

UNIVERSIDAD DE MÁLAGA  
ESCUELA TÉCNICA SUPERIOR DE  
INGENIERÍA DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

MÓDULO DE SEGURIDAD HARDWARE  
PARA FIRMAS BLS

GRADO EN INGENIERÍA TELEMÁTICA

PABLO DE JUAN FIDALGO  
MÁLAGA, 2021



## **Módulo de Seguridad Hardware para firmas BLS**

Autor: Pablo de Juan Fidalgo

Tutor: Isaac Agudo Ruiz

Departamento: Departamento Lenguajes y Ciencias de la Computación

Titulación: Grado en Ingeniería Telemática

Palabras clave: blst, cadena de bloques, esquema de firmas BLS, Ethereum, módulo de seguridad hardware.

### **Resumen**

Este trabajo es uno de los 25 proyectos becados por Ethereum Foundation dentro del programa Eth2 Staking Community Grants que surgió como impulso para mejorar el ecosistema de validación en el lanzamiento de la nueva red de Ethereum.

El objetivo es ayudar a la escalabilidad de la cadena de bloques y permitir que los clientes devuelvan la firma de un bloque cuando esta sea requerida mientras escuchan a través de una API. Además, la ejecución de las operaciones criptográficas en un entorno seguro como puede ser un módulo de seguridad hardware aporta al usuario mayor confianza en sus operaciones.

La implementación se ha llevado a cabo a través de dos vías: una se ha desarrollado con una biblioteca estática y la otra con el código fuente suministrado por la biblioteca blst de Supranational. Ambos trabajos han reflejado buenos resultados y han puesto de manifiesto que microcontroladores de bajo coste, como la placa nRF9160 DK usada, también pueden participar y funcionar correctamente en tecnologías demandantes como es *blockchain*.

## **Hardware Security Module BLS Signer**

Author: Pablo de Juan Fidalgo

Supervisor: Isaac Agudo Ruiz

Department: Computer Science Department

Degree: Grado en Ingeniería Telemática

Keywords: blockchain, BLS signature scheme, blst, Ethereum, hardware security module.

### **Abstract**

This work is one of the 25 grantee projects from the Eth2 Staking Community Grants from Ethereum Foundation. The goal of this programme was to enhance the staking ecosystem while the new Ethereum network was being launched.

The objective is to help in blockchain's scalability and to allow clients to return the requested block signature while listening to an API. Also, the execution of cryptographic operations in a secure environment such as a hardware security module contribute to improve the user's trust in his work.

The implementation of this project has gone through two ways: one of them has been developed with a static library and the other one with source code files from Supranational blst library. Both works presented good results and they have shown that low-cost microcontrollers like nRF9160 DK board can succeed in challenging technologies such as Blockchain.

# Agradecimientos

A Isaac Agudo por ejercer de tutor en este Trabajo Fin de Grado y sobre todo por brindarme la oportunidad de desarrollarlo a través de Decentralized Security, que obtuvo una beca de Ethereum Foundation en febrero de 2021.

A mis amigos y compañeros del Grado con quienes he compartido multitud de momentos a lo largo de estos cuatro años.

A Raquel Morente por la revisión del manuscrito y sus indicaciones como traductora.



*A mis padres Isabel y Miguel  
por su continuo apoyo, indispensable para llegar hasta aquí  
y para seguir en adelante*





# Índice del contenido

<b>Capítulo 1. Introducción y objetivos.....</b>	<b>15</b>
1.1. Introducción .....	15
1.2. Objetivos.....	16
<b>Capítulo 2. <i>Blockchain</i>.....</b>	<b>19</b>
2.1. Introducción .....	19
2.2. Bitcoin .....	20
2.3. Ethereum.....	22
2.4. Eth2 .....	24
2.5. Mecanismos de consenso.....	26
2.6. Almacenamiento de las claves: <i>cold wallets</i> .....	28
<b>Capítulo 3. Tecnologías implicadas.....</b>	<b>31</b>
3.1. Esquema de firmas BLS .....	31
3.2. Microcontrolador nRF9160 DK.....	34
3.2.1. ARM TrustZone .....	35
3.2.2. Zephyr.....	36
3.2.3. SEGGER Embedded Studio .....	37
<b>Capítulo 4. Análisis .....</b>	<b>41</b>
4.1. Objetivo.....	41
4.2. Impacto y retos .....	42
4.3. Trabajos previos.....	44
<b>Capítulo 5. Desarrollo .....</b>	<b>47</b>
5.1. Arquitectura.....	47
5.2. Diseño.....	48
5.3. Implementación.....	54

5.3.1. Primera iteración .....	54
5.3.2. Segunda iteración.....	56
5.3.3. Pruebas y resultados.....	57
<b>Capítulo 6. Conclusiones y líneas futuras .....</b>	<b>59</b>
6.1. Conclusiones .....	59
6.2. Líneas futuras.....	60
<b>Referencias .....</b>	<b>63</b>

## Índice de figuras

Figura 3.1. Arquitectura del nRF91 SoC.....	34
Figura 3.2. División de zonas con ARM TrustZone .....	35
Figura 3.3. Instrucción Go To Definition.....	37
Figura 3.4. Control de expresiones en SEGGER Embedded Studio.....	38
Figura 3.5. Tamaño de la pila durante la ejecución del código .....	38
Figura 3.6. Estado de los registros .....	39
Figura 5.1. Arquitectura tradicional en Cliente de Ethereum.....	47
Figura 5.2. Arquitectura con módulo de seguridad hardware .....	48
Figura 5.3. Generación de claves.....	51
Figura 5.4. Comando publickey .....	51
Figura 5.5. Traza de la ejecución del comando signature .....	52
Figura 5.6. Comando getkeys .....	53

## Índice de tablas

Tabla 5.1. Análisis de los tiempos promedios empleados en cada operación.....	58
---	----

## Lista de Acrónimos

API	Application Programming Interface
BLS	Boneh–Lynn–Shacham
CPU	Central Processing Unit
DAO	Decentralized Autonomous Organization
EVM	Ethereum Virtual Machine
HSM	Hardware Security Module
HTTP	Hypertext Transfer Protocol
IKM	Interpretative Key Management
IoT	Internet of Things
LTE	Long Term Evolution
PoA	Proof of Authority
PoS	Proof of Stake
PoW	Proof of Work
SDK	Software Development Kit
SHA	Secure Hash Algorithm
SiP	System in Package
SoC	System on a Chip
TLS	Transport Layer Security
USB	Universal Serial Bus



# Capítulo 1. Introducción y objetivos

## 1.1. Introducción

En los últimos años debido al auge de las criptomonedas en general y Bitcoin en particular, la tecnología de cadena de bloques, comúnmente llamada *blockchain*, se ha convertido en el foco de atención de muchas empresas para modernizar sus aplicaciones, además de multitud de casos de uso adaptados a este nuevo paradigma. El concepto primigenio se podría establecer en 1991 con la descripción [1] de este sistema por S. Haber y W. S. Stornetta. Sin embargo, no es hasta 2008 con la aparición del *white paper* sobre Bitcoin [2], escrito por Satoshi Nakamoto, donde se explica en profundidad la implementación de este mecanismo para crear una moneda virtual que hará uso de una red *peer-to-peer* (P2P) para almacenar el conjunto de transacciones que se realicen sobre ella.

La criptografía es fundamental para el funcionamiento de los protocolos de seguridad que protegen las redes y en particular Internet. En el caso de las redes de *blockchain* lo es más aún, sobre todo si estamos hablando de criptomonedas, debido a que el robo de una clave puede suponer una importante pérdida de fondos. Por ello se están definiendo diferentes soluciones para la custodia de las claves mediante el uso de monederos electrónicos, firmas compartidas o soluciones basadas en computación segura multiparte.

La meta de este trabajo no es otra que la de proteger las claves privadas de los nodos que conforman la red de Eth2. Para ello, todas las operaciones

criptográficas como la generación de claves y las firmas se trasladarán a un elemento *hardware* externo, fuera del cliente. Al separar estas operaciones se podrán desplegar de forma segura nodos en entornos mucho menos seguros como servicios en la nube. Otro beneficio es el de replicar el nodo de forma fácil sin comprometer todo el material criptográfico o incluso lanzar en paralelo un nodo de respaldo que comparta el mismo conjunto de claves.

Este proyecto ha sido uno de los premiados entre los 25 que recibieron financiación por parte de Ethereum Foundation como parte del programa de *Staking Grants* [3] lanzado en diciembre de 2020 y resuelto en febrero de 2021. Se encuentra dentro de la categoría «*New tooling*» que hace referencia a nuevas herramientas para que los validadores tengan un entorno más cómodo en el que participar en la nueva red de Ethereum.

## 1.2. Objetivos

El objetivo que se pretende abordar es dar el salto a sistemas distribuidos, en los que las operaciones y los datos del usuario se encuentran en un sistema de iguales como las redes P2P, en lugar de la ya conocida arquitectura Cliente/Servidor. Por tanto, se debe prestar especial cuidado al tratamiento de las claves, para garantizar al usuario la fiabilidad de dicho sistema tanto con sus datos como con sus operaciones.

En este proyecto se pretende analizar toda esta problemática desde tres puntos fundamentales:

- los usos con respecto a los datos (almacenamiento seguro y operaciones sobre datos cifrados)
- la gestión propiamente dicha de las claves (generación, almacenamiento y gestión)
- los usos de las claves con respecto al usuario (como la autenticación y la autorización).

Concretamente, se trata del diseño e implementación de una interfaz de programación de aplicaciones (API, por sus siglas en inglés) que permita a los clientes validadores de la *blockchain* de Eth2, la nueva versión de la red de



Ethereum, almacenar las claves privadas y que se puedan ejecutar en entornos más escalables. De esta forma, los nodos estarán escuchando a la API y devolverán la firma cuando sea requerida. Así pues, se pasará a un esquema de suministro de firmas BLS a la carta.



## Capítulo 2. *Blockchain*

### 2.1. Introducción

*Blockchain* aporta confianza sin la necesidad de un intermediario, a través de un sistema que permite anotar cualquier información directamente en un registro compartido entre varios usuarios con la tranquilidad de saber que esta anotación es legítima y perdurable; además asegura e impide que ninguno de estos usuarios podrá cambiar esa anotación en el futuro.

No existen servidores ni registros centrales que aporten la confianza que se necesita, solo los propios miembros o “nodos” de la red conectados entre sí, y son precisamente estos los que se aportan confianza recíprocamente.

De forma simplificada, la legitimidad de cada anotación en el registro es validada por consenso entre los nodos de la red sin que intervenga ningún registro central y, una vez validadas, las anotaciones se consolidan en la *blockchain* mediante un mecanismo de *hashes* encadenados y de firma digital que hace que ninguno de los nodos pueda modificar esta anotación en el futuro.

En la actualidad nos encontramos con diferentes casos de uso para esta tecnología, como la trazabilidad alimentaria, pero hay uno de ellos que destaca por encima del resto: los contratos inteligentes o *smart contracts*.

Los contratos inteligentes son el mejor ejemplo de la delegación y renuncia de la presencia de una tercera entidad que medie entre dos partes. A través de un código que se programa se pueden ejecutar órdenes una vez se satisfagan unas condiciones. Y todo ello sucede ocultando la información del contrato

para que sea de carácter confidencial entre los implicados. Si trasladamos este caso de uso a nuestro día a día veremos cómo procedimientos de traspasos o pólizas se pueden realizar con mayor agilidad y sin la necesidad de que un tercero compruebe y verifique el movimiento.

