Comparativa de pruebas de conocimiento cero en Iota y Ethereum
Comparing zero-knowledge proofs in Iota and Ethereum

Trabajo de Fin de Máster
Curso 2023–2024

Autor
Sergio Garrido de Castro

Director
Samer Hassan Collado

Máster en Internet de las Cosas
Facultad de Informática
Universidad Complutense de Madrid

Comparativa de pruebas de conocimiento cero en Iota y Ethereum
Comparing zero-knowledge proofs in Iota and Ethereum

Trabajo de Fin de Máster en Internet de las Cosas
Departamento de Ingeniería del Software e Inteligencia Artificial

Autor
Sergio Garrido de Castro

Director
Samer Hassan Collado

Convocatoria: *Septiembre 2024*
Calificación: *7.5*

Máster en Internet de las Cosas
Facultad de Informática
Universidad Complutense de Madrid

19 de Septiembre de 2024

Agradecimientos

A mis padres por el apoyo incondicional durante este último año tan difícil.
Y a todo mi circulo de confianza en Medina del Campo y en Valladolid que siempre sirven de sostén y apoyo en cada etapa vital y en cada paso que doy.

Resumen

Comparativa de pruebas de conocimiento cero en Iota y Ethereum

Las pruebas de conocimiento cero (ZKP) han aparecido, gracias a la tecnología Blockchain, como una solución para garantizar privacidad y seguridad en sistemas descentralizados. Esta solución es particularmente interesante en Ethereum e IOTA, dos sistemas descentralizados de distinta naturaleza, en su búsqueda por equilibrar la descentralización, la seguridad y la eficiencia en entornos de alto rendimiento. Los ZKP en Ethereum e IOTA se utilizan para aumentar la privacidad de las transacciones, permitiendo su verificación sin revelar información sensible.

Este trabajo analiza el rendimiento de tres tipos de ZKP -Groth16, Plonk y STARK- implementados en las plataformas Ethereum e IOTA. Se hace énfasis en variables clave como son el tiempo de generación y verificación de pruebas, el tamaño de las pruebas y el consumo de gas. Los resultados obtenidos indican que Ethereum, a pesar de ser un sistema más robusto en términos de seguridad y descentralizado, sufre de un mayor consumo de gas en comparación con IOTA. En cuanto a los tiempos de verificación, se comprueba que son más elevados en IOTA, aunque se muestran más constantes y dependen en mayor medida del algoritmo de ZKP utilizado y de la manera en la que se configura el entorno de pruebas. Los resultados obtenidos inciden en la importancia de elegir correctamente la plataforma y el algoritmo de ZKP según el contexto

Palabras clave

Blockchain, Criptografía, Curvas elípticas, Ethereum, Iota, Métricas, Prueba de conocimiento cero, Solidity, SNARK, STARK.

Abstract

Comparing zero-knowledge proofs in Iota and Ethereum

Zero-knowledge proofs (ZKP) have appeared, thanks to Blockchain technology, as a solution to ensure privacy and security in decentralized systems. This solution is particularly interesting in Ethereum and IOTA, two decentralized systems of different nature, in their quest to balance decentralization, security and efficiency in high-performance environments. ZKPs in Ethereum and IOTA are used to increase the privacy of transactions, allowing their verification without revealing sensitive information.

This paper analyzes the performance of three types of ZKPs-Groth16, Plonk, and STARK-implemented on Ethereum and IOTA platforms. Emphasis is placed on key variables such as proof generation and verification time, proof size, and gas consumption. The results obtained indicate that Ethereum, despite being a more robust system in terms of security and decentralized, suffers from higher gas consumption compared to IOTA. As for verification times, it is found that they are higher in IOTA, although they are shown to be more constant and depend to a greater extent on the ZKP algorithm used and the way the testing environment is configured. The results obtained emphasize the importance of choosing the right platform and ZKP algorithm according to the context.

Keywords

Blockchain, Cryptography, Elliptic curve, Ethereum, Iota, Metrics, Solidity, SNARK, STARK, Zero Knowledge Proof.

Índice

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

En los últimos años Blockchain ha surgido como una tecnología innovadora para construir sistemas distribuidos. Blockchain es una tecnología basada en una base de datos compartida y distribuida entre distintos nodos donde la información registrada está almacenada en bloques ligados entre sí a través de criptografía, y validados de forma descentralizada a través de un protocolo común. Es decir, no hay ningún tipo de autoridad central que controle la red. La información se guarda en los nodos de la red y se mantiene una copia idéntica de la información almacenada en cada nodo, garantizando así la integridad. Blockchain es un tipo de DLT (Distributed Ledger Technology), es decir, es un libro de contabilidad compartido, descentralizado y cuyo histórico es inmutable. Blockchain registra y verifica todas las transacciones realizadas en una red. Se puede considerar a blockchain como una base de datos a la que solo se pueden añadir nuevas transacciones, pero no modificar ni eliminar las transacciones previas. Al estar enlazados los bloques de forma criptográfica la cadena de bloques generada es inmutable, por lo que las transacciones registradas no se pueden alterar.

La aplicación más conocida de blockchain son las criptomonedas o monedas digitales. Este caso de uso tiene como intención descentralizar las transacciones financieras sin necesidad de una autoridad central como puede ser un banco. La criptomoneda más conocida es Bitcoin. Bitcoin se suele utilizar para transferencias internacionales, reservas de valor, pagos on-line, inversiones, micropagos, finanzas descentralizadas (DeFi), pagos sin cuenta bancaria, intercambio entre pares (peer-to-peer), etc. Por ejemplo, Bitcoin es muy útil en un escenario para compras on-line donde se admiten pagos con Bitcoin. Un usuario seleccionará el servicio o producto que desea adquirir y a la hora de pagar, este usuario elige Bitcoin como método de pago. El sitio web le muestra una dirección de Bitcoin, que es una cadena única de caracteres donde se debe enviar el pago, junto con la cantidad exacta de Bitcoin que debe transferir. El usuario abre su monedero digital, pega la dirección de Bitcoin. Después, introduce la cantidad exacta de Bitcoin que debe enviar y confirma la transacción. Esta transacción se transmite a la red de Bitcoin, donde es verificada por los nodos y mineros. Cuando se verifica la transacción el sitio web de compra on-line recibe la notificación de que el pago ha sido realizado correctamente.

Pero esta no es la única aplicación existente. Blockchain se puede utilizar para crear contratos inteligentes o smart contracts, votaciones electrónicas, gestión de cadenas de suministros y gestión de identidad digitales. Por ejemplo, al aplicarla a identidad digital, permite a los usuarios tener control sobre su información personal, reducir el riesgo de

fraude, y aumentar la privacidad.

El uso de la tecnología de Blockchain ha supuesto un impacto significativo en los ámbitos del Internet de las Cosas, innovando en el ámbito de los sistemas distribuidos, al ofrecer una estructura segura y descentralizada. Blockchain se basa en algoritmos de criptografía bien conocidos, como son SHA-1 o hashing, para conformar su estructura de cadena de bloques. Cada bloque contiene el histórico de las transacciones, lo que garantiza la inmutabilidad y la integridad del historial de datos. Este enfoque permite que las trasacciones realizadas sean públicas y abiertas. Blockchain ofrece una capa más de seguridad en los sistemas distribuidos al permitir transacciones seguras sin necesitar de intermediarios ni de una estructura centralizada. Al ser una estructura cuyo histórico es inmutable, Blockchain permite a los usuarios tener un mayor control gracias a la implementación de mecanismos que aseguran que los datos sólo se pueden compartir con el consentimiento explícito del usuario y que cualquier transacción debe ser realizada transparente. Blockchain ha impulsado la investigación de nuevos modelos como son los ZKP para garantizar mayor privacidad en una transacción. Estos permiten validar transacciones sin revelar detalles sensibles, mejorando la privacidad sin sacrificar la seguridad.

Zero Knowledge Proof (ZKP) o prueba de conocimiento cero es un método entre dos partes "prover"(probador) y "verifier"(verificador). Es una tecnología emergente, que permite a una parte probar a otra que una prueba es verdadera sin revelar información adicional [16]. Un caso de uso práctico que ejemplifica el funcionamiento de una prueba de conocimiento cero es el siguiente. Imagine una empresa de alquiler de vehículos, en la que la empresa requiere que la persona proporcione una copia de su permiso de conducir y su tarjeta de crédito para verificar que tiene una licencia válida y que tiene suficiente crédito disponible. La persona que quiere alquilar un vehículo registra previamente su identidad, permiso de conducir y tarjeta de crédito en una plataforma que soporta Zero Knowledge Proof. Cuando se solicita el alquiler de un vehículo la plataforma genera una prueba ZKP en vez de enviar la información personal. Con esa prueba de conocimiento cero en este escenario permite demostrar que la persona tiene un permiso de conducir valido y que la tarjeta de crédito tiene crédito suficiente. La empresa de alquiler recibe estas pruebas criptográficas y las verifica sin necesidad de visualizar detalles del permiso de conducir o de la tarjeta de crédito. Una vez verificadas las pruebas ZKP la empresa de alquiler aprueba la solicitud de alquiler.

En el contexto de Blockchain, ZKP se utiliza para aumentar la privacidad de las transacciones, permitiendo que los participantes en la red verifiquen la validez de una transacción sin necesidad de conocer los detalles específicos de la misma. Este enfoque es importante en aquellas aplicaciones donde la privacidad es crítica, como pueden ser las transacciones financieras o la gestión de identidades digitales. Las pruebas de conocimiento cero son herramientas muy útiles en el diseño de protocolos seguros. Sin embargo, el concepto de prueba de conocimiento es muy sutil y se necesita mucho cuidado para obtener una formalización satisfactoria.

1.1. Motivación

El máster de Internet de las Cosas proporciona a los alumnos que cursamos dicho máster herramientas y competencias relativas a IoT. El Internet de las Cosas o Internet of Things (IoT) es un campo compuesto por múltiples sistemas de información global. Estos

sistemas están compuestos por dispositivos masivos heterogéneos y descentralizados que pueden identificarse, detectarse y procesarse mediante protocolos de comunicación estandarizados e interoperables.

El objetivo de esta tecnología es conectar una gran variedad de dispositivos permitiendo cambiar la forma de interacción del usuario con el mundo físico. Estos dispositivos pueden aprovechar la naturaleza descentralizada de blockchain estableciendo confianza, mejorando la seguridad y facilitando la comunicación e intercambio de datos. La transparencia del histórico de Blockchain ofrecen ventajas a las aplicaciones IoT.

En la gran mayoría de ocasiones los datos de IoT se almacenan en diferentes servidores en la nube, y se procesan de manera distribuida. Este enfoque puede provocar brechas de seguridad haciendo susceptibles de ciberataques a los dispositivos IoT. Los sistemas descentralizados pueden proporcionar privacidad y escalabilidad, sin necesidad de depender de una autoridad central o de un hardware específico. En este contexto, las tecnologías de libro mayor distribuido (DLT) y similares como es el caso de Blockchain pueden ser importantes en el entorno IoT.

La implementación más importante de blockchain en IoT es el proyecto IOTA. IOTA es una criptomoneda que promete alta escalabilidad, con transferencias casi instantáneas a coste cero, enfocada a soluciones de Internet de las cosas. Fundado por la IOTA Foundation, el proyecto busca resolver algunos de los problemas de las tecnologías blockchain tradicionales, adaptándose específicamente a las necesidades de IoT.

Actualmente las blockchains se enfrentan a los retos de ser descentralizadas, seguras y escalables al mismo tiempo. Este dilema es conocido como el Trilema de la Escalabilidad[66], el cual postula que un sistema software solo puede tener dos de las tres propiedades entre descentralización, seguridad y escalabilidad. No en términos absolutos, pues siempre se pueden tener algo de las tres, pero lo que sugiere Vitalik Buterin es que en un sistema blockchain, como los que se estudiarán, si se quiere ganar de una propiedad de las tres, se debe perder de otra de las restantes. Las blockchains tradicionales como Bitcoin y Ethereum, por ejemplo, son descentralizadas y seguras, pero a menudo luchan con problemas de escalabilidad, lo que se traduce en altas tasas de transacción y tiempos de procesamiento prolongados. Sin embargo, existen soluciones que han intentado llegar a altas cotas en las tres características, aun sin cumplimentarlas al 100 %, como pueden ser sharding, cadenas laterales o sidechains, capa 2 de Ethereum, Proof of Stake (PoS), Directed Acyclic Graphs (DAGs), nuevos algoritmos de consenso o cadenas híbridas.

Estos problemas de diseño son particularmente importantes en Blockchain, como la presencia de tasas, el alto tiempo de procesamiento y la falta de escalabilidad, que no encajan bien en un escenario de dispositivos heterogéneos como los que surgen de la nueva economía surgida con IoT. En un entorno IoT donde multitud de dispositivos necesitan comunicarse y realizar transacciones de datos de manera eficiente y segura, las limitaciones de escalabilidad y los costos asociados con las blockchains tradicionales no son viables. Para mitigar estos problemas, IOTA creó su propia DLT, que se llama Tangle [56]. Al eliminar los mineros y las tarifas de transacción, y al permitir que cada transacción valide al menos dos transacciones anteriores, el Tangle proporciona un sistema más eficiente y escalable para el ecosistema de IoT, a costa de perder descentralización. Esta perdida de descentralización sigue el Trilema de la escalabilidad.

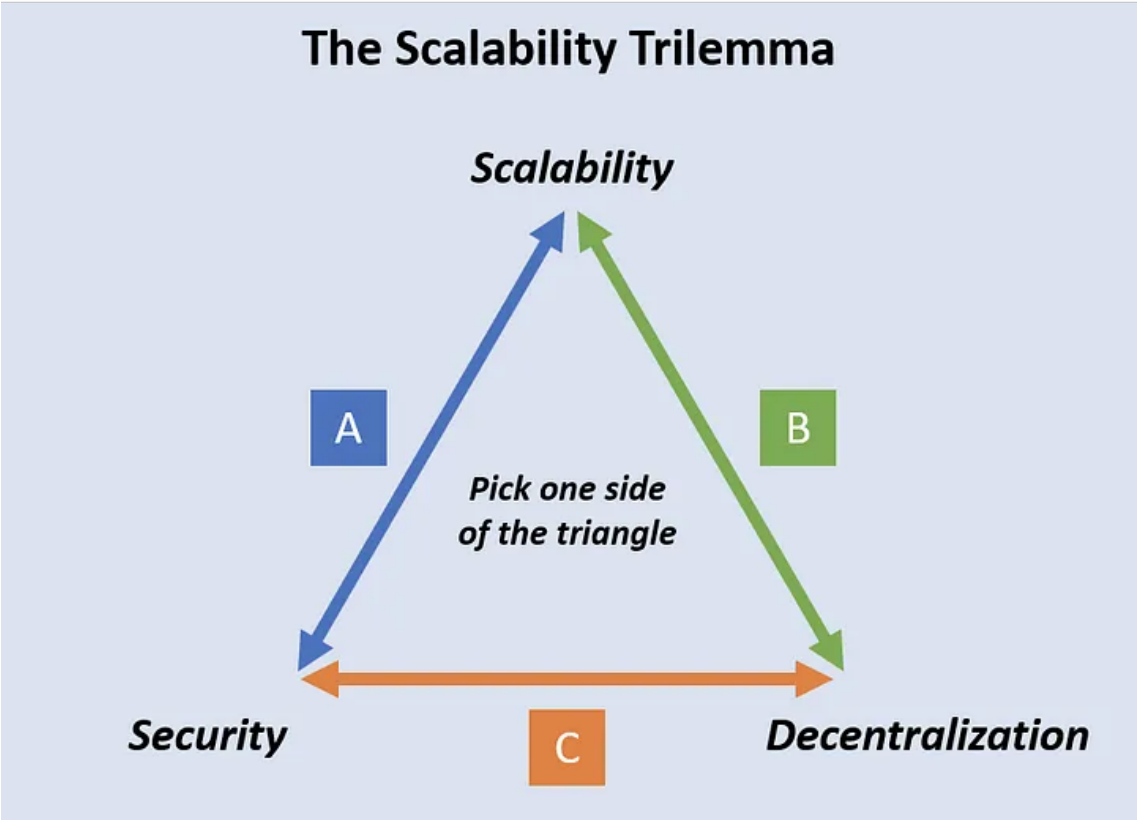

Figura 1.1: Trilema de la escalabilidad [45]

La presencia de Blockchain ha impactado en la privacidad de los datos utilizados en un entorno IoT permitiendo nuevas aplicaciones, pero también presentando nuevos retos en la gestión de los datos y en la seguridad de los mismos. En el contexto de IoT es vital contar con un sistema que garantice la integridad y privacidad de los datos, así como la protección contra accesos no autorizados. Por tanto las identidades digitales se convierten esenciales en IoT, pues pueden asegurar que solo dispositivos autorizados accedan a la red y puedan comunicarse entre si. Blockchain e IOTA proporcionan una solución robusta para la gestión de identidades digitales en IoT. Las características de inmutabilidad y descentralización aseguran que las identidades no puedan ser falsificadas o alteradas sin ser detectadas. Cada dispositivo puede tener su identidad registrada en un bloque distribuido, permitiendo una verificación rápida y confiable de su autenticidad.

