemente grandes. Se usa como una estimación para el recuento de puertas lógicas clásicas y cuánticas necesarias para ataques exitosos de recuperación de claves y ataques de colisión.

También se establecen otros principios cualitativos que se deben tener en cuenta en el desarrollo de soluciones post-cuánticas.

- *Perfect Forward Secrecy*: se debe hacer una evaluación precisa del uso de algoritmos que garanticen la *Sequential Linear Programming* (SLP) dado que su uso puede agregar otros costes adicionales, como el uso de algoritmos de generación de claves muy lento, como en RSA, que deben evitarse. Por lo tanto, es necesario considerar la diferencia significativa entre el coste y la seguridad práctica de un algoritmo en cuanto a SLP.

- Resistencia a ataques de canal lateral: como ya hemos explicado con anterioridad, son ataques que se aprovechan de los efectos secundarios del procesamiento de máquinas, como fugas de información, tiempo de ejecución y recursos computacionales utilizados en la ejecución de tareas a través de la irradiación de ondas de campo electromagnéticas, calor disipado o sonido. El NIST nos informa de que los algoritmos candidatos deben tratar de resistir estos ataques al menor coste posible para que el rendimiento del algoritmo no se vea afectado.

- Resistencia a ataques multi-tecla: deben buscar formas de evitar que el atacante pueda realizar ataques usando varias claves a la vez con el objetivo de comprometer a muchas claves o tan solo una de ellas.

- Resistencia a las Malas Prácticas (Uso Indebido): los esquemas criptográficos deben ser lo suficientemente fuertes para evitar fallos en sus ejecuciones, ya sea por errores

en la codificación de los algoritmos, mal funcionamiento de los generadores de números aleatorios, reutilización de nonces o reutilización de pares de claves en cifrado con claves efímeras, así como su utilización de forma no adecuada [NIS14].

Capítulo 6

Análisis de cambios propuestos a VPN Wireguard

6.1. WireGuard

WireGuard es una VPN simple, rápida y moderna que utiliza criptografía de última generación. Es una VPN de propósito general con un alto grado de compatibilidad y adaptación. Inicialmente fue lanzado para el kernel de Linux, ahora es multiplataforma (Windows, *Macintosh Operating System* (macOS), *Berkeley Software Distribution* (BSD), *iPhone operating system* (IOS), Android). Se encuentra bajo un proceso de desarrollo y mejora continuo y hoy en día podría considerarse como la solución VPN más segura, fácil de usar y simple. Pretende ser tan fácil de configurar e implementar como SSH. Una conexión VPN se realiza simplemente intercambiando claves públicas, exactamente como intercambiar claves SSH y WireGuard maneja todo lo demás de manera transparente. Incluso es capaz de hacer roaming entre direcciones *Internet Protocol* (IP), al igual que Mosh. No hay necesidad de administrar conexiones, preocuparse por el estado, administrar demonios o preocuparse del funcionamiento interno. Puede ser fácilmente implementado en muy pocas líneas de código (unas 4000) y auditable de una manera rápida y sencilla estando enfocado a ser revisado de manera integral por una sola persona.

En cuanto a seguridad utiliza criptografía de última generación, como el marco del protocolo Noise, Curve25519, ChaCha20, Poly1305, blake2s, SipHash24, *Hash based Key Derivation Function* (HKDF) y construcciones seguras y confiables. Toma decisiones conservadoras y razonables y ha sido revisado por criptógrafos. Gracias a la eficiencia y rendimiento de sus algoritmos y a la adición WireGuard al kernel de Linux, este ofrece redes seguras de alta velocidad. Puede ser implementado en pequeños dispositivos integrados y teléfonos inteligentes y enrutadores de red troncal completamente cargados [Wir].

6.2. Algoritmos usados en WireGuard

6.2.1. Blake2s-256

Es una función *hash* criptográfica que forma parte de la familia de funciones blake2 [Aum]. Esta familia es una mejora respecto a su antecesora blake y cuenta con mecanismos *hash* rápidos y altamente seguros, que proporcionan una seguridad similar a la de familia

SHA3 [GB11] y que ofrece resultados compilados de tamaños de 160 a 512 *bits*. Esta función puede ser utilizada por protocolos de firma digital, autenticación y mecanismos de protección de la integridad, así como para aplicaciones en PKI, protocolos de comunicación seguros y almacenamiento en la nube entre otros, estando además optimizado para plataformas de 8 *bits* a 32 *bits*, dando como resultado archivos compilados de tamaños entre 1 y 32 *bytes* [GB11]. La función *hash* criptográfica blake2 en su variación blake2s está optimizada para plataformas de 8 *bits* a 32 *bits* y puede producir resultados compilados entre 1 y 32 *bytes* de tamaño. Los parámetros adoptados en esta variación se enumeran en la siguiente tabla.

Parámetro	**Representación**	**Valor**
***Bits* por palabra**	w	32
Rondas en F	r	10
***Bytes* por bloque**	bb	64
Bytes* del *hash	nn	1-32
***Bytes* de la clave**	kk	0 - 32
***Bytes* de entrada**	11	0 - 2^{64}
Constantes de rotación de G	R1, R2, R3, R4	16, 12, 8, 7

Tabla 6.1: Parámetros de blake2s-256

Sus fundamentos de procesamiento se basan en dos funciones básicas: una de mezcla G y una de compresión F, tal como se define en su *Remote Function Call* (RFC) [Aum] La función de mezcla G toma dos palabras x e y y las baraja entre cuatro palabras de índices a, b, c y d en un vector funcional v de 16 posiciones, de modo que la función devuelve el vector modificado con su debidas rotaciones, como se puede ver en el siguiente pseudocódigo [Aum].

Algoritmo 1: Función de barajeo de blake2s

function G$(v[0..15], a, b, c, d, x, y)$
$v[a] := (v[a] + v[b] + x) \bmod 2^w$
$v[d] := (v[d] \wedge v[a]) >>> R1$
$v[c] := (v[c] + v[d]) \bmod 2^w$
$v[b] := (v[b] \wedge v[c]) >>> R2$
$v[a] := (v[a] + v[b] + y) \bmod 2^w$
$v[d] := (v[d] + v[a]) >>> R3$
$v[c] := (v[c] + v[d]) \bmod 2^w$
$v[b] := (v[b] + v[c]) >>> R4$
return $v[0..15]$
end function

A continuación, una función de compresión F, que tiene como parámetros un vector de estados h, un vector de bloque de mensaje m, un contador de compensación con tamaño 2w *bits* y un indicador de fin de bloque f; toma el vector local v resultante de la función de mezcla y lo usa en un procesamiento de 10 rondas para el caso de blake2s como se muestra a continuación en pseudocódigo [Aum].

Algoritmo 2: Función de compresión de blake2s

function F$(h[0..7], m[0..15], t, f)$
  $v[0...7] := h[0..7]$
  $v[8..15] := IV[0..7]$
  $v[12] := v[12] \wedge (t \bmod 2^w)$
  $v[13] := v[13] \wedge (t >> w)$
If $(f = TRUE)$
  $v[14] := v[14] \wedge 0xFF..FF$
EndIf
For $(i = 0$ to $r - 1$ $)$
  $s[0..15] := SIGMA[imod10][0..15]$
  $v := G(v, 0, 4, 8, 12, m[s[0]], m[s[1]])$
  $v := G(v, 1, 5, 9, 13, m[s[2]], m[s[3]])$
  $v := G(v, 2, 6, 10, 14, m[s[4]], m[s[5]])$
  $v := G(v, 3, 7, 11, 15, m[s[6]], m[s[7]])$
  $v := G(v, 0, 5, 10, 15, m[s[8]], m[s[9]])$
  $v := G(v, 1, 6, 11, 12, m[s[10]], m[s[11]])$
  $v := G(v, 2, 7, 8, 13, m[s[12]], m[s[13]])$
  $v := G(v, 3, 4, 9, 14, m[s[14]], m[s[15]])$
EndFor
For$(i = 0$ to $7)$
  $h[i] := h[i] \wedge v[i] \wedge v[i + 8]$
EndFor
  return $h[0..7]$
end function

Se describen el relleno de datos y el cálculo de compilación de blake2s completamente en el RFC 7693 [Aum].

6.2.2. ChaCha20

ChaCha20 es un cifrado de flujo diseñado por D. J. Bernstein. Refina el algoritmo Salsa20, y utiliza una clave de 256 *bits*. El nombre del algoritmo es "ChaCha". "ChaCha20" es una instancia específica donde se utilizan 20 "rondas" (u 80 cuartos de ronda). Existen otras variantes, con 8 o 12 vueltas, pero en este solo trataremos el ChaCha de 20 rondas ya que es el usado por WireGuard.

Se trata de un cifrado de alta velocidad considerablemente más rápido que AES, haciéndolo alrededor de tres veces más eficiente en plataformas que carecen de hardware AES especializado.

La operación básica del algoritmo ChaCha20 es el cuarto de vuelta, la cual opera en cuatro enteros sin signo de 32 *bits*, denotados a, b, c y d. La operación en notación tipo C es la siguiente [Lan] :

Algoritmo 3: Operación de cuarto de vuelta de ChaCha20

1: $a + b; d \wedge = a; d <<<= 16;$
2: $c+ = re; b \wedge = c; b <<<= 12;$
3: $a+ = b; d \wedge = a; d <<<= 8;$
4: $c+ = re; b \wedge = c; b <<<= 7;$

Donde "+" denota módulo de suma de enteros 2^{32}, "∧" denota *bit* a *bit* OR exclusivo (*Exclusive Or* (XOR)), y "<<< n" denota una rotación a la izquierda de n *bits* (hacia los *bits* altos).

El estado de ChaCha20 no tiene 4 números enteros, tiene 16. La operación de cuarto de vuelta funciona solo en cuatro de ellos, por lo que cada cuarto de ronda se opera en cuatro números predeterminados en el estado ChaCha20 los cuales denotaremos como parámetros de la función en CUARTERROUND(x,y,z,w). La función de bloque ChaCha20 transforma un estado ejecutando varios cuartos de vuelta. Sus entradas son:

- Una clave de 256 *bits*, tratada como una concatenación de ocho claves de 32 *bits* de enteros representados en *little-endian*.
- Un *nonce* de 96 *bits*, tratado como una concatenación de tres *bits* de 32 enteros en *little-endian*.
- Un parámetro de conteo de bloques de 32 *bits*, tratado como un *little-endian* de 32 *bits* entero.
- Una salida es de 64 *bytes* de aspecto aleatorio.

ChaCha20 ejecuta 20 rondas, alternando entre "rondas columna" y "rondas diagonales". Cada ronda consta de cuatro cuartos de ronda, y se ejecutan de la siguiente manera: Los cuartos de vuelta 1-4 son parte de una "ronda columna", mientras que 5-8 son parte de una "ronda diagonal".

Algoritmo 4: Rondas de ChaCha

```
1: CUARTO DE RONDA (0, 4, 8, 12)
2: CUARTO DE RONDA (1, 5, 9, 13)
3: CUARTO DE TERRENO (2, 6, 10, 14)
4: CUARTO DE TERRENO (3, 7, 11, 15)
5: CUARTO DE RONDA (0, 5, 10, 15)
6: CUARTO DE RONDA (1, 6, 11, 12)
7: CUARTO DE TERRENO (2, 7, 8, 13)
8: CUARTO DE TERRENO (3, 4, 9, 14)
```

En cuanto a cifrado se refiere, ChaCha20 llama sucesivamente a la función de bloque Cha-Cha20, con la misma clave y *nonce*, y con un contador de parámetros de bloque que aumenta sucesivamente para luego serializar el estado resultante escribiendo los números en orden *little-endian*, creando un bloque de flujo de claves. La concatenación de los bloques sucesivos forma un flujo de claves. Después, la función ChaCha20 realiza un XOR de este flujo de claves con el texto sin formato. Alternativamente, cada bloque de flujo de claves puede ser XORed con un bloque de texto sin formato antes de proceder a crear el siguiente bloque, ahorrando algo de memoria. No hay ningún requisito para el texto sin formato como un múltiplo entero de 512 *bits*. Si hubiera *extra keystream* del último bloque, este se descartaría. Los protocolos específicos pueden requerir que el texto sin formato y el texto cifrado tengan cierta longitud. Dichos protocolos deben especificar cómo se rellena el texto sin formato.

Las entradas a ChaCha20 son:

- Una clave de 256 *bits*.
- Un contador inicial de 32 *bits* el cual se puede establecer en cualquier número, pero suele ser 0 o 1. Tiene sentido usar 1 si usamos el bloque 0 para otra cosa, como generar una única vez la clave de autenticación como parte de un algoritmo *Authenticated Encryption with Associated Data* (AEAD).
- Un *nonce* de 96 *bits*. En algunos protocolos, esto se conoce como *Initialization Vector* (IV)
- Un texto sin formato de longitud arbitraria.

La salida es un mensaje encriptado, o texto cifrado, de la misma longitud.

El descifrado se realiza de la misma manera. La función de bloque ChaCha20 se utiliza para expandir la clave en un flujo de claves, que se XORed con el texto cifrado devolviendo el texto plano.

Algoritmo de cifrado ChaCha20 en pseudocódigo [Lan].

Algoritmo 5: Algoritmo de cifrado de ChaCha

[H]
function CHACHA20ENCRYPT($key, counter, nonce, plaintext$)
For ($j = 0$ upto floor $(len(plaintext)/64) - 1$)
$key\_stream = chacha20\_block(key, counter + j, nonce)$
$block = plaintext[(j \times 64)...(j \times 64 + 63)]$
$encrypted\_message+ = block \wedge key\_stream$
EndFor
If $(len(plaintext)\ \%64)! = 0)$
$j = floor(len(plaintext)/64)$
$key\_stream = chacha20\_block(key, counter + j, nonce)$
$block = plaintext[(j \times 64)...len(plaintext) - 1]$
$encrypted\_message+ = (block \wedge key\_stream)[0..len(plaintext)\ \%64]$
EndIf
return *encrypted_message*
end function

Se describen más aspectos sobre ChaCha20 así como ejemplos de uso en el RFC 7539 [Lan].