## 2.2. Bitcoin

Bitcoin es un proyecto de moneda virtual que fue lanzado en 2009 por Satoshi Nakamoto. La idea era crear una divisa que no dependiese de bancos centrales o Estados y que se gestionase a través de una base de datos pública sin la aprobación de terceros. La transparencia del protocolo, que es de código abierto, y su base matemática, apoyada en la criptografía, hacen que la confianza y adopción cada vez sea mayor por el público en general.

Curiosamente la identidad de su creador es desconocida y, a pesar de existir numerosas especulaciones sobre su persona, hoy en día todo son incógnitas. Fue en 2010 cuando G. Andresen tomó el control del proyecto tras recibir el acceso al repositorio de Nakamoto y, posteriormente, se concibió la Fundación Bitcoin en 2012.

Uno de los puntos que se suele comentar acerca de Bitcoin es su similitud con el oro y su característica como reserva de valor debido a la escasez. La cantidad máxima de la moneda está establecida en 21 millones de *bitcoins*, de los cuales existen más de 19 millones en circulación porque ya han sido minados. Como se puede observar la oferta es limitada, a diferencia del dinero fiduciario que depende de la emisión de la Reserva Federal en el caso del dólar o del Banco Central Europeo con el euro. La emisión de *bitcoins* depende de la recompensa ofrecida a los mineros que minan el nuevo bloque. El concepto de minado se define de forma más detallada en la explicación del algoritmo de *Proof of Work*, correspondiente al subcapítulo «Mecanismos de consenso». Actualmente está fijada en 6,25 *bitcoins*, pero esta cantidad se regula aproximadamente cada cuatro años, por un proceso llamado *halving*. Para entonces se añadirán 210.000 nuevos bloques y se habrá reducido a la mitad este premio.

Un problema recurrente dentro del mundo *blockchain* es la escalabilidad de las plataformas. En Bitcoin cada bloque dentro de la red se mina cada 10 minutos, tiempo determinado por la definición del protocolo y que es inmutable, y este incluye 2.188 transacciones de promedio. Si se quiere establecer a Bitcoin como una alternativa real a otras plataformas de pago como Visa y Mastercard, que soportan alrededor de 50.000 transacciones por segundo, es obvio que habría que aumentar el número de transacciones que se incluyen por bloque u ofrecer una alternativa tecnológica que se ajuste al protocolo para soportar estas necesidades. En la actualidad, la preferencia para que unas transacciones se efectúen por delante de otras se hace a través de comisiones a los mineros, pero esto es inviable cuando hablamos de movimientos de poca cantidad de fondos, donde la comisión puede superar el número de *Bitcoins* que se desea traspasar.

Debido a esto, surge *Lightning Network* [4] como una posible solución a esta encrucijada. Con ella se pretende que los nodos trabajen sin necesidad de guardar cada transacción en la cadena de bloques. El único requisito necesario para la creación de estos canales de pagos es sellar el estado inicial y final de los usuarios involucrados en la *blockchain*, es decir, al abrir y cerrar dichos canales. Todo este sistema independiente a Bitcoin se conoce como una solución *off-chain* o de capa 2.

Las transacciones que se realicen en esos canales son inmediatas ya que no requieren ninguna confirmación. Además, la cantidad mínima que se puede transferir equivale a 0,00000001 *bitcoins* mientras que en las transacciones que no hacen uso de *Lightning Network* esta cantidad asciende a 0,00000546 *bitcoins*. Esto es un escenario ideal para el mundo de los micro pagos, aunque también afloran posibles aplicaciones para el mundo de los contratos inteligentes. Cabe destacar que los canales son entre pares, pero se pueden interconectar más intervinientes siempre que haya un punto en común. Un usuario A puede tener un canal abierto con un usuario B, y a su vez este con un usuario C. En este caso, el usuario A también podría realizar transferencias al usuario C dando lugar a una potencial red de grandes magnitudes.

Como se ha mencionado anteriormente, en estos canales de pares se pueden realizar multitud de operaciones y el cierre se suele hacer de forma orgánica y

consensuada. Puede darse el caso de que solo una de las partes involucradas clausure el medio y suba a la *blockchain* el estado del canal con una transacción que le favorezca ya que no es necesario que sea la última. Por ejemplo, si el usuario A y el usuario B aportan 5 *bitcoins* cada uno al canal de pago y en primera instancia A recibe 2 *bitcoins* de B y posteriormente A envía 1 *bitcoin* a B, el saldo definitivo del primer usuario sería de 6 *bitcoins* y del segundo de 4 *bitcoins*. Pues técnicamente es posible que A cierre el canal con el estado obtenido tras la primera transacción, que le dejaría un saldo de 7 *bitcoins*. Sin embargo, para prevenir este fraude existe un mecanismo de espera de 3 días que impide que aquel que haya cerrado el canal de forma unilateral no pueda retirar los fondos. Es en ese lapso en el que el otro usuario puede formalizar una queja a través de una prueba de violación y si esta es fructífera puede suponer la pérdida de los fondos del usuario que ha realizado el fraude. Teniendo en cuenta las consecuencias y los largos tiempos de espera es poco coherente actuar de manera maliciosa en la red.

La implantación de *Lightning Network* cada vez está aumentando más, pero no se considera una solución definitiva por las limitaciones técnicas que conlleva. De todos modos, es un paso más para el avance de la red.

## 2.3. Ethereum

Ethereum es la segunda plataforma más importante dentro del mundo *blockchain*. Su popularidad viene dada por haber impulsado los contratos inteligentes, *smart contracts*, y las aplicaciones descentralizadas dentro de la red, a través de la Máquina Virtual de Ethereum (EVM, por sus siglas en inglés). Surgió en 2013 [5] cuando uno de sus cofundadores, Vitálik Buterin decidió apostar por un proyecto alternativo que solucionase las deficiencias de Bitcoin.

El mecanismo de consenso que usa en la actualidad es el de prueba de trabajo, pero desde hace unos años los desarrolladores están sentando las bases para realizar una migración de la red y que se use el algoritmo de prueba de participación.

A la hora de realizar las transacciones en la red de Ethereum hay que tener en cuenta el concepto de gas, el cual regula la ejecución de los contratos inteligentes y en función de su valor los mineros confirmarán antes o después el bloque en el que se encuentra dicha transacción. Es la unidad que nos permite medir el trabajo que se realiza en la red ya que no es lo mismo ejecutar un fragmento pequeño de código a poner en marcha un contrato inteligente que implementa varios bucles. El valor máximo es de 4,7 millones de unidades de gas por bloque.

Para programar los contratos inteligentes se han de usar lenguajes específicos como Serpent o Solidity. Estos lenguajes de programación son Turing completos [6], que significa que tienen un poder computacional igual a la Máquina de Turing Universal y podrían realizar cualquier cálculo si se tuviese acceso a recursos físicos ilimitados. Si se lleva al terreno de la cadena de bloques, se refiere a la aptitud para implementar bucles de los contratos inteligentes. Estos lenguajes hacen que la red sea sensible a ataques de *hackers*. De hecho, en 2016 ocurrió un evento [7] en el cual se produjo un robo de *ether* equivalente a 50 millones de dólares. El *ether* es el *token*, o vale, que usa la red de Ethereum. Como resultado se produjo una bifurcación o *hard fork* en el proyecto, por un lado, Ethereum Classic (ETC) se mantuvo fiel a la versión original y por otro, Ethereum Foundation modificó el código tras el ataque.

Este ataque se basó en la explotación de una vulnerabilidad que se hallaba en un contrato inteligente. El contrato inteligente lo que permitía era la compra de *tokens* de la empresa Decentralized Autonomous Organization (DAO). Los inversores recibían 100 DAO *tokens* a razón de 1 *ether*, con los cuales adquirirían derechos de voto para la selección de proyectos y su posterior financiación. El problema estaba en que dentro de las líneas de código que permitían la ejecución de ese *smart contract*, había una llamada recursiva a una función que permitió al ataque sustraer una inmensa cantidad de *tokens* cuyo valor en dólares superaba las decenas de millones. Ese fallo se produjo por un error en el diseño del proyecto, mediante el cual los creadores de “The DAO” buscaban proteger a la minoría y que pudiesen retirar sus fondos cuando una propuesta que no era de su gusto fuese elegida. Ese *ether* se transfirió a

una nueva DAO, llamada DAO hija, que tenía las mismas características y restricciones que la original. La función que permitía dividir los fondos estaba escrita de forma que primero retiraba el *ether* y luego actualizaba el balance y, dado que era una llamada recursiva, el atacante estuvo retirando los depósitos múltiples veces antes de que el código comprobase el balance. La comunidad debatió con profundidad acerca de este asunto ya que técnicamente no se había hecho nada ilegal en el sentido de que el programa permitía esas acciones y por tanto eran legítimas. Sin embargo, la confianza en el sistema quedó en entredicho y se barajaron tres opciones: no hacer nada y dejar la red tal cual estaba, realizar un *soft fork* por el cual los mineros destruirían el *ether* de la DAO hija, pero no se verían afectadas las transacciones realizadas antes de este ataque, o ejecutar un *hard fork* que sobrescribiese el historial y restaurase la red al punto previo del ataque. Finalmente, se optó por esta última alternativa.

## 2.4. Eth2

Al igual que Bitcoin, Ethereum también sufre de problemas de escalabilidad. Actualmente la red de Ethereum solo soporta unas 15 transacciones por segundo, un caudal muy bajo para una plataforma que soporta el peso de tantas aplicaciones diferentes. El caso más destacado de esta deficiencia es el de *CryptoKitties*, un juego que se apoya sobre la cadena de bloques de Ethereum y que en diciembre de 2017 congestionó la red al alcanzar un pico máximo de transacciones que ocuparon el 25 % de todo el tráfico de la red [8]. A pesar de este trágico accidente, parece que la historia se repite debido a que el mundo de las finanzas descentralizadas, *DeFi* en inglés, está aumentando cada día su actividad, por lo que la actualización del proyecto paralelamente resulta más necesaria. Para solucionar este cuello de botella surge Eth2, la nueva versión del protocolo que incluye varias implementaciones que permiten escalarlo.



Los principales cambios que se pretenden implementar dentro de la red de Eth2 son los siguientes:

- Cambio del mecanismo de consenso

Se pasará del algoritmo de prueba de trabajo donde los mineros crean nuevos bloques a la cadena del algoritmo de prueba de participación. Ahora ya no se necesitarán potentes equipos a nivel de *hardware*, que resuelvan problemas criptográficos, si no validadores que hayan depositado 32 *ether* a un *smart contract* específico. De entre todos los validadores se seleccionará a uno en función de unos parámetros específicos y será el encargado de añadir el nuevo bloque a la red. Ahora la recompensa recibida por ello será en base a comisiones.

- Introducción de *Beacon Chain*

De forma muy simplificada, *Beacon Chain* [9] se podría considerar como una solución de segunda capa. Se encargará de coordinar las cadenas de bloques individuales que correrán en paralelo, llamadas cadenas de fragmentos o *shard chains*. Estas no incluirán el registro completo por lo que serán más veloces al no tener que cargar sobre ellas el historial de la red. *Beacon Chain*, por tanto, hará la función de coordinador entre las cadenas de fragmentos y la cadena principal de prueba de trabajo, que será el ancla.