Ethereum es una plataforma de blockchain descentralizada y de código abierto que permite la creación y ejecución de contratos inteligentes. Fue propuesta en 2013 por el programador Vitalik Buterin y su desarrollo comenzó en 2014, con su red principal lanzada en julio de 2015. La importancia de Ethereum en el contexto de IoT y la gestión de identidades digitales radica en su capacidad para ejecutar contratos inteligentes, que son programas que se ejecutan exactamente como están configurados, sin servidor central, y sin posibilidad de interrupción, censura, fraude o interferencia de terceros.

Los contratos inteligentes en Ethereum pueden automatizar y asegurar diversas operaciones en un entorno IoT, desde la gestión de identidades hasta la ejecución de transacciones y el intercambio de datos entre dispositivos. La flexibilidad y la robustez de

Ethereum lo convierten en una plataforma ideal para experimentar con tecnologías como los Zero-Knowledge Proof. Estos permiten mejorar la privacidad y la seguridad al permitir la verificación de información sin necesidad de revelar los datos subyacentes, lo cual es crucial en el manejo de datos sensibles en IoT.

Este trabajo de fin de máster propone un estudio comparativo sobre los distintos tipos de ZKP implementados e integrados tanto en una blockchain como Ethereum como en el Tangle de IOTA.

1.2. Objetivos

Las Zero Knowledge Proof se han convertido en un área de estudio muy novedoso en el ámbito de blockchain, en concreto en lo relativo a la privacidad y criptografía. El objetivo de este estudio es analizar cómo se comportan los ZKP tanto en Ethereum como en IOTA. Se plantea el siguiente conjunto de objetivos de desarrollo.:

- Objetivos generales entre los que se encuentran: Estudio de las tecnologías relacionadas con Zero-Knowledge Proof, utilización de métricas de comparación para analizar ZKP en Ethereum e IOTA y el desarrollo de pruebas que permitan realizar dicho análisis
- Objetivos específicos:
  - Desarrollo de 3 prototipos distintos de ZKP capaces de generar pruebas.
  - Desarrollo de smart contracts en Solidity que verifiquen cada prueba generada.
  - Utilización de las métricas seleccionadas y realizar el análisis en base a dichas métricas.
  - Desarrollo del escenario de pruebas para realizar la comparativa. En este escenario se debe desplegar cada smart contract en Ethereum y en IOTA. También se debe realizar mediciones y repetir el proceso 20 veces.

Una vez se hayan obtenido todos los datos necesarios se procederá a analizar los resultados obtenidos teniendo en cuenta las siguientes preguntas de investigación:

- ¿Que tipo de ZKP es mejor, en función de distintas métricas?
- Entre el blockchain de Ethereum y el Tangle de IOTA, ¿en que plataforma tiene mejor rendimiento el ZKP?
- ¿Cual es el tipo de ZK cuya prueba tiene menor tamaño?

1.3. Plan de trabajo

Este trabajo ha tenido un desarrollo aproximado de 6 meses, desde febrero a septiembre, abarcando distintas etapas que se pueden definir en las siguientes:

- Fase de definición del tema a tratar: En esta fase se ha planteado cual era un caso de uso interesante en IOTA, eligiendo así las Identidades Digitales.
- Fase de investigación de tecnologías: Para trabajar sobre identidades digitales se eligió investigar sobre ZKP. En esta etapa se ha revisado literatura sobre IOTA y ZKP, investigando sobre los distintos tipos de ZKP.

- Fase de prototipado: En esta fase se han diseñado los distintos prototipos de ZKP y los distintos smart contracts para Ethereum e IOTA que los validaran.
- Fase de de ejecución de pruebas: Una vez desarrollados los prototipos se realizan las pruebas desplegando los smart contracts con los verificadores en el blockchain de Ethereum y en el Tangle de IOTA.
- Fase de análisis comparativo: En esta fase se ha realizado el análisis de las pruebas realizadas teniendo en cuenta una serie de métricas definidas, para poder comparar el rendimiento de los distintos tipos de ZKP elegidos en Ethereum e IOTA
- Fase de redacción: En esta etapa se ha redactado la memoria, definiendo los conceptos teóricos necesarios y detallando el análisis realizado así como las conclusiones obtenidas.

1.4. Estructura del documento

El documento está estructurado en 7 capítulos que describen el desarrollo realizado sobre el estudio de los zero-knowledge proofs y su integración tanto en Ethereum como en IOTA. A continuación se presenta un resumen breve del contenido de cada capítulo:

- Capitulo 1. Introducción: Se da una introducción a los ZKP, estableciendo las motivación que hacen de este tema relevante y novedoso. Se explican los objetivos del estudio y se presenta una visión general del contenido del documento.
- Capitulo 2. Tecnologías y Estado del Arte: En este capitulo se exploran los fundamentos teóricos de las distintas tecnologías y herramientas que se utilizan a lo largo del trabajo. Se exploraran conceptos relativos a los zero-knowledge proof, la criptografía que utilizan las variedades de ZKP que se eligen, conceptos sobre blockchain, Ethereum, IOTA y las herramientas utilizadas para el desarrollo del trabajo. En una sección de este capítulo se desarrollará el estado del arte. En él se exploraran trabajos, proyectos y artículos existentes que sirven de base e inspiración para el desarrollo del trabajo. Es resultado de un trabajo de investigación previo. A lo largo de este subcapitulo se ofrecen los resultados de dicha investigación sobre Zero Knowledge Proof, sobre estudios comparativos entre distintos tipos de ZKP y las métricas que e utilizan.
- Capitulo 3. Metodología: En este capitulo se detallará la metodología seguida en el proceso del trabajo. Se describe cómo se han llevado a cabo las distintas fases del proyecto, desde la investigación y desarrollo hasta las pruebas y la escritura de la memoria. También se incluye el diagrama general que siguen los contratos inteligentes y las pruebas realizadas.
- Capitulo 4. Desarrollo: En este capítulo se detallan los tres contratos inteligentes desarrollados, junto con sus pruebas. Se explica cómo estos contratos siguen el diagrama general descrito en el capítulo anterior y se proporcionan detalles técnicos sobre su implementación y funcionamiento.
- Capítulo 5. Métricas: En este capitulo se explican las métricas que se utilizarán a lo largo del análisis comparativo. Se utilizarán métricas que permitan comparar tiempos de generación de las pruebas, tiempos de verificación de las pruebas y el gas consumido durante el proceso. También se definirán métricas que permitirán evaluar la complejidad de los contratos inteligentes desarrollados.

- Capitulo 6. Resultados del análisis comparativo: En este capitulo se realizará un análisis comparando distintos tipos de ZKP implementados tanto en Ethereum como en IOTA teniendo en cuenta una serie de métricas elegidas para comparar su rendimiento. Se incluyen subcapítulos que detallan los resultados obtenidos en función de diversas métricas elegidas para evaluar su rendimiento. Además se discutirán los resultados obtenidos tras realizar el análisis comparativo y se analizan sus implicaciones.
- Capitulo 7. Conclusiones y trabajo futuro: En este capítulo se plantean las conclusiones generales que se han extraído a la hora de realizar el análisis comparativo de ZKP desplegados en Ethereum y en IOTA. Se resumen los hallazgos claves y se responde el cumplimiento de los objetivos planteados inicialmente. También se ofrecen ideas y recomendaciones para futuras investigaciones que pretendan ampliar este tema de investigación

Capítulo 2

Tecnologías y Estado del Arte

2.1. Blockchain

La tecnología Blockchain encuentra sus orígenes en la crisis financiera de 2008 cuando una persona o serie de personas, con el pseudónimo de Satoshi Nakamoto, esbozaron el protocolo de un nuevo sistema de pago electrónico directo y entre iguales (peer-to-peer o P2P) que usaba una criptomoneda llamada Bitcoin, descentralizada, distribuida y validada mediante criptografía. Este protocolo establece una serie de normas, en forma de computación distribuida, que garantiza la integridad de la información intercambiada entre esos millones de ordenadores sin pasar por terceros [64]

Blockchain podría considerarse un libro de contabilidad público, en el que todas las transacciones comprometidas se almacenan en una cadena de bloques ("block chain"). Esta cadena crece continuamente cuando se le añaden nuevos bloques. La tecnología blockchain tiene características clave como la descentralización, la persistencia, el anonimato y la auditabilidad. Blockchain funciona habitualmente en un entorno descentralizado. Esto se consigue gracias a la integración de varias tecnologías básicas como el hash criptográfico, la firma digital (basada en la criptografía asimétrica) y el mecanismo de consenso distribuido. Con la tecnología blockchain una transacción puede realizarse de forma descentralizada. Como resultado, blockchain en sistemas distribuidos puede ahorrar costes y mejorar la eficiencia. [75] Blockchain tiene más casos de uso más allá de las criptomonedas, convirtiéndose así en una de las tecnologías más prometedoras de la siguiente generación de sistemas de interacción en internet. Ofrece posibilidades como la creación de contratos inteligentes, la mejora en el acceso a diferentes servicios públicos utilizando identidades digitales, la integración con el internet de las cosas (IoT), la implementación de sistemas de reputación y el desarrollo de servicios de seguridad.

Dentro de las distintas soluciones que ofrece esta tecnología la que más nos interesa para este trabajo son las Identidades Digitales. Hoy en día es necesaria una prueba de identidad digital para interactuar con gran parte de los servicios web públicos y privados. Esta prueba de identidad digital se puede basar en un documento de identidad, pasaporte o simplemente una fecha de nacimiento. Este tipo de gestión de identidad tiene el inconveniente de que los documentos físicos están sujetos a perdidas, suplantación, robo o fraude. Al implantarse la identidad digital se aumenta la seguridad al utilizar métodos criptográficos, como puede ser el uso de funciones hash y esquemas de firma criptográfica

Pero antes de es necesario explicar conceptos necesarios de blockchain, empezando por la estructura que tiene la Blockchain. Como se ha mencionado Blockchain es una cadena de bloques, la cual almacena un registro de cada transacción validada en el bloque. Como se indica en la figura 8.1 donde se muestra la estructura de blockchain se puede apreciar como cada bloque apunta al bloque anterior referenciando a un valor hash del bloque anterior o bloque padre.

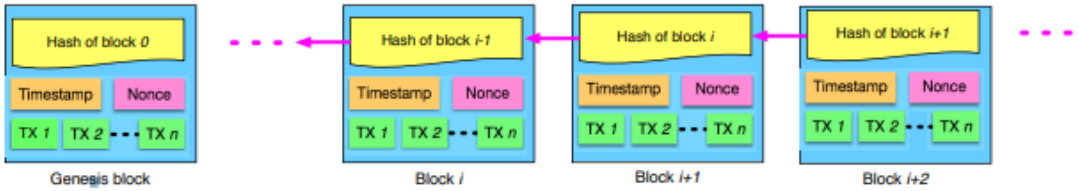

Figura 2.1: Estructura de blockchain

Un bloque se compone de una cabecera y de un cuerpo, donde la cabecera incluye: la versión del bloque, el hash del bloque padre, el hash raíz del árbol Merkle [79] , un timestamp, nBits (especifica el nivel de dificultad de la minería del bloque) y Nonce (es un número que los mineros modifican en cada intento de encontrar un hash válido para un bloque).

2.1.1. Funcionamiento de Blockchain

La blockchain o cadena de bloques toma su nombre de la manera en que se representa el almacenamiento de la información. El libro mayor se compone de bloques como se ha mencionado anteriormente. Los bloques, que son el elemento clave en una blockchain son una colección de transacciones, un sello temporal y un enlace criptográfico al bloque anterior mediante un hash. Ese diseño permite que el histórico de las transacciones sea inmutable y seguro, ya que para modificarlo requiere modificar cada bloque anterior al actual, lo cual es muy costoso.

Basándome en el ejemplo que propuso Satoshi Nakamoto [43] en el artículo original sobre bitcoin se puede explicar como funciona una blockchain. Nakamoto describe un sistema descentralizado para validar transacciones mediante una red de nodos. En este sistema, cada transacción es registrada públicamente y verificada a través de un consenso distribuido, eliminando así la necesidad de intermediarios.

Para fomentar la seguridad y prevenir lo que Nakamoto denomina como doble gasto, este propone un mecanismo llamado prueba de trabajo o proof of work (PoW). Los nodos en la red o mineros compiten entre si para resolver problemas matemáticos complejos. Este proceso consume grandes recursos, lo cual puede asegurar que el bloque minado sea valido. Solo el primer nodo en resolver dicho problema es aquel puede agregar su bloque a la cadena. Este proceso también actúa como un sistema de votación donde la cadena más larga es la más confiable, ya que representa la mayor cantidad de trabajo computacional invertido. Los mineros son incentivados a través de recompensas en forma de comisiones. El consenso en este proceso se alcanza cuando la mayoría de los nodos confirma la transacción.

Aunque las transacciones son publicas, las identidades detrás de ellas pueden permanecer anónimas mediante el uso de claves públicas. Esto permite un nivel de privacidad mayor.

2.2. Ethereum

Ethereum, introducido por Viterik Buterin [9], aborda varias limitaciones de Bitcoin al ofrecer una cadena de bloques con un lenguaje de programación Turing completo. La plataforma de Ethereum se utilizará para la creación de smart contracts auto-ejecutables con condiciones específicas. Según Vitalik Buterin, el propósito de Ethereum es crear un protocolo alternativo a Bitcoin que permita crear aplicaciones descentralizadas. Ethereum se enfoca en ayudar a que el tiempo de desarrollo sea menor.

Solidity es el lenguaje de programación utilizado por Ethereum para el desarrollo de contratos inteligentes. Es un lenguaje de programación Turing completo. Un lenguaje Turing completo es aquel que es capaz de realizar cualquier cálculo computacional descrito por un algoritmo, siempre que se disponga de tiempo y recursos suficientes. Esta característica es importante en Solidity pues permite implementar lógica más sofisticada en los smart contracts. [67].

Ethereum introduce el concepto de gas, que es una unidad interna utilizada para medir y limitar el trabajo computacional requerido para la ejecución de operaciones en la red [22]. Es fundamental para evitar comportamientos indeseados como bucles infinitos o comportamientos maliciosos. Será una medida a tener en cuenta a lo largo del trabajo.

2.2.1. Smart Contracts de Ethereum

Un smart contract o contrato inteligente [33], es un programa que se ejecuta en la blockchain de Ethereum. Estos programas se consideran como una colección de código (funciones) y datos (estado) que reside en un dirección en especifico en la cadena de bloques.

Un contrato inteligente tienen saldo y pueden actuar como objeto de transacciones. Estos serán desplegados en la red y se ejecutaran según lo que se ha programado. Gracias a esta cualidad de un contrato inteligente se puede interpretar que pueden actuar como una cuenta de Ethereum. Una cuenta de usuario interactúa con un contrato desplegado en la blockchain enviando transacciones que ejecutan una de las funciones definidas en dicho contrato.

2.2.2. Ethereum Virtual Machine

La Ethereum Virtual Machine [42] o maquina virtual de Ethereum (EVM) es un entorno virtualizado que ejecuta código en nodos de la cadena de bloques de Ethereum. Dicha ejecución ha de ser coherente y segura en cada nodo. Estas ejecuciones se realizan consumiendo gas, que es una métrica que mide el esfuerzo computacional necesario para las operaciones llevadas a cabo, mientras se garantiza una gestión eficiente de los recursos y de la seguridad en la red.