6.2.3. POLY1305-AES

Es un autenticador de una sola vez diseñado por D. J. Bernstein. Este toma una clave única de 32 *bytes* y un mensaje y produce una etiqueta de 16 *bytes* que se utiliza para autenticar el mensaje. Independientemente de cómo se genere la clave, esta se divide en dos partes, llamadas r y s. El par (r, s) debe ser único e impredecible para cada invocación (es por eso que ha sido obtenido originalmente mediante el cifrado de un *nonce*), mientras que r puede ser constante, pero debe modificarse de la siguiente manera antes de usarse, (r se trata como un número *little-endian* de 16 octetos):

- r[3], r[7], r[11] y r[15] deben tener sus cuatro primeros *bits* claros (ser más pequeños que 16)

- r[4], r[8] y r[12] deben tener sus dos *bits* inferiores claros (ser divisible por 4)

El siguiente código muestra la fijación de r para que sea apropiada.

Algoritmo 6: Algoritmo de fijación de r en poly

```
#include "poly1305aes_test.h"
function POLY1305AES_TEST_CLAMP( unsigned char r[16])
    r[3] = 15;
    r[7] = 15;
    r[11] = 15;
    r[15] = 15;
    r[4] = 252;
    r[8] = 252;
    r[12] = 252;
end function
```

La s debería ser impredecible, pero es perfectamente aceptable para generar r y s de forma única cada vez. Ya que cada uno de ellos es de 128 *bits*, generarlos pseudoaleatoriamente también es aceptable. Las entradas a Poly1305 son una clave de un solo uso de 256 *bits* y un mensaje de longitud arbitraria. La salida es una etiqueta de 128 *bits*.

Primero, el valor r debe fijarse. A continuación se debe establecer la constante prima P en 2^{130-5}: 3ffffffffffffffffffffffffffffffb. También se establece una variable *acumulador* a cero. Después se divide el mensaje en bloques de 16 *bytes*. El último podría ser más corto.

- Se lee el bloque como un número *little-endian*.

- Se añade un *bit* más allá del número de octetos. Para un bloque de 16 *bytes*, esto es equivalente a sumar 2^{128} al número, para los bloques más cortos, puede ser 2^{120}, 2^{112}, o cualquier potencia de dos que sea uniformemente divisible por 8 hasta 2^8 .

- Si el bloque no tiene 17 *bytes* (el último bloque), se rellena con ceros. Esto no tiene sentido si se están tratando los bloques como números.

- Se agrega este número al acumulador.

- Se multiplica por r.

- Se pone el acumulador en el resultado módulo p. Para resumir: Cuenta=((Acc+bloquear)*r) % pág.

Finalmente, el valor de la clave secreta s se suma al acumulador, y los 128 *bits* menos significativos se serializan en *little-endian* en orden para formar la etiqueta.

Algoritmo 7: Algoritmo de cifrado de POLY1305-AES

$clamp(r) : r = 0x0ffffffc0ffffffc0ffffffc0fffffff$
function POLY1305_MAC(msg, key)
  $r = (le\_bytes\_to\_num(key[0..15])$
  $clamp(r)$
  $s = le\_num(key[16..31])$
  $accumulator = 0$
  $p = (1 \ll 130) - 5$
For ($for\ i = 1$ upto ceil(msg length in $bytes/16$)
  $n = le\_bytes\_to\_num(msg[((i-1) \times 16)..(i \times 16)] \mid [0x01])$
  $a+ = n$
  $a = (r \times a)\ \%p$
EndFor
  $a+ = s$
  return $num\_to\_16\_le\_bytes(a)$
end function

6.2.4. CURVE25519

El algoritmo Curve25519 se basa en una función Curve-Elliptic-Diffie-Hellman de intercambio de claves de última generación [DJBCNDHSRIMYM06], y se distingue por tener un gran desempeño en velocidad de procesamiento y una fuerte seguridad [Berb]. Provee de un nivel de seguridad de 128 *bits* con una longitud de clave de 256 *bits* [Bera]. La curva utilizada $y^2 = x^3 + 486662x^2 + x$ es una curva de Montgomery, sobre el campo primo definido por el número primo $2^{255} - 19$, y usa el punto base x=9 . Este punto genera un subgrupo cíclico cuyo orden es el primo $2^{252} + 27742317777372353535851937790883648493$ y es del índice 8. Utiliza un punto elíptico comprimido (sólo coordenadas X), lo que permite un uso eficiente de la escalera de Montgomery para *Elliptic Curve Diffie-Hellman* (ECDH), utilizando sólo coordenadas XZ.

Está enfocada a evitar problemas de implementación. Es inmune a los ataques de tiempo, acepta cualquier cadena de 32 *bytes* como clave pública y no necesita validar que un punto dado pertenezca a la curva o que sea generado por el punto base. En la generación de claves bajo la función Curve25519, para una explicación de alto nivel, tenemos el siguiente mecanismo: cada parte en un proceso de autenticación y cifrado entre dos partes tiene una clave pública y una privada, ambas de 32 *bytes* de largo y ambas partes intercambian un secreto también de 32 *bytes* [Ber06]. Se utiliza un *hash* del secreto Curve25519(a, Curve25519(b,9)) como clave para autenticación de mensajes en un sistema basado en claves privadas, o para sistemas basados en autenticación y encriptación simultáneamente [Ber06].

6.2.5. SIPHASH24

SipHash es una familia de *Pseudo-Random Functions* (PRF) optimizadas para velocidad en mensajes cortos. El código C de referencia de SipHash es portátil, simple, optimizado para claridad y depuración [Webo]. Fue diseñado en 2012 por Jean-Philippe Aumasson y Daniel J. Bernstein como una defensa contra los ataques *Denial of Service* (DoS) de inundación de *hash*.

Algunas características de SipHash son:

- Es más simple y rápido en mensajes cortos que los otros algoritmos criptográficos, como los *Macintosh* (MAC) basados en hashing universal.
- Nos ofrece rendimiento competitivo con algoritmos no criptográficos inseguros, como fhhash.
- Es criptográficamente seguro, sin signos de debilidad a pesar de los múltiples proyectos de criptoanálisis de los principales criptógrafos.
- Pruebas realistas, con integración exitosa en sistemas operativos (Linux kernel, OpenBSD, FreeBSD), lenguajes (Perl, Python, Ruby, etc.), bibliotecas (OpenSSL li-bcrypto, Sodium, etc.) y aplicaciones (Wireguard, Redis, etc.).

Como función pseudoaleatoria segura (también conocida como función *hash* con clave), SipHash también se puede usar como un código de autenticación de mensajes (MAC) seguros. Pero SipHash no es un *hash* en el sentido de una función *hash* sin clave de propósito general como blake3 o SHA-3. Por lo tanto, SipHash siempre debe usarse con una clave secreta para estar seguro.

El modo de funcionamiento predeterminado es SipHash-2-4, toma una clave de 128 *bits*, realiza 2 rondas de compresión, 4 rondas de finalización y devuelve una etiqueta de 64 *bits*. Se espera que (Half) SipHash-c-d con $c \geq 2$ y $d \geq 4$ proporcione la máxima seguridad PRF para cualquier función con la misma clave y tamaño de salida. El objetivo de seguridad PRF estándar permite que el atacante acceda a la salida de SipHash en los mensajes elegidos de forma adaptativa por el atacante. La seguridad está limitada por el tamaño de la clave (128 *bits*). Los atacantes que buscan claves 2^s tienen una probabilidad de 2^{s128} de encontrar la clave SipHash. La seguridad también está limitada por el tamaño de salida. En particular, cuando se usa SipHash como MAC, un atacante que prueba a ciegas las etiquetas 2^s tendrá éxito con una probabilidad de 2^{s-t}, si t es el tamaño de *bits* de esa etiqueta [Webo].

6.2.6. HKDF

Una función de derivación clave (*Key Derivation Function* (KDF)) es un componente básico y esencial de sistemas criptográficos. Su objetivo es tomar alguna fuente de información inicial del material de clave y derivar de él una o más claves secretas criptográficas fuertes. Está diseñado para usarse en una amplia variedad de aplicaciones KDF, beneficiándose de la simplicidad y la naturaleza polivalente de HKDF, así como de su fundamento analítico [Webn].

Se prevé que algunas aplicaciones no sean capaces de usar HKDF debido a requisitos operativos específicos o podrán usarlo pero sin todos los beneficios del esquema. Un ejemplo significativo es la derivación de criptografía de claves de una fuente de baja entropía, como la contraseña de un usuario. El paso de extracción en HKDF puede concentrar la entropía existente pero no puede amplificar la entropía. En el caso de los KDF basados en contraseña, un objetivo principal es ralentizar los ataques de diccionario usando dos ingredientes, un valor de sal, y la ralentización intencional del cálculo de derivación de claves. HKDF se adapta naturalmente al uso de sal; sin embargo, una desaceleración en el mecanismo no forma parte de esta especificación [Kal00].

A pesar de la simplicidad de HKDF, hay muchas medidas de seguridad que se han tenido en cuenta en el diseño y análisis de esta construcción.

6.3. Implementación Post-Cuántica a WireGuard

Una vez introducidos y explicados los algoritmos criptográficos de WireGuard, vamos a analizarlos evaluando su seguridad y viendo si estos serían resistentes a un escenario post-cuántico tanto con respecto a las premisas impuestas por el NIST como con premisas básicas externas.

6.4. CURVE25519

Es una curva elíptica que proporciona seguridad de 128 *bits* con una clave de 256 *bits*, diseñada para el esquema de acuerdo clave de ECDH. Es una de las curvas ECC más rápidas [Berc]. Demuestra ser resistente a los ataques clásicos, tales como ataques por los métodos de *kangaroo* o rho de Pollard, que calcula algoritmos discretos contenidos en un intervalo [a,b] en grupos de ciclos arbitrarios.

Se estima que un atacante, utilizando los métodos *kangaroo* o rho de Pollard en máquinas clásicas paralelizadas tendrían posibilidades de éxito al calcular el logaritmo discreto de 251 *bits* de una ECDH Curve25519 de aproximadamente 2^{-90}, que caracteriza una posibilidad muy baja a muy alto coste. Así podemos ver que el algoritmo permanece seguro con el uso de claves de más de 251 *bits*, en términos de coste y tiempo para romperlo. Sin embargo, si el atacante poseyera una computadora cuántica universal lo suficientemente grande, equipada con una implementación del algoritmo de Shor, entonces este análisis colapsaría dramáticamente, ya que rompería fácilmente todos los sistemas mencionados, incluido ECC. Si bien no está claro si dicho dispositivo será técnicamente realizable algún día, las versiones en miniatura han demostrado que el principio funciona. Se estima que 1600 *qubits* serían suficientes para romper Curve25519, mientras que se necesitan 6147 *qubits* para romper RSA-3072.

Este escenario post-cuántico, con el descubrimiento de la isogenia de curvas supersingulares y su resistencia ante esta computación, han llevado a diseñar un protocolo de intercambio de claves post-cuántico que lleva el nombre de SIDH. Concretamente el protocolo funciona de la siguiente manera: dos usuarios acuerdan un grupo de curvas y una curva inicial E. El primero de ellos aplica una secuencia aleatoria secreta de dos isogenias, lo que da como resultado una curva elíptica E A. De manera similar, el segundo recorre en secreto un rastro aleatorio de tres isogenias para terminar en una curva E B. Después de intercambiar E A y E B públicamente, el primer usuario vuelve a aplicar su secuencia de dos isogenias a partir de la curva E B del segundo, y este hace lo mismo a partir de la curva E A del primero. Con esto, ambos usuarios, eventualmente terminan con la misma curva elíptica E AB. Este es el secreto común listo para usar en futuras comunicaciones, por ejemplo, para crear una clave AES.

Entre sus características están los parámetros del sistema bastante pequeños, con claves públicas que se pueden forzar muy por debajo de los 3072 *bits*. Además, la idea subyacente es versátil y se puede adoptar para crear protocolos para pruebas de identidad de conocimiento cero, para el cifrado de claves públicas y para firmas digitales.

Un candidato del NIST en tercera ronda es *Supersingular Isogeny Key Encapsulation* (SIKE), el cual hace uso de este tipo de isogenia de curvas y que explicaremos a continuación como posible alternativa.

6.4.1. SIKE

Es una suite de encapsulación de claves basada en isogenia de caminatas pseudoaleatorias en gráficos de isogenia supersingulares, que se envió al proceso de estandarización NIST en criptografía post-cuántica. Contiene dos algoritmos:

- Un algoritmo de cifrado de clave pública seguro de *Chosen Plaintext Attack* (CPA) SIKE.PKE.
- Un mecanismo de encapsulación de claves seguro *Chosen Ciphertext Attack* (CCA) SIKE.KEM, cada uno instanciado con cuatro conjuntos de parámetros: SIKEp434, SIKEp503, SIKEp610 y SIKEp751.

Podría contemplarse como una alternativa post-cuántica a la implementación de WireGuard.

6.5. CHACHA20 Y POLY1305

WireGuard utiliza el algoritmo ChaCha20-Poly1305, es decir, una combinación de ambos algoritmos en un algoritmo de cifrado autenticado con datos asociados donde se usa Cha-Cha20 para el cifrado simétrico y Poly1305 para la autenticación.

Las entradas a este algoritmo son una clave de 256 *bits*, un *nonce* de 96 *bits* (que es diferente para cada uso de la misma clave), una parte de tamaño arbitrario de texto sin formato y una *Additional Authentication Data* (AAD) de tamaño arbitrario. ChaCha20 es un cifrado de bloque que tiene la seguridad al nivel de un cifrado de bloque de 256 *bits*, cumpliendo el criterio de resistencia cuántica de Nivel V según los criterios establecidos por el NIST, que en teoría proporciona una alta seguridad contra los ataques de búsqueda de claves, es inmune a los ataques Padding-Oracle, tales como el Lucky13, que afecta el modo *Cipher-Block Chaining* (CBC) que se usa en TLS. También es resistente a los ataques de sincronización. Para una discusión sobre cómo romper el algoritmo ChaCha20 mediante un ataque forzado *raw by key lookup*, usando por ejemplo una computadora cuántica, es relevante mencionar que, al menos en un enfoque clásico, es imposible lograr el rendimiento de máquinas paralelas con precio razonable. Por el enfoque de la resistencia a los ataques de canales laterales, Bernstein enumera que "Los ataques de *timing* contra Salsa20 son por lo tanto tan difíciles como el puro criptoanálisis de los resultados de Salsa20. Las operaciones en Salsa20 también se encuentran entre las más fáciles de proteger contra ataques de energía y otros ataques de canal lateral".