La hoja de ruta de este proyecto se desglosa en las siguientes tres fases:

- Fase 0: implementación de *Beacon Chain* que almacena y gestiona el registro de validadores y el mecanismo de consenso de prueba de participación.
- Fase 1: comienzo de despliegue de la red que permitirá alcanzar nuevas tasas de transacciones, unas 64 veces superior a la actual.
- Fase 2: cadena de bloques totalmente funcional y compatible con los *smart contracts*, además de mayor libertad a los desarrolladores para lanzar sus implementaciones.

## 2.5. Mecanismos de consenso

El algoritmo de consenso en una red de *blockchain* es el mecanismo que se utiliza para que los nodos estén conformes con la adición de nuevos bloques y de esta forma se puedan registrar. Así pues, se verifica que el elemento más reciente represente de forma correcta las transacciones efectuadas y así evitar datos que no se correspondan.

El más popular es el de prueba de trabajo, *Proof of Work* (PoW abreviado en inglés), ya que fue el pionero por haberse aplicado en la red de Bitcoin. Ahora bien, con el paso de los años surgen más algoritmos como la prueba de participación, *Proof of Stake* (PoS abreviado en inglés), que ofrecen ventajas pero también inconvenientes frente al archiconocido PoW.

El mecanismo de *Proof of Work* es el que está más extendido y destaca su uso en la *blockchain* de Bitcoin y Ethereum. Las transacciones se almacenan en una zona de memoria esperando a ser registradas y son los mineros los que, a través de la generación de un *hash* con un número  $N$  de ceros correcto, envían el bloque que las agrupa al resto de la red. Una vez aceptado dicho bloque, el usuario que ha resuelto el problema recibe las monedas como recompensa. La resolución de la operación se basa en la capacidad computacional de los nodos, ya que averiguar la cadena que originó dicho *hash* solo se puede hacer mediante fuerza bruta por tratarse de una operación criptográfica de un solo sentido. Con el paso de los años, la potencia de los elementos *hardware* encargados de resolver estos acertijos ha aumentado por lo que el número  $N$  de ceros se va modificando en el protocolo para que siempre se mantenga un tiempo medio entre bloques de diez minutos en el caso de Bitcoin.

Por otro lado, está el mecanismo de *Proof of Authority* (PoA abreviado en inglés), que se utiliza en las redes privadas. En este caso, los usuarios que participan en la red son entidades notorias por lo que no es necesario un despliegue similar al de *Proof of Work*. Aquellos participantes encargados de subir los bloques se establecen como autoridades y, a través de la firma con su certificado digital, validan las transacciones. Se elimina completamente el anonimato dentro de la red pero, por el contrario, se aceleran los tiempos, puesto que hay una confianza preestablecida.

El algoritmo de *Proof of Stake*, adquiere reconocimiento últimamente por ser el mecanismo de consenso que va a implementar la nueva red de Ethereum, Eth2, además de otras plataformas como Cardano. En este caso no hay que resolver ningún puzle matemático para ser el encargado de crear el bloque como ocurría en la Prueba de Trabajo, sino que se elegirá de manera aleatoria entre todos los participantes de la red que hayan enviado sus fondos a un depósito especial, y tienen mayor probabilidad de validar la operación aquellos que posean un mayor número de *tokens*. Con esta forma se reduce en gran medida el consumo de energía por parte de los participantes y, además, se consigue crear un verdadero interés de participación en la red.

Si analizamos las diferencias entre ambos mecanismos, observamos que PoW tiene una mayor huella ambiental que PoS y en la actualidad es una gran desventaja teniendo en cuenta que la red de Bitcoin supera en consumo anual de energía a grandes países como Noruega o Chile [10]. Además, los particulares están dejando de lado el minado de la moneda porque cada vez es más difícil competir contra las granjas de tarjetas gráficas dedicadas única y exclusivamente a esta tarea. Estos *pools* de minería concentran la mayor parte de la potencia de *hash* de la red, y provocan una centralización del ecosistema [11], [12] que atenta directamente contra los valores de *blockchain* y un mundo descentralizado. En el caso de PoS esto no sucede ya que para participar en la red simplemente se deben bloquear los *tokens* en un contrato inteligente (*staking*) y mientras se cumpla con los requisitos fijados, que pueden ser en función del tiempo o la cantidad de monedas, ya se opta a validar nuevos bloques. Esta diferencia de inversión inicial hace que se desincentive el gasto en *hardware* para redes PoW y en cambio se opte por redes PoS.

También se debe considerar entre sus diferencias la seguridad de la red. Siempre se ha puesto a PoW como un ejemplo de mecanismo que evita un ataque de denegación de servicio (DDoS, según sus siglas en inglés) donde el 51 % de la red opera de forma maliciosa. Sin embargo, como se ha descrito anteriormente, cada vez los *pools* de minería se están centralizando más y no se podría descartar un ataque de este tipo. Por el contrario, con PoS habría que poseer el 51 % de los *tokens* en circulación, lo cual es algo poco probable por el importe económico que ello implica. De todos modos, si esto sucediese,

el mecanismo incluye penalizaciones a aquellos validadores que se comporten de forma inadecuada, por lo que verían reducido su capital y, probablemente, el valor de la moneda también disminuiría.

## 2.6. Almacenamiento de las claves: *cold wallets*

Debido a la cantidad de robos de fondos de criptomonedas, el almacenamiento seguro de estas siempre ha sido un tema recurrente.

Los monederos fríos, comúnmente llamados *cold wallets*, son la opción que permite guardar las claves sin la necesidad de estar conectados a Internet. Su función se podría equiparar a la de una cuenta bancaria tradicional donde se guardan los ahorros. Debido a este hermetismo son el prototipo ideal para la protección frente a hackeos o sustracción de claves.

Las carteras o monederos de criptomonedas tienen dos vínculos, por un lado, la clave pública y por otro la clave privada.

La primera de ellas corresponde a la identificación del monedero y es la que se debe compartir para enviar o recibir *tokens*. En cambio, la clave privada es el acceso a la cartera y es de vital importancia mantenerla en secreto. Cabe recordar que los *tokens* no se almacenan en las carteras si no en la *blockchain*, y que lo único que se guarda en ellas son las claves.

A pesar de la comparación con la cuenta bancaria, la posesión de una *cold wallet* presenta un paso más allá en el mundo de la descentralización. Mientras que en el primer caso es el banco quien almacena y custodia los ahorros, con un monedero frío es el mismo titular quien asume esa responsabilidad. Ahora ya no se depende de un tercero como ocurre al mantener los fondos en el mercado de intercambio (*exchange*), los cuales ya han sido atacados a través de brechas de seguridad o repentinos cierres sin que los usuarios pudiesen hacer nada. Dentro de la jerga del ecosistema se suele hacer alusión a la expresión «*not your keys, not your coins*», que significa que si no se posee las claves realmente se delegan los fondos y no se tendría la posesión real de los mismos.

Además, en el caso de que el dispositivo *hardware* fallase, siempre existirá la vía de restaurar las claves a través de una semilla o palabras que guardaría el usuario.

No obstante, hace años cuando se hablaba de *cold wallets* solo existía la opción de monederos de papel, y con el paso del tiempo aparecieron los *hardware wallets* que son el objeto sobre el que se centra este trabajo.

Las carteras de papel ofrecen un nivel más alto de seguridad que las *hot wallets* respecto a posibles ataques informáticos. Eso sí, se pueden perder o estropear si no se tiene un cuidado mínimo. En el papel se guardan las claves públicas y privadas. Algunos ejemplos son Wallet Generator y Bitcoin Paper Wallet.

En cuanto a *hardware wallets*, como su propio nombre indica son instrumentos físicos de forma similar a un disco de almacenamiento externo USB. Como se ha mencionado anteriormente, el hecho de que su funcionamiento sea *offline* las protege de posibles robos. Además, ofrecen la protección vía PIN para proteger su uso frente a terceros en caso de pérdida. Las marcas más reconocidas que ofrecen este tipo de dispositivos son Trezor [13] y Ledger [14].



## Capítulo 3. Tecnologías implicadas

### 3.1. Esquema de firmas BLS

El esquema de firma digital, denominado BLS [15], [16] creado en 2001 por Boneh, Lynn y Shacham es un esquema de firma criptográfica que permite verificar la autenticidad de una firma. Se trata de un concepto muy atractivo ya que se puede comprimir un grupo de firmas  $S_1, S_2, S_3 \dots S_n$  en una sola firma que identifique a todos los usuarios. Si se traslada esto hacia el campo de la *blockchain* se observa un aumento en la velocidad de las transacciones, puesto que se puede verificar más rápido. A pesar de sus múltiples propiedades, la adopción fue lenta y es a partir de 2019 cuando el ritmo se acelera, en parte debido al impulso dado por el Crypto Forum Research Group (CFRG), que se encarga del proceso de estandarización de la criptografía para el Grupo de Trabajo de Ingeniería de Internet (IETF, por sus siglas en inglés). Además, en 2020 se empezaron a usar en Eth2, la nueva versión de Ethereum, para que los validadores verificasen las transacciones. Esto permitirá desatascar el cuello de botella a medio plazo en la *blockchain* aunque no son una solución definitiva puesto que se trabaja sobre criptografía, que de momento no es totalmente segura frente a la computación cuántica.

Todo parte del concepto de emparejamientos en las matemáticas, en concreto de la rama del álgebra lineal, y con aplicación directa en la criptografía. Si consideramos los grupos  $G_1$  y  $G_2$ , los cuales son finitos, cíclicos y aditivamente formulados. Se puede tomar un elemento de cada uno de ellos y se puede mapear hacia un tercer grupo  $G_T$ , que es finito, cíclico y formulado multiplicativamente. Las operaciones que se realizan tanto en  $G_1$  como en  $G_2$

son intercambiables, sin embargo, la elección de uso de uno y otro ofrece diferentes ventajas. El grupo  $G_1$  contiene puntos pequeños y es más rápido mientras que el grupo  $G_2$  contiene puntos grandes y es más lento. Al aplicar esto a la red Eth2, se usa al grupo  $G_1$  para las claves públicas por dos motivos, primero porque la agregación de claves públicas se realiza más a menudo que la agregación de firmas y segundo por el almacenamiento de las claves públicas a través de los validadores. Las firmas, por tanto, serán puntos del grupo  $G_2$ .

El nivel de seguridad se mide en *bits* y lo que se busca es un número lo suficientemente grande que haga muy complicado la resolución del problema del logaritmo discreto mediante  $2^n$  operaciones. Hoy en día  $n$  debe ser superior a 100. Para ello se trabaja sobre una curva elíptica, concretamente sobre la familia de curvas BLS12-381. El número 12 indica el grado de incrustación de la curva y el número 381 significa el número de *bits* del número primo en el campo finito.

En un principio, BLS12-381 pretendía ofrecer un nivel de seguridad de 128 bits, pero después de un análisis profundo se ha detectado un nivel de seguridad entre 117 y 120 *bits*, de modo que es un nivel de seguridad adecuado y suficiente.

La ecuación que define la curva para  $G_1$  es

$$y^2 = x^3 + 4 \quad (3.1)$$

mientras que la que define la curva de  $G_2$  es

$$y^2 = x^3 + 4(1 + \mathbf{j}) \quad (3.2)$$

Dentro del esquema de firma digital aparecen cuatro apartados fundamentales:

- Generación de la clave privada y clave pública: para la clave privada se selecciona un escalar aleatorio  $x$  dentro de la curva. Su tamaño es de 32 *bytes*. A partir de ella se genera la clave pública  $p$ , que tiene un tamaño de 48 *bytes*, de la siguiente forma:

$$p = xG \quad (3.3)$$

donde  $G$  es un punto dentro de la curva elíptica.