La EVM ha adquirido gran relevancia fuera de la red Ethereum. Se busca que el resto de redes blockchain sean compatibles con la EVM de Ethereum. Esto permite que los programas en Solidity desarrollados para Ethereum puedan ser ejecutados en otras blockchain.

2.2.3. Layer 1 y Layer 2

Una opción que se plantea para resolver el Trilema de la escalabilidad es la capa 2 o L2 a la cadena de bloques de Ethereum. L2 es una solución propuesta por Ethereum para abordar los problemas de escalabilidad a los que se enfrentan. La capa 2 de Ethereum es una cadena de bloques separada que extiende Ethereum al mismo tiempo que hereda sus garantías de seguridad [10].

La capa 1 o L1 es la cadena de bloques principal y funciona como la base sobre la que se sustentan las redes de la capa 2. La capa 1 incluye cualquier cambio en el protocolo básico de Ethereum que afecte directamente al procesamiento y validación la información en la cadena principal. Estos cambios son muy importantes para la seguridad y descentralización de la red. La capa 1 contiene una red de nodos para asegurar y validar la red, una red de productores de bloques, la cadena de bloques y su historial de transacciones, y el mecanismo de consenso de dicha red.

La capa 2 funciona construyendo distintas soluciones fuera de la cadena principal que interactúan con ella. El objetivo de la capa 2 reducir la carga de la capa 1, al reducir la congestión y permitir la escalabilidad. Hay diferentes tipos de capa 2, entre los que destacan los rollups [10]. Un rollup acumula una cantidad grande de transacciones en una sola transacción de capa 1 [76]. Las comisiones se reparten entre todas las transacciones, haciéndolas mas baratas para cada usuario. Los datos de la transacción rollup se envían a la capa 1, pero su ejecución no se realiza en la capa 1. Una vez enviados los datos a L1, el rollup hereda la seguridad de Ethereum, pues deshacer un rollup equivaldría a revertir Ethereum. Hay dos tipos principales de rollups:

- Rollups optimistas: Asume por defecto que las transacciones son validas. Solo se revisan si se cree que ha producido una transacción fraudulenta, pero para ello se han de aportar pruebas de fraude. Se basan en un esquema de prueba de fraude para detectar dichos casos. Si la prueba de fraude es exitosa, el rollup vuelve a ejecutar las transacciones. Si no se revisa todo el paquete de transacciones, se considera como valido y estas se ejecutan. [50]

- Rollups de conocimiento cero o zk-rollups: Utilizan pruebas de conocimiento cero para garantizar que las transacciones son validas y así no enviar datos relativos a las transacciones. Para actualizar el estado de un rollup, los nodos deberán aportar una prueba de su validez para la verificación, la cual será una prueba de conocimiento cero. Esto evita publicar todo el historial de transacciones en la cadena principal. Los ZK-rollups escriben transacciones en Ethereum como calldata, que almacenan datos, incluidas las llamadas externas en las funciones de un smart contract. Los datos almacenados calldata se publican en la blockchain, esto permite reconstruir el estado del rollup. [51]

Dentro de las distintas soluciones de capa 2 existentes la más relevante para el marco de este trabajo es Starknet [60]

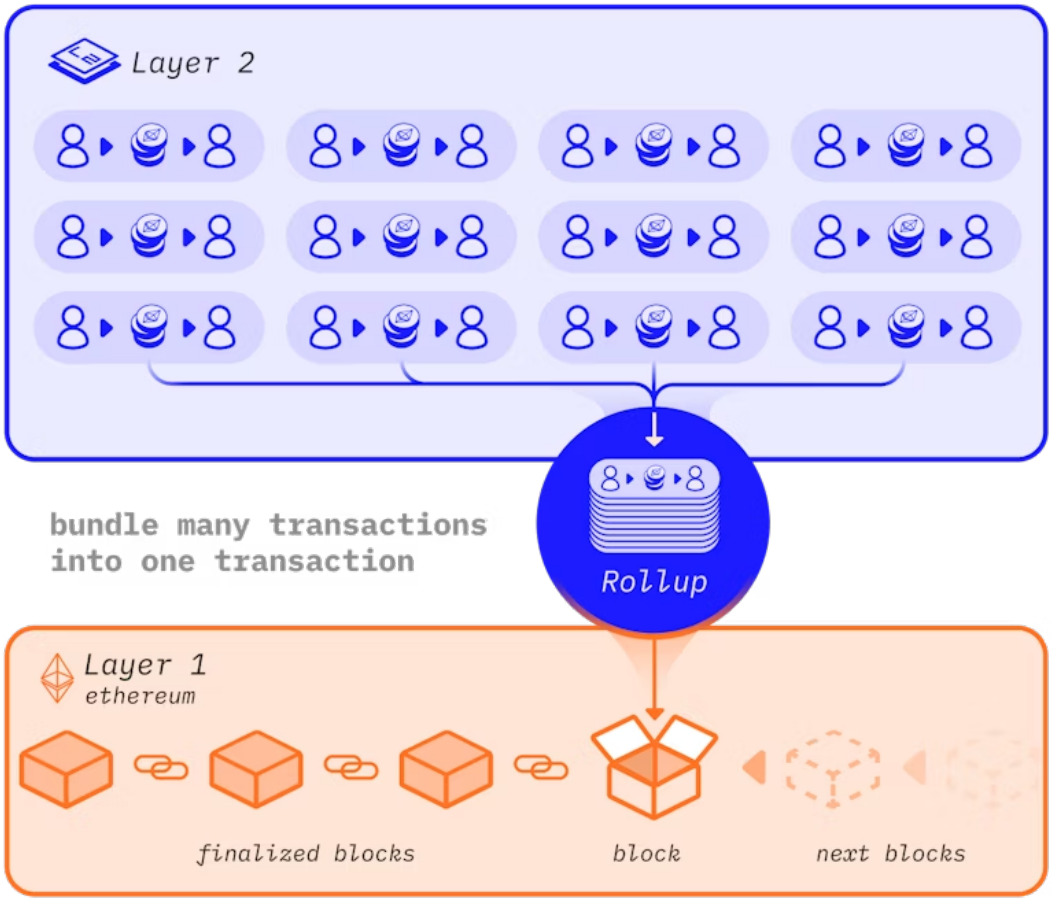

Figura 2.2: Funcionamiento L1 y L2 de Ethereum. Fuente: [10]

2.2.3.1. StarkNet

StarkNet es una solución de escalabilidad de la capa 2 de Ethereum. StarkNet es un zero-knowledge rollup y permite a las aplicaciones descentralizadas aumentar su escalabilidad sin prescindir de la seguridad de Ethereum. El objetivo de esta red es realizar transacciones seguras y con un alto rendimiento gracias a las pruebas criptográficas zk-stark. Los contratos inteligentes que se despliegan en Starknet están escritos en un lenguaje de programación propio y especializado llamado Cairo [61].

La arquitectura de Starknet se conforma como un sistema coordinado en el que cada nodo desempeña su papel. No es una red descentralizada al 100 %, aunque según asegura la propia empresa desarrolladora, StarkWare, se está avanzando en dicha dirección. [60]

En la figura 2.3 se muestra el funcionamiento de StarkNet donde se destacan tres elementos claves: Mempools, Sequencer y Prover. El proceso seguido por Starknet se inicia cuando llegan transacciones al Mempool, que actúa como gateway por donde llegan las transacciones. Las transaccciones se marcan con una etiqueta de received y se envian al secuenciador. Una vez en el secuenciador se han recibido las transacciones, este las ordena, produce bloques y ejecuta las transacciones. Estas transacciones pasan a ser etiquetadas como Accepted_on_L2. Posteriormente el prover genera pruebas STARK para los bloques y las transacciones mediante CairoVM. Estas pruebas STARK son enviadas a la capa 1

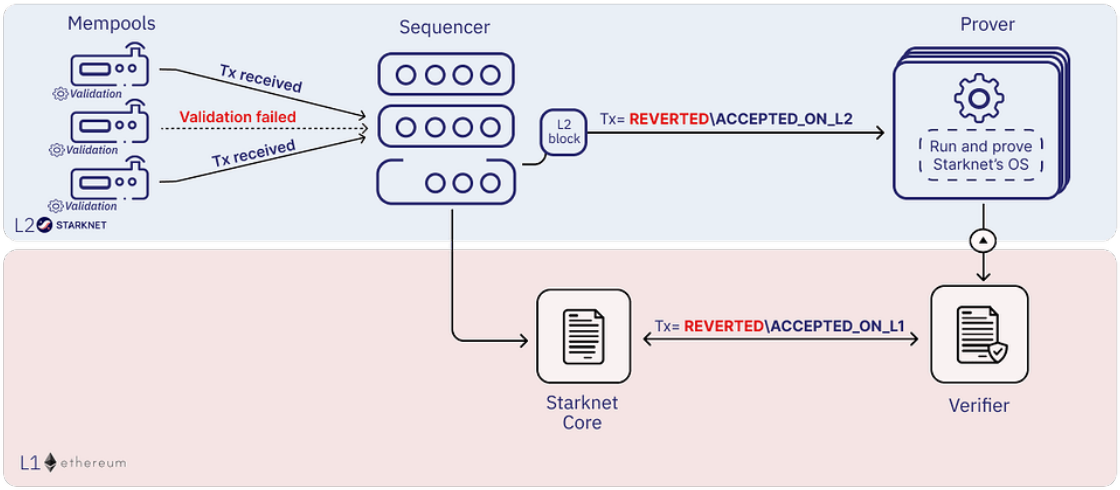

Figura 2.3: Funcionamiento de StarkNet. Fuente: [60]

(Ethereum) para su verificación final. Si las pruebas son válidas, se actualiza el estado de StarkNet en la capa 1, garantizando así la seguridad de las transacciones procesadas fuera de la cadena principal.

2.3. IOTA

IOTA es una tecnología de libro mayor distribuido (DLT) que permite a los usuarios tener control sobre sus datos privados, ejecutar programas a prueba de manipulaciones y participar en la propiedad y el comercio de activos sin necesidad de intermediarios. La DLT como su nombre indica, mantiene un registro en múltiple nodos. A diferencia de los sistemas centralizados, las DLT requieren consenso entre nodos independientes para acordar el estado del libro de contabilidad. Esto plantea el reto de proteger la red de agentes maliciosos. IOTA adopta un enfoque único que lo diferencia de otros protocolos. La versión inicial de IOTA se lanzó en 2016 y es conocida como Legacy Trinary. Es una versión que esta basada en un algoritmo de consenso probabilístico que utiliza recorridos aleatorios en un DAG (Grafo Aciclico Dirigido).

2.3.1. Tangle de IOTA

El Tangle de IOTA es el núcleo de IOTA y la principal innovación presentada por IOTA Foundation[1] [14]. Es una estructura de datos replicada a través de una red de nodos. Al utilizar el Tangle se evitan los cuellos de botellas provocados por los sistemas blockchain más clásicos. En la figura 2.4 se ilustra como la tecnología Tangle con la aplicación del DAG puede aliviar dichos cuellos de botella.
El Tangle [56] de IOTA es una tecnología que aumenta la velocidad de las transacciones en comparativa con Blockchain. Permite que se puedan realizar dichas transacciones sin coste. Los sistemas basados en Blockchain como Bitcoin o Ethereum utilizan la Blockchain que tiene bloques secuenciales, con múltiples transacciones dentro de un bloque. El tipo de grafo acíclico dirigido (DAG) que utilizan las cadenas de bloques clásicas restringe la conexión a un único camino, es decir, encamina al siguiente bloque. En el caso de IOTA

[1] `https://www.iota.org`

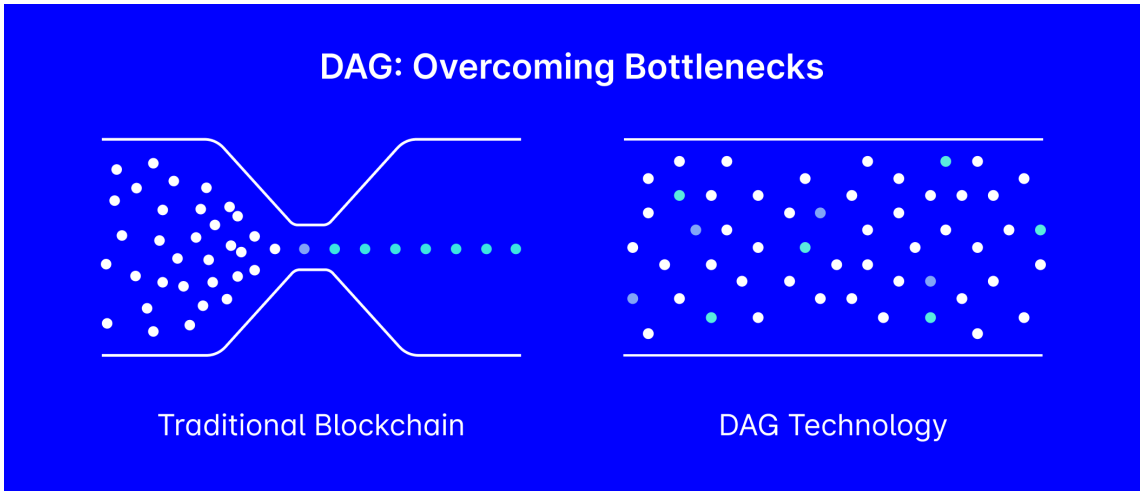

Figura 2.4: Cuellos de botella en Blockchain y en Tangle [14]

se utiliza otro tipo de DAG menos restrictivo denominado Tangle. En la red Tangle de IOTA no existen bloques, cada nueva transacción hace referencia a las dos transacciones anteriores y no es necesario para obtener un consenso inmediato para añadir una nueva transacción. Cuando un participante quiere añadir una nueva transacción a Tangle debe aprobar las dos transacciones previamente añadidas. El consenso está relacionado con el número de transacciones que aprobaron una determinada transacción. La transacción que recibe un mayor numero de aprobaciones tiene un mayor nivel de confianza. Dado que son los propios usuarios de la red quienes validan las transacciones, no es necesaria la figura del minero. El coste de la transacción depende del coste computacional de la validación y no está relacionado con comisiones a mineros. Esto reduce significativamente los costos, permitiendo que las transacciones en la red IOTA sean prácticamente gratuitas, algo ideal para un entorno IoT.

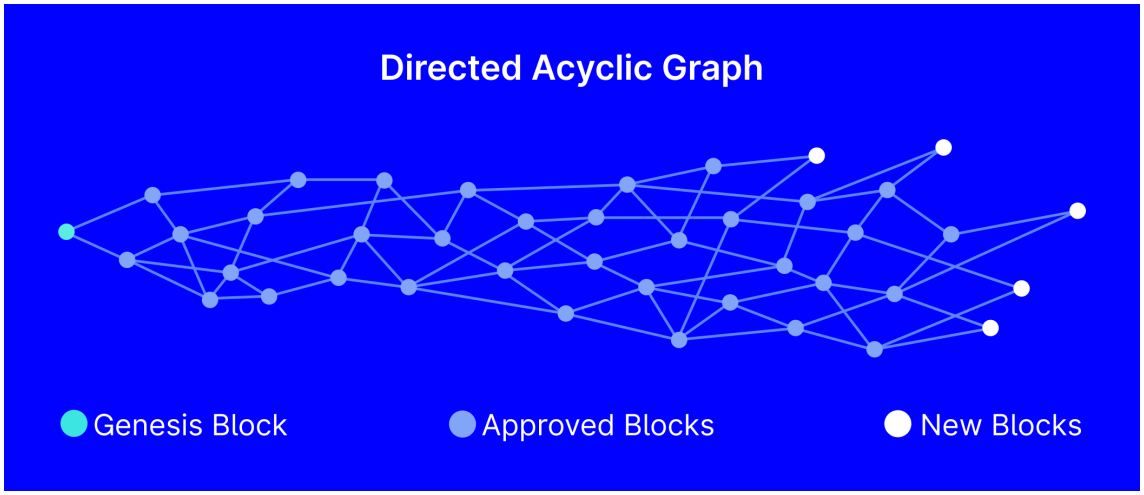

Figura 2.5: Tangle de IOTA [14]

En la figura 2.5 se plantea el Tangle de IOTA como una malla donde cada transacción apunta a sus dos transacciones predecesoras, creando así una estructura de grafo acíclico dirigido (DAG). Una Blockchain tradicional, donde las transacciones se agrupan en bloques formando una cadena, difiere del Tangle propuesto por IOTA en que permite una mayor

escalabilidad e intenta evitar cuellos de botellas gracias a la estructura del DAG. [55]

2.3.2. Redes y actualizaciones

Actualmente IOTA esta desarrollando IOTA 2.0 [34], que es el futuro del protocolo. Como se puede ver en la figura 2.6 se observas diferencias notables entre la versión 2.0 y la versión Legacy de IOTA. Entre las diferencias destacables se puede ver que IOTA 2.0 provee de soporte para smart contracts, el tamaño de una transacción es considerablemente menor en IOTA 2.0, que es totalmente descentralizada y permite mayor escalabilidad con respecto a la versión Legacy.