El algoritmo MAC Poly1305 utilizado en la autenticación de datos garantiza que la única forma de romper el cifrado es romper el algoritmo AES involucrado en el cifrado, que es AES de 128 *bits*, lo que le otorga individualmente al cifrado una fuerza de Nivel I. Además, según el RFC del algoritmo ChaCha20-Poly1305, Poly1305 rechaza mensajes falsificados con una probabilidad de 1-(n2102) para un mensaje de tamaño $16n$ *bytes*, incluso enviando 264 mensajes auténticos, lo que hace que el algoritmo sea altamente

resistente a ataques de mensaje elegido. Por lo tanto no sería necesaria su modificación o sustitución en WireGuard en vista de un escenario post-cuántico. Aun así podríamos aumentar la seguridad de este algoritmo con el uso de XChaCha20.

6.5.1. XChaCha20

Una mejora respecto de este algoritmo es el uso de XChaCha20, una variante de ChaCha20 con un *nonce* extendido, lo que permite que los *nonce*s aleatorios sean seguros. No requiere ninguna tabla de búsqueda y evita la posibilidad de ataques de tiempo. Internamente, XChaCha20 funciona como un cifrado de bloque utilizado en modo contador. Utiliza la función *hash* HChaCha20 para derivar una subclave y un *subnonce* de la clave original y un *nonce* extendido, y un contador de bloques de 64 *bits* dedicado para evitar incrementar el *nonce* después de cada bloque. Generalmente se recomienda sobre ChaCha20 simple debido a su tamaño de *nonce* extendido y su rendimiento comparable por lo que, pese a estar seguros usando Cha-Cha20, podríamos implementar esta simple alternativa mejorando la seguridad sin afectar notablemente al rendimiento y, al ser una mejora de la misma familia, su implementación sería sencilla [Webp].

6.6. Blake2s-256

Los usos de la función *hash* de blake2s en WireGuard están en modo de 256 *bits* de salida. Esta función también se utiliza para HKDF, como un dispositivo de simulación de aleatoriedad, es decir, como una función pseudoaleatoria, que es el tipo de mecanismo más cercano para simular la aleatoriedad en la computación clásica. Esta implementación en arquitectura de 32 *bits* y también con salida de 32 *bits* ofrece una seguridad de colisión de 2^{128}. De esta forma, no alcanza ninguno de los niveles de resistencia cuántica estipulada por NIST, donde el criterio mínimo para la resistencia de una función *hash* es 2^{256}, ofrecido por ejemplo por el cifrado SHA-512. Por tanto, este algoritmo no sería resistente a un ataque realizado por computadoras cuánticas según los criterios del NIST, por lo que habría que sustituirlo.

La familia blake2 cuenta con el modo blake2b512 que ofrece una seguridad de choque de 2^{256}, que sería una solución viable en un análisis sin tener en cuenta las implicaciones sobre el rendimiento de WireGuard utilizando este algoritmo. Otra opción es el uso de uno de los algoritmos propuestos en el concurso del NIST, se trata de Falcon, un algoritmo de firma basado en celosías y que es resistente a la computación clásica y cuántica.

A continuación se evaluarán ambas alternativas en motivos de seguridad y compatibilidad con WireGuard para ofrecer una solución viable.

6.6.1. Blake2b-512

Una posible alternativa al uso de blake2s-256 es el uso de otro algoritmo de su propia familia: blake2b-512 el cual nos brinda una seguridad ante colisiones de 2^{256}, seguridad considerada post-cuántica según el NIST, frente a la de 2^{128} que nos ofrecía blake2s-256. Al ser de la misma familia su sustitución sería sencilla ya que comparten gran parte de las funciones, teniendo tan solo que modificar algunos parámetros para que el número de variables y el tamaño de entrada y salida concuerden. Antes de comenzar a explicar la

posible implementación de esta propuesta, vamos a ver sus características respecto del algoritmo actual de WireGuard.

Blake2s está optimizado para plataformas de 8 a 32 *bits* y produce resúmenes de cualquier tamaño entre 1 y 32 *bytes*, mientras que blake2b (o simplemente blake2) está optimizado para plataformas de 64 *bits* y produce resúmenes de cualquier tamaño entre 1 y 64 *bytes*. Se cree que tanto blake2b como blake2s son altamente seguros y funcionan bien en cualquier plataforma, software o hardware. blake2 no requiere una construcción especial "*Hash-Based Message Authentication Code* (HMAC)" para la autenticación de mensajes con clave, ya que tiene una clave incorporada mecanismo [Aum].

		blake2b-512	**blake2s-256**
***Bits* en palabra**	w	64	32
Rondas en F	r	12	10
***Bytes* por bloque**	bb	128	64
Bytes* del *hash	nn	1 - 64	1 - 32
***Bytes* de la clave**	kk	0 - 64	0 - 32
***Bytes* de entrada**	11	0 - 2^{128}	0 - 2^{64}
Constantes de rotación de G	R1, R2, R3, R4	32, 24, 16, 63	16, 12, 8, 7
Rondas	Rondas	12	10

Tabla 6.2: Comparación de parámetros de blake2b-512 y blake2s-256

El IV: blake2b = sha-512, blake2s = sha-256

Enfocándonos en las permutaciones de palabras de mensaje para cada ronda, blake2s tiene 10 rondas mientras que blake2b tiene 12 rondas usando SIGMA[10..11] = SIGMA[1..1].

La función de mezcla G es similar en ambas versiones del algoritmo. La función de compresión F varía en el número de rondas *r*, siendo 10 para blake2s y 12 para blake2b.

ID del algoritmo	**Objetivo Arch**	**Seguridad ante colisiones**	***Hash* nn**	***Hash* ASN.1 sufijo OID**
id-blake2b-512	64 *bits*	2^{256}	64	x.1.16
id-blake2s-256	32 *bits*	2^{128}	32	x.2.8

Tabla 6.3: Comparativa de blake2b-512 y blake2s-256

Las dos variantes de blake2

- blake2b IDENTIFICADOR DE OBJETO ::= {hashAlgs 1}
- blake2s IDENTIFICADOR DE OBJETO ::= {hashAlgs 2}

Respectivos identificadores

- id_blake2b512 IDENTIFICADOR DE OBJETO ::= {blake2b 16}
- id_blake2s256 IDENTIFICADOR DE OBJETO ::= {blake2s 8}

La creación de nuevos objetos vendrían dadas por la invocación a las funciones constructor [Aum].

hashlib.blake2b (*data*=b, *, *digest_size*=64, *key*=b, *salt*=b, *person*=b, *fanout*=1, *depth*=1, *leaf_size*=0, *node_offset*=0, *node_depth*=0, *inner_size*=0, *last_node*=False, *usedforsecurity*=True)

hashlib.blake2s (*data=b*, *, *digest_size*=32, *key*=b, *salt*=b, *person*=b, *fanout*=1, *depth*=1, *leaf_size*=0, *node_offset*=0, *node_depth*=0, *inner_size*=0, *last_node*=False, *usedforsecurity*=True)

Siendo así funciones compatibles, cambiando solo el tamaño de los datos de entrada, salida y las constantes.

6.6.2. Falcon

Se trata de un algoritmo actualmente en evaluación en la tercera ronda del concurso de criptografía post-cuántica del NIST [Weba]. Se basa en el marco teórico de Gentry, Peikert y Vaikuntanathan para esquemas de firma basados en celosías. Se instancia ese marco sobre redes *N-th degree Truncated polynomial Ring Units* (NTRU), con un muestreado de trampilla llamado "muestreo rápido de Fourier". El problema difícil subyacente es el problema de *Short Integer Solution* (SIS) sobre redes NTRU, para el cual actualmente no se conoce ningún algoritmo de resolución eficiente en el caso general, incluso con la ayuda de computadoras cuánticas.

$$Falcon = GPV\,framework + NTRU\,lattices + Fast\,Fourier\,sampling$$

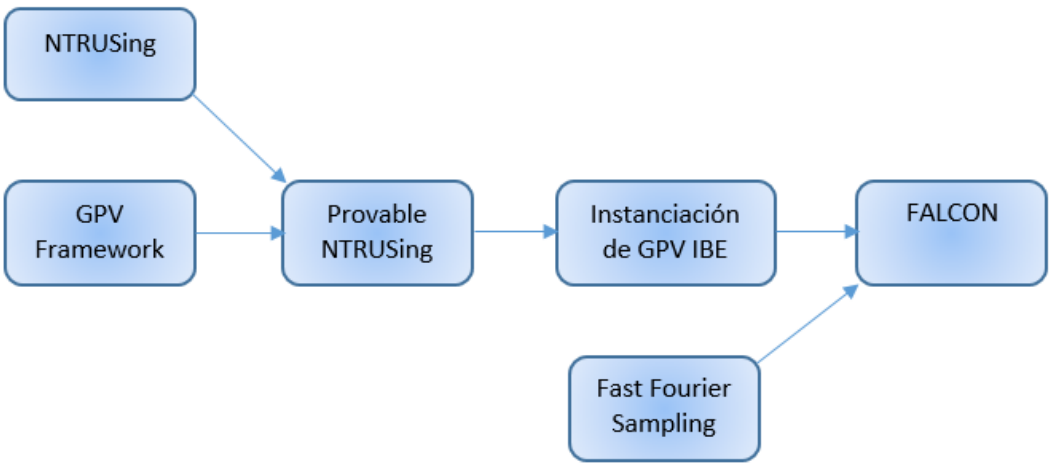

Figura 6.1: Diagrama de Falcon

Falcon ofrece las siguientes características:

- Seguridad: internamente se utiliza un verdadero *sampler* gaussiano, lo que garantiza una fuga de información inapreciable sobre la clave secreta hasta un número prácticamente infinito de firmas (más de 2^{64}).

- Compacidad: gracias al uso de celosías NTRU, las firmas son sustancialmente más cortas que en cualquier esquema de firma basado en celosía con las mismas garantías de seguridad, mientras que las claves públicas tienen aproximadamente el mismo tamaño.

- Velocidad: el uso del muestreo rápido de Fourier permite implementaciones muy rápidas, en miles de firmas por segundo en una computadora común; la verificación es de cinco a diez veces más rápida.

- Escalabilidad: las operaciones tienen un coste de $O(nlogn)$ para el grado n, lo que permite utilizar parámetros de seguridad de muy largo plazo a un costo moderado.

- Economía de *Random Access Memory* (RAM): el algoritmo mejorado de generación de claves de Falcon utiliza menos de 30 kilo*bytes* de RAM, una gran mejora con respecto a diseños anteriores, por lo que es compatible con dispositivos integrados pequeños con limitaciones de memoria.

Si bien la resistencia a las computadoras cuánticas es el principal impulso para el diseño y desarrollo de Falcon, el algoritmo también es razonablemente eficiente en nuestra computación actual. Usando la implementación de referencia en una computadora de escritorio común (Intel® Core® i5-8259U a 2,3 Gigahercio (GHz), TurboBoost deshabilitado), Falcon logra el siguiente rendimiento:

Variante	**Generador de claves(ms)**	**Generador de claves (RAM)**	**Señales**	**Verificaciones**	**Tamaño de pub**	**Tamaño de firma**
Falcon-512	8.64	14336	5948.1	27933.0	897	666
Falcon-1024	27.45	2872	2913.0	13650.0	1793	1280

Tabla 6.4: Parámetros de Falcon

Los tamaños (uso de RAM de generación de clave, tamaño de clave pública, tamaño de firma) se expresan en *bytes*. El tiempo de generación de claves se da en milisegundos. El tamaño de la clave privada (no mencionado anteriormente) es aproximadamente tres veces mayor que el de una firma y, en teoría, podría comprimirse hasta una pequeña semilla *Pseudo-Random Number Generator* (PRNG) (por ejemplo, 32 *bytes*) si el firmante acepta ejecutar el algoritmo de generación de claves cada vez que la clave deba ser cargada.

Para dar un punto de comparación, Falcon-512 es más o menos equivalente, en términos de seguridad clásicos, a RSA-2048 cuyas firmas y claves públicas usan 256 *bytes* cada una. En el sistema específico en el que se tomaron estas medidas la implementación de ensamblado completamente optimizada de *Open Secure Sockets Layer* (OpenSSL) logra alrededor de 1140 firmas por segundo; por lo tanto, la implementación de referencia de Falcon, que es portátil y no usa ensamblaje en línea en CPU x86, ya es más de cinco veces más rápida y escala mejor a tamaños más grandes (para seguridad a largo plazo).

6.7. Sustitución de blake2s-256 por blake2b-512

6.7.1. Análisis del código de WireGuard

Se empezará analizando el código fuente de WireGuard alojado en un repositorio de GitHub [Webb].