- Firma: su tamaño es de 96 *bytes*. Con la posesión de la clave privada  $x$  y un mensaje  $m$  se realiza la firma. Primero se realiza un *hash* del mensaje tal que

$$h = H(m) \quad (3.4)$$

para después obtener la firma de la siguiente manera

$$\sigma = h^x \quad (3.5)$$

- Verificación de firma: este paso se realiza al comprobar con la clave pública,  $p$ , y la firma,  $\sigma$ , que se cumple esta igualdad:

$$e(\sigma, G) = e(H(m), p) \quad (3.6)$$

- Agregación de firmas: esta opción aporta versatilidad ya que se reduce enormemente el tamaño del bloque sin renunciar a la información de tener cada firma. Además, es imposible conocer la firma original que dio paso a la firma agregada.

Un posible ataque [17] cuando se están agregando firmas sobre un mensaje puede darse cuando un usuario A en vez de publicar su clave pública devuelve una clave pública que es la clave pública más la inversa de otro usuario B. Es decir, el usuario B publica  $pk_B$  y el usuario A envía

$$pk_{A'} = x_A G_1 - pk_B. \quad (3.7)$$

Ahora el usuario B podrá firmar un mensaje  $m$  y publicarlo anunciando que ambos usuarios lo han firmado ya que la verificación se cumple. Una posible defensa ante este ataque, que ya se usa en Eth2, es forzar a los validadores a demostrar que poseen la clave privada y que corresponde con la clave pública que anuncian a la red. Así cuando un usuario se registre, firmará con una clave pública que se mantendrá para el resto de las firmas.

### 3.2. Microcontrolador nRF9160 DK

Nordic Semiconductor es uno de los fabricantes más importantes dentro del mundo de sistemas empujados de ultra bajo consumo. Sus «sistemas en un chip» (System on a Chip, SoC, abreviado en inglés) son empleados por varias de las marcas líderes a nivel mundial en multitud de diseños de Internet de las Cosas (IoT, abreviado en inglés). Dentro de su línea de productos destaca la placa nRF9160 DK, que ha sido la plataforma hardware sobre la que se ha sustentado este proyecto.

El microcontrolador nRF9160 DK es una de las mejores soluciones en cuanto a eficiencia energética y diseño compacto para IoT. Contiene un módem que soporta LTE-M y NB-IoT que lo hace ideal para aplicaciones que requieren de uso bajo de potencia.

En el ámbito de la seguridad cuenta con la tecnología ARM TrustZone la cual permite el aislamiento y la protección de zonas comunes y zonas seguras del *firmware* y la memoria. El rendimiento no se ve comprometido y se pueden construir aplicaciones IoT sólidas y seguras. Además, también se cuenta con el acelerador criptográfico ARM CryptoCell, el cual mejora todavía más la seguridad ya que ofrece recursos de criptografía que permiten proteger las implementaciones desarrolladas frente a las amenazas de los atacantes.

En la Figura 3.1 se detallan todos los elementos implicados en la placa.

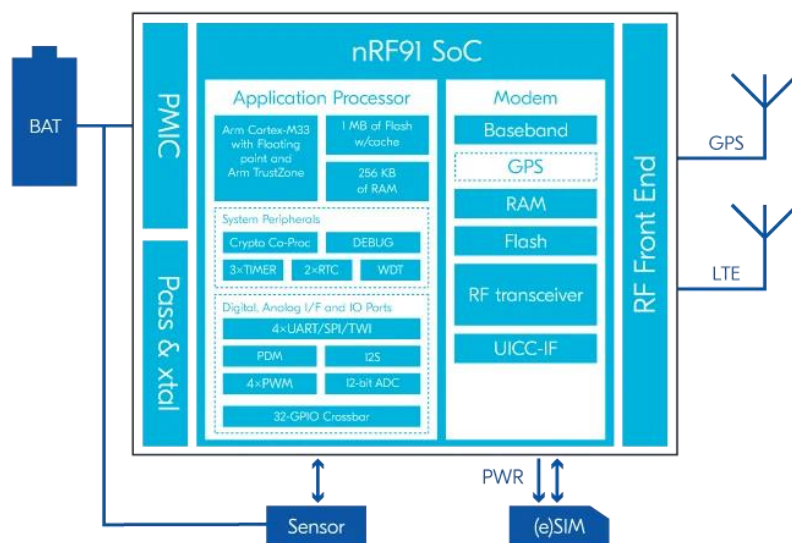


Figura 3.1. Arquitectura del nRF91 SoC

### 3.2.1. ARM TrustZone

El procesador Cortex-M33 de la placa nRF9160 hace uso de la arquitectura ARMv8-M. Una de las funcionalidades más interesantes que ofrece esta arquitectura es la tecnología llamada ARM TrustZone.

A parte de los típicos estados de operación de la CPU como puede ser el código tradicional y el manejador de excepciones, se introducen los atributos seguros y no seguros. Después de reiniciar la CPU siempre comienza en el modo seguro y puede acceder a todas las regiones de memoria y usar los periféricos sin restricciones.

La idea de TrustZone es la de ocultar todo lo que sea relevante a nivel de seguridad como código, datos y periféricos en una pequeña región del dispositivo. La división entre zonas seguras y no seguras depende del usuario al hacer uso de la biblioteca *Secure Partition Manager*, que ofrece la funcionalidad necesaria para controlar los entornos de ejecución de confiables. Esta separación se puede ver de forma más detallada en la Figura 3.2. Además, una vez se salta al *firmware* del modo no seguro no se puede acceder directamente a ninguna región del modo seguro sin que se lance una excepción de seguridad. Para ello, se debe invocar una pasarela especial que ofrece el modo seguro.

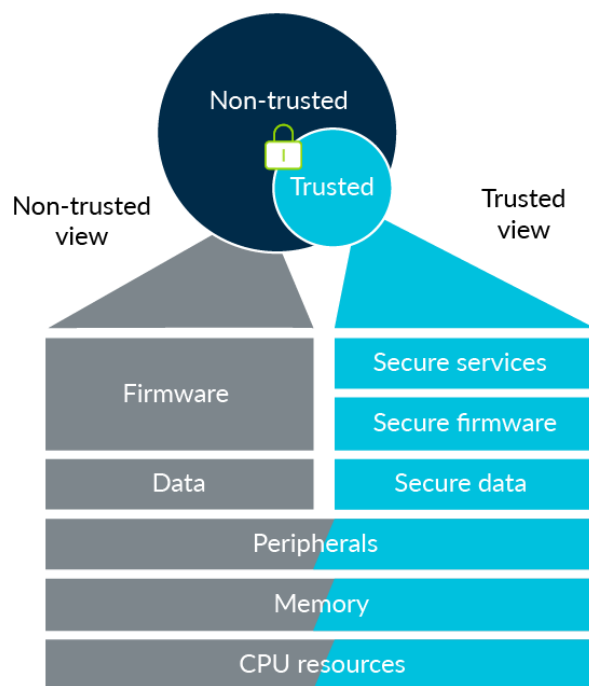


Figura 3.2. División de zonas con ARM TrustZone

Esta pasarela consiste en usar la etiqueta de llamada no segura, non-secure callable (NSC en inglés). De esta forma, la memoria segura reduce las restricciones y la pasarela puede acceder a esas instrucciones. Después, la pasarela tomará otro camino para tener acceso a la implementación de las funciones seguras que no han sido etiquetadas como NSC. Así, el proceso de llamadas al modo seguro desde el modo no seguro consiste en dos pasos. Por suerte, los compiladores de ARMv8-M se encargan de realizar internamente este desarrollo y el programador lo único que tiene que hacer es señalar mediante un atributo de modo no seguro que la función va a ser llamada desde ese entorno. A continuación, se muestra un ejemplo de cómo sería la cabecera de la función:

```
__attribute__((cmse_nonsecure_entry)) void ControlCriticoIO(){  
    // Código de la función  
}
```

### 3.2.2. Zephyr

Zephyr es un sistema operativo de tiempo real de código abierto que fue lanzado en 2016 y está especialmente diseñado para sistemas empotrados. Su desarrollo depende de Linux Foundation y desde su comienzo ha recibido multitud de apoyos por la comunidad y fabricantes. Actualmente es el sistema operativo de tiempo real que más contribuciones recibe comparado con sus competidores. Además, su sistema de compilación está basado en CMake que permite que las aplicaciones que hayan sido desarrolladas se puedan compilar tanto en Linux, macOS y Microsoft Windows.

Dentro de sus características principales se puede destacar la variedad de protocolos que incluye y permiten que las aplicaciones IoT diseñadas no tengan carencias. La apuesta por la seguridad es otro de los pilares básicos de este proyecto. Hay un equipo dedicado exclusivamente al análisis del núcleo y con el esfuerzo de la comunidad se mantiene un código sin apenas vulnerabilidades. De hecho, tiene añadido el *framework* Trusted Execution Environment.

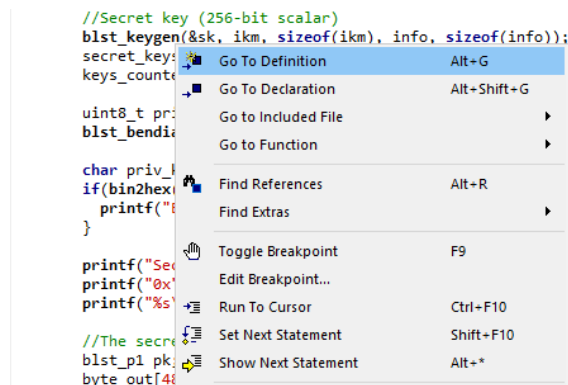
### 3.2.3. SEGGER Embedded Studio

En cuanto al entorno de desarrollo se ha usado SEGGER Embedded Studio (SES), el cual está recomendado por el propio fabricante de la placa nRF9160, Nordic Semiconductor. La principal herramienta que ofrece SES es la depuración para dispositivos ARM Cortex, que ha sido de mucha utilidad a lo largo del proyecto para realizar un análisis paso a paso de la ejecución del código.

Entre las diferentes herramientas usadas para la depuración destacan las siguientes:

- Go To Definition

Gracias a ella (Figura 3.3) se puede realizar un seguimiento exhaustivo de las funciones que se van encadenando internamente dentro de la biblioteca.



**Figura 3.3. Instrucción Go To Definition**

- Control de expresiones

Esta herramienta de depuración (Figura 3.4) permite controlar el valor de las variables línea a línea. En el caso del presente proyecto se ha vigilado concretamente que el paso de binario a hexadecimal y viceversa sea el correcto y que las claves no sean corruptas.

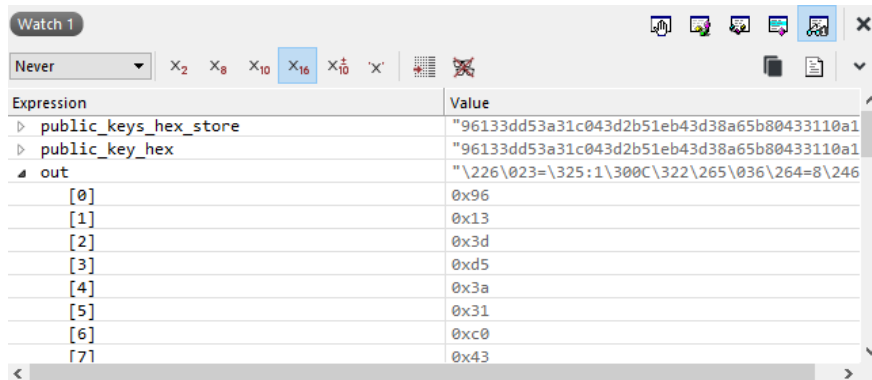


Figura 3.4. Control de expresiones en SEGGER Embedded Studio

- Tamaño de la pila

En el desarrollo de trabajos sobre plataformas *hardware* de capacidad limitada resulta muy importante un control exhaustivo de la gestión de la memoria. En la Figura 3.5 se muestra el tamaño de la pila tras la ejecución parcial del comando de generación de clave privada y clave pública y las hebras implicadas.