Feature	IOTA 2.0	Legacy IOTA
Smart contracts	Supported	Not supported
Digital asset support	Yes	No
Transaction size	100 bytes	1700 bytes
Decentralization	Fully decentralized	Coordinator as point of centralization
Sybil protection	Mana reputation system	None
Spam prevention	Lightweight Adaptive PoW	PoW
Address types	Reusable	One-time use
Consensus mechanism	FPC binary voting protocol	Weighted MCMC tip selection with coordinator
Scalability	Very scalable (increased TPS with increased network size)	Limited (coordinator and milestones limit scalability)
Approvement finality	Based on consensus mana approval weight – no orphans	Based on MCMC weight magnitude – orphans possible

Figura 2.6: Diferencias entre Legacy IOTA e IOTA 2.0 [53]

2.3.3. Smart Contracts de IOTA

Los smart contracts o contratos inteligentes son programas autónomos ejecutados en la capa 2 por el comité autorizado (fuera del Tangle) [3]. El Comité de Nodos actualiza colectivamente el libro mayor enviando transacciones firmadas al Tangle usando la firma umbral. Los contratos inteligentes de IOTA requieren menos recursos que la blockchain. Esto es importante en el ámbito de IoT, donde se espera que los micro-contratos y las micro-transacciones sean normales. Los contratos inteligentes de IOTA se clasifican como una máquina de estados inmutable:

- Máquina de Estados: Cada contrato inteligente tiene un estado vinculado al Tangle. El estado incluye datos como saldos de cuentas y condiciones de entrada. Cada actualización de estado representa un cambio de estado en el Tangle.
- Inmutable: El código del estado y el programa del contrato inteligente son permanentes ya que se almacenan en el Tangle. El estado puede actualizarse de manera incremental vinculando nuevas transacciones al Tangle.

El Tangle crea una pista de auditoría verificable para las transiciones de estado. Asume que las transiciones de estado son precisas y no pueden ser manipuladas por nodos maliciosos o incorrectos. Los contratos inteligentes de IOTA tienen una forma natural de ejecutar computación distribuida. Cada contrato inteligente puede ejecutarse en un ámbito localizado sin requerir que toda la red lo ejecute. Este enfoque asegura que los contratos inteligentes de IOTA no se conviertan en un obstáculo para escalar la red IOTA en el futuro. [2]

Los smart contracts se despliegan en IOTA a través de la capa 2, donde los comités autorizados y los nodos de validación son los encargados de procesar las transacciones y asegurar que se cumplan las condiciones establecidas en el contrato. Estos contratos pueden abarcar una amplia gama de aplicaciones, desde la automatización de micro-pagos hasta la creación de sistemas autónomos de toma de decisiones en el ámbito de IoT.

Para facilitar el despliegue de los contratos y la interacción con los mismos se emplea Metamask. Metamask se integra con la Máquina Virtual Ethereum (EVM) [18] utilizada por la IOTA Smart Contracts (ISC). Actúa como puente entre el usuario y la red. Esto significa que Metamask actúa como una interfaz que permite a los usuarios interactuar con los contratos inteligentes en la IOTA VM, de igual misma manera que sucede en la Ethereum VM.

2.4. Zero-Knowledge Proof

Un Zero-Knowledge Proof o prueba de conocimiento cero (ZKP) es una prueba de una afirmación que no revela nada más que la veracidad de una afirmación. Una prueba es un protocolo aleatorio mediante el cual una parte (llamada el prover) desea convencer a otra parte (llamada verificador) de que una afirmación dada es verdadera.

Según la definición que se aporta en el paper escrito por Austin Mohr [39] se puede definir un ZKP como ún sistema interactivo para un conjunto S es un juego de dos partes entre un verificador que ejecuta una estrategia probabilística en tiempo polinomial y un probador que ejecuta una estrategia computacionalmente ilimitada que satisface:

- **Completeness:** Para cada $x \in S$, el verificador siempre acepta después de interactuar con el prover sobre la entrada común x.
- **Soundness:** : Para un polinomio p, se cumple que para cada$x \notin S$ y cada estrategia potencial P^*, el verificador rechaza con probabilidad al menos $\frac{1}{p(|x|)}$ después de interactuar con P^* sobre la entrada común x.

La clase de problemas que tienen sistemas de prueba interactivos se denota **IP**.

Esto quiere decir que una prueba se considerará completa si el verificador esta siempre convencido de una afirmación por parte del probador. Ambas partes se suponen honestas. También se deduce que una prueba es solida si un probador "deshonesto"puede convencer a un verificador que una prueba falsa sea verdadera con una pequeña probabilidad. En otras palabras, una prueba es solida si las pruebas falsas se reconocen como tal, y no como verdaderas. Según lo que plantea Austin Mohr [39] se considera que una prueba es de conocimiento cero si cumple con la definición: Úna estrategia A es de conocimiento cero sobre el conjunto S si, para cada estrategia factible B^*, existe un cálculo factible C^* tal que los siguientes dos conjuntos de probabilidad son computacionalmente indistinguibles:

- el resultado de B^* después de interactuar con A sobre la entrada común $x \in S$
- el resultado de C^* sobre la entrada $x \in S$'

En la definición anterior, el primer conjunto representa el resultado de una ejecución real del protocolo del sistema de pruebas, mientras que el segundo conjunto (llamado la "simulación") es el resultado de un procedimiento independiente que no es parte de ningún

sistema interactivo. Una prueba se llama de conocimiento cero si el resultado de cualquier estrategia B^* utilizada por un verificador tramposo también podría ser producido por el cálculo no interactivo C^*. En otras palabras, cualquier información que pueda ser aprendida interactuando con A sobre alguna entrada x también puede ser extraída de x sin interactuar con A. [39]

2.4.1. Propiedades de Zero-Knowledge Proof

Un Zero-Knowledge Proof para denominarse como tal debe cumplir las propiedades conocidas [28] como: *completeness, soundness, y zero-knowledge*. A continuación se detallaran en que consisten estas propiedades.

- *Completeness*: En el caso de una afirmación sea verdadera, un verificador honesto puede ser convencido por un probador honesto de que este posee conocimiento sobre la entrada correcta.

- *Soundness*: En el caso de una afirmación sea falsa, ningún verificador deshonesto podrá convencer a un verificador honesto de que este posee conocimiento sobre la entrada correcta.

- *Zero-knowledge*: Si el enunciado es cierto, el verificador no aprende nada del prover más allá de que el enunciado es verdadero.

Dentro de los casos de uso de los ZKP se destaca la privacidad. La finalidad es aumentar la privacidad en ámbitos como las identidades digitales, la cual es el caso de uso que se ha tomado para el ejemplo. Pero este no es el único caso de uso en el que los zkp son aplicables en ámbitos como las finanzas, votaciones electrónicas o intercambio de información confidencial.

2.4.2. SNARK

Los zk-SNARKs (Zero-Knowledge Succinct Non-interactive Arguments of Knowledge) han ganado cada vez más interés debido a su uso para escalar el rendimiento de blockchain y proporcionar formas seguras de realizar transacciones. Los zk-SNARK son útiles para los rollups de blockchain en los que el prover quiere mantener en secreto el testigo de su cálculo.[12]

zk-SNARKs permite a una de las partes (el prover) genere una prueba sucinta[2] que permita a la otra parte (el verificador) comprobar de forma eficiente la corrección de la prueba. Lo que hace especialmente atractivas a las zkSNARK es que el prover y el verificador pueden generarse automáticamente mediante un circuito aritmético. Un circuito aritmético es un conjunto de ecuaciones polinómicas sobre un campo finito Por tanto, dado un cálculo f en la entrada x, los desarrolladores de aplicaciones ZK sólo necesitan construir un circuito aritmético C tal que y= f (x) sólo si C(x,y) es verdadero. Entonces, dado dicho circuito C, se puede utilizar un generador de pruebas zkSNARK para generar un prover y un verifier, lo que permite al prover establecer que y = f (x) sin revelar ninguna información sobre x al verificador. [69]

[2]Definición de sucinta `https://dle.rae.es/sucinto`

2.4.2.1. Circuitos aritméticos

Los zk-snarks utilizan los circuitos aritméticos como base para generar las pruebas. Un circuito aritmético es una representación matemática y lógica de un conjunto de operaciones que deben ser verificadas por la prueba de conocimiento cero.
El circuito aritmético implementa operaciones aritméticas básicas. Opera sobre un campo finito, que es un conjunto finito de elementos, típico de criptografía, donde se definen las operaciones de suma y multiplicación, con propiedades específicas como cerradura, asociatividad, y la existencia de elementos neutros e inversos. Al realizar operaciones sobre un campo finito, las entradas, salidas y las pruebas pueden ser gestionadas de manera eficiente y segura. Esto es debido a que las operaciones se limitan a un conjunto de valores fijado.
Es importante la estructura del campo finito para garantizar que las pruebas generadas sean verificables eficientemente y seguras, asegurando que la verificación de la computación pueda ser realizada sin revelar la información privada del prover.

2.4.3. Groth16

Groth16 es un protocolo de ZK-SNARK introducido en 2016 por Jens Groth. [28] Groth16 está basado en emparejamiento y puede garantizar privacidad. Es zero-knowledge debido a que es un tipo de ZK-SNARK, por tanto es necesario de una configuración confiable.
Las pruebas generadas por ZK-SNARk se destacan por ser breves y concisas, por lo que se requiere menos espacio para almacenarlas y transmitirlas. El probador puede generar una prueba sucinta que atestigua la validez de una declaración, mientras que el verificador puede verificar eficientemente la prueba. Una prueba contiene sólo 3 elementos de grupo, y la verificación consiste en comprobar el producto par de una ecuación utilizando 3 emparejamientos en total. La brevedad de las pruebas es una propiedad interesante debido a las limitaciones que ofrecen las cadenas de bloques.
El protocolo de Groth16 consta de los siguientes pasos:

- Configuración: Se generan los parámetros comunes, como la elección de curvas elípticas[3] adecuadas, la definición de la función de emparejamiento bilineal y el establecimiento de los parámetros públicos necesarios.
- Prueba: En este paso se genera la prueba realizando una serie de transformaciones matemáticas y cálculos sobre los datos de entrada, asegurándose de que siguen siendo válidos y no revelan ninguna información adicional.
- Verificación: En este paso el verificador utiliza los parámetros públicos y la prueba proporcionada por el prover para realizar una serie de comprobaciones y cálculos matemáticos. Si la prueba supera todas las comprobaciones de verificación, el verificador acepta la prueba y concluye que la declaración es válida con una alta probabilidad.

Groth16 será uno de los algoritmos que se utilizarán a lo largo del desarrollo del proyecto.

2.4.4. Plonk

Plonk (Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge) [21] es un protocolo de ZK-SNARK planteado por Ariel Gabizon et al. que

[3]Mas sobre curvas elípticas: `https://math.uchicago.edu/~may/REU2020/REUPapers/Shevchuk.pdf`

plantea mejoras sobre protocolos anteriores de zk-snarks como pueden ser Groth16. Se plantea Plonk como un protocolo universal y actualizable, esto quiere decir que puede aplicarse a una variada selección de circuitos aritméticos sin que exista necesidad de cambiar su configuración inicial.

Plonk utiliza una configuración basada en circuitos aritméticos. Cada circuito se transforma en un conjunto de polinomios de grado limitado que describen las relaciones aritméticas entre las variables de entrada, salida y las puertas intermedias.
Las polinomios se evalúan en puntos específicos conocidos como bases de Lagrange[4]. Estos puntos son utilizados para interpolar los polinomios, facilitando la verificación eficiente de las pruebas sin necesidad de reconstruir el circuito completo. Esto se traduce en una mayor eficiencia computacional tanto para el probador como para el verificador.
Plonk ha sido diseñado para ser soportado en distintas blockchains, DLT y sistemas de identidad digital. A diferencia de algunos esquemas que requieren una configuración de confianza, Plonk puede operar sin esta característica, lo cual incrementa la confianza en el protocolo.
Plonk será el otro algoritmo de Snark que se utilizará para el desarrollo del proyecto.

2.4.5. SNARKs recursivos

Otro tipo de SNARKs que se consideran son los snarks recursivos. Es un sistema que genera pruebas de manera paralela para varios bloques de una transacción y las combina para crear una única prueba que se envía a la cadena de bloques. Esto quiere decir que una prueba puede verificar a otra prueba. Los SNARKs recursivos utilizan la capa L2 de Ethereum. Las pruebas recursivas verifican transacciones en múltiples bloques en L2 , las cuales pasan a ser validas una vez la prueba general, la que engloba a todas las anteriores, es aceptada en L1 de Ethereum.

Estas pruebas recursivas son diferentes a un ZK-SNARK normal porque los ZK-SNARK recursivos pueden verificar más de un bloque de transacciones combinando SNARK generados para diferentes bloques L2 en una única prueba de validez que se envía a la cadena L1. Estos pueden verificar transacciones que han sido verificadas por los zk-snarks normales sin utilizar las entradas originales. Por tanto, el smart contract ejecutado en la blockchain posee un mayor conjunto de cálculos fuera de la cadena.

El proceso de recursión tiene las siguientes etapas:

1. Probar la corrección del calculo de cada capa.
2. Componer los SNARKs recursivamente para generar una prueba general.
3. Verificar la prueba utilizando el algoritmo de verificación que se prefiera.

2.4.6. STARK

Zk-STARK (Zero-Knowledge Scalable Transparent Arguments of Knowledge) son un tipo de ZKP propuesto por Eli-Ben Sasson et al. [6] [7]. Se considera una evolución y una variante mas eficiente que los SNARKs

[4] `https://www.ingenieria.unam.mx/pinilla/PE105117/pdfs/tema4/4-1_lagrange.pdf`

Las pruebas SNARKs no son interactivas, es decir, que el probador genera la prueba y y la envía al verificador de una sola vez. En cambio, las pruebas STARK son interactivas y precisan de una comunicación de ida y vuelta entre probador y verificador. Las pruebas STARK también son transparentes, es decir, pueden ser consultadas públicamente. En cambio en los SNARKs las pruebas no son transparentes: a pesar de ser sucintas estas no son transparentes. Los SNARKs, aun siendo eficientes, pueden tener dificultades para circuitos de gran complejidad, mientras que las pruebas STARK son altamente escalables, haciéndolas aptas para cálculos complejos. Otra caracteristica que diferencian a los Stark de los Snark es la falta de una fase de configuración externa en la que confíe un tercero, por lo que la aleatoriedad que se utiliza para generar la prueba es publica y verificable, lo cual provoca que las pruebas Stark sean más seguras que las Snark. Al eliminar la etapa de configuración de confianza, se evita el riesgo de manipulación de las pruebas por parte de un tercero.

En resumen, se puede establecer las siguientes características como propias e identificables de los STARKs:

- Interactividad: Quiere decir que la prueba requiere un proceso interactivo entre probador y verificador. Esto significa que requieren una comunicación bidireccional entre el probador y el verificador, lo que permite ajustar la prueba en función de las respuestas del verificador. Este proceso de interacción puede aumentar la flexibilidad y seguridad del sistema, aunque a costa de una mayor complejidad en la implementación.
- Transparencia: No depende de una configuración confiable inicial. La aleatoriedad utilizada para generar la prueba es publica y verificable.
- Escalabilidad: Permite manejar cálculos de alta complejidad y grandes volúmenes de datos. Esto se logra gracias a su estructura basada en técnicas de aritmetización que permiten mantener la eficiencia sin sacrificar la robustez del sistema.