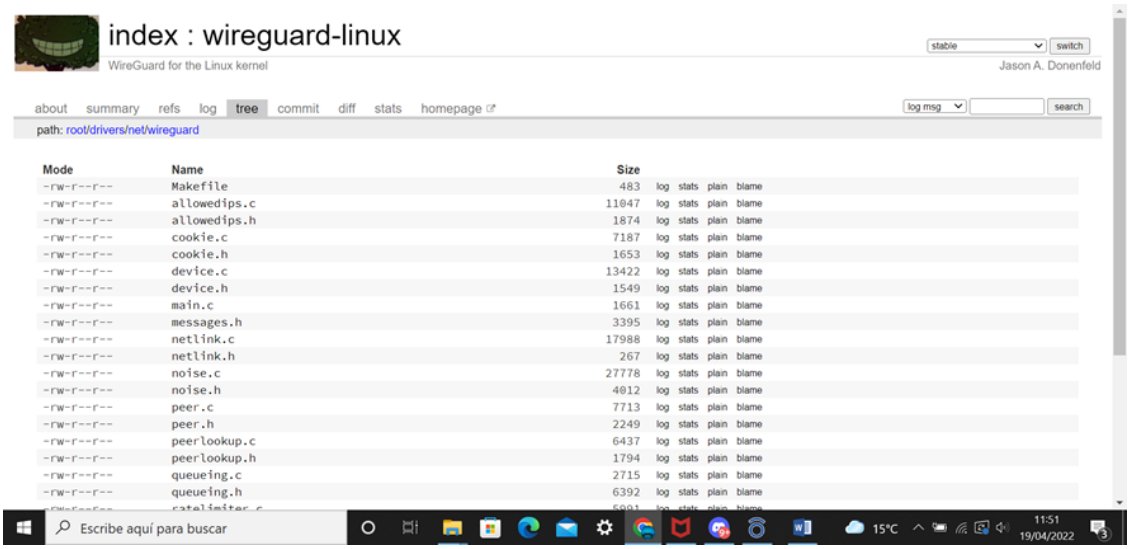

Figura 6.2: Repositorio de código de WireGuard

La primera modificación propuesta es la de localizar los módulos del programa donde aparecen referencias a blake2s y cambiarlas por referencias a blake2b, ya que las llamadas al parecer son compatibles al ser de la misma familia. Esto implicaría cambios en variables y llamadas a las funciones así como la importación de la librería "crypto/blake2b", librería que actualmente se encuentra en los repositorios de WireGuard.

En cuanto al cambio de parámetros de tamaños de entrada y salida, estos vienen definidos dentro de la librería de cada uno de los blake2, por lo que al no estar definidos localmente en el código de WireGuard, con un simple cambio de librerías quedarían modificados. A continuación vamos a realizar dichas modificaciones en cada uno de los archivos en este código fuente dónde se usa blake2s [Webc].

- *cookie.c*. Sustituimos las funciones referentes a blake2s por las referentes a blake2B, así como la sustitución a la librería correspondiente.

```
1  // SPDX-License-Identifier: GPL-2.0
2  /*
3   * Copyright (C) 2015-2019 Jason A. Donenfeld <Jason@zx2c4.com>. All Rights Reserved.
4   */
5
6  #include "cookie.h"
7  #include "peer.h"
8  #include "device.h"
9  #include "messages.h"
10 #include "ratelimiter.h"
11 #include "timers.h"
12
13 #include <crypto/blake2s.h>          #include <crypto/blake2b.h>
14 #include <crypto/chacha20poly1305.h>
15
16 #include <net/ipv6.h>
17 #include <crypto/algapi.h>
18
```

Figura 6.3: Cambios propuestos 1 (cookie.c)

```
static void precompute_key(u8 key[NOISE_SYMMETRIC_KEY_LEN],
                           const u8 pubkey[NOISE_PUBLIC_KEY_LEN],
                           const u8 label[COOKIE_KEY_LABEL_LEN])
{
        struct blake2s_state blake;                    struct blake2b_state blake;

        blake2s_init(&blake, NOISE_SYMMETRIC_KEY_LEN);      blake2b_init (&blake, NOISE_SYMETRIC_KEY_LEN);
        blake2s_update(&blake, label, COOKIE_KEY_LABEL_LEN); blake2b_update(&blake, label, COOKIE_KEY_LABEL_LEN);
        blake2s_update(&blake, pubkey, NOISE_PUBLIC_KEY_LEN); blake2b_update(&blake, pubkey, NOISE_PUBLIC_KEY_LEN);
        blake2s_final(&blake, key);                    blake2b_final(&blake, key);
}
```

Figura 6.4: Cambios propuestos 2 (cookie.c)

```
75 static void compute_mac1(u8 mac1[COOKIE_LEN], const void *message, size_t len,
76                          const u8 key[NOISE_SYMMETRIC_KEY_LEN])
77 {
78         len = len - sizeof(struct message_macs) +
79               offsetof(struct message_macs, mac1);
80         blake2s(mac1, message, key, COOKIE_LEN, len, NOISE_SYMMETRIC_KEY_LEN);
81 }       blake2b(mac1, message, key, COOKIE_LEN, len, NOISE_SYMETRIC_KEY_LEN);
82
83 static void compute_mac2(u8 mac2[COOKIE_LEN], const void *message, size_t len,
84                          const u8 cookie[COOKIE_LEN])
85 {
86         len = len - sizeof(struct message_macs) +
87               offsetof(struct message_macs, mac2);
88         blake2s(mac2, message, cookie, COOKIE_LEN, len, COOKIE_LEN);
89 }       blake2b(mac2, message, cookie, COOKIE_LEN, len, COOKIE_LEN);
```

Figura 6.5: Cambios propuestos 3 (cookie.c)

```
static void make_cookie(u8 cookie[COOKIE_LEN], struct sk_buff *skb,
                        struct cookie_checker *checker)
{
        struct blake2s_state state;                    struct blake2b_state state;

        if (wg_birthdate_has_expired(checker->secret_birthdate,
                                     COOKIE_SECRET_MAX_AGE)) {
                down_write(&checker->secret_lock);
                checker->secret_birthdate = ktime_get_coarse_boottime_ns();
                get_random_bytes(checker->secret, NOISE_HASH_LEN);
                up_write(&checker->secret_lock);
        }

        down_read(&checker->secret_lock);

        blake2s_init_key(&state, COOKIE_LEN, checker->secret, NOISE_HASH_LEN);   blake2b_init_key(&state, COOKIE_LEN, checker->secret, NOISE_HASH_LEN);
        if (skb->protocol == htons(ETH_P_IP))
                blake2s_update(&state, (u8 *)&ip_hdr(skb)->saddr,                blake2b_udate(&state, (u8 *)&i_hdr(skb)->saddr;
                               sizeof(struct in_addr));
        else if (skb->protocol == htons(ETH_P_IPV6))
                blake2s_update(&state, (u8 *)&ipv6_hdr(skb)->saddr,              blake2b_udate(&state, (u8 *)&iv6_hdr(skb)->saddr;
                               sizeof(struct in6_addr));
        blake2s_update(&state, (u8 *)&udp_hdr(skb)->source, sizeof(__be16));     blake2b_udate(&state, (u8 *)&ud_hdr(skb)->source, sizeof(__be16));
        blake2s_final(&state, cookie);                 blake2b_final(&state, cookie);

        up_read(&checker->secret_lock);
}
```

Figura 6.6: Cambios propuestos 4 (cookie.c)

- *noise.c*. Modificamos las referencias a blake2s sustituyéndolas por blake2b, así como la sustitución de librerías.

```
static const u8 handshake_name[37] = "Noise_IKpsk2_25519_ChaChaPoly_BLAKE2s"; "Noise_IKsk2_25519_ChaChaoly_BLAKE2b";
static const u8 identifier_name[34] = "WireGuard v1 zx2c4 Jason@zx2c4.com";
static u8 handshake_init_hash[NOISE_HASH_LEN] __ro_after_init;
static u8 handshake_init_chaining_key[NOISE_HASH_LEN] __ro_after_init;
static atomic64_t keypair_counter = ATOMIC64_INIT(0);
```

Figura 6.7: Cambios propuestos 5 (noise.c)

```
void __init wg_noise_init(void)
{
        struct blake2s_state blake;                    struct blake2b_state blake;

        blake2s(handshake_init_chaining_key, handshake_name, NULL,       blake2b(handshake_init_chainining_key, handshake_name, NULL,
                NOISE_HASH_LEN, sizeof(handshake_name), 0);              NOISE_HASH_LEN, sizeof(handshake_name), 0);
        blake2s_init(&blake, NOISE_HASH_LEN);                          blake2b_init(&blake, NOISE_HASH_LEN);
        blake2s_update(&blake, handshake_init_chaining_key, NOISE_HASH_LEN); blake2b_update(&blake, handshake_init_chaining_key, NOISE_HASH_LEN);
        blake2s_update(&blake, identifier_name, sizeof(identifier_name));  blake2b_update(&blake, identifier_name, sizeof(identifier_name));
        blake2s_final(&blake, handshake_init_hash);                      blake2b_final(&blake, handshake_init_hash);
}
```

Figura 6.8: Cambios propuestos 6 (noise.c)

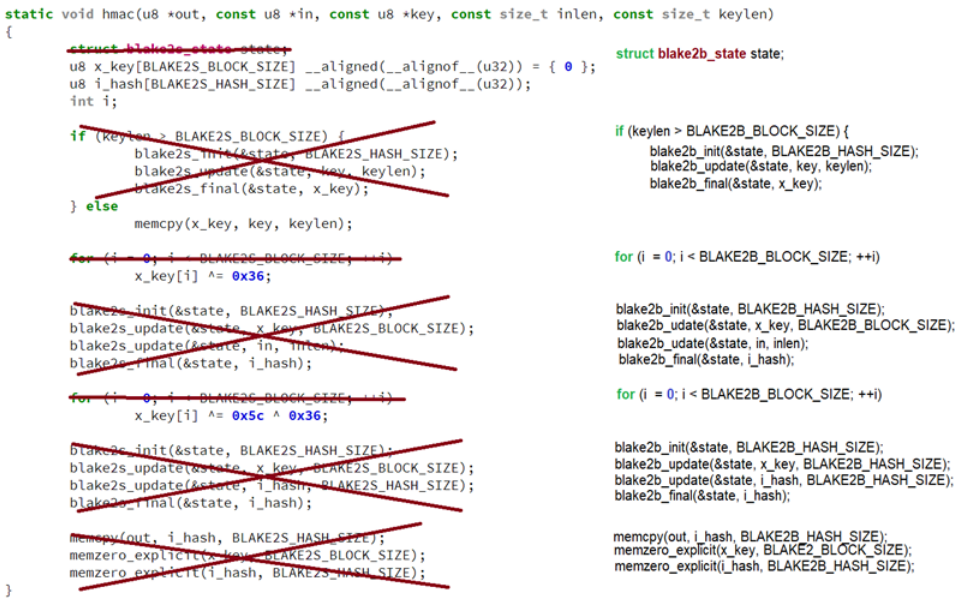

Figura 6.9: Cambios propuestos 7 (noise.c)

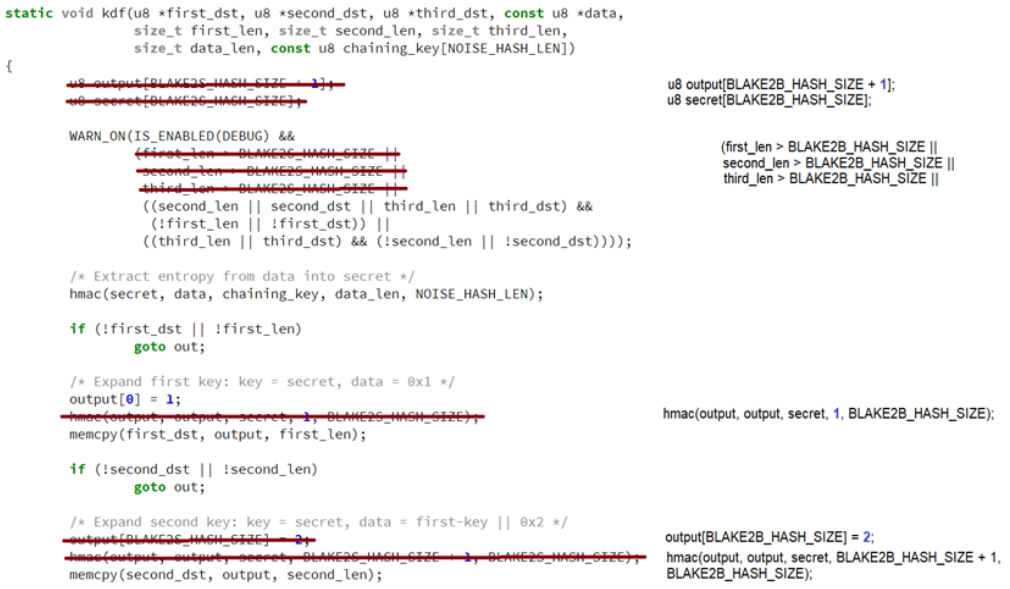

Figura 6.10: Cambios propuestos 8 (noise.c)

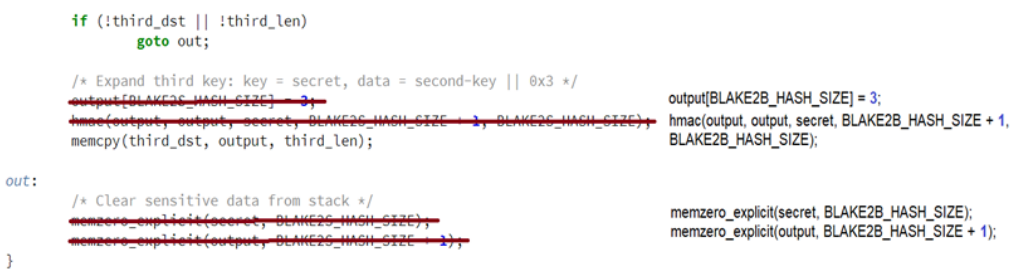

Figura 6.11: Cambios propuestos 9 (noise.c)

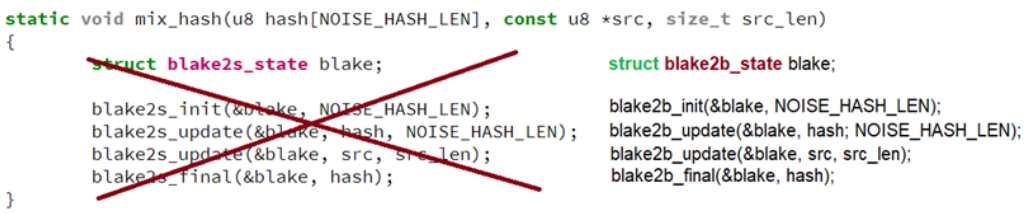

Figura 6.12: Cambios propuestos 10 (noise.c)

Una vez llevada a cabo esta modificación habría que tener en cuenta las longitudes, ya que tanto los *bits* de entrada como los de salida son de distinto tamaño entre blake2s y blake2b, a lo que sumamos una *key* de distinta longitud, como ya hemos visto antes en la tabla explicativa, por tanto habría que ver las implicaciones que tendrían estos cambios de longitudes en el resto de algoritmos que usen blake2 como parámetro.