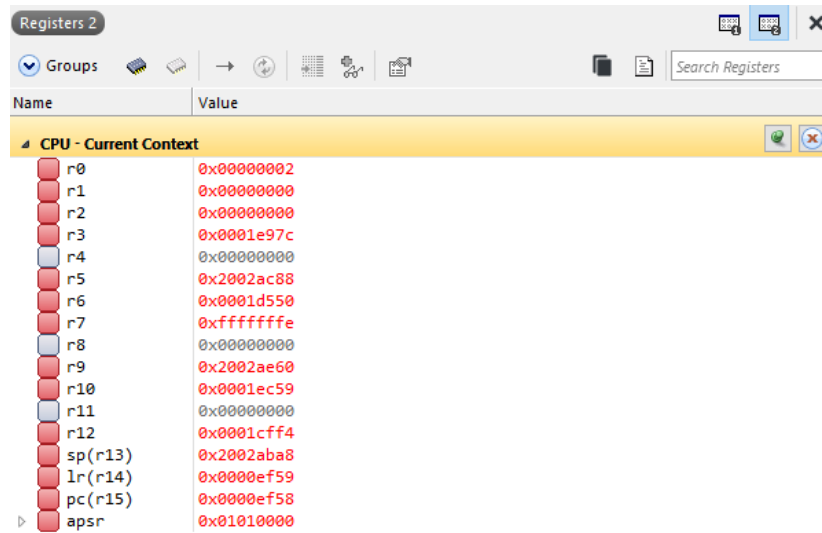
The screenshot shows the 'Call Stack' window in SEGGER Embedded Studio. It displays a list of functions, their call addresses, stack pointers, and the amount of stack used. The functions are: 'int cmd\_keygen(const shell\* sh', 'int execute(const shell\* shell=0', 'void shell\_process(const shell\*', 'void shell\_signal\_handle(const', 'void shell\_thread(void\* shell\_h', and 'void z\_thread\_entry(void(void \*,', '0xAAAAAA6', '0x2002AFC0', '0 bytes'.

Function	Call Address	Stack Pointer	Stack Used
int cmd_keygen(const shell* sh	0x0000EF58	0x2002ABA8	1,048 bytes
int execute(const shell* shell=0	0x0001197C	0x2002AE18	424 bytes
void shell_process(const shell*	0x00011A98	0x2002AEB8	264 bytes
void shell_signal_handle(const	0x0001AAF8	0x2002AF88	56 bytes
void shell_thread(void* shell_h	0x0001224C	0x2002AFA0	32 bytes
void z_thread_entry(void(void *,	0x00019EB2	0x2002AFB8	8 bytes
0xAAAAAA6	0x2002AFC0	0 bytes	

Figura 3.5. Tamaño de la pila durante la ejecución del código

- Registros

En función de las necesidades del proyecto puede ser preciso realizar un control a bajo nivel. En la Figura 3.6 se muestra como SES ofrece la posibilidad de realizar este tipo de análisis.



Name	Value
<b>CPU - Current Context</b>	
r0	0x00000002
r1	0x00000000
r2	0x00000000
r3	0x0001e97c
r4	0x00000000
r5	0x2002ac88
r6	0x0001d550
r7	0xffffffff
r8	0x00000000
r9	0x2002ae60
r10	0x0001ec59
r11	0x00000000
r12	0x0001cff4
sp(r13)	0x2002aba8
lr(r14)	0x0000ef59
pc(r15)	0x0000ef58
apsr	0x01010000

**Figura 3.6. Estado de los registros**

Por último, y de forma complementaria se ha combinado la depuración con el terminal *Putty* mediante el puerto serie con una configuración de 115.200 baudios.





## Capítulo 4. Análisis

### 4.1. Objetivo

El problema de ejecutar operaciones criptográficas fuera de la lógica principal del programa se suele resolver usando un entorno seguro o un módulo de seguridad *hardware*. Desafortunadamente, el soporte para firmas BLS es escaso en la actualidad. Hay algunas implementaciones en diferentes lenguajes de programación, pero no existen soluciones definitivas para usar en clientes de firmas BLS externos. Esto es un factor limitante para los validadores de Eth2 que quieran proteger su material criptográfico.

Existen algunas medidas en esta línea que intentan mejorar la situación actual. Por un lado, la Propuesta de Mejora de Ethereum 3030 (EIP-3030 [18], según sus siglas en inglés) abre la puerta a la delegación de firmas a una entidad externa usando una conexión TLS segura. Este mismo enfoque también lo apoya parcialmente Prysmatic Labs [19] y Consensys [20] para sus clientes, aunque el código aún no está listo para un uso definitivo, puesto que no está completamente implementado. Por ejemplo, ninguno de ellos soporta el uso de un módulo de seguridad *hardware* para firmas BLS y generación de claves, simplemente permiten el almacenamiento de claves en un HSM.

El almacenamiento de claves solo protege la clave cuando se migre el nodo. Pero en un nodo operacional si el atacante logra comprometerlo, es decir, gana acceso de forma remota, no hay diferencia entre guardar la clave en texto plano o en un HSM, ya que nada podría parar al atacante de leer la clave. Por tanto, hay una clara necesidad de desarrollo de un módulo externo que sea

capaz de realizar firmas BLS y, a la vez, que proteja todo el material criptográfico usando *hardware* seguro, que pueda ser conectado directamente al sistema o incluso de forma remota.

## 4.2. Impacto y retos

Para alcanzar el objetivo se ha usado la placa nRF9160 que almacena la clave privada y opera con ella. Gracias a sus características *hardware* como la tecnología ARM TrustZone, es posible aislar y proteger zonas seguras y no seguras del *firmware* y elementos como la memoria o periféricos.

Además, cuenta con soporte 4G/LTE que proporciona conectividad completa y LEDs RGB que sirven para monitorizar el correcto funcionamiento del nodo y así el usuario puede tener retroalimentación.

Los impactos y beneficios que este proyecto aporta sobre el ecosistema de validación de Eth2 cuando cualquier usuario que lance un nodo validador son:

- incrementar la seguridad y resiliencia de la red.
- reducir la presión a los desarrolladores y operadores de los clientes de la red.
- proveer una herramienta lista para usar, que protege todo el material criptográfico necesario para ejecutar un nodo.

Durante la realización de este proyecto se han encontrado una serie de retos que ha habido que gestionar y resolver para la consecución del desarrollo final.

Las bibliotecas que implementan criptografía BLS en el lenguaje de programación C para sistemas empujados no son un recurso abundante. Además, las placas del fabricante Nordic Semiconductor no ofrecen soporte específico para este tipo de esquema de firma, por lo tanto, había que lograr la implementación de código que procesase rápidamente las operaciones para que alcanzase la máxima eficiencia en la *blockchain* de Eth2.

El análisis preliminar del proyecto se inició con la revisión de emparejamiento criptográfico y sus posibles implementaciones en C. Algunas de las diferentes opciones que se barajaron fueron la biblioteca blst de Supranational [21], “The

Pairing-Based Cryptography library” de Ben Lynn [22], RELIC toolkit [23] y “The MIRACL Crypto SDK” [24].

Ethereum Foundation y Protocol Labs han apostado por la biblioteca de blst, para la cual realizaron una solicitud de propuesta [25] (Request For Proposal, RFP) con la finalidad de realizar una revisión de seguridad al código y establecer una referencia para futuras implementaciones de los desarrolladores. A partir de esta premisa, se ha elegido que el proyecto también pivote sobre este *software*. Así pues, blst se ha convertido en la biblioteca de firma criptográfica por defecto para los clientes de Eth2. Para asegurarse de que se cumplen con los estándares requeridos para la red principal (*mainnet*) se ha buscado a una organización para que realice una auditoría tanto a bajo nivel, como de optimización de código y vinculaciones con otros lenguajes, como pueden ser Rust y Go. En este caso ha sido NCC Group, empresa puntera en el sector de la consultoría de ciberseguridad quien se ha encargado de este trabajo [26].

Paralelamente, se ha realizado un estudio en profundidad del protocolo de Eth2 y sus requisitos para verificar que la solución encaja en la nueva implementación de la *blockchain*.

Dentro de los desafíos que han aparecido a la hora de diseñar el proyecto destacan los siguientes:

- Encontrar el *hardware* adecuado.

Se ha hecho una búsqueda de diferentes plataformas sobre las que trabajar y, finalmente, se ha decidido que la implementación se realice sobre nRF9160 SiP de Nordic Semiconductor, que incluye el procesador ARM Cortex-M33 con soporte a la tecnología TrustZone. Esta permite proteger el material criptográfico y el procesador es lo suficientemente potente como para lanzar la API de BLS. Además, la placa nRF9160 consta de un módem LTE-M que permitirá en el futuro implementar, por un lado, un módulo de seguridad *hardware* tradicional y, por otro, uno remoto en el mismo *hardware*.

- Implementar BLS en nRF9160.

Uno de los desafíos más importantes que surgieron a lo largo del proyecto ha sido trasladar la biblioteca de blst, optimizada para arquitecturas como x86 o ARMv8 a la arquitectura de nuestro microcontrolador. En este caso se ha realizado una compilación cruzada con la toolchain de GNU para sistemas empujados del archivo binario de blst para la arquitectura Cortex-M33 con los *flags* adecuados para el correcto funcionamiento.

- Implementación de una interfaz USB.

En paralelo con el trabajo de desarrollo de la funcionalidad de BLS, este se ha plasmado vía USB con la UART. Se han estudiado las interfaces de Prysmatic Labs y Consensys para desarrollar una API similar que trabaje por el puerto serie y así integrar nuestro módulo de seguridad *hardware* en sus proyectos fácilmente.

### 4.3. Trabajos previos

La elección del sistema *hardware* ha estado motivada por tres factores principales:

- el bajo coste del dispositivo, que hace que sea más accesible para todos los usuarios que quieran formar parte de este ecosistema.
- la comunidad de usuarios que está constantemente apoyando de forma activa en foros del fabricante.
- la versatilidad de estos microcontroladores para realizar operaciones criptográficas computacionalmente pesadas en ellos, que se muestra en una tesis doctoral [27].

Ese trabajo ha demostrado que la placa nRF9160 es capaz de soportar el emparejamiento bilineal y las funciones relacionadas con este tipo de criptografía han sido testadas para comprobar que, efectivamente, el procesador Cortex-M33 de 64 MHz es capaz de ejecutar el código, no sin antes realizar algunas optimizaciones.

Estas optimizaciones se resumen en rutinas de multiplicación y distribución del código a la RAM. Dado que el procesador soporta instrucciones de

multiplicación de 64 bit, las operaciones de cuadratura, reducción de campo finito y multiplicaciones de Karatsuba [28] (algoritmo que permite la multiplicación de números grandes de forma eficiente) fueron optimizadas. Por ejemplo, con esta última se logró reducir de 4.580 ciclos a 2.770 ciclos. De todos modos, como la caché de nRF9160 es de solo 2.048 *bytes* hubo que mover dichos algoritmos a la RAM sin afectar a la búsqueda de instrucciones puesto que tienen una velocidad similar.