También cabe mencionar que los zk-stark utilizan criptografía basada en la teoría de la información, lo cual ofrece protección contra ataques cuánticos. Sin embargo la computación cuántica aun esta a años de poder tener un implementación más cotidiana.

2.4.6.1. Representación algebraica intermedia (AIR)

En el contexto de ZK-STARK el concepto de aritmetización es primordial en la verificación de computaciones sin revelar detalles confidenciales. Este proceso consiste en la conversión de un programa a un formato matemático que sea adecuado para los ZK-STARKs, denominado Representación Algebraica Intermedia (AIR)

Según Eli-Ben Sasson [7] se define la representación algebraica intermedia (AIR) como un conjunto de polinomios de bajo grado con coeficientes en un campo finito F, definidos sobre dos conjuntos de variables $\tilde{X}$ y $\tilde{Y}$, que representan el estado actual y el estado siguiente de la computación, respectivamente. De manera más precisa, la AIR se formaliza como un conjunto

$$P = \{P_1(\tilde{X}, \tilde{Y}), \ldots, P_s(\tilde{X}, \tilde{Y})\}$$

donde cada P_i es un polinomio de bajo grado.
El conjunto P define la relación de transición de la computación C. Un par de estados

$$(\tilde{x}, \tilde{y}) \in F^w \times F^w$$

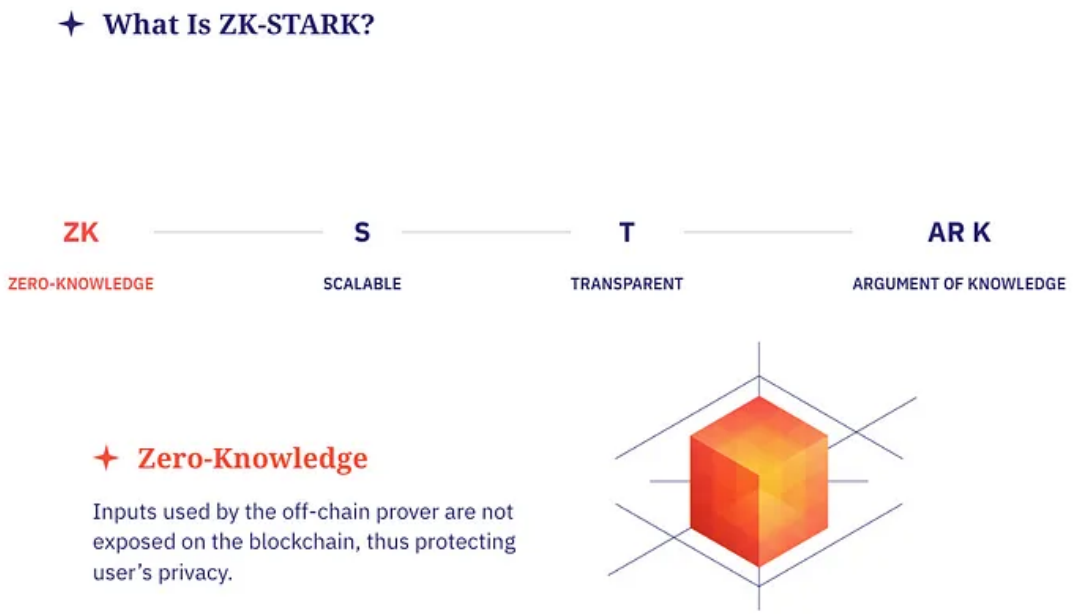

Figura 2.7: Propiedades de ZK-STARK [59]

corresponde a una transición válida de C si, y solo si,

$$P_1(\tilde{x}, \tilde{y}) = \ldots = P_s(\tilde{x}, \tilde{y}) = 0$$

es decir, si $(\tilde{x}, \tilde{y})$ es una solución común del sistema algebraico P.

Los parámetros clave de P influyen directamente en la complejidad tanto del probador como del verificador. Estos parámetros incluyen el grado de AIR

$$deg(P) = \max_{i=1}^{s} deg(P_i)$$

el ancho del estado w, que es el número de variables necesarias para representar un estado, el tamaño de AIR s, que es el número de restricciones, y el conteo de ciclos c, que representa el número de ciclos de máquina necesarios para ejecutar C. En escenarios donde el programa procesa un gran número n de elementos de datos, el interés se centra en el número de ciclos por elemento c, y el conteo total de ciclos es $c \cdot n$. En otros sistemas, se calcula también el número total de puertas de multiplicación en un circuito expandido, ya que tanto este número y la profundidad del circuito son medidas de complejidad influyentes en los costos de verificación y prueba.

Un factor significativo en la complejidad del probador dentro es el costo de probar la integridad computacional de invocaciones repetidas de una función hash criptográfica. Por lo tanto, la elección de la función hash H y su definición en términos de P es crucial. La función hash criptográfica seleccionada debe ser compatible con el campo binario, es decir, su AIR debe tener parámetros de complejidad pequeños cuando se define sobre campos binarios.

2.4.7. Bulletproofs

En 2017 Benedikt Bunz, Jonathan Bootle et al [8], propusieron un nuevo concepto de conocimiento cero que denominaron como Bulletproofs, el cual puede demostrar que un

secreto se encuentra dentro de un rango determinado. Un bulletproof es una prueba de conocimiento cero corta y no interactiva que no necesita de una configuración de confianza. Este tipo de pruebas pueden convencer a un verificador de que un valor cifrado esta dentro de un rango establecido sin revelar ningún tipo de información sobre dicho valor. Los

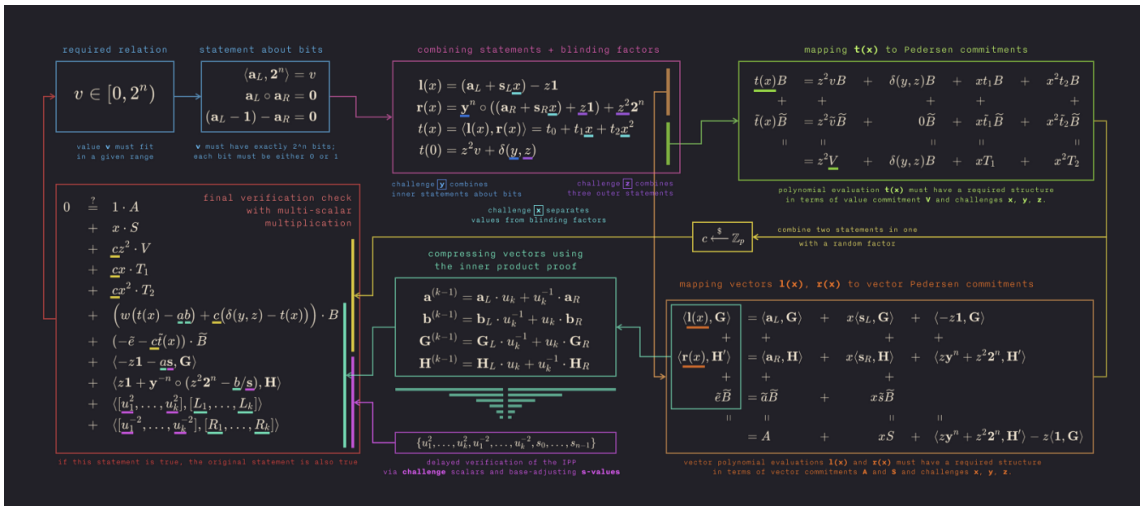

Figura 2.8: Ilustración de Oleg Andreev para el protocolo de Bulletproofs [74]

bulletproofs emplean técnicas similares a las utilizadas por zk-snark y zk-stark. De manera similar a zk-stark, esta no requiere de una configuración de confianza, sin embargo son mas pequeñas que los zk-stark. Con respecto a zk-snark, bulletproofs tardan mas tiempo en verificarse.

2.5. Estado del Arte

En esta sección se explorarán los trabajos existentes sobre ZKP y los estudios existentes sobre comparativas entre los distintos tipos de ZKP.

2.5.1. Investigación sobre ZKP

Los Zero-Knowledge Proof son un campo de conocimiento novedoso que da pie a investigaciones recientes. En estas investigaciones y trabajos existentes se exponen conceptos matemáticos avanzados que sirven para definir los ZKP. Siguiendo las definiciones matemáticas se extrae la definición de un ZKP como una prueba (proof) en la que una parte (el probador) puede demostrar a otra parte (el verificador) que una afirmación es verdadera, sin necesidad de revelar ningún tipo de información adicional más allá de la veracidad de la afirmación. Estas definiciones matemáticas que plantean el concepto de Zero Knowledge Proof se establecen en Goldreich et al. en el artículo [25]. También se puede encontrar información sobre la composición de los ZKP de manera matemática en el estudio realizado por Goldreich et al. en el artículo [24]. Groth escribió el artículo [28] donde se define el protocolo Groth16, con sus características y propiedades. Este artículo también proporciona una base teórica para el desarrollo de zk-snarks.

El concepto de Zero Knowledge Proof se ha convertido en una herramienta interesante por su potencial en la aplicación para identidades digitales. En el estudio realizado por

Yang y Li [73] se explora la implementación de algoritmos de ZKP para mejorar los modelos existentes de gestión de identidades digitales en blockchain. ZKP juega un papel clave al permitir la verificación de identidad sin necesidad de revelar información sensible, sugiriéndose como parte imprescindible en la gestión de identidades descentralizadas. Saarela en el articulo [52] lleva más allá el planteamiento anterior al aplicar ZKP en el marco de protocolos que confirman la Identidad Digital Europea (EUDI). El uso de ZKP permite una solución eficiente y segura para gestionar identidades digitales para proteger los derechos de los ciudadanos europeos. En el articulo [1] Akram y Sen explora un enfoque descentralizado utilizando ZKP para transformar la identidad digital clásica como contraseñas o biometría a una identidad digital más segura en el sector bancario. En este artículo se adopta un enfoque más específico que en los dos anteriores artículos. En este caso ZKP se plantea como un elemento clave para solventar vulnerabilidades clásicas de los sistemas tradicionales de autenticación. Todos los artículos exploran el uso de ZKP como una herramienta para mejorar la privacidad y seguridad en la gestión de identidades digitales. Mientras que [73] se centra más en aspectos teóricos, [52] y Akram y Sen se centran más en aplicaciones más especificas.

Otra aplicación de ZKP es ZCash, que se plantea en el artículo escrito por Hopwood et al. [29], la cual es una implementación de del esquema de pago anónimo descentralizado Zerocash, al cual se le ha mejorado el rendimiento, se le ha añadido funcionalidad y seguridad a mayores. Se utiliza ZKP para combinarlo con esquema de pago transparente transparente utilizado por Bitcoin. La intención de esta fusión es conseguir un esquema de pago blindado.

Dentro de los Zero Knowledge Proof se pueden categorizar en dos grandes grupos: las pruebas interactivas y las pruebas no interactivas. Las pruebas interactivas fueron formuladas por Goldwasser, Micali y Rackoff en un trabajo pionero hasta el momento [26]. En dicho trabajo se planteó que este tipo de pruebas requieren de interacción directa entre probador y verificador, donde cada parte intercambian mensajes para demostrar la validez de una afirmación sin revelar ninguna información subyacente. En este trabajo se han establecido las bases para sistemas interactivos que pueden ser utilizados en la verificación segura de identidades y otras aplicaciones de seguridad. Por otra parte, Blum, Feldman y Micali demostraron en [15] que tan solo es necesario que el probador 'hable'y que el verificador éscuche'sin necesidad de que se produzca una interacción directa entre ambas partes. De este modo nace el concepto de prueba no interactiva, de la que saldrán después las ideas que conducen al desarrollo de SNARK y STARK. Siguiendo esta linea de investigación en [72] Wu y Wang revisan y analizan los principios básicos del sistema ZKP no interactivo así como sus posibles aplicaciones en el mundo real. El artículo ofrece una visión exhaustiva sobre cómo las pruebas no interactivas pueden ser aplicadas en diversos sectores. Este trabajo ayuda a conectar los conceptos teóricos desarrollados en los artículos anteriores con el desarrollo de aplicaciones prácticas. Estos tres artículos son fundamentales en un trabajo sobre ZKP, pues ayudan a entender la evolución de los ZKP desde su planteamiento interactivo en un inicio hasta la forma interactiva, pasando por las aplicaciones prácticas.

En la literatura disponible a los sistemas ZKP no interactivos se encuentran trabajos relativos a lo que se conoce como ZK-SNARKs. En los trabajos [12] y [44] se plantean los fundamentos técnicos que sostienen zk-snarks y aplicaciones novedosas como zkEVM. En el artículo [48] Pinto plantea la implementación de zk-snarks en el ecosistema blockchain, adaptándolos para cada aplicación específica. Esta integración de zk-snarks en blockchain

permite utilizar las pruebas como datos de transacción, se ocultan los detalles privados garantizando simultáneamente la integridad y exactitud de la transacción. Las pruebas se verificarían en la cadena con mediante contratos inteligentes.

En el estudio [70] realizado por Wen et al. se analiza la seguridad de los circuitos de Zero-Knowledge Proof. Se aporta información relevante sobre circuitos realizados Circom y las diferentes curvas elípticas que soportan los ZKP. Con respecto al trabajo desarrollado aporta una base teórica para los circuitos de Circom. En el estudio escrito por Nitulescu [44] se plantea una base teórica muy solida para entender de una manera más accesible el funcionamiento de ZK-SNARK, aportando los fundamentos básicos y desarrollo matemático y criptográfico de este tipo de pruebas. Para este trabajo servirá de base teórica para definir correctamente las pruebas de conocimiento cero Snark

Respecto a STARK, la otra variante de las pruebas no interactivas, Ben-Sasson, Goldberg y Levit presentaron un articulo donde se definían las pruebas STARK [6]. En este artículo se presenta la definición matemática de una prueba STARK, explica el concepto de AIR, y el proceso de generación de una prueba STARK a partir del AIR. Este artículo sirve de base teórica para el desarrollo del prover y verifier STARK que se desarrollará. En este artículo, complementa al articulo anterior, explicando de manera simple que son las pruebas STARK.

En el estudio realizado por Tennant [65] se ofrecen distintas alternativas para mejorar la privacidad en IOTA, entre esas alternativas se propone la utilización de pruebas ZKP. Este estudio puede proporcionar una base para implementar pruebas ZKP en IOTA

2.5.2. Trabajos relacionados sobre comparativas con distintos ZKP

El interés creciente en los Zero Knowledge Proof y sus distintos tipos de implementación han dado pie a nuevas investigaciones y a una serie de estudios comparativos que analizan sus características, ventajas y desventajas. Uno de estos trabajos es [41], en el cual en la sección 6 se plantea una comparación entre distintos algoritmos planteados. Se comparan los tamaños de la prueba generada y la complejidad de los algoritmos del prover y verifier. En relación con el trabajo a desarrollar, es interesante ver los resultados que ofrece el algoritmo Groth.

En el estudio realizado por Sun et al. [63] se comenta de manera superficial una comparativa de la complejidad de los algoritmos del prover, verifier y el tamaño de la prueba generada en distintos modelos de zk-snarks. Esta comparativa se puede resumir en la imagen 2.9:

	Ben-Sasson's model	Ligero	Bulletproofs	Hyrax	Aurora	Libra
Trusted setup	Yes	N/A	N/A	N/A	N/A	Yes
Prover algorithm	$O(c \log c)$	$O(c \log c)$	$O(c)$	$O(c \log c)$	$O(c \log c)$	$O(c)$
Verification algorithm	$O(1)$	$O(c)$	$O(c)$	$O(\sqrt{s} + d \log c)$	$O(c)$	$O(d \log c)$
Proof size	$O(1)$	$O(\sqrt{c})$	$O(\log c)$	$O(\sqrt{s} + d \log c)$	$O(\log^2 c)$	$O(d \log c)$
Implementation technique	Quadratic arithmetic programs	Interactive oracle proofs	Discrete logarithm	Interactive proofs	Interactive oracle proofs	Interactive proofs

TABLE 1. Comparison of improved zkSNARK models.

Figura 2.9: Comparación de la complejidad de modelos de zk-snark [63]

En el trabajo realizado por Gong et al. [7], en el capitulo 1 se plantea una comparativa de ZK-STARKS con otros sistemas ZK. En este estudio se acota a sistemas ZK Turing completo y que se han realizado en código. La finalidad de este estudio es realizar una comparativa en términos de escalabilidad del prover y el verifier, así como también se incide en su seguridad post-cuántica.