Una propuesta para solucionar dicho problema sería truncar los valores a la misma longitud que blake2s, siendo así compatible totalmente. Pero, a pesar del aumento de rondas (de 10 a 12), se perdería gran parte de la seguridad, la cual reside precisamente en la longitud de los resúmenes obtenidos, por lo que con esta propuesta de truncamiento no se conseguiría una verdadera seguridad post-cuántica. También habría que modificar el archivo *messages.h* donde se definen el control de tamaños constantes de entradas y salidas entre los distintos algoritmos, así como los mensajes intercambiados durante el *handshake*, los cuales habría que adatar al uso de blake2b.

Capítulo 7

Laboratorio de firmas digitales

7.1. Descripción

Con la finalidad de realizar comparaciones en tiempo real entre algoritmos de *hash*, se ha implementado una herramienta donde, además de poder obtener el resumen de varios algoritmos, nos muestra aspectos de rendimiento, tales como el tiempo de cálculo y la entropía de Shannon del resumen. Se permite la elección de uno o varios algoritmos y para estos la introducción de una única preimagen o un conjunto de preimagenes. Estando en este último caso cada una de ellas delimitada por corchetes para una correcta interpretación por el sistema. Para el caso de los algoritmos de la familia blake2 se permite el uso de una *key* para reforzar la seguridad, el sistema se encargará de aplicar dicha key sólo a los algoritmos correspondientes. La entrada puede realizarse tanto de manera manual, en el área de texto destinada para ello, como en forma de fichero de entrada con un tamaño máximo de 10 MB. Para ficheros superiores a 2 MB, no se mostrará su contenido en el área de texto, sino que se realizará en memoria, proporcionando las salidas de forma similar al resto de ficheros.

Dado que la funcionalidad de la aplicación se centra principalmente en la parte de *Front-End*, se ha prestado especial cuidado en que la navegación propuesta entre acciones sea, en todos casos, la más adecuada. Se persigue de cara al usuario final que el funcionamiento sea intuitivo y con resultados claros en función de la acción elegida. Según la decisión de entrada el resultado que se muestra es diferente y adecuado para que la interpretación de datos sea la mejor facilitando así su análisis.

- Entrada manual o por fichero de una preimagen: nos muestra, en la zona derecha de la pantalla, el resumen obtenido a partir de la entrada con cada uno de los distintos algoritmos de *hash*, así como la indicación de si se tratan de algoritmos con seguridad post-cuántica. Esta sería la forma indicada para obtener el resumen asociado a la preimagen introducida.

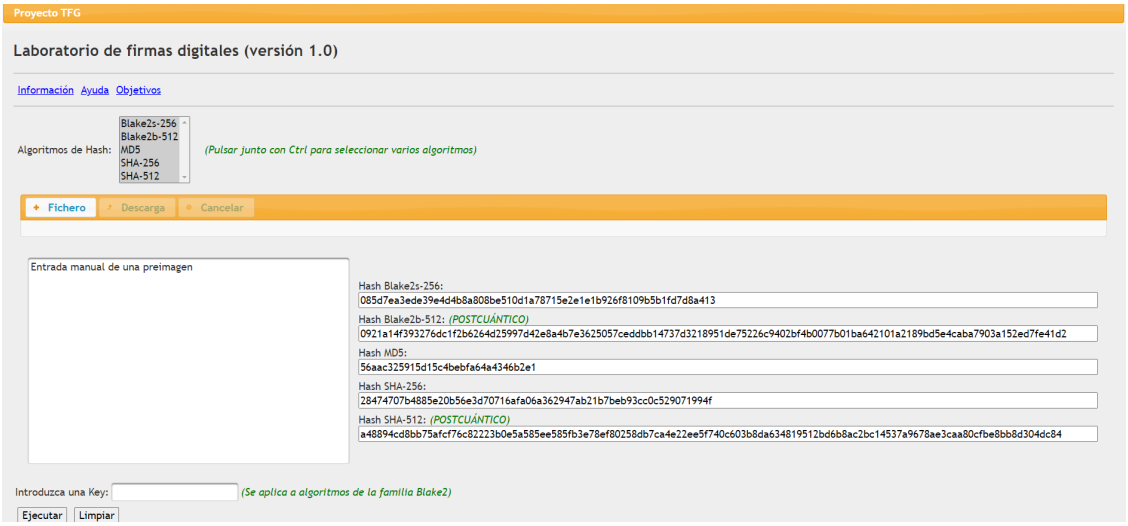

Figura 7.1: Ejemplo de entrada de una preimagen

- Entrada manual o por fichero de varias preimágenes: nos muestra, en la zona derecha de la pantalla, los datos del tiempo de ejecución y entropía de Shannon respecto a la longitud de la preimagen en forma de gráficos de líneas, siendo esta una forma más visual de realizar comparaciones y ver tendencias.

Esta funcionalidad está enfocada a un análisis de los datos de los algoritmos, no a la obtención de los resúmenes, lo que dificultaría la claridad y facilidad de análisis ocupando espacio en la pantalla.

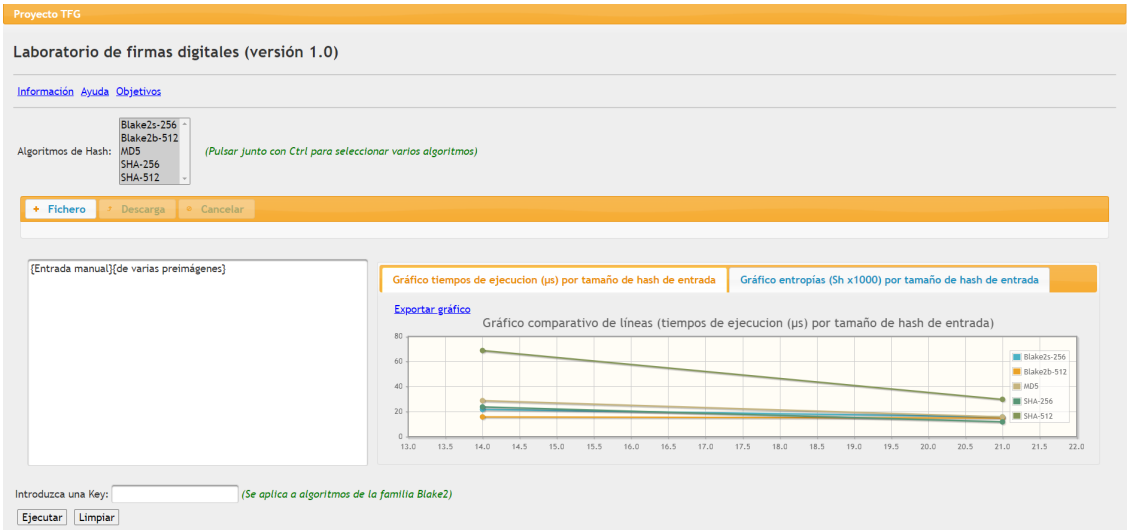

Figura 7.2: Ejemplo de entrada de varias preimágenes

En ambos casos, en la parte inferior de la pantalla, se mostrará una tabla asociada a los datos obtenidos de cada algoritmo al realizar el resumen. Es posible la ordenación de cada una de las columnas.

Exportar a Excel

Nombre Hash	Núm. caracteres entrada	Tiempo de ejecución (µs)	Entropía de Shannon (Sh)	Núm. caracteres Key
Blake2s-256	14	22	3.9593225166542543	0
Blake2b-512	14	16	3.9031245336210816	0
MD5	14	29	3.391126767019874	0
SHA-256	14	24	3.7829638216885826	0
SHA-512	14	69	3.8792613780547294	0
Blake2s-256	21	16	3.8797352677511157	0
Blake2b-512	21	15	3.8805961436461365	0
MD5	21	16	3.616729296672175	0
SHA-256	21	12	3.848988235582062	0
SHA-512	21	30	3.9117650959698134	0

Figura 7.3: Tabla de salida

Se permite la exportación de los datos presentados en la tabla a un archivo Excel que es especialmente útil para los casos en los que se eligen varios algoritmos y se pretenden analizar varias entradas. Trabajar con datos en Excel posibilita un manejo de información más óptimo ya que se permiten filtros, ordenación múltiple de campos, creación de gráficos y, entre otras, copia de datos a otras plataformas. En resumen, se permite tratar la información de una forma más potente. También podemos encontrar la opción de exportar los gráficos a imagen e incluirlos en documentos de estudio.

Para facilitar la representación de las medidas en los gráficos se ha decidido usar para la entropía de Shannon "Sh x 1000" y para los tiempos de ejecución "microsegundos", ajustando así el tiempo a una buena precisión dado que se trata de cálculos muy rápidos. En el caso de la entropía de Shannon se muestra el orden, más bien el desorden, implícito del resumen obtenido, cuanto mayor sea la entropía, mayor información contendrá un conjunto de datos, mientras que una menor entropía implica menos desorden lo que conlleva la existencia de menos alternativas y por tanto los mismos datos contendrán menos información, por lo que será un *hash* más seguro y más difícil de descifrar. En cuanto al tiempo de cálculo cabe decir que cuanto menor sea este, estaremos hablando de un algoritmo de *hash* más eficiente. Cuanto menor sea el espacio de mensajes y menor la entropía, existirá un menor riesgo de colisión en el tratamiento del *hash* pero la reidentificación, es decir, la obtención de la preimagen a partir del resumen, será más probable. Por el contrario, a mayor entropía la posibilidad de una colisión, es decir, la obtención de dos resúmenes iguales a partir de preimágenes distintas será mayor, pero el riesgo de reidentificación será menor.

En la parte superior de la pantalla se dispone de un menú informativo, donde se encuentra breve información asociada a cada algoritmo de *hash* así como una ayuda para el uso y los objetivos.

7.2. Estructura de la aplicación

A la hora de elegir una estructura para realizar este proyecto se ha optado por implementar una aplicación Java Web apoyada en la gestión de bibliotecas de Maven y con la capa de *Front-End* realizada con ayuda del *framework* *Java Server Faces* (JSF). Como refuerzo a la maquetación y presentación de la capa web se utiliza la biblioteca Print-Faces. Para soportar su funcionamiento se ha decidido usar un servidor Tomcat local desplegado en el puerto 8080 de *localhost*. El servidor debe estar levantado y la ruta de lanzamiento es: https://localhost:8080/TFG-AnalizadorHash/

Cabe mencionar que todas las herramientas utilizadas en esta aplicación web son gratuitas y de fácil uso e instalación en cualquier *Personal Computer* (PC) de uso doméstico.

7.3. Tecnologías y herramientas utilizadas

A continuación se tratarán las tecnologías utilizadas para la realización de esta aplicación web así como el motivo de su elección. Se dará una explicación detallada y razonada de cada tecnología usada y su finalidad. Más adelante se explicará también cómo se han implementado estas tecnologías dentro de la aplicación.

- Java es un lenguaje de alto nivel muy usado destinado para el desarrollo de aplicaciones web. Ofrece una gran cantidad de prestaciones así como la creación de clases que nos ayudan mucho a una correcta compartimentación e implementación.
- *Java Development Kit* (JDK) es un conjunto de herramientas, utilidades, documentación y ejemplos para el desarrollo de aplicaciones Java. Es imprescindible para dar soporte a la ejecución de los programas.
- Eclipse es un *Integrated Development Environment* (IDE) multilenguaje construido alrededor de un *workspace* al que pueden incluirse un gran número de *plugins* proporcionando funcionalidades concretas en el desarrollo de una aplicación. En nuestro caso usamos Eclipse con la extensión del lenguaje Java.
- Maven Repository es un repositorio de librerías importables en Java para nuestro proyecto y que nos aportan funcionalidades concretas de formas testada, confiable y sencilla gracias a su amplio repositorio de librerías útiles.
- JSF es una tecnología y *framework* para aplicaciones Java Web que simplifica el desarrollo de interfaces de usuario. Proporciona un Modelo Vista Controlador (MVC) para la correcta organización de las aplicaciones web. Ofrece una serie de ventajas:
  - Facilidad de aprendizaje y uso, ya que el código JSF con el que las vistas son creadas es muy parecido al *HyperText Markup Language* (HTML) por lo que los desarrolladores y diseñadores web podrían usarlo fácilmente.
  - Permite introducir JavaScript en la página, acelerando la respuesta de la interfaz en el cliente y proporcionándole una mayor comodidad de uso.
  - Proporciona un ciclo de vida de acorde al MVC y permite el uso sencillo de *Asynchronous JavaScript And XML* (AJAX) mostrando los datos de forma asíncrona, sin tener que recargar nuevamente la página aportando mayor fluidez.

Desde un enfoque más técnico, es reseñable la compatibilidad de este *software* al cambio de versiones del resto de elementos con los que trabaja e interopera y cabe destacar que JSF forma parte del estándar *Java 2nd Platform Enterprise Edition* (J2EE) el cual nos proporciona un estándar de desarrollo de aplicaciones de empresa multinivel.

- PrimesFaces es una biblioteca de código abierto de componentes para potenciar JSF. Ofrece una serie de marcas que posibilitan acciones modularizadas que además aportan un aspecto más profesional. Algunas de las ventajas que aporta son:
  - Ofrece un amplio conjunto de componentes de interfaz de usuario como Data-Table, AutoComplete, HtmlEditor, Charts...
  - Para su uso no se requiere ninguna configuración adicional, ni son necesarias dependencias.

  - Tiene AJAX incorporado para un mayor dinamismo hacia el cliente.
  - Contiene temas ya creados que mejoran el aspecto final de la aplicación.
  - Se trata de una biblioteca amplia y claramente documentada con códigos de ejemplo para su uso.
  - Alto grado de compatibilidad con otras librerías de componentes.
  - Realiza un uso de JavaScript no intrusivo, es decir, no aparece en línea dentro de los elementos, sino dentro de un bloque con la etiqueta $\prec$ *script* $\succ$.
  - Es un proyecto de código abierto, activo y bastante estable entre versiones.
  - Facilita la maquetación y organización de componentes dentro de las páginas.