El resultado tras la ejecución de la biblioteca arrojó un tiempo de 410 ms, que es suficiente para los casos de uso en los que no se requiera de forma frecuente el cálculo computacional de estas operaciones. De hecho, fue el mejor entre diferentes plataformas de baja potencia (<100 MHz) aunque la mayoría de las implementaciones se realizan sobre procesadores de 1 GHz o en FPGAs personalizadas.

Otro punto que se ha tenido en cuenta de dicha tesis es que se realizó sobre la curva BN254, la cual no es la más eficiente y se compara con las curvas BLS que ofrecen una reducción de 80 ms y son las que se usan en este proyecto. De todos modos, para conseguir un rendimiento por debajo de 100 ms habría que cambiar a otras soluciones *hardware*.



## Capítulo 5. Desarrollo

### 5.1. Arquitectura

En la actualidad los clientes que se conectan a la *blockchain* de Ethereum almacenan las claves y realizan las operaciones de firma en la misma lógica del sistema que conforma el nodo. Como se puede observar en la Figura 5.1 esto es un punto crítico y si los atacantes consiguen explotar alguna vulnerabilidad de la máquina podrían tener total acceso a las claves y fondos del usuario. A pesar de los esfuerzos de los desarrolladores por ofrecer soluciones seguras a este planteamiento nunca se podrá generar una confianza total del usuario a este sistema, ya que de forma intrínseca es potencialmente vulnerable. Es cierto que la seguridad plena es un concepto utópico, pero sin duda hay enfoques, como el que se describe a continuación, que pueden implementar esto de forma más fiel.

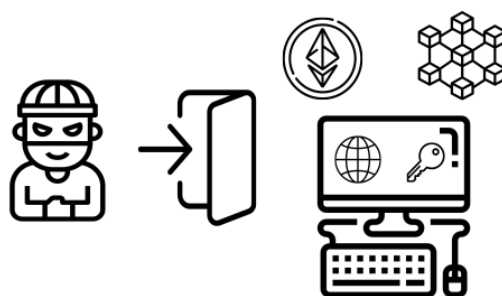


Figura 5.1. Arquitectura tradicional en Cliente de Ethereum

La arquitectura del módulo de seguridad *hardware* permite trasladar todo el material crítico a un entorno seguro. El nodo que está conectado a la cadena de bloques se comunicará por el puerto serie a través de comandos con la placa y esta devolverá la información requerida. En la Figura 5.2 se observa

que de esta forma se añade una capa más de seguridad al sistema. Además, dicha capa no presenta el potencial de ser atacada por un *hacker*, como si ocurría en la arquitectura tradicional, ya que la conexión es *offline* y la operabilidad con el HSM está muy restringida. Ahora toda la operativa ha de realizar un paso más. Como en todo proyecto de ingeniería se busca un compromiso entre seguridad y eficiencia, sin embargo, y como se refleja en el apartado Pruebas y resultados, los tiempos requeridos para realizar la operativa criptográfica en un sistema mucho menos potente no suponen un factor limitante a la hora de interactuar con la cadena de bloques y además se logra crear una barrera de protección adicional.

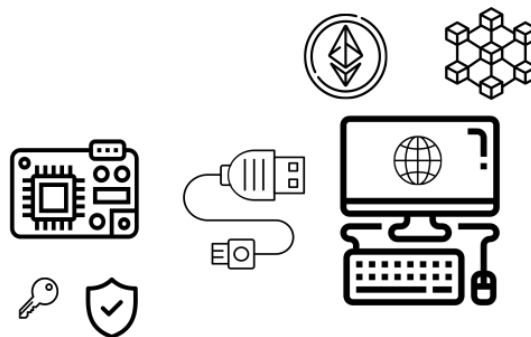


Figura 5.2. Arquitectura con módulo de seguridad hardware

## 5.2. Diseño

El diseño del sistema consiste en replicar el estándar EIP-3030 de la API HTTP para firmas BLS de forma remota y que sería consumido por los clientes de los validadores a la hora de firmar bloques en Eth2.

Esta solución comprime el esquema de ejecutar un nodo especializado, con acceso exclusivo a las claves privadas, que escucha a una API y devuelve las firmas solicitadas. Para ello, la especificación define una serie de comandos.

- GET /upcheck  
Si el nodo está operativo devuelve en una estructura JSON la respuesta «OK».
- GET /keys  
Devuelve los identificadores de las claves públicas que se encuentran disponibles para la firma.



- POST /sign/:identifier

Dentro del cuerpo de la solicitud se incluyen varios campos obligatorios: el dominio de la firma BLS, los datos que se van a firmar, un objeto *fork* que incluye versiones previas y actuales, y la raíz del génesis de los validadores que se trata de un *hash* SHA-256.

Puede devolver la firma en el campo JSON *signature* a través de un array de *bytes* codificado como una cadena de caracteres en hexadecimal. Si por el contrario se produce un error en la solicitud de la firma o el identificador de la clave no existe se devuelve un error.

El protocolo solo recoge estos tres métodos. No se definen métodos adicionales para la autenticación, administración de claves o transporte cifrado.

Tras el estudio del protocolo se ha procedido a desarrollar un código que emula el funcionamiento de una API a través de unos comandos por el puerto serie. Además de incluir aquellos que están recogidos en el documento EIP-3030 se han elaborado otros propios.

Los comandos soportados son los siguientes:

- keygen

Genera una clave privada de 32 *bytes*. Para ello se reciben cuatro argumentos de entrada: un vector IKM (*Interpretative Key Management*) de al menos 32 *bytes* que no sea posible deducir y por lo tanto se genera de una fuente aleatoria, el tamaño de IKM, un vector info que es opcional y se usa para derivar múltiples claves a partir del mismo vector IKM, y el tamaño de info. Por defecto es una cadena vacía. En nuestro caso se ha optado por un array de 32 *bytes* lleno de ceros.

Para la generación aleatoria de IKM se ha optado por un método disponible en los Servicios Seguros de la SDK de Nordic Semiconductor. Está implementado para el *firmware* seguro pero que se encuentra disponible para ser llamado desde la zona no segura de este. Esta función devuelve un número aleatorio de 144 *bytes* que sirve como entrada para realizar la operación *hash* SHA-256 que retorna un parámetro de 32 *bytes*. A continuación, se presentan las líneas de código encargadas de realizar las operaciones anteriores.

```
ret = spm_request_random_number(random_number,
random_number_len, &olen);

ocrypto_sha256(ikm, random_number, random_number_len);
```

Una vez se tienen todos los parámetros listos se llama a la función *keygen*, la cual devuelve un escalar y al ejecutar *bendian\_from\_escalar* ya se consigue la representación en binario de esa clave privada.

```
//Secret key (256-bit scalar)

blst_keygen(&sk, ikm, sizeof(ikm), info, sizeof(info));

blst_bendian_from_scalar(priv_key_bin, &sk);
```

Por último, se usan funciones propias de Zephyr, el sistema operativo sobre el que se ejecuta la placa nRF9160, para tener en hexadecimal el valor de la clave.

```
bin2hex(priv_key_bin, sizeof(priv_key_bin), priv_key_hex2,
sizeof(priv_key_hex2));
```

Posteriormente se genera una clave pública de 48 *bytes* a partir de la clave privada. Para ellos se usa la función *sk\_to\_pk\_in\_g1*, ya que se está en la variante de mínimo tamaño de clave pública [29] donde estas son puntos en  $G_1$  y las firmas son puntos en  $G_2$ . Además, se llama al método que nos permite obtener el punto afín de clave pública, que luego será un parámetro de entrada para la verificación de la firma. También se guarda en un array para su almacenamiento y posterior elección del usuario a la hora de la firma del mensaje.

```
blst_sk_to_pk_in_g1(&pk, &sk);

blst_p1_to_affine(&pk2, &pk);

blst_p1_compress(out, &pk);
```

En la Figura 5.3 se ilustra la comunicación entre cliente y HSM para realizar la operativa descrita anteriormente.

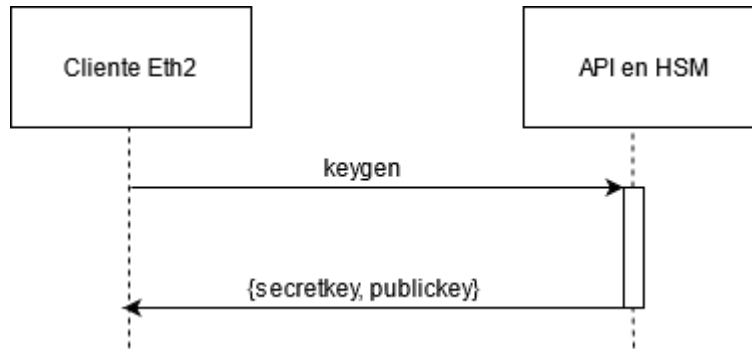


Figura 5.3. Generación de claves

- `publickey`

Muestra la última clave pública que se genera a través del comando `keygen`, que produce un par de llaves para el usuario (Figura 5.4).

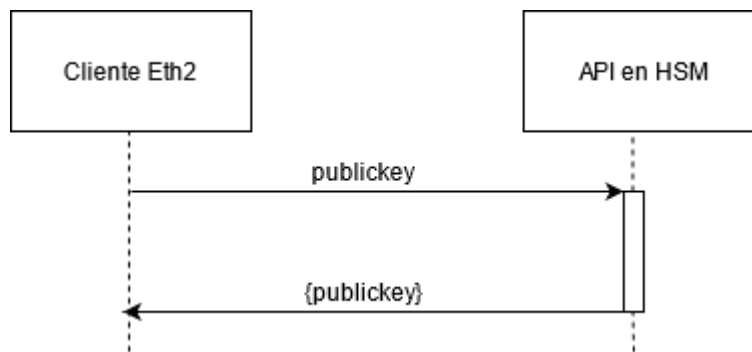


Figura 5.4. Comando `publickey`

- `signature <publickey> <message>`

Genera una firma de 96 *bytes* sobre el mensaje recibido como parámetro y con la clave pública que haya sido seleccionada. Previamente se ha de ejecutar un método propio llamado `public_key_to_sk` que nos devuelve el escalar correspondiente a la clave pública recibida como una cadena de caracteres en hexadecimal. El escalar es uno de los parámetros necesarios para realizar la operación de firma.

Después, se realiza un *hash* del mensaje recibido como argumento. En este caso se llama al método `hash_to_g2` donde se obtiene como parámetro de salida el punto en  $G_2$ , que representa al *hash* del mensaje. Esto ocurre tras recibir como entrada el mensaje en binario y su tamaño, la etiqueta de cifrado

correspondiente a la prueba de posesión [30] «BLS\_SIG\_BLS12381G2\_XMD:SHA-256\_SSWU\_RO\_POP» y el tamaño de la cadena.

```
blst_hash_to_g2(&hash, msg_bin, sizeof(msg_bin), dst,
sizeof(dst), NULL, 0);
```

Finalmente se usa la función *sign\_pk\_in\_g1* que devuelve el punto en  $G_2$ , que representa a la firma, una vez se ha recibido como parámetros de entrada el punto  $G_2$  correspondiente al *hash* del mensaje y el escalar de la clave privada. A parte se consigue el punto afín de la firma para la verificación y se serializa en un array para su posterior representación en hexadecimal.

```
blst_sign_pk_in_g1(&sig, &hash, &sk_sign);
blst_p2_to_affine(&sig2, &sig);
blst_p2_compress(out2, &sig);
```

En la Figura 5.5 se refleja la sucesión de pasos entre las partes involucradas para la ejecución del comando *signature*.