Otro trabajo que analiza en mas detalle entre distintos tipos de ZKP es [27]. Se comparan cuatro esquemas de ZKP (ZK-SNARK, ZK-STARK, MPC y Bulletproofs) atendiendo a cuatro enfoques distintos:

- Configuración confiable o trusted stup, donde se explora la necesidad de la generación de un conjunto de parámetros públicos y privados para que las pruebas sean correctas y seguras.
- Complejidad, atendiendo a los algoritmos del verificador y del probador.
- Aplicaciones, teniendo en cuenta los escenarios donde se puede aplicar cada esquemas de ZKP
- Amenaza cuántica o quantum threat, donde se explora la seguridad de cada esquema ZKP a ataques producidos por ordenadores cuánticos

Se realiza un análisis de Snark y Stark a todos los niveles. Primeramente se compara la necesidad de una configuración confiable y después la complejidad de la comunicación, de la prueba y del verificador. Es un estudio que se plantea de manera teórica y sirve de una base muy solida para el trabajo que se está realizando, y así aplicar esta base teórica a un escenario real.

Para este trabajo se plantea un escenario en el que se pueda comparar pruebas de conocimiento cero en IOTA y Ethereum. En la literatura disponible se encuentran casos de estudio que utilizan benchmarks con la finalidad de realizar una comparación cualitativa. En el estudio [36] realizado por Kobelt, Sober y Schulte se comparan distintos esquemas de ZKP, y entre ellos se compara también las curvas y esquemas que se utilizan durante la generación de dichas pruebas. En este estudio comparativo se utilizan tres métricas diferentes: Wall Time, CPU Time y Peak Memory. Estas métricas miden la eficiencia y el rendimiento de los esquemas ZKP escogidos. Según lo establecido en el estudio comentado, Wall Time es el tiempo total que toma generar y verificar una prueba, CPU Time es el tiempo que la CPU invierte en los cálculos necesarios para generar y verificar una prueba, mientras que Peak Memory es la cantidad máxima de memoria que se requiere en cualquier momento durante la generación y verificación de una prueba. Es interesante con respecto a este trabajo ver las distintas métricas escogidas y la metodología utilizada para el estudio estadístico realizado. Aporta una base sobre la que realizar el análisis y las métricas a utilizar sobre la comparación de ZKP.

Entre la literatura existente se han encontrado una serie de trabajos de fin de máster de distintas universidades europeas que se pueden relacionar con este trabajo de fin de máster. En el primero de ellos, [11], se realiza un análisis de IOTA y Ethereum para la comunicación de dispositivos IoT. Este trabajo es relevante debido a que el enfoque del estudio es muy similar al trabajo que se está llevando a cabo aquí. Se explora el despliegue de ZKP en IOTA y Ethereum centrándose en la capacidad de comunicación entre dispositivos IoT, mientras que en nuestro caso se añadirá una capa adicional al explorar el rendimiento

de las pruebas de conocimiento cero en ambas plataformas. En el segundo de ellos, [40], se explora la utilización de smart contracts de IOTA. La implementación de estos contratos inteligentes en la red de IOTA es clave para entender cómo las pruebas ZKP pueden interactuar con la red. Este trabajo ofrece una base para entender como IOTA maneja los contratos inteligentes. En el tercero de ellos, [35], se realiza una comparación en la implementación de smart contracts en Ethereum e IOTA. Esta comparación proporciona información crucial sobre las diferencias técnicas entre ambas plataformas, lo que puede influir en cómo los distintos algoritmos de ZKP se implementan y funcionan en cada una de las plataformas. También proporciona información relevante para este estudio sobre la forma de comparar smart contracts en las dos plataformas y las métricas utilizadas. Este trabajo utiliza como métricas el coste, el tiempo de ejecución y el gas utilizado por el smart contract. El trabajo actual nuestro va más allá al estudiar no solo los contratos si no también las pruebas de conocimiento cero. En el cuarto de los trabajos de fin de máster encontrados, [58], se plantea una evaluación de la seguridad de IOTA en un entorno IoT. Este trabajo permite comprender IOTA desde el punto de vista de la seguridad, la cual se puede complementar con los ZKP aplicados a la privacidad de los datos. En el quinto y último TFM encontrado, [5], se habla sobre implementación de zk-snarks en Ethereum. Este trabajo sirve como base para comprender como Ethereum trabaja con zk-snarks. El trabajo que se desarrolla aquí compara diferentes enfoques de pruebas de conocimiento cero, Snark y Stark, y como se despliegan estos en Ethereum e IOTA.

En el estudio realizado por Ibba et al. [30] se realiza un estudio sobre métricas asociadas a la complejidad los smart contracts de Solidity. Entre las métricas estudiadas se observa la versión de Solidity, métricas asociadas al numero de líneas de código, variables de estado, funciones, parámetros y llamadas a funciones. También se mide el acoplamiento entre contratos, la dependencia con librerías y la complejidad ciclomática. Este estudio ofrece una base solida para realizar un análisis de la complejidad de los contratos desarrollados en este trabajo.

En el articulo [54] se comparan ZK-SNARK y ZK-STARK, donde se explica de forma accesible las diferencias entre ambos tipos de ZKP, sus características propias y el proceso de comparación. Con respecto al trabajo actual, es interesante destacar las métricas utilizadas para el análisis. Entre esas métricas se destacan el tiempo del prover, el tamaño de la prueba, el tiempo en la verificación y el gas utilizado en la verificación. En este otro artículo [37] se comparan Snark de manera general, es decir, se comparan entre distintos algoritmos. En relación a este trabajo, es relevante la comparación entre Plonk y Groth16. En este caso, se enfoca en medir métricas relativas a tamaño y tiempo de verificación y generación. En este trabajo se añadirá mayor granularidad a la comparación al tener también en cuenta el gas consumido durante la verificación. El articulo [77] complementa a los anteriores ofreciendo una visión más gráfica de las comparaciones entre Stark y Snark. Se incide en la evolución durante el tiempo de la complejidad de la comunicación y de los tiempos de generación y verificación.

2.6. Herramientas utilizadas

En esta sección se desglosaran todas las herramientas utilizadas a lo largo del desarrollo de este trabajo.

2.6.1. Remix IDE

Remix IDE [68] es la herramienta que se ha utilizado para desarrollar los contratos inteligentes para probar los ZKP. Remix IDE es un entorno de desarrollo integrado basado en web, aunque existe una versión de escritorio, que facilita la escritura, compilación, implementación y depuración de contratos inteligentes escritos en el lenguaje de programación Solidity. Remix proporciona una interfaz de usuario intuitiva y diversas funcionalidades como la capacidad de conectarse a diferentes redes de Ethereum, desde redes de prueba hasta la red principal.

El editor de código de Remix incluye resaltado de sintaxis, auto completado y análisis estático para detectar errores potenciales en el código. El compilador permite utilizar diferentes versiones de Solidity a la hora de compilar los contratos inteligentes. Incluye un herramienta de depuración para analizar el comportamiento de los contratos y realizar seguimiento de las transacciones.

Además permite a integración con otros servicios y herramientas como pueden ser Metamask y Truffle. Remix tiene una arquitectura de plugins que permite a los desarrolladores añadir funcionalidades adicionales y personalizar su entorno de desarrollo. Entre los plugins que se han utilizado a lo largo del desarrollo de este trabajo se encuentran los siguientes:

- Circom ZKP Compiler: Compilador para circuitos de pruebas de conocimiento cero (ZKP) escritos en Circom.
- Debugger: Herramienta que permite depurar contratos inteligentes paso a paso.
- Remixd: Conecta con el entorno local para trabajar con archivos locales.
- Solidity Analyzers: Proporciona análisis estático para detectar posibles errores y optimizaciones en contratos Solidity.
- Zokrates: Herramienta para crear y verificar pruebas de conocimiento cero usando Zokrates.
- ZKsync: Permite interactuar con soluciones de escalabilidad mediante ZK Rollups en Ethereum.
- Starknet: Integración para interactuar con la red Starknet y sus soluciones de escalabilidad mediante STARKs.

2.6.2. Zokrates

Zokrates [78] [32] es un toolbox para zkSNARKS en Ethereum. Zokrates facilita la creación de estas pruebas proporcionando una serie de herramientas y un lenguaje de programación propio. Para desarrollar las pruebas Zokrates proporciona una serie de herramientas:

- Un lenguaje de programación de alto nivel especifico para escribir programas que se desean convertir en pruebas de conocimiento cero.
- Un compilador que convierte los programas escritos en Zokrates en sistemas de ecuaciones algebraicas que pueden ser utilizados para generar y verificar pruebas zkSNARK.

- Herramientas para la configuración de parámetros, la generación de claves públicas y privadas, y la creación de pruebas a partir de los programas compilados.
- Facilita la verificación de pruebas zkSNARK directamente en la cadena de bloques de Ethereum gracias a la generación contratos inteligentes. Estos contratos inteligentes se pueden incluir en otros contratos inteligentes.

Para el desarrollo del trabajo se ha utilizado el plugin de Zokrates de Remix IDE, el cual agrupa todas las herramientas mencionadas anteriormente. Este plugin se ha utilizado para desarrollar las pruebas de Groth16.

2.6.3. Circom

Circom [31] es un lenguaje de programación que permite diseñar circuitos aritméticos que pueden utilizarse para generar pruebas de conocimiento cero. El compilador de Circom es un compilador del lenguaje Circom escrito en Rust que puede utilizarse para generar un archivo R1CS (Rank-1 Constraint Systems) con un conjunto de restricciones asociadas y un programa (escrito en C++ o WebAssembly) para calcular eficientemente una asignación válida a todos los hilos del circuito. Esta herramienta permite trasladar los cálculos matemáticos necesarios para crear zk-SNARKs, para ello se utiliza SNARK.js, que es una implementación de JavaScript y PureWeb Assembly de esquemas ZK-SNARK. Utiliza el protocolo Groth16 y PLONK.

Para diseñar los circuitos aritméticos Plonk que se utilizan en este trabajo se ha utilizado el plugin de Circom para Remix IDE.

2.6.4. Cairo

Cairo es un lenguaje de programación diseñado exclusivamente para el desarrollo de pruebas de conocimiento cero eficientes. Se utiliza sobre todo en la construcción de sistemas de escalabilidad de capa 2 y en la creación de aplicaciones descentralizadas que requieren la verificación de cálculos complejos sin revelar datos sensibles. Cairo permite la creación de circuitos altamente optimizados que se pueden verificar de manera rápida y segura en la blockchain.

Cairo difiere con otros lenguajes de programación tradicionales, sobre todo en lo referido a gasto y ventajas. Los programas en Cairo se ejecutan de dos maneras distintas

- Cuando es ejecutado por el probador: el comportamiento es similar al de los lenguajes tradicionales, donde se realizan todos los cálculos necesarios y se genera una prueba de conocimiento cero.
- Cuando es ejecutado por el verificador: Solo se verifica la prueba generada, lo cual es un proceso mucho más rápido y eficiente en términos de recursos computacionales y de gas en la blockchain.

En Cairo el acceso a memoria es inmutable por lo que una vez que un valor se escribe en la memoria, este no se puede cambiar. Cairo 1 proporciona abstracciones que ayudan a los desarrolladores a trabajar con estas restricciones, pero no simula completamente la mutabilidad. Por lo tanto, se debe pensar cuidadosamente sobre cómo gestionar la memoria y las estructuras de datos para optimizar el rendimiento.

En un principio Cairo se utilizó para desarrollar un contrato que permitiera verificar una prueba Stark, pero ante la falta de documentación y de ejemplos se decidió descartar la utilización de esta herramienta.

2.6.5. Winterfell

Winterfell [19] es un software desarrollado por Meta que funciona como probador y verificador STARK para cálculos arbitrarios. Es un proyecto que aún está en desarrollo cuyo objetivo es crear un prover STARK rico en características, fácil de usar y de alto rendimiento que pueda generar pruebas de integridad para cálculos muy grandes. En este proyecto se utilizará para desarrollar un programa que sea capaz de generar y verificar una prueba Stark.

Winterfell es un prover y verificador para STARK multihilo que posee de una interfaz sencilla para realizar cálculos sencillos [46]. Winterfell tiene soporte aleatorio de AIR, lo cual permite, comprobaciones de conjuntos múltiples y permutaciones similares a las disponibles en los sistemas PLONKish. Winterfell es capaz de generar pruebas multihilo. Permite la configuración en la selección de campos y de funciones hash. Aunque es una herramienta desarrollada íntegramente en Rust, esta es compatible con WebAssembly. Por ultimo, también ofrece verificación de pruebas de manera asíncrona.

2.6.6. Metamask

Metamask [38] es una extensión de navegador web que funciona como monedero virtual que puede interactuar con blockchains como Ethereum. Metamask permite a los usuarios conectarse a redes DLT como IOTA así como blockhains. Al ser una extensión de navegador proporciona una interfaz sencilla para realizar transacciones y desplegar smart contracts. En este proyecto se utilizará como puente para desplegar contratos inteligentes en la EVM de IOTA, y para poder interactuar con dichos contratos desde el entorno web de Remix.

En la figura 2.10 se observa la arquitectura de Metamask, la cual proporciona una visión de su funcionamiento. Un usuario interactúa con Metamask a través de una extensión de navegador. El usuario tiene el control sobre sus claves privadas, que se almacenan en el dispositivo del usuario, lo que garantiza la seguridad. Metamask cifra y almacena estas claves en el navegador. Una aplicación que quiere interactuar con Metamask utilizará el Metamask SDK, el cuál actúa como puente entre la aplicación y las instancias de Metamask (extensión del navegador) utilizando JSON-RPC. Cuando el usuario interactúa con una app descentralizada, Metamask envía las transacciones a la red blockchain, a través de un nodo proveedor que retransmite las transacciones a la red blockchain. Toda transacción debe ser firmada y autorizada. En definitiva, Metamask actúa como un puente seguro entre el usuario y la blockchain, donde el usuario controla sus claves y puede interactuar con diferentes aplicaciones descentralizadas.

2.6.7. Solidity Metrics

Solidity Metrics[13] es una extensión de VSCode que sirve para analizar y medir diversas características y métricas del código de contratos inteligentes escritos en Solidity. Sirve para evaluar la calidad, complejidad y mantenibilidad del código. En este proyecto se utiliza para obtener metricas de la complejidad de todos los smart contracts utilizados

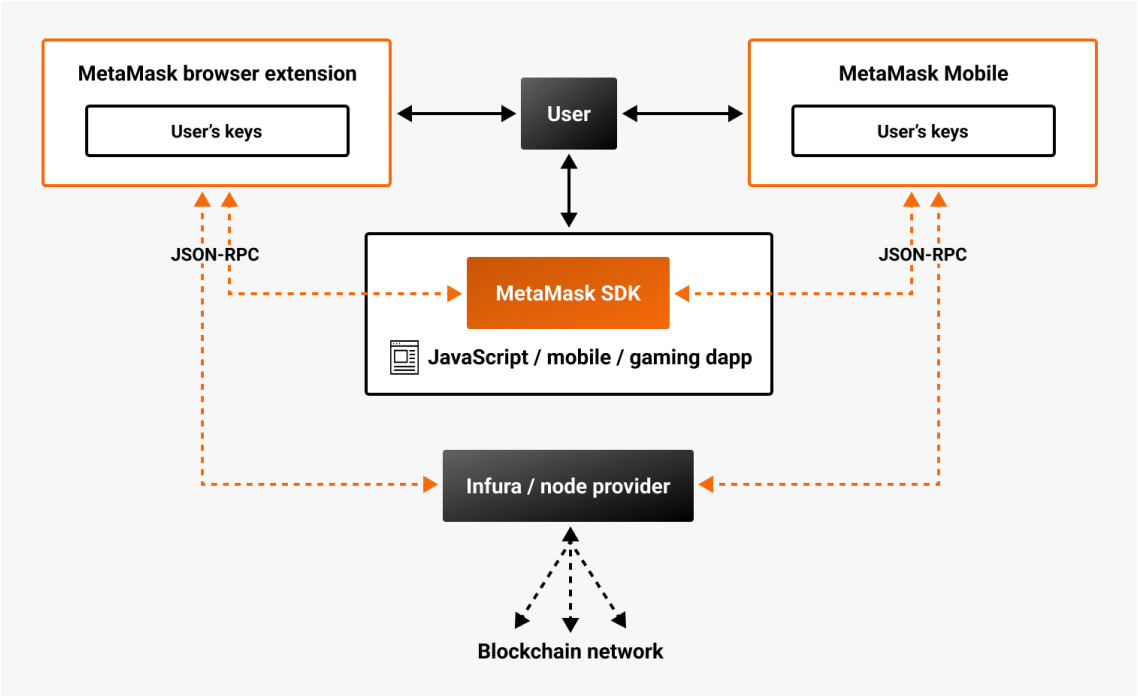

Figura 2.10: Arquitectura de Metamask [4]

a lo largo del proyecto

Al ejecutar el plugin sobre el código, se generan informes detallados y visuales del proyecto. Proporciona métricas relativas al numero de lineas de código, funciones y dependencias entre diferentes smart contracts. Se proporciona una puntuación de la complejidad de cada contrato. También se añaden gráficos que permiten visualizar las dependencias entre las distintas partes del proyecto para detectar posibles problemas.