- Managed Beans es una clase regular de Java Bean registrada con JSF. Son objetos administrados por un contenedor con servicios enriquecidos admitidos, como inyección de recursos, devoluciones de llamadas e interceptores del ciclo de vida. Estos contienen los métodos *get* y *set*, la lógica de presentación y la comunicación con el controlador. Los Managed Beans funcionan como modelo para el componente *User Interface* (UI) de un modo fácil de administrar.

- JUnit se utiliza para implementar las pruebas unitarias en las que se verifica si los casos de entrada son correctos. Es plenamente compatible con Java.

- Apache Tomcat es un servidor *open source* de *servlets* que se puede usar para compilar y ejecutar aplicaciones web realizadas en Java. Posibilita la ejecución en entorno local de la aplicación web.

7.4. Versiones de sofware y librerías utilizadas

Para esta aplicación se ha usado:

- Java 1.8.0.
- Eclpise 2022-03 (4.23.0)
- JSF 2.2.17
- PrimeFaces 11.0.0
- JUnit 4.12
- Apache Tomcat 8.5
- Jblake 0.4
- blake2b 1.0.0
- Pitaya 0.4

7.5. Pruebas unitarias

Este tipo de pruebas se han realizado con JUnit de modo programático. Se ha decidido verificar el cálculo de todos los algoritmos y el cálculo de la entropía de Shannon.

Para cada algoritmo se prueba que sean correctos los resúmenes asociados a distintos tipos de entradas: con caracteres seguidos, con separación con espacios, con saltos de línea y con caracteres especiales y acentos. Los algoritmos que permiten refuerzo de clave tienen métodos adicionales que cubren este caso.

Para la entropía de Shannon se comprueba que la función utilizada ofrezca resultados de salida correctos, ya que en ellos se basa el estudio de medidas.

Los casos no programados se tratan con las pruebas funcionales.

7.6. Pruebas funcionales

Se ha probado el funcionamiento en los navegadores de uso más común: Edge, Chrome y Firefox donde la aplicación se presenta y funciona correctamente. Para cada acción se han comprobado los casos válidos permitidos y las entradas que pueden provocar anomalías por presentar algún problema. En caso de anomalía, esta se captura y se muestra por pantalla el mensaje de error correspondiente para que el usuario pueda actuar en consecuencia. Se han hecho pruebas con palabras de entrada individual de tamaño pequeño, contemplando entradas con caracteres seguidos, con espacios, con saltos de línea y con caracteres especiales incluidos los acentos. Para entradas de tamaño grande, superior a 2 *MegaByte* (MB), se recurre a la carga por fichero. Con datos de prueba el flujo es el previsto y los resultados los esperados. Para entradas manuales de hasta 2 MB también se ha comprobado el correcto funcionamiento.

7.7. Dificultades encontradas

A lo largo del proceso de desarrollo de la aplicación web se han encontrado una serie de dificultades a destacar:

- Problema con los saltos de línea: en el *inputTexArea* utilizado, los saltos de línea se toman como \n. Para realizar de forma adecuada el resumen, los saltos de línea debían tomarse como en Windows, es decir \n. por lo que se ha implementado una función que toma la entrada, tanto del fichero como del *inputTexArea* y cambia los \n por \r \n (Retorno de carro y salto de línea).
- Problema de codificación del texto de entrada para realizar los resúmenes: se daban problemas con los caracteres especiales y acentos, por lo que se ha decidido usar UTF8 como codificación, evitando así errores de este tipo.
- Problemas con la salida de los resúmenes: se producían errores a la hora de transformar el texto del resumen (en bytes) a hexadecimal. Para lo que se ha utilizado una función conversora evitando así salidas de caracteres extraños.
- Problema en la salida de *hash*: cuando el resumen empezaba por un cero o una cadena de ceros se veían truncados, ya que los ceros a la izquierda se anulaban. Para

ello, cuando la longitud del resumen es más corta de lo esperado, se rellena a la izquierda con ceros hasta completar la longitud del resumen, solventando así este problema.

- Problema de tamaño de fichero: para tamaños de ficheros superiores a 2 MB se producían errores. Como en la aplicación web se quiere permitir la entrada de ficheros de hasta 10 MB (valor modificable), se ha decidido trabajar directamente sobre la memoria, no mostrando el texto en el *inputTexArea* en caso de que el tamaño de entrada supere los 2 MB permitidos.