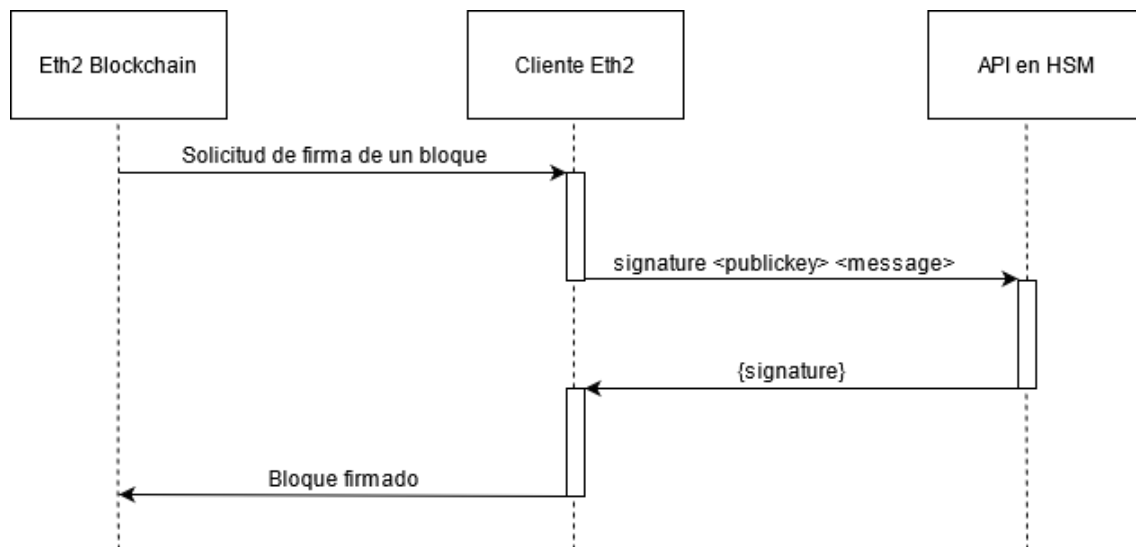


Figura 5.5. Traza de la ejecución del comando *signature*

- verification

Se comprueba la firma a partir de la clave pública. Este comando hace uso de la función `core_verify_pk_in_g1` para recibir los puntos afines de la clave pública y la firma, el mensaje y su tamaño, y la etiqueta de cifrado y su tamaño. En caso de que la verificación sea exitosa se devuelve BLST\_SUCCESS.

```
blst_core_verify_pk_in_g1(&pk2, &sig2, 1, msg_bin,
sizeof(msg_bin), dst, sizeof(dst), NULL, 0);
```

- getkeys

Este comando lista un máximo de 10 claves públicas, las cuales han sido generadas previamente. Solamente a través de ellas se podrá realizar la firma de un mensaje con éxito (Figura 5.6).

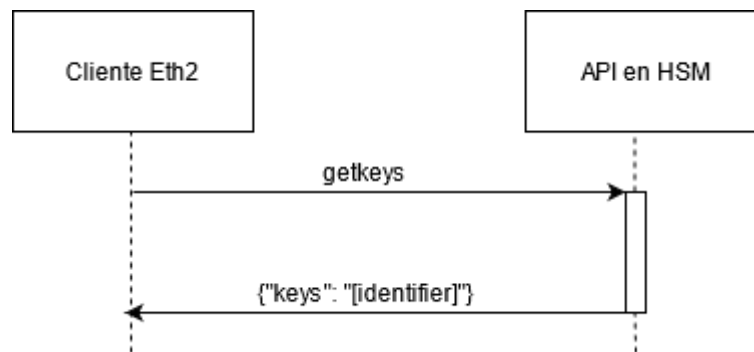


Figura 5.6. Comando getkeys

- benchmark

Ejecuta el código desde la generación del par de claves hasta la firma y devuelve los tiempos que se han empleado para la realización de cada operación criptográfica. Para las medidas de los tiempos se han usado los temporizadores propios del sistema operativo de tiempo real Zephyr.

```
K_TIMER_DEFINE(my_timer, NULL, NULL);
k_timer_start(&my_timer, K_MSEC(3000), K_NO_WAIT);
time_left = k_timer_remaining_get(&my_timer);
k_timer_stop(&my_timer);
```

## 5.3. Implementación

El desarrollo del código ha seguido dos fases diferenciadas. En la primera se trabajó sobre un código estático donde no había interacción con el usuario y en la segunda se dotó de dinamismo a través de una comunicación por comandos. Dentro de cada fase había dos vías distintas, la que usaba la biblioteca estática y la que usaba los archivos del código fuente.

Por un lado, el primer enfoque fue con la biblioteca estática pero debido a los desbordamientos de pila sufridos con la inclusión de esta se optó por dejarla en un segundo plano. Cuando los avances con la implementación vía código fuente de los propios archivos del repositorio de blst permitieron retomar de nuevo esta primera opción se volvió a ello ya que era una versión más atractiva por la sencillez de cara al usuario final para cuando finalmente el proyecto viese la luz.

En el repositorio de GitHub [31] se puede observar la implementación de ambas versiones, además de contar con un manual de instalación del entorno para el usuario.

### 5.3.1. Primera iteración

Al principio lo que se buscaba era obtener los mismos resultados que los vectores de prueba ofrecidos en los distintos repositorios de las implementaciones de BLS12-381 para Eth2. Para ello se programó un código de formato similar al que se ejecuta con el comando *benchmark* pero se eliminó la aleatoriedad en la generación de la clave privada y usó una prefijada en formato hexadecimal.

Anteriormente se han comentado los problemas que hubo con la implementación a través de la biblioteca estática. Esta agrupa las funcionalidades criptográficas que son de interés, pero la interfaz que sirve de comunicación simplemente es un fichero que actúa como contenedor con los diferentes archivos de código objeto. Así, a la hora de depurar los diferentes problemas nos encontrábamos con una «caja negra» que recibía una entrada y devolvía una salida, pero de la cual se desconocía su funcionamiento interior.

La única referencia al código visible era el archivo de cabeceras *blst.h*<sup>1</sup>, el cual solo incluía los prototipos de las funciones, sin saber en qué línea y en qué función se producía el fallo. De todos modos, la implementación de la versión con la biblioteca estática y la versión con los ficheros fuente iba a guardar muchas similitudes por lo que todo el desarrollo que se lograra en una parte se podría trasladar a la otra y contando con que la agilidad para la depuración iba a ser mayor. De hecho, las únicas diferencias que hay son la inclusión de los ficheros de código fuente, que implementan las funciones cuya cabecera está en *blst.h*, y los tipos que definen los puntos en  $G_1$  y  $G_2$  y sus puntos afines.

Finalmente se optó por trabajar sobre la línea del código fuente y se obtuvieron resultados más concluyentes. Cuando se llegaba a una asignación de memoria en el fichero *e2.c* aparecía el siguiente error:

```
z_arm_bus_fault()

UsageFault_Checking Error

#error Unknown ARM architecture

#endif /* CONFIG_ARMV6_M_ARMV8_M_BASELINE */

SECTION_SUBSEC_FUNC(TEXT, __fault, z_arm_exc_spurious)

    mrs r0, MSP

    mrs r1, PSP

    push {r0, lr}
```

Siguiendo las llamadas recursivas de las funciones de *blst* se observó que había llamadas a funciones-macro en el archivo *ec\_mult.h*<sup>2</sup>, las cuales recibían parámetros en tiempo de ejecución y podían causar eventos críticos en el avance del código. Para resolver esta encrucijada, se editaron esas funciones de forma que los parámetros que se iban a recibir vendrían ya definidos. Atacando este problema desde una perspectiva estática se podría prever una solución al problema de desbordamiento de pila. El problema de esta solución

---

<sup>1</sup> Disponible en <https://github.com/supranational/blst/blob/master/bindings/blst.h>

<sup>2</sup> Disponible en [https://github.com/supranational/blst/blob/master/src/ec\\_mult.h#L68](https://github.com/supranational/blst/blob/master/src/ec_mult.h#L68)

es que se modificaba directamente el código elaborado por Supranational en lugar de hacer un uso puro de la herramienta. A pesar de todo, los problemas no se resolvieron por completo y finalmente la solución pasó por aumentar la variable que definía el tamaño de pila en los archivos generados al compilar el proyecto, concretamente el fichero `.config` dentro de la ruta `/build/zephyr`.

Dado que la solución de reescribir parte del código no era una buena práctica y después de observar que la variación del tamaño de la pila sí fue un hecho definitivo en la compilación del proyecto, se deshizo el paso de edición de las macros. Ahora ya se podía ejecutar la implementación de blst en la placa nRF9160 sin que la configuración por defecto del proyecto para el *hardware* afectase y sin haber editado el código original salvo por una excepción. Dicha excepción se encontró en la línea 370 del fichero `vect.h`<sup>3</sup> y ha sido comentada omitiendo así su presencia para no sufrir errores de compilación. El traslado a la versión de la biblioteca estática no requirió de ningún esfuerzo adicional y además arrojaba mejores resultados debido a la optimización producida al encapsular todo el código fuente en un archivo binario.

### 5.3.2. Segunda iteración

Tras obtener los resultados correctos, se dividió el código en bloques para ser ejecutado por comandos como una API, partiendo de uno de los ejemplos que ofrece Zephyr. Las modificaciones en la configuración del ejemplo base han estado enfocadas en evitar los desbordamientos de pila previamente mencionados gracias al aumento de esta a 49.152 *bytes*. Estos desbordamientos fueron detectados con el analizador de hebras al ir ejecutando de manera consecutiva los comandos implementados. En el archivo `prj.conf` se habilitó esta funcionalidad además de seleccionar el tamaño de la hebra UART.

---

<sup>3</sup> Disponible en <https://github.com/supranational/blst/blob/master/src/vect.h#L370>



El siguiente fragmento de código muestra lo descrito anteriormente.

```
CONFIG_THREAD_ANALYZER=y  
CONFIG_THREAD_ANALYZER_USE_PRINTK=y  
CONFIG_THREAD_ANALYZER_AUTO=y  
CONFIG_THREAD_ANALYZER_AUTO_INTERVAL=10  
CONFIG_SHELL_STACK_SIZE=49152
```

Una característica adicional al proyecto es la inclusión del *backend* de seguridad de Nordic y la biblioteca *nrf\_oberon*, que consta de una colección de algoritmos criptográficos optimizados para la arquitectura Cortex-M33 entre otras. Dentro del archivo de configuración *prj.conf* se encuentran las líneas que ajustan estos detalles:

```
CONFIG_NORDIC_SECURITY_BACKEND=y  
CONFIG_NRF_OBERON=y
```

### 5.3.3. Pruebas y resultados

Una vez implementado el código que permite el funcionamiento de la API se han hecho pruebas para medir el tiempo empleado en realizar cada operación criptográfica. La evaluación se ha realizado tanto para la versión que hace uso de la librería estática como para la que hace uso del código fuente de la librería *blst*.

Las pruebas han consistido en la ejecución de 100 iteraciones del código, almacenando el valor máximo, el valor mínimo y el valor promedio. Entre los dos primeros la diferencia era de un milisegundo. Además, también se ha analizado la diferencia entre los resultados obtenidos en las primeras iteraciones frente a las últimas, sin observar cambios notables.