Capítulo 3

Metodología

A lo largo de este capitulo se describirá la metodología seguida en el proceso del trabajo. Se describe cómo se han llevado a cabo las distintas fases del proyecto, desde la investigación y desarrollo hasta las pruebas y la escritura de la memoria. También se incluye el diagrama general que siguen los contratos inteligentes y las pruebas realizadas.

3.1. Fases del proyecto

El proyecto ha tenido una duración de 7 meses, desde febrero hasta agosto. A lo largo del mismo, se han implementado las siguientes fases:

- Investigación: Inicialmente, se investigó sobre qué es IOTA y sus posibles casos de uso. Una vez determinado que el caso de uso más interesante era el de las identidades digitales, se exploraron las tecnologías que podrían aplicarse a este ámbito. A partir de aquí, se sugirió la investigación y desarrollo de Zero Knowledge Proof (ZKP). Durante esta etapa, se recopiló material entre la literatura existente sobre ZKP y sus diferentes variantes. Tras la propuesta de realizar un análisis comparativo entre tres tipos de ZKP, se procedió a investigar y recopilar material entre la literatura existente sobre trabajos similares. Se investigo también las herramientas necesarias para el desarrollo de las pruebas y el posterior análisis, así como se establecieron una serie de métricas necesarias para el análisis.

- Desarrollo: En esta fase se desarrollaron los tres prototipos de ZKP.

- Pruebas: Una vez terminada la fase de desarrollo, en esta etapa se han ejecutado todas las pruebas.

- Análisis: Con los datos obtenidos durante la ejecución de las pruebas se realiza los análisis pertinentes. Se realiza un análisis numérico atendiendo a las métricas escogidas. Se calculan medias y desviaciones típicas de los datos numéricos obtenidos. Y se realizan gráficas a partir de dichos datos. También se obtienen datos y gráficas relativas a la complejidad de los smart contracts proporcionados por la herramienta Solidity Metrics.

- Escritura de la memoria: En esta fase final del proyecto se ha escrito la memoria siguiendo las indicaciones del tutor y la normativa que señala la Universidad. Se ha

sintetizado los conceptos teóricos necesarios para este proyecto. Se ha desarrollado un estado del arte con los trabajos más relevantes en relación con nuestro trabajo de entre toda la literatura disponible. Y por ultimo se ha sintetizado el desarrollo de los prototipos y de los resultados obtenidos durante la etapa de análisis. Por ultimo se dan lineas a seguir en trabajos futuros así como también las conclusiones que se han extraído durante todo el proceso del proyecto.

A lo largo del proyecto se han tenido reuniones con el director, con una frecuencia de 2 semanas donde se supervisó el correcto desarrollo del proyecto, se resolvieron las dudas pertinentes y se ofreció feedback.

3.1.1. Proceso de desarrollo de los prototipos ZKP

Para el proceso de desarrollo de los prototipos se utilizó el siguiente código como referencia.

- Tutorial sobre la implementación de Zero Knowledge Proof para Identidades digitales en Solidity y Ethereum. [49] Se siguió el circuito aritmético descrito, el método para generar una prueba ZK-SNARK y el smart contract para verificar dicha prueba.
- Ejemplo proporcionado por Meta para la herramienta Winterfell. [23] Para la generación de pruebas STARK, se tomó como base este ejemplo, modificándolo según las condiciones necesarias.
- Template de snarkjs para generar un smart contract de Solidity para Plonk. [57]
- Smart Contract de Solidity que verifica firmas realizadas por ECDSA, o curva elíptica. [47]
- Guía de para desplegar smart contracts en EVM por medio scripts en Remix IDE [62]

A partir del ejemplo básico de ZK-SNARKs, se planteó la siguiente expresión para generar las pruebas:

$$a.a = b$$

Donde a y b son las entradas publicas de los circuitos aritméticos y se valida que el cuadrado de a sea igual al valor de b. De esta manera, un probador honesto puede demostrar la veracidad de la prueba ante un verificador proporcionando los valores de a y b que cumplan dicha condición. Esta expresión es la que se utilizará para la generación de las pruebas.

Dado que el objetivo principal de este proyecto es comparar distintos zkp, se ha decidido desarrollar tres prototipos distintos de zero-knowledge proofs. Tras analizar entre las distintas variantes de zero-knowledge-proofs, se optó por desarrollar 2 prototipos de zk-snark y un prototipo de zk-stark. Las razones de esta elección atienden a la mayor disponibilidad de recursos en Solidity y Ethereum para desarrollar zero-knowledge proofs de tipo snark, mientras que reservar un prototipo para ZK-STARK permite ampliar el análisis comparativo debido a las diferencias significativas entre ambos enfoques.
Se ha optado por Groth16 y por Plonk como los dos prototipos de ZK-Snark elegidos. Han sido seleccionados por la disponibilidad de herramientas que permiten su desarrollo, generación de pruebas y verificación de manera satisfactoria.

El primer prototipo desarrollado fue Groth16, siguiendo el ejemplo mencionado anteriormente. Tanto para el circuito como para el smart contract se siguió el tutorial. Es un ejemplo sencillo que permite una comprensión efectiva del de funcionamiento de zk-snark y de los procesos de generación y verificación de una prueba. Debido a que Zokrates exporta un smart contract generado automáticamente para verificar la prueba, entonces se crea un smart contract que interactúe con este. Esto es necesario porque todas las funciones en dicho smart contract son de vista interna, por lo que no es posible interactuar con el de manera directa, solo se puede interactuar a través de otro smart contract.

Para el segundo prototipo se modifico lo desarrollado en el anterior, pero atendiendo a las particularidades de Plonk y de la herramienta Circom. En el circuito aritmético se siguió implementando la expresión que se ha definido anteriormente. De igual manera que en el caso anterior Circom genera también automáticamente un smart contract para verificar las pruebas. Y como en el caso anterior, este contrato generado tiene también las funciones internas, por lo que no se puede interactuar con el directamente.

El ultimo prototipo desarrollado fue el prototipo STARK, y en este caso se siguió un ejemplo proporcionado por Meta. Se utilizo la librería Winterfell para crear tanto el prover como el verifier. La verificación debido a limitaciones se realizara en dos fases, primero off-chain y seguido se verificara en un smart contract.

Para desplegar un prototipo en IOTA sería necesario crear un contrato inteligente compatible con la red IOTA. Sin embargo, los contratos inteligentes de Solidity son compatibles en cualquier red [17]. Es decir, no serán necesarias ninguna modificación de los contratos desarrollados para que estos funcionen en IOTA. Por lo que se utilizarán los mismos contratos inteligentes en ambas redes.

El entorno que se ha utilizado, mayoritariamente, durante todo el desarrollo ha sido Remix IDE debido a su soporte para trabajar con Ethereum y con Solidity.

3.2. Esquema general

La idea principal sobre la que se basa el proyecto es la comparación de tres tipos distintos de prototipos ZKP para realizar un análisis. Estos prototipos se desplegaran en la blockchain de Ethereum y posteriormente en el Tangle de IOTA mediante la extensión de Metamask.

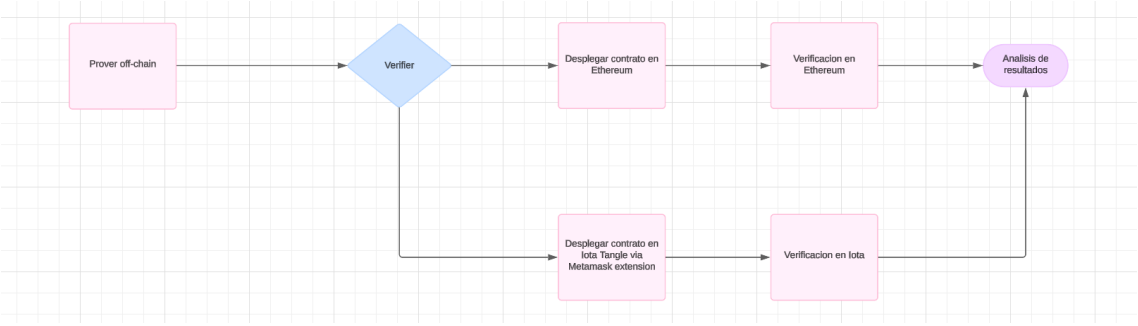

Figura 3.1: Esquema General

En la figura 3.1 se muestra el esquema general que va a seguir la ejecución de los prototipos desarrollados. Se tiene un prover que generará una prueba ZKP fuera de la cadena, esta se devuelve en un JSON y por pantalla. Posteriormente se decide si se utilizara en que red se desplegara el verificador, entre Ethereum e IOTA. Una vez desplegado, se interactúa con el contrato para verificar la prueba obtenida en el primer paso. Posteriormente se evalúan los resultados obtenidos durante todo el proceso. El usuario interactúa con el prover fuera de cadena, y con el smart contract a través de transacciones.

3.3. Ejecución de las pruebas

Para realizar la ejecución de las pruebas se plantea un escenario donde se ejecuten 20 ejecuciones de cada prototipo. Cada prototipo se compondría de un probador, utilizando la herramienta correspondiente para cada prototipo, y de un verificador que estaría implementado en un smart contract de Solidity.

El flujo de ejecución para generar una prueba SNARK con el plugin Zokrates se muestra en la imagen 3.2.Utilizando el plugin de Zokrates primeramente se compila un circuito, un fichero con extensión .zok. Esto genera un archivo de ABI y un archivo de código compilado del circuito. Después de la compilación, se calcula un testigo para el programa compilado al introducir valores de a y b. Posteriormente se realiza la configuración conocida como trusted setup, la cual crea una clave de comprobación y una clave de verificación. Ambas se generan en sendos ficheros JSON. Con estos pasos realizados, se procede a generar la prueba. Y por ultimo se genera el verifier en Solidity, con el que se interactuara a través de un smart contract.

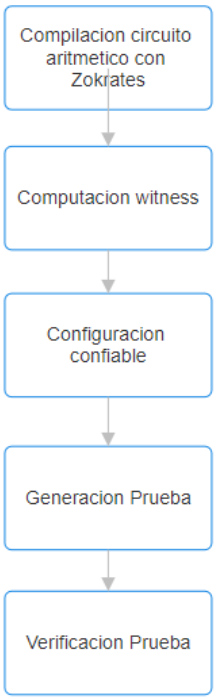

Figura 3.2: Flujo de ejecución de Zokrates para generar una prueba

De manera similar funciona Circom, que es la herramienta que se utiliza para generar las

pruebas Plonk. Ambos flujos de trabajo de Zokrates y de Circom se automatizarán para poder medir las métricas requeridas durante el proceso de generación de pruebas. Esta automatización se llevara a cabo mediante scripts en JavaScript. Debido a que Groth16 y Plonk son dos tipos de SNARK la manera de ejecutar las pruebas será la siguiente:

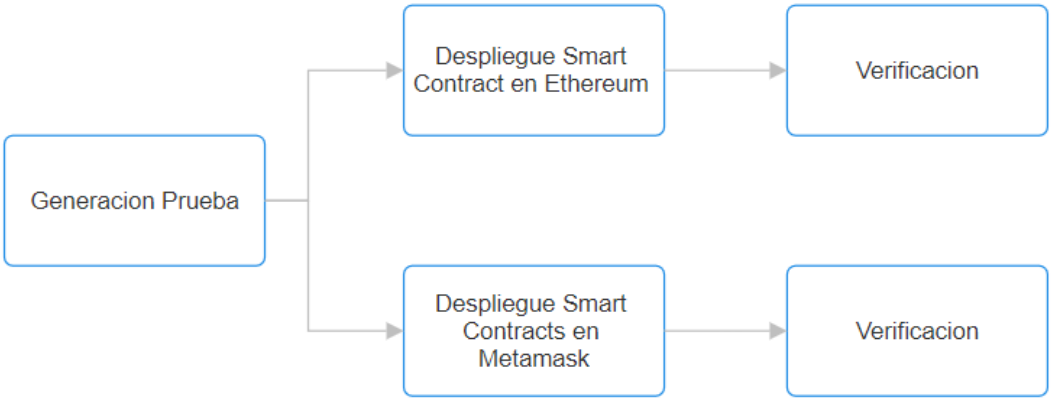

Figura 3.3: Flujo de ejecución de Circom para generar una prueba

Cada vez que se genera una prueba, posteriormente se desplegara en Ethereum los dos smart contract, para después interactuar con ellos y verificar la prueba. De manera similar se despliegan los smart contract en Metamask y se verifica la prueba. Este proceso se llevara a cabo durante 20 veces y se anotarán los resultados numéricos que procedan.
Se plantean dos escenarios distintos de pruebas con la intención de comparar los tres prototipos en Ethereum e IOTA. El escenario consistirá en un escenario donde las entradas serán constantes, este escenario se plantea para obtener mayor granularidad en los datos recolectados y observar que sucede con una prueba desplegada repetidas veces.

Capítulo 4

Desarrollo

A lo largo de este capitulo se van a explicar los tres prototipos de ZKP desarrollados. También se explicaran los circuitos que generan las pruebas y los smart contracts que verifican los ZKP. Un prototipo consta de un prover que genera la prueba y de un verifier implementado en un smart contract en Solidity. El código desarrollado se encuentra en dos repositorios de Github `https://github.com/SergioGarridoDeCastro/codigoZKP_TFM`, donde se encuentra el código relativo a los smart contracts con los verifiers, los script de generación de pruebas para Groth16 y Plonk, y los scripts de despliegue para los tres prototipos. En definitiva es todo el código con el que se ha trabajado en el entorno web de Remix IDE. El otro repositorio utilizado es `https://github.com/SergioGarridoDeCastro/codigoStark_TFM`. En este repositorio se almacena el codigo utilizado para generar y verificar off-chain pruebas Stark. Ambos repositorios tienen licencia MIT License, la cual permite a los usuarios hacer prácticamente cualquier cosa con el código (copiar, modificar, redistribuir), siempre y cuando se mantenga el aviso de copyright original y la licencia en cualquier copia o derivado.

4.1. Groth16

El primer prototipo de ZKP que se ha desarrollado ha sido un zk-snark de tipo Groth16. En la sección 2.4.3 se explica más a fondo el funcionamiento de este algoritmo. En la figura 4.1 se muestra un esquema de funcionamiento de como funciona el algoritmo Groth16.

Un prover recibe información publica proporcionada por un usuario, información privada y una clave privada generada en el proceso de configuración segura. Todos esos elementos los utiliza para generar una prueba segura. El Prover procesa la información privada junto con la pública y, mediante el uso de la clave privada, genera una prueba que luego es enviada al Verifier. Por su parte, el Verifier recibe también una clave generada durante el proceso de configuración segura, así como también recibe la misma información publica proporcionado por un usuario. La clave publica permite verificar la validez de la prueba recibida. El Verifier combinando la información publica y la clave publica puede determinar la validez de la afirmación realizada por el prover. Permite no revelar la información privada asociada a la prueba.