7.8. Diagramas UML

7.8.1. Casos de uso

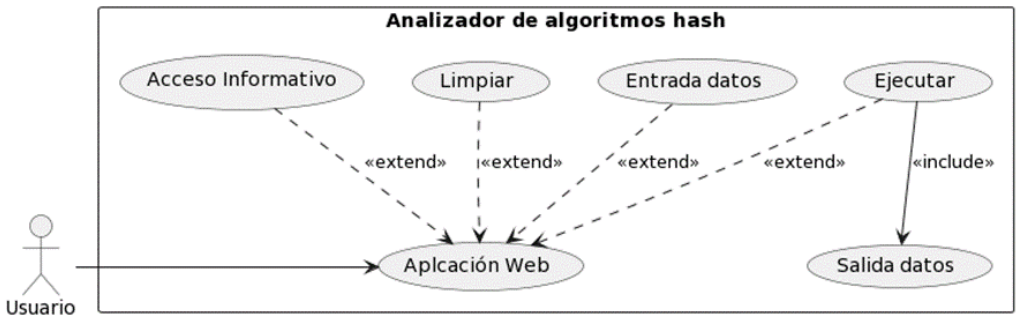

Figura 7.4: Caso de uso analizador de algoritmos hash

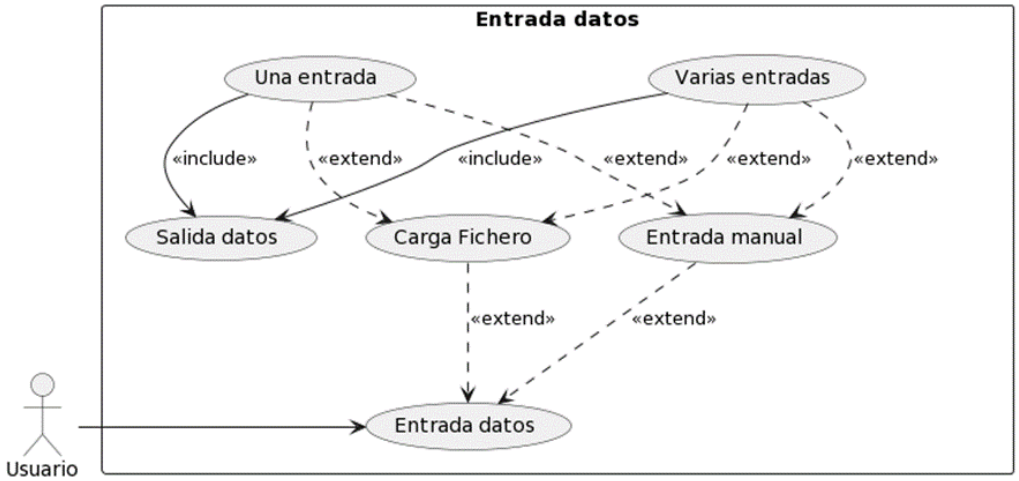

Figura 7.5: Caso de uso entrada de datos

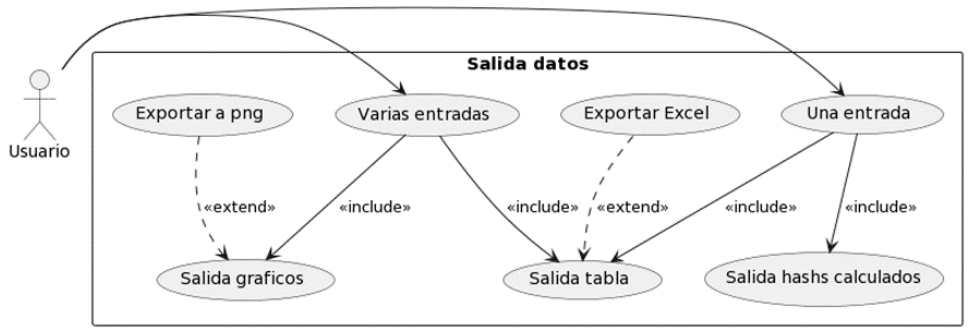

Figura 7.6: Caso de uso salida de datos

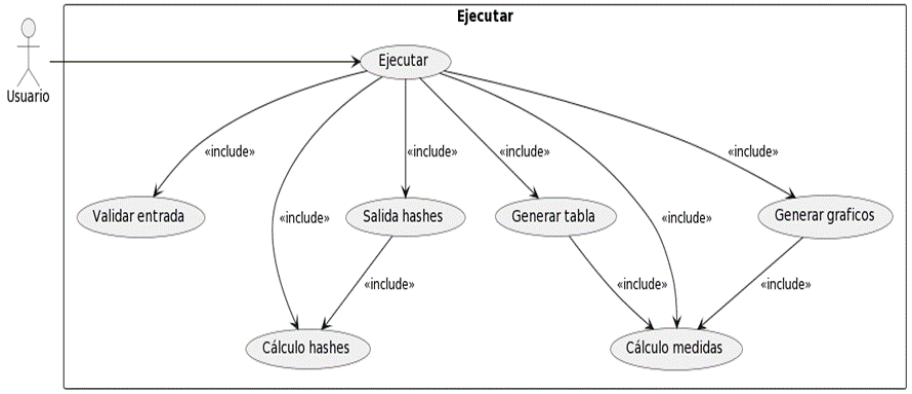

Figura 7.7: Caso de uso ejecutar

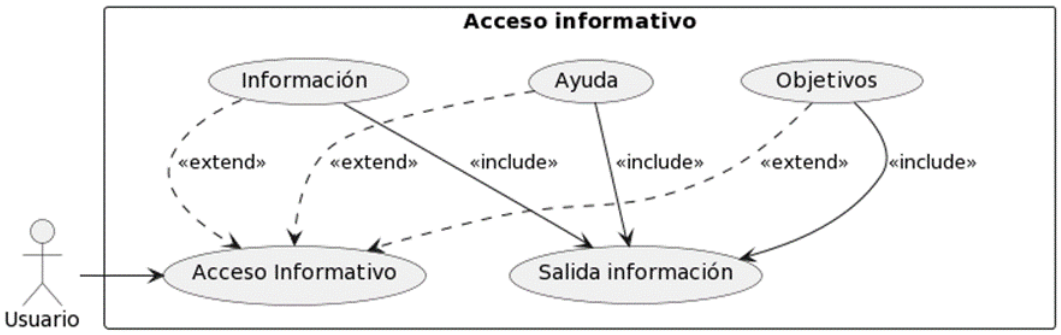

Figura 7.8: Caso de uso acceso informativo

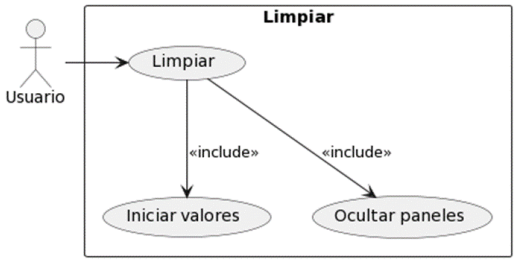

Figura 7.9: Caso de uso limpiar

7.8.2. Diagrama de estados

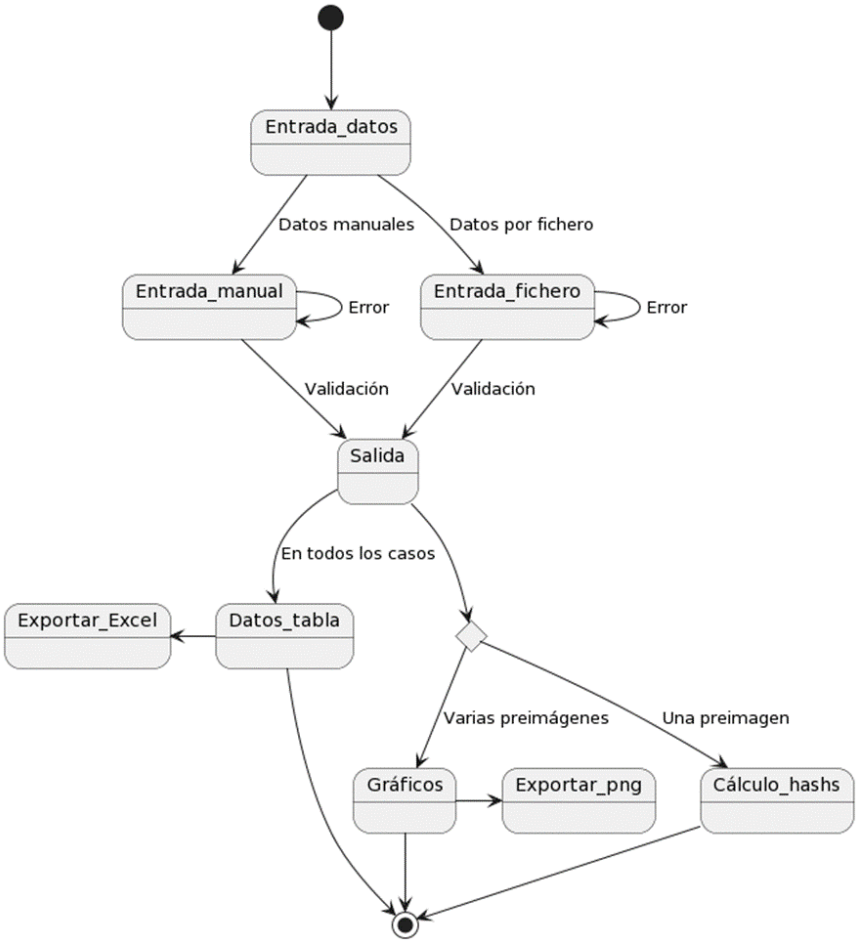

Figura 7.10: Diagrama de estados

7.8.3. Diagrama de clases

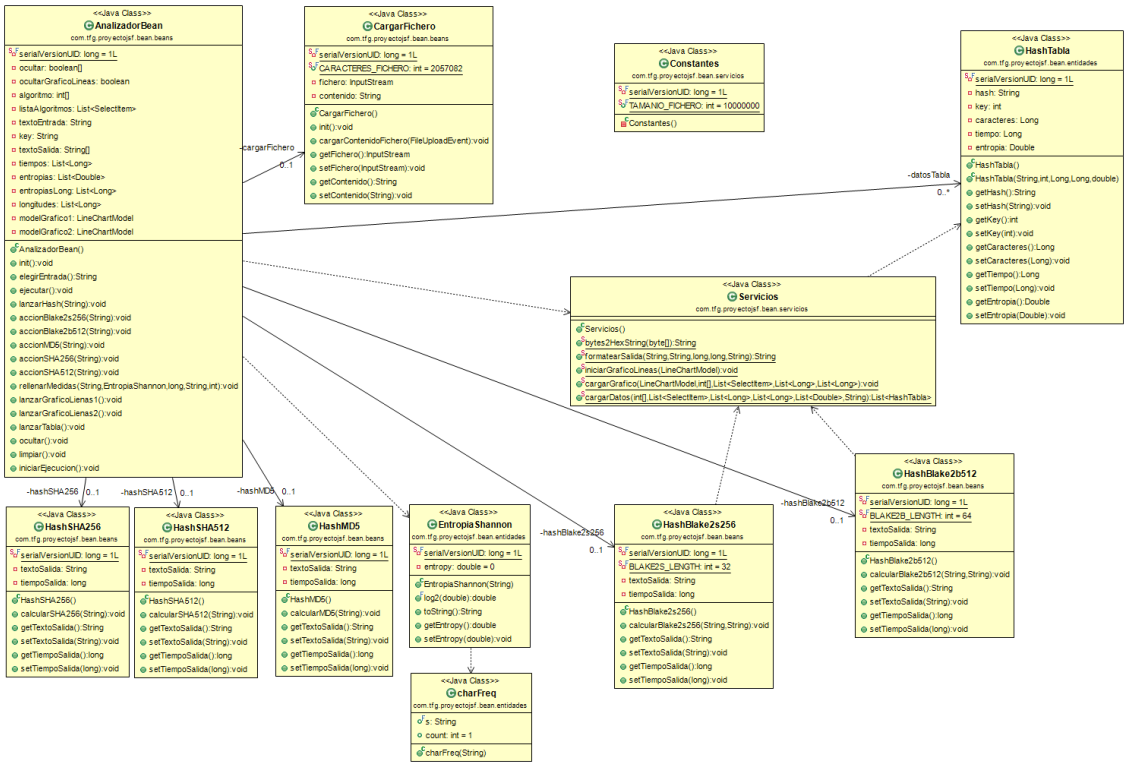

Figura 7.11: Diagrama de clases

7.8.4. Visión crítica y análisis de los resultados

A continuación se van a mostrar algunas ejecuciones de la aplicación web comparando datos obtenidos.

Dichas ejecuciones serán analizadas en base a los conocimientos acerca de *hashes*.

Figura 7.12: Salida de resúmenes

Se ha introducido la palabra “Prueba” como preimagen dando lugar a las funciones de resumen vistas. Podemos observar aspectos tales como la extensión de dichos resúmenes, siendo MD5 el menos extenso, con una longitud de 32 caracteres, seguido de blake2s-256 y SHA-256 con 64 caracteres y con SHA-512 y blake2b-512 como resúmenes de mayor extensión, contando con 128 caracteres. También podemos ver que tanto SHA-512 como blake2b-512 son post-cuánticos.

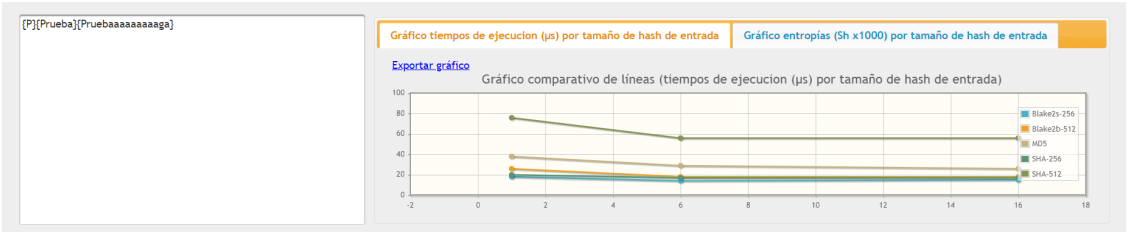

Figura 7.13: Gráfico de tiempo de ejecución

Para este segundo análisis se realizan tres entradas: {P}{Prueba}{Pruebaaaaaaaaaga} obteniendo los resultados gráficos que se analizan a continuación. En cuanto a tiempos de ejecución podemos observar que MD5, pese a ser el algoritmo con un resumen de menor longitud, no es el más rápido. Blake2s-256 y SHA-256 tienen unos tiempos de ejecución bastante similares, debido a que ambos nos ofrecen tamaños de resumen de 64 caracteres. Cabe mencionar que en caso de una máquina de 32 *bits* el algoritmo blake2s-256 sería más eficiente, ya que está orientado a dicho tipo de máquinas. Se observa que SHA-512 tiene el tiempo de ejecución más alto, ya que nos ofrece el mayor tamaño de resumen. Por último se ve que el algoritmo blake2b-512 tiene un tiempo de ejecución similar al de otros algoritmos con resúmenes de 64 caracteres, pero ofreciendo resúmenes de 128. Esto es debido a que blake2b-512 esta optimizado para máquinas de 64 *bits* (como es el ordenador desde el que se ejecuta esta aplicación web) alcanzando así su rendimiento óptimo en estas condiciones y reduciendo mucho su tiempo de ejecución.

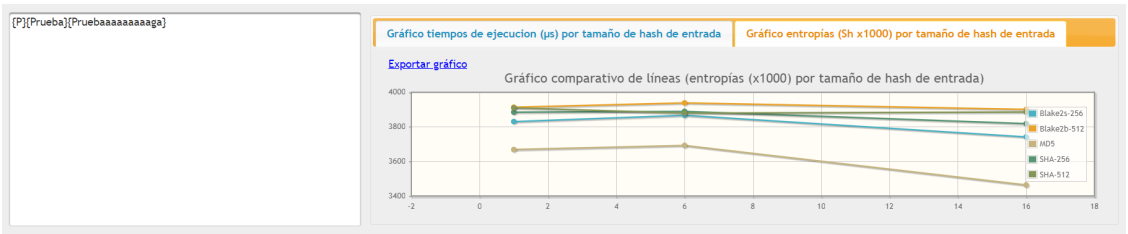

Figura 7.14: Gráfico de entropía de Shannon

En cuanto a entropía de Shannon, se observa que MD5 está muy por debajo del

resto de algoritmos mostrando así un resumen poco seguro y con baja resistencia a ataques de obtención de preimagen. Tanto SHA-256 como blake2s-256 ofrecen una entropía moderada, acorde con su tamaño de resumen y tiempo de ejecución. SHA-512 tiene una entropía superior a la de los algoritmos mencionados, resultado esperable, ya que presenta resúmenes más extensos con un tiempo de ejecución más elevado. Por último se observa que el algoritmo con mayor entropía y por tanto, con mayor resistencia ante ataques de obtención de preimagen es el blake2b-512, lo que podría sorprendernos, ya que nos aporta la mayor seguridad con un tiempo de ejecución bajo.

Nombre Hash	Núm. caracteres entrada	Tiempo de ejecución (µs)	Entropía de Shannon (Sh)	Núm. caracteres Key
Blake2s-256	1	18	3.832108306496812	0
Blake2b-512	1	26	3.915143346304342	0
MD5	1	38	3.67095859334435	0
SHA-256	1	20	3.8865877974034166	0
SHA-512	1	76	3.911417963838221	0
Blake2s-256	6	14	3.868443118360758	0
Blake2b-512	6	18	3.94035946536658	0
MD5	6	29	3.6945488277869583	0
SHA-256	6	17	3.8917831585689373	0
SHA-512	6	56	3.8816014737797335	0
Blake2s-256	16	15	3.7431929241263275	0

Figura 7.15: Tabla de salida de datos

Aquí se exponen los datos obtenidos en el análisis en forma de tabla para ver y contrastar los datos de manera numérica.

Como conclusión de este análisis se señala que el mejor candidato a la hora de implementar un algoritmo de *hash* en una máquina de 64 *bits* es el blake2b-512, gracias a sus altas prestaciones tanto en rendimiento como en seguridad, a lo que hay que añadirle que se trata de un *hash* de carácter post-cuántico.

Capítulo 8

Conclusiones y trabajo futuro

8.1. Conclusiones

Tras la lectura de este trabajo deducimos que estamos frente al escenario de la llegada inminente de la computación cuántica, para el cual es necesario prepararse. Para ello, primero es necesario comprenderlo así como entender en qué compromete a la actual criptografía.

Tras centrarnos en la VPN WireGuard se ha visto que no es resistente a este tipo de computación, por lo que se han analizado sus algoritmos y propuesto otros post-cuánticos para su sustitución.

Gracias al uso del laboratorio de firmas digitales se observa que blake2b-512 hoy en día está muy por encima en cuestiones de rendimiento y seguridad respecto de sus competidores, por lo que sería un gran aporte a WireGuard, siendo este además un algoritmo post-cuántico.

Aún queda trabajo por hacer, pero esta investigación es un paso importante en la comprensión de la computación cuántica y de los algoritmos de seguridad de WireGuard, para los cuales se han mostrado unas primeras indicaciones de sustitución, centrándose en la de blake2s-256 por blake2b-512 siendo este más eficiente y además resistente a la computación cuántica.

El laboratorio de firmas es una herramienta de gran utilidad, ya sea destinada a fines didácticos de comprensión de *hashes* o como comparador y analizador a un mayor nivel que el de la enseñanza y que puede crecer fácilmente con las adición de nuevas firmas y nuevas medidas a analizar.

8.2. Trabajo Futuro

Como posibles trabajos futuros pueden señalarse los siguientes:

- Sustitución de blake2s-256 por blake2b-512 en WireGuard a partir de los pasos indicados.
- Sustitución del resto de algoritmos de WireGuard con la finalidad de hacerlo post-cuántico siguiendo las indicaciones y propuestas de este trabajo.

- Mejora del laboratorio de análisis de firmas digitales con la adición de nuevos algoritmos de *hash* o nuevas variables a analizar.

Capítulo 9

Introduction

9.1. Motivation

Cryptography is the technique of writing with a secret key, it is responsible for writing and encrypting messages so that they are difficult to read by any third party that intercepts our information. Etymologically it comes from the Greek *crypto*, which means hidden or secret, and *graphy*, which is its graphic representation. Cryptanalysis deals with decryption methods without knowledge of the key that has been used to encrypt.

In classical Greece they used torches to spell words and send messages. In the Roman Empire, unintelligible messages were sent thanks to the placement of the letters in such a way that no words could be formed and to decipher them the order of the letters had to be changed. In more modern times, other methods have been used, such as mirrors that reflected sunlight using Morse Code. Until now, monoalphabetic substitution ciphers were used.

Leon Battista Alberti wrote in 1466 *De Componendis Cyphris*, the oldest known book on cryptography in the Western world where this type of encryption is analyzed. Historically he is the first to propose polyalphabetic ciphers.

During the World Wars, most messages were transmitted by radio, but this had the drawback that anyone with a receiver tuned to the right frequency could hear them, so they had to be encrypted. The German army used in World War II the most famous encryption machine in history: Enigma. Alan Turing was primarily responsible for cracking Enigma, thereby helping to shorten the war. From this moment modern cryptography was developed.

El medio de telecomunicaciones más utilizado en la actualidad es Internet, que tiene como pilar básico el uso de la criptografía como forma de garantizar la seguridad y operatividad de sus recursos con respecto a la confidencialidad, integridad, autenticidad y no repudio de los mensajes intercambiados. Lo que garantiza este apoyo es la imposibilidad práctica de romper la criptografía en los sistemas y aplicaciones con los recursos computacionales que existen hoy en día. Sin embargo, con la aparición de los primeros computadores cuánticos muchas de las claves de cifrado, actualmente seguras, en un futuro podrían ser vulnerables dejando al descubierto todo el sistema de cifrado actual, por ello es necesario preparar nuestra criptografía para hacerla resistente a la computación cuántica.

En este TFG se tratarán los principios básicos de la criptografía así como sus distintos

tipos y utilidades comentando los principales ataques, también se tratará la computación cuántica teniendo en cuenta sus ataques teóricos y los posibles problemas que esta generaría en la seguridad clásica actual, para centrarse en VPN WireGuard y su adaptación para un futuro cuántico. En este trabajo introductorio se propondrán una serie de indicaciones para la sustitución de los algoritmos de WireGuard por unos post-cuánticos, centrándonos en el cambio de blake2s-256 por blake2b-512, para lo cual se detallarán las acciones teóricas necesarias para llevar a cabo esta tarea. Como parte práctica surgió la idea de la creación de una herramienta de laboratorio de firmas digitales, que permitiera analizar de forma fácil y comparativa distintos parámetros de algoritmos de *hash* y que, gracias a su análisis de datos y comparación, sirvió para apoyar la decisión del cambio de *hash* propuesto para WireGuard.

9.2. Object of the Investigation

Understand what quantum computing is and how it works, more specifically in its application in cybersecurity and how this scenario would affect current classical encryption and cryptography.

Choosing a VPN that is currently widely used and open source and facing the study of the changes necessary to adapt it to a post-quantum behavior.

Implementation of a Java Web application for hashing algorithm analysis that, in addition to summaries of pre-images introduced, enables dynamic and visual comparison between different algorithms as well as obtaining relevant measurements when choosing one summary function or another.