	Proyecto con biblioteca estática	Proyecto con código fuente
<b>Generación de clave privada y clave pública</b>	242 ms	294 ms
<b>Hash del mensaje</b>	453 ms	582 ms
<b>Firma del mensaje</b>	532 ms	631 ms

**Tabla 5.1. Análisis de los tiempos promedios empleados en cada operación**

Como se puede observar, en la Tabla 5.1 los tiempos en la implementación con la biblioteca estática son inferiores frente a la versión con el código fuente. El hecho de optimizar el código dentro de una biblioteca realizada a medida para la arquitectura Cortex-M33 resultó en una diferencia de casi 100 ms entre ambas firmas. Es cierto que las disparidades de resultados no son determinantes ya que los tiempos que se manejan no provocarían un cambio sustancial de rendimiento en la red. Además de que la operación de generación de claves solo se haría una sola vez, no continuamente como puede ser la firma al realizar la tarea de validación.

## Capítulo 6. Conclusiones y líneas futuras

### 6.1. Conclusiones

La tecnología *blockchain* no ha conseguido implantarse de forma generalizada todavía y sin duda la apuesta por ella es hacia el medio y largo plazo, pero sin olvidar que en el presente ya se ejecutan aplicaciones en ella muy prometedoras. Como todo proyecto que se encuentra en las primeras fases, el trabajo de la comunidad es vital para que el desarrollo sea lo más eficiente y transparente posible. Por este motivo, en este Trabajo Fin de Grado se muestran las características de un proyecto de ingeniería en un ámbito que presenta mucho potencial pero que apenas está desarrollado.

Desde el comienzo ha habido que realizar tareas de investigación sobre el entorno en el que se sitúa, entender el contexto previo e interpretar cómo encajar esta idea dentro de la hoja de ruta de un proyecto que tiene una cotización de miles de millones de dólares. Sin duda lo más complejo ha sido pasar de un deseo teórico a un proyecto tangible que se apoye en las herramientas que se encuentran alrededor, aunque muchas veces estas también estén en una fase de desarrollo paralelamente al transcurso de la implementación.

Este módulo de seguridad *hardware* se ha desarrollado en base a un vacío y una necesidad del mercado, el cual cada vez es más grande y requiere que los usuarios finales puedan participar en un nicho que augura mucho potencial. La pieza angular ha sido una placa de bajo coste, que en ningún caso es un factor limitante tanto por presupuesto como capacidad computacional.

Las características principales del trabajo que se verán aplicadas al ecosistema de validación en la red Eth2 son el incremento de la seguridad y resiliencia, la reducción de la presión a los desarrolladores de los clientes, y el suministro de una herramienta disponible para usar, que protege todo el material criptográfico necesario para ejecutar un nodo.

La seguridad, concretamente la rama de criptografía, y la programación de microcontroladores son disciplinas que se han estudiado durante el transcurso del Grado, concretamente el estudio teórico de la criptografía asimétrica y sus diferentes usos como el desarrollo de aplicaciones en C y en plataformas como FreeRTOS sobre material del fabricante Texas Instruments. En cambio, el esquema de firmas BLS y el trabajo sobre las placas de Nordic Semiconductor son novedades conceptuales, pero esta característica no ha supuesto ningún inconveniente y ha permitido desarrollar la capacidad de adaptación y aprendizaje sobre nuevas tecnologías. Considero que son capacidades que un futuro ingeniero debe adquirir en su formación.

## 6.2. Líneas futuras

La hoja de ruta del proyecto se divide en varias fases, la primera de ellas es la correspondiente a este trabajo de fin de grado. En los próximos meses se realizarán las implementaciones que completarán más el prototipo y se detallan a continuación:

- Traslado de la ejecución del código a la zona segura con TrustZone.

Una de las características más interesantes del proyecto será llevar a cabo el aislamiento del código en la CPU. Se implementará con la tecnología ARM TrustZone y habilitará robustos niveles de protección además de permitir el cambio entre el entorno seguro y el entorno no seguro vía hardware para transiciones más rápidas.

- Implementación de una interfaz LTE.

Otro de los desafíos será llevar a cabo la EIP-3030 en la placa para permitir un HSM dual, es decir, que opere vía USB y a la vez de forma independiente (*standalone*) vía LTE-M. Esto requerirá interactuar con el procesador criptográfico para todas las conexiones TLS, implementación de una API REST

y conectarlo a la funcionalidad de BLS. La administración del cliente y el emparejamiento serán también un desafío además de ofrecer una retroalimentación al usuario mientras usa el módulo de seguridad *hardware*.

- Mejora de la experiencia del usuario.

El último reto al que habrá que enfrentarse será unir todas las piezas descritas anteriormente e implementarlas en un dispositivo amigable desde el punto de vista del consumidor. Para esta tarea se usará la plataforma Thingy:91 [32] que incluye tanto el *hardware* de la placa nRF9160 como el de nRF52840. Consta de botones y LEDs RGB que permitirán usarlos para notificaciones, entre otras prestaciones. Además, tiene una batería y es de un tamaño muy reducido.



## Referencias

- [1] S. Haber y W. S. Stornetta, «How to time-stamp a digital document,» *Journal of Cryptology*, 1991.
  
- [2] S. Nakamoto, «Bitcoin: A Peer-to-Peer Electronic Cash System,» 2009. [En línea]. Available: <https://bitcoin.org/bitcoin.pdf>.
  
- [3] Ethereum Foundation Blog, «Staking community grantee announcement,» [En línea]. Available: <https://blog.ethereum.org/2021/02/09/esp-staking-community-grantee-announcement/>.
  
- [4] J. Poon y T. Dryja, «The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments,» 14 January 2016. [En línea]. Available: <https://lightning.network/lightning-network-paper.pdf>.
  
- [5] V. Buterin, «A Next-Generation Smart Contract and Decentralized Application Platform,» [En línea]. Available: <https://ethereum.org/en/whitepaper/>.

- [6] A. Preukschat, «Ethereum es Turing completo ¿y eso qué es?,» 18 Diciembre 2017. [En línea]. Available: <https://www.eleconomista.es/economia/noticias/8817210/12/17/Ethereum-es-Turing-completo-y-eso-que-es.html>.
- [7] O. G. Güçlütürk, «The DAO Hack Explained: Unfortunate Take-off of Smart Contracts,» 1 August 2018. [En línea]. Available: <https://ogucluturk.medium.com/the-dao-hack-explained-unfortunate-take-off-of-smart-contracts-2bd8c8db3562>.
- [8] BBC Tech, «CryptoKitties craze slows down transactions on Ethereum,» 5 December 2017. [En línea]. Available: <https://www.bbc.com/news/technology-42237162>.
- [9] Ethereum Foundation, «The Beacon Chain,» [En línea]. Available: <https://ethereum.org/en/eth2/beacon-chain/>.
- [10] M. M. Roa, «Bitcoin consume más electricidad que países enteros,» 6 Mayo 2021. [En línea]. Available: <https://es.statista.com/grafico/18630/consumo-de-electricidad-anual-de-bitcoin/>.
- [11] D. Chester, «The Dangers of Mining Pools: Centralization and Security Issues,» 4 November 2019. [En línea]. Available: <https://cointelegraph.com/news/the-dangers-of-mining-pools-centralization-and-security-issues>.
- [12] N. Reiff, «Why Centralized Cryptocurrency Mining Is a Growing Problem,» 10 March 2021. [En línea]. Available: <https://www.investopedia.com/investing/why-centralized-crypto-mining-growing-problem/>.



- [13] Satoshi Labs, «Trezor Hardware Wallet | The original and most secure hardware wallet,» [En línea]. Available: <https://trezor.io/>.
  
- [14] Ledger, «Hardware Wallet - State-of-the-art security for crypto assets,» [En línea]. Available: <https://www.ledger.com/>.
  
- [15] D. Boneh, B. Lynn y H. Shacham, «Short signatures from the Weil pairing,» [En línea]. Available: <https://www.iacr.org/archive/asiacrypt2001/22480516.pdf>.
  
- [16] D. Boneh, S. Gorbunov, R. Wahby, H. Wee y Z. Zhang, «IETF BLS Signature V4,» 10 September 2020. [En línea]. Available: <https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-bls-signature-04>.
  
- [17] B. Edgington, «Rogue key attacks,» [En línea]. Available: <https://hackmd.io/@benjaminion/bls12-381#Rogue-key-attacks>.
  
- [18] H. Junge, «EIP-3030: BLS Remote Signer HTTP API Standard,» 30 Septiembre 2020. [En línea]. Available: <https://github.com/ethereum/EIPs/blob/master/EIPS/eip-3030.md>.

- [19] P. Labs, «Eth2 Remote Signer,» [En línea]. Available: <https://github.com/prysmaticlabs/remote-signer>.
- [20] Consensys, «Web3Signer, an open-source signing service,» [En línea]. Available: <https://docs.web3signer.consensys.net/en/latest/#what-is-web3signer>.
- [21] Supranational, «blst, Multilingual BLS12-381 signature library,» [En línea]. Available: <https://github.com/supranational/blst>.
- [22] B. Lynn, «A free pairing-based cryptography library in C,» [En línea]. Available: <https://crypto.stanford.edu/pbc/>.
- [23] D. F. Aranha, C. P. L. Gouvêa, T. Markmann, R. S. Wahby y K. Liao, «RELIC is an Efficient Library for Cryptography,» [En línea]. Available: <https://github.com/relic-toolkit/relic>.
- [24] MIRACL Ltd, «MIRACL Cryptographic SDK,» [En línea]. Available: <https://github.com/miracl/MIRACL>.

- [25] Ethereum Foundation and Protocol Labs, «blst audit,» [En línea]. Available: <https://notes.ethereum.org/@djrtwo/blst-rfp>.
- [26] NCC Group's Cryptography Services team, «BLST Cryptographic Implementation Review,» [En línea]. Available: <https://static1.squarespace.com/static/5eb0c72a0914c75f344e55db/t/5fd3f2a0c22572489c5ca171/1607725730065/NCC+Group+Audit+-+blst.pdf>.
- [27] M. Pesonen, «Performance Evaluation of Optimal Ate Pairing on Low-Cost Single Microprocessor Platform,» [En línea]. Available: [https://www.utupub.fi/bitstream/handle/10024/150659/Pesonen\\_Mikko\\_opinneyte.pdf?sequence=1&isAllowed=y](https://www.utupub.fi/bitstream/handle/10024/150659/Pesonen_Mikko_opinneyte.pdf?sequence=1&isAllowed=y).
- [28] A. Karatsuba y Y. Ofman, «Multiplication of many-digit numbers by automatic computers,» *Doklady Akademii Nauk*, vol. 145, nº 2, p. 293–294, 1962.
- [29] D. Boneh, S. Gorbunov, R. Wahby, H. Wee y Z. Zhang, «IETF BLS Signatures (Variants),» [En línea]. Available: <https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-bls-signature#section-2.1>.
- [30] D. Boneh, S. Gorbunov, R. Wahby, H. Wee y Z. Zhang, «IETF BLS Signatures (Proof of possession),» [En línea]. Available: <https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-bls-signature#section-4.2.3>.

[31] P. de Juan Fidalgo, «HSM BLS Signer,» [En línea]. Available: <https://github.com/decentralizedsecurity/bls-hsm>.

[32] Nordic Semiconductor, «Nordic Thingy:91, Multi-sensor cellular IoT prototyping platform,» [En línea]. Available: <https://www.nordicsemi.com/Software-and-tools/Prototyping-platforms/Nordic-Thingy-91>.