9.3. Workplan

Four phases have been completed

1. **Research** This first phase has mainly consisted of searching for information and familiarizing myself with the content of the project, especially with quantum computing and with the WireGuard VPN tool. After the first meetings with the professors, they guided me towards the search for information on cryptography, including hash algorithms, the basic concepts of post-quantum cryptography, the associated mathematical concepts and what this type of computing implies for the future. Information was collected from different sources and bibliography as well as the application to be modified was studied and tested to understand its operation.

2. **Development**: Once the concepts are understood and presented, we move on to a phase of practical development. A recognized VPN, such as WireGuard, is taken and the changes to be made so that one of its algorithms become post-quantum are considered. To do this, the code is accessed and the different files where the algorithm to be modified appears are analyzed. All the indications are noted down and collected in the memory of this project. To check the validity of this change, and what advantages the integration of a post-quantum algorithm provides, the idea of implementing a web application that facilitates this step arises. Its implementation is decided in order to serve as a comparator and analyzer of different hashing

algorithms, where, based on an input preimage, it provides us with the calculated summary accompanied by a series of measures that have been considered useful, all based on the entered word size. The output is accompanied by comparative graphs, a table with the data and with exports to Excel and graphs. In addition, brief information on the algorithms chosen for the study is included. In the specific case that concerns us, it was necessary to compare Blake2s-256 with Blake2b-512. In addition, it is decided to include MD5, SHA256 and SHA512 as representation of the most used algorithms. The application is scalable so that, if necessary, more algorithms and measurements could be incorporated.

3. **Carrying out tests**: The tests that the entire application must include to verify its correct operation are developed. It is verified with the implementation of unit tests for all the calculations of the algorithms and the Shannon entropy, where for various inputs it is verified that they return the expected values. Functional tests are also performed to test everything that cannot be covered by unit tests. Within this phase, the study of the behavior of the algorithms based on the results and how this can be applied to the study of the proposed changes in WireGuard stands out. Conclusions reflected in the memory are reached and a preview of the replacement work in WireGuard is given, which remains open as a continuation.

4. **Memory**: It implies collecting everything developed, cataloging it and including it in its corresponding sections so that the information is presented structured and the entire process carried out is reflected. The conclusions and the future work that opens from this introductory project are added.

9.4. Struture of the Work

The work is organized in 8 chapters, before starting with the chapters the images and tables are cataloged and the acronyms are identified.

Chapter 1 includes the introduction and motivation of the TFG. It details the aspects of why and how this work has been carried out, explaining the context and the work plan followed in its realization.

Chapter 2 deals with quantum computing. This new type of computing is introduced, its operation and usefulness are explained and its main algorithms are discussed. A comparison is also made with the current classical computing and the possible inconveniences that this would cause in the field of cybersecurity are detailed.

Chapter 3 shows the importance of cryptography. It comments on the interest in this field today as well as an explanation of the different types of classical cryptography.

Chapter 4 deals with quantum-resistant cryptography, also called post-quantum cryptography. New types of encryption techniques that have emerged to deal with the quantum computing scenario are explained here.

Chapter 5 introduces quantum computing resistant VPN features. Here the concept of VPN is defined and the post-quantum algorithm contest carried out by NIST is explained.

Chapter 6 introduces the WireGuard VPN tool as well as the full range of cryptographic algorithms it uses. After the explanation, a security analysis of each of them is carried out, proposing post-quantum solutions. In the case of the replacement of Blake2s-256 by Blake2b-512, a detailed theoretical proposal for change is offered.

Chapter 7 includes the digital signature laboratory web application, where, once the structure and tools used have been commented, its functionalities are presented, as well as proper functioning tests and corresponding UML diagrams. Finally, an example of real analysis using it is shown.

Chapter 8 shows the main conclusions of this work, the future lines of research and the possible derivable works.

Chapters 9 and 10 are the English translations of the Introduction and Conclusions.

Finally, you can find the bibliography used.

Capítulo 10

Conclusions and Future Work

10.1. Conclusions

After reading this work we can conclude that we are facing a scenario of the imminent arrival of quantum computing, for which it is necessary to prepare. To do this, it is first necessary to understand it as well as understand what it compromises the current cryptography.

After focusing on the WireGuard VPN, it has been seen that it is not resistant to this type of computation, so its algorithms have been analyzed and other post-quantum algorithms have been proposed for its replacement.

Thanks to the use of the digital signature laboratory, we have been able to observe that blake2b-512, today is far above its competitors in terms of performance and security, so it would be a great contribution to WireGuard, as this is also a post-quantum algorithm.

There is still work to be done, but this work is an important step in understanding quantum computing and WireGuard's security algorithms, for which we have shown early indications of substitution, focusing on blake2s-256 by blake2b-512 being this more efficient and also resistant to quantum computing.

The signature laboratory is a very useful tool, whether it is intended for educational purposes of understanding hashes or as a comparator and analyzer at a higher level than that of teaching and that can easily grow with the addition of new signatures and new measures to analyze.

10.2. Future Work

As possible future works following this investigation line, there are proposed the following ones:

- Substitution of blake2s-256 for blake2b-512 in WireGuard from the indicated steps.
- Substitution of the rest of the WireGuard algorithms in order to make it PostQuantum following the indications and proposals of this work.
- Improvement of the digital signature analysis laboratory with the addition of new hash algorithms or new variables to analyze.

Bibliografía

[Aum] J.-P. Aumasson. The BLAKE2 Cryptographic Hash and Message Authentication Code (MAC),” RFC 7693. https://datatracker.ietf.org/doc/html/rfc7693.

[Bera] Daniel J. Bernstein. A state-of-the-art Diffie-Hellman function by Daniel J. Bernstein ”My curve25519 library com-putes the Curve25519 function at very high speed. The library is in the public domain”. *public domain.*

[Berb] Daniel J. Bernstein. Irrelevant patents on elliptic-curve cryptography. *cr.yp.to.*

[Berc] Daniel J. Bernstein. Patentes irrelevantes sobre criptografía de curva elíptica. *cr.yp.to.*, 3.

[Ber06] Daniel J. Bernstein. Curve25519: new Diffie-Hellman speed records,” pem Public Key Cryptography”. *pem Public Key Cryptography. Lecture Notes in Computer Science*, 3958, 2006.

[Ber09a] Daniel J. Bernstein. Introducción a la criptografía post-cuántica. 2009.

[Ber09b] Daniel J. Bernstein. Preguntas y respuestas con el investigador de criptografía de computación cuántica Jintai Ding. 2009.

[BO15] Daira; Hulsing Bernstein, Daniel J.; Hopwood and O'Hearn. .).SP HINCS: prácticas firmas basadas en hashs inestado. Apuntes de conferencias en Ciencias de la Computación.9056.Spri. *Zooko (2015).Oswald, Elisa beth; F ischlin, Marc (eds).*, 2015.

[Buc11] Erik; Hulsing Andreas Buchmann, Johannes; Dahmen. *XMSS: un esquema práctico de firma segura hacia adelante basado en suposiciones de seguridad mínimas”. Criptografía post-cuántica.* PQCrypto 2011. Apuntes de conferencias en Ciencias de la Computación. 7071. pags. 117-129. CiteSeerX 10.1.1.400.6086 . doi : 10.1007 /978-3-642-25405-58, 2011.

[Buy21] Adarsh Kumar Carlo Ottaviani Sukhpal Singh Gill Rajkumar Buyya. Securing the future internet of things with post-quantum cryptography. *Senior Member, IEEE*, November 2021.

[CC] CCN-CERT. Gestión de eventos de seguridad. https://www.inteco.es/glossary/Formacion/Glosario/.

[Din05] Jintai; Schmid Ding. *Arco iris, un nuevo esquema de firma polinomial multivariable.* En Ioannidis, John (ed.). Tercera Conferencia Internacional, ACNS 2005, Nueva York, NY, EE. UU., 7 al 10 de junio de 2005. Actas. Apuntes de conferencias en Ciencias de la Computación. 3531. pags. 64-175. doi: 10.1007, 7 de junio de 2005.

[DJBCNDHSRIMYM06] Aggelos-Kiayias Daniel J. Bernstein. Curve25519: New Diffie-Hellman Speed Records. In Moti Yung, Yevgeniy Dodis and Tal Malkin. PKC 2006, 9th International Conference on Theory and Practice of Public-Key Cryptography, New York, NY, USA, April 24-26, 2006, Proceedings, volume 3958 of Lecture Notes in Computer Science. *USA, April 24-26, 2006*, 3958:207–228, January 14 2006.

[FC] Tiago M. Fernández-Caramés. Pre-Quantum to Post-Quantum IoT Security: A Survey on Quantum-Resistant Cryptosystems for the Internet of Things.

[FC20] Tiago M. Fernández-Caramés. From Pre-Quantum to Post-Quantum IoT Security: A Survey on Quantum-Resistant Cryptosystems for the Internet of Things. *Senior Member, IEEE*, 7(7), July 2020.

[GB11] M. Peeters e G. V. Assche G. Bertoni, J. Daemen. The Keccak SHA-3 submission. January 14 2011.

[Hig13] Peter Higgins. Impulsando un secreto perfecto hacia adelante, una importante protección de la privacidad en la Web. *Fundación Frontera Electrónica.*, 2013.

[HW01] Hwang SW Hwang WY, Ahn DD. *Eavesdropper's optimal information in variations of Bennett–Brassard 1984 quantum key distribution in the coherent attacks.* Phys Lett A. 2001; 279 (3–4):133-138, 2001.

[jea10] T. Klein jung et al. *Factorization of a 768-bit RSA modulus.* Proc. 30th Annu. Conf. Adv. Cryptol., Santa Barbara, CA, USA, pp. 333–350, 2010.

[Kal00] B Kaliski. PKCS 5: Password-Based Cryptography Specification Version 2.0", RFC 2898. https://datatracker.ietf.org/doc/html/rfc2898, September 2000.

[Ket] Ket.G. computación cuántica. https://www.youtube.com/channel/UCedVPGH5fnWBmsR4oHVOX6w.

[Lan] A. Langley. RFC 7539: ChaCha20 and Poly1305 for IETF Protocols," Internet Research Task Force (IRTF). https://datatracker.ietf.org/doc/html/rfc7539.

[LMT21] SARA RICCI1 RAIMUNDAS MATULEVIČIUS JAN HAJNY GAUTAM SRIVASTAVA(Senior Member IEEE) ABASI-AMEFON O. AFFIA4 MARYLINE LAURENT(Member IEEE) NAZATUL HAQUE SULTAN LUKAS MALINA, PETR DZURENDA and QIANG TANG. Post-Quantum Era Privacy Protection for Intelligent Infrastructures. *Senior Member, IEEE*, May 2021.

[Mos18] M. Mosca. Cybersecurity in an era with quantum computers: Will we be ready. *IEEE Security Privacy, vol. 16*, 33.(no. 5):38–41, 2018.

[N07] Nikolina Ilic N. The Ekert protocol. J Phy. 33.:(1):22, 2007.

[NIS] NIST. Post-Quantum Cryptography," Information Technology Laboratory - Computer Security Resource Center. https://csrc.nist.gov/projects/post-quantum-cryptography.

[NIS14] NIST. Standards-research-and-applications-in-cryptograph. *csrc.nist.gov*, 3, Octubre 2014.

[Sou21] Higor Souza. Availacao de metodos criptograficos quantico-Resistentes WireGuard. *Universidad de Brasília, Faculdade de Tecnologia*, 2021.

[Taq10] Laurent Vleeschouwer Christophe de Taquet, Maxime Jacques. Invariant Spectral Hashing of Image Saliency Graph. 3, 2010.

[Weba] Obtenido de la página oficial de falcon. https://www.di.ens.fr/~prest/Publications/falcon.pdf.

[Webb] Obtenido del repositorio de código oficial de wireguard. https://git.zx2c4.com/wireguard-linux/tree/drivers/net/wireguard/.

[Webc] Obtenido del repositorio de código oficial de blake2b, más en concreto en la carpeta de funciones. https://github.com/franziskuskiefer/blake2b/blob/master/src/blake2b_functions.c.

[Webd] Origen de la computación cuántica. https://es.wikipedia.org/wiki/ComputaciC3B3n_cuC3A1ntica#Origen_de_la_computaciC3B3n_cuC3A1ntica.

[Webe] Principios fundamentales de la computación cuántica. https://www.researchgate.net/publication/347112503_PRINCIPIOS_FUNDAMENTALES_DE_COMPUTACION_CUANTICA.

[Webf] Shor's algorithm. https://quantum-computing.ibm.com/composer/docs/iqx/guide/shors-algorithm.

[Webg] Algoritmo de shor. http://diccionario.sensagent.com/Algoritmo20de20Shor/es-es/.

[Webh] Algoritmo de grover. https://bizinsightsnews.com/es/qc-el-algoritmo-de-grover/.

[Webi] Qué es un hash y cómo funciona. https://latam.kaspersky.com/blog/que-es-un-hash-y-como-funciona/2806/.

[Webj] Criptografia sistemas simetricos y asimetricos. https://sites.google.com/site/francolirusso3c/home/software-ii-4c/criptografia---sistemas-simetricos-y-asimetricos.

[Webk] Criptografía asimétrica. https://es.wikipedia.org/wiki/CriptografC3ADa_asimC3A9trica.

[Webl] Criptografía, conceptos de seguridad. https://unifiednetworks.es/ciberseguridad/conceptos-de-ciberseguridad/criptografia/.

[Webm] Tunneling-protocol. https://latam.kaspersky.com/resource-center/definitions/tunneling-protocol.

[Webn] Getting-to-know-hkdf. https://www.nearform.com/blog/getting-to-know-hkdf/.

[Webo] Obtenido del respositorio de GitHub oficial de SIPHash. https://github.com/veorq/SipHash/blob/master/README.md.

[Webp] Obtenido de la página oficial de xchacha20. https://doc.libsodium.org/advanced/stream_ciphers/xchacha20.

[Webq] What is an vpn. https://www.avast.com/es-es/c-what-is-a-vpn.

[Wir] WireGuard. Obtenido de la página de WIreGuard. https://www.wireguard.com/.