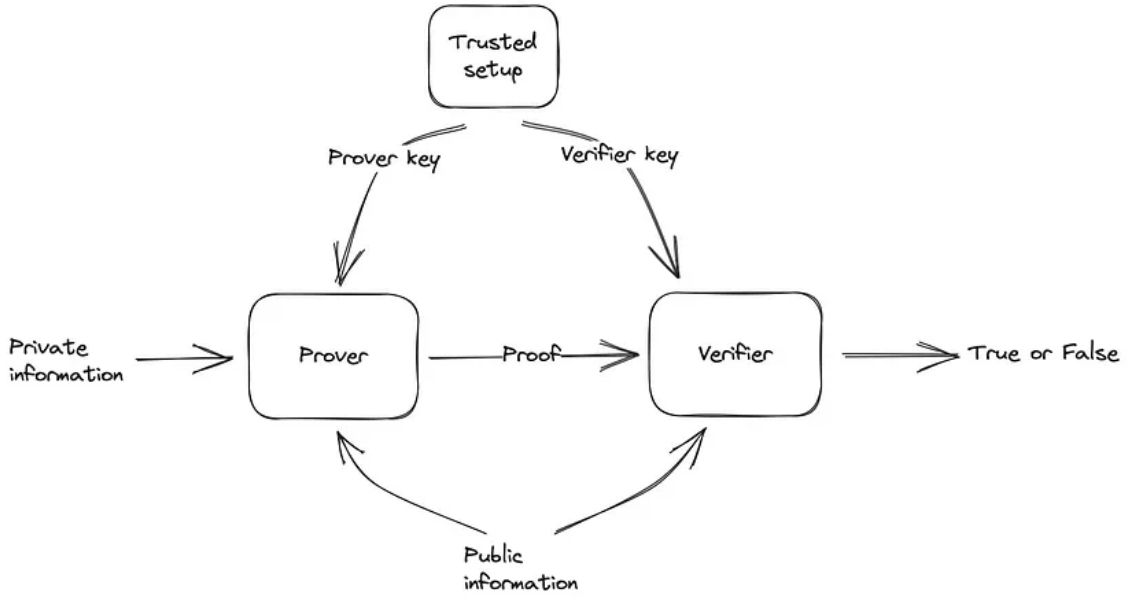

Figura 4.1: Esquema de funcionamiento de Groth16 [20]

4.1.1. Circuito aritmético

Para la creación del circuito aritmético se ha seguido la expresión anterior implementándolo en un fichero.zok.

```
def main(private field a, private field b){
    assert(a * a == b);
    return;
}
```

El circuito aritmético mostrado solo tiene una función main que es el punto de entrada principal del programa. Tomas dos argumento como entrada, a y b, ambos de tipo field. Un tipo field es el tipo de dato principal de Zokrates y representan un elemento de un campo finito, por lo que las operaciones se realizaran dentro de un campo finito. Se indica que las entradas son private, es decir, que estos valores son entradas privadas. Las entradas entonces no serán visibles para el verificador mientras que si serán visibles para el probador.

El assert presente verifica que la condición sea verdadera, es decir, en este caso verifica que $a * a == b$. Si esta condición falla no se ejecutará y fallará la ejecución, por lo que no se generará una prueba valida. En caso de que la condición se cumpla, entonces se generará correctamente una prueba snark. La ausencia de valor de retorno de debe a que en Zokrates una función no necesita devolver un valor explícito, debido a que lo importante es la generación de la prueba que demuestra que los cálculos se realizaron correctamente.

Suponiendo un ejemplo en el que el probador conoce $a = 3$, entonces entonces puede demostrar que $b = 9$ sin revelar el valor exacto de a La prueba snark que se genere será una prueba que puede ser verificada por un verificador sin que este conozca el valor exacto de a, pero sabrá que existe un a tal que $a.a = b$.

4.1.2. Smart Contract

El verificador se implementa dentro de un smart contract desarrollado en Solidity. Del mismo modo que en el circuito aritmético, se ha seguido como inspiración el ejemplo mencionado en el capitulo 3.

Al utilizar Zokrates, como se ha comentado anteriormente, a parte de la prueba también se genera un smart contract de manera automática que funciona como verificador. Por tanto el smart contract desarrollado se encargara de implementar las funcionalidades del verifier generado automáticamente. El smart contract que se ha desarrollado para interactuar con el anterior contrato es un smart contract sencillo. Se utiliza una licencia MIT, que indica que es de código abierto y puede ser utilizado libremente por cualquier usuario. Se indica que es compatible con la versión 0.8.0 y superiores de Solidity. Dentro de él se importa el verifier. Se define una estructura User, que representaría en el ejemplo a seguir la identidad digital que debe validar la prueba generada. En ese struct se incluyen datos de un usuario (email, pais y nombre). Se tienen dos mappings, el primero que asocia una dirección de Ethereum con un User, es decir, permite almacenar la información de un usuario asociada a su dirección. El otro mapping almacena un valor booleano para cada dirección, indicando si ha sido verificada o no. Hay dos funciones en este contrato, una publica que se llama verifyIdentity, y otra interna, verifyProof. Ambas funciones reciben como parámetros la prueba. La primera verifica la identidad del usuario mediante la prueba snark. Verifica que la dirección del remitente no haya sido verificada previamente. Llama a la función verifyProof y si la prueba es valida se marca la dirección como verificada. Esta función queda abierta a futuras mejoras que trabajen con la identidad digital. La función verifyProof solamente llama a la función verifyTx del contrato Verifier para verificarla. Devuelve si la prueba es válida o no. Se ha decidido separar ambas funciones para garantizar una modularidad mayor, con vistas a implementar en un futuro una gestión más completa de una identidad general.

El smart contract generado automáticamente por Zokrates implementa la verificación de la prueba generada también por Zokrates. Este contrato utiliza la librería Pairing, la cual permite realizar operaciones con curvas elípticas. Dentro del contrato verifier.sol, se define una función principal, llamada verifyTx, cuya entrada será una prueba zk-SNARK. Esta función realiza varias operaciones matemáticas utilizando las primitivas definidas en la librería Pairing, como la suma y multiplicación de puntos en las curvas elípticas, para verificar si la prueba es válida.

El proceso de verificación realizado por verifyTx conlleva inicialmente el procesamiento de puntos específicos en las curvas elípticas G1 y G2. Esto lo lleva a cabo las funciones P1 y P2. Utilizando las primitivas criptográficas, se realiza un emparejamiento bilineal, permite comprobar que la prueba es consistente con los datos públicos y las entradas privadas, sin ser reveladas estas ultimas. Si el emparejamiento es correcto, entonces la prueba es correcta. En caso contrario la prueba es incorrecta.

4.2. Plonk

El segundo prototipo de ZKP diseñado es de tipo Plonk. Plonk es un protocolo similar a Groth16 dado que ambos son protocolos para zk-snarks. En el caso de Plonk, el prover recibe también información publica proporcionada por un usuario junto a información privada.

Durante el proceso de configuración, se genera un par de claves públicas que utilizarán el Prover y el Verifier. El Prover utilizando la información pública junto a la información privada y a la clave pública es capaz de realizar una serie de cálculos criptográficos para generar una prueba. Dicha prueba se envía al verifier que utiliza la otra clave publica junto a la información publica para verificar la prueba.
En resumen, prover y verifier interactúan entre si de la misma manera que en el caso de Groth16. La diferencia entre ambos protocolos recae en el proceso de configuración.

4.2.1. Circuito aritmético

De manera similar al circuito del prototipo de Groth16 se ha adaptado la expresión $a.a = b$ como un circuito aritmético para Circom.

```
1   pragma circom 2.0.0;
2
3   template circuit() {
4       signal input a;
5       signal input b;
6       signal output out;
7
8       out <== a * a;
9       b === out;
10  }
11
12  component main = circuit();
```

En la primera linea establece la versión del lenguaje Circom que se utiliza, la versión 2.0.0. Se define una plantilla de circuito donde se definen dos señales de entradas, a y b, donde a será utilizada para realizar el cálculo y b será utilizada para validar el resultado obtenido en dicho calculo. Las entradas se han definido como signal input. Una señal en Circom es una variable interna que representa un valor en el circuito y es operada sobre los elementos del campo finito. Se define también una señal de salida, que será el valor generado a partir del calculo en el interior del circuito. La operación $out <== a * a$; es la que se realiza dentro del circuito, y el valor obtenido al calcular el cuadrado de la variable a se asigna a la señal de salida. La señal de salida se compara con la variable de entrada b. Si dicha igualdad se cumple, la ejecución fallara y no se generará correctamente la prueba snark. De igual modo, si la comparación funciona entonces se genera la prueba sin ningún tipo de problema.

Recuperando el ejemplo explicado en el circuito de Groth16, el probador en este caso puede demostrar que conoce el valor de a tal que $a.a = b$ sin necesidad de demostrar el valor de a. Mientras que el verificador puede demostrar que la prueba es valida sin conocer el valor de a.

4.2.2. Smart Contract

Como en el caso del prototipo de Groth16, en este caso el verificador también se implementara en un smart contract desarrollado en Solidity. Se sigue la misma idea de implementar un smart contract que interactúe con un smart contract generado, en este caso por Circom.

El smart contract es muy similar al de Groth16, siendo un contrato sencillo. Define una estructura User, también para indicar una identidad digital, donde se incluyen nuevamente

los datos del usuario (nombre, país y email). De igual manera se encuentran los dos mappings presentes en el prototipo de Groth16. Un mapping permite almacenar la información de un usuario asociada a su dirección, mientras que el segundo almacena un valor booleano para cada dirección, indicando si ha sido verificada o no. El contrato VerificationPlonk interactúa con un contrato verificador (PlonkVerifier) generado por Circom. La función verifyIdentity, como se ha comentado, esta diseñada para verificar la identidad digital. Se comprueba que el usuario no haya sido verificado previamente. Posteriormente, se llama a verifyProof, función que permite la interacción con el contrato generado por Circom. Esta función toma como parámetros la prueba y las señales publicas devueltas por Circom. Si la verificación es exitosa, se emite un evento para registrar el resultado de la verificación y se marca como validada la dirección del usuario.

El smart contract PlonkVerifier realiza la verificación de pruebas zk-SNARK utilizando PLONK y ha sido generado automáticamente con Circom. la función verifyProof toma como entradas proof, una matriz de 24 elementos que representa la prueba snark, y pubSignals, una matriz de 1 elemento que representa las señales públicas asociadas a la prueba. Realiza una serie de cálculos en ensamblado utilizando el algoritmo euclidiano extendido.[1] Como resultado de estos cálculos se devuelve un valor booleano, que indicara si la prueba es valida o no.

4.3. STARK

En la etapa de desarrollo de ZK-STARK se ha utilizado primordialmente la herramienta Winterfell para generar las pruebas. Para desarrollar el prover se ha partido del ejemplo básico proporcionado por Meta para la herramienta Winterfell. Winterfell es una librería que permite implementar sistemas de pruebas STARK.

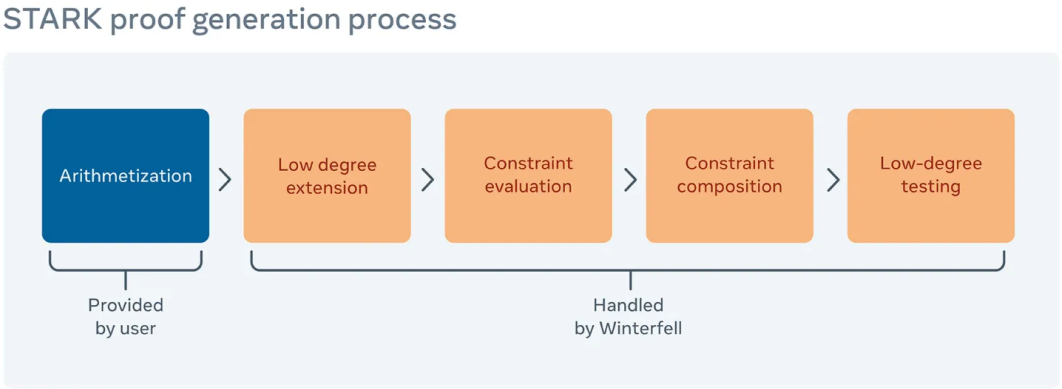

Figura 4.2: Proceso de generación de pruebas STARK

En la figura 4.2, se muestra el proceso de generación de pruebas STARK por parte de Winterfell. En este proceso el usuario solo se encarga de proporcionar la aritmetización, mientras que el resto de fases las realiza Winterfell. La aritmetización es el primer paso, y es el que provee el usuario, es un proceso en el que se convierte el problema a probar en

[1] Para mas información `https://esfm.egormaximenko.com/linalg/gcd_extended_es.pdf`

un conjunto de ecuaciones algebraicas. Este conjunto de ecuaciones algebraicas definen las restricciones del calculo, las cuales debe respetar el prover durante las siguientes etapas del proceso.

Una vez el usuario proporciona la aritmetización, Winterfell continua con el proceso. Primero realiza una extensión de bajo grado, esto quiere decir que las ecuaciones definidas se expanden a un polinomio de grado mayor. Esta operación implica que las propiedades matemáticas requeridas para la prueba sean cumplidas. Posteriormente se evalúan las restricciones necesarias para que la prueba sea valida. Tras analizar dichas restricciones, Winterfell combina todas las restricciones en una única prueba. Por ultimo, se comprueba que el polinomio resultante cumple las restricciones definidas en la aritmetización, asegurando que la prueba sea válida y eficiente computacionalmente.
En resumen, Winterfell facilita el proceso de generación de pruebas de manera automatizada, permitiendo al usuario concentrarse en la definición del problema, sin tener que implementar los detalles complejos de la prueba criptográfica.

4.3.1. Prover

Como se ha mencionado anteriormente, se comienza siguiendo el tutorial para definir un AIR para un cálculo sencillo. En el ejemplo original la operación que se realiza es a^3+42, pero como se ha comentado en el capitulo 3 la operación a implementar es $a^2 = b$.

Para definir un Algebraic Intermediate Representation (AIR) y su traza que cumpla la condición $a^2 = b$ primeramente se debe definir la relación de la transición. Se define una relación simple en la que en cada paso de la traza se verifica que la relación $a_i.a_i = b_i$ se mantiene. La traza tendrá dos estados:

- Estado 0: Corresponde al valor de a en cada paso de la traza.
- Estado 1: Corresponde al valor de b en cada paso de la traza. Siguiendo la relación sería el resultado de a_i^2

Se define un polinomio de restricción, que asegura si las condiciones definidas en la relación se cumplen a lo largo de la traza. Para que una prueba STARK sea valida, el polinomio debe igualarse a cero en cada etapa de la traza. Por tanto, en este caso el polinomio quedaría definido tal que

$$P(a_i, b_i) = a_i.a_i - b_i = 0$$

.

La traza es una secuencia de valores o estados de varios pasos, normalmente potencia de 2, y como mínimo 8 pasos. Sabiendo que los estados de la traza son los mencionados anteriormente, la traza sería la siguiente si $a = 3$:

$$[a_0, a_1, ..., a_n] = [3, 3, ..., 3]$$

$$[b_0, b_1, ..., b_n] = [9, 9, ..., 9]$$

De esta manera se cumpliría la relación propuesta. Una vez definida la relación, el polinomio de restricción y la traza, quedaría implementarlo sobre el ejemplo proporcionado por Winterfell.

En un inicio, antes de establecer la traza tal y como está definida ahora mismo se planteó lo siguiente:

```
    pub fn build_quadratic_trace(a: BaseElement, b: BaseElement, n:
        usize) -> TraceTable<BaseElement> {
    let trace_width = 2;
    let mut trace = TraceTable::new(trace_width, n);
    trace.fill(
        |state| {
            state[0] = a; // El primer estado es 'a'
            state[1] = b;     // b = a * a
        },
        |_, state| {
            state[0] = state[1];                // a toma el valor
                anterior de b
            state[1] = state[0]*state[0];  // b se actualiza a a^2
        },
    );
    trace
}
```

Este planteamiento dio errores significativos a la hora de generar la prueba. La traza se comportaba bien en los primeros pasos, pero en el séptimo paso comenzó a generar valores incorrecto en el calculo del polinomio de restricción, distintos de cero. Estos resultados extraños sugerían que se estaba produciendo un desbordamiento de entero. El desbordamiento ocurre cuando los valores exceden el rango máximo representable por un tipo de dato. En este caso se usaban enteros de tipo u128, cuyo limite es amplio pero se alcanza fácilmente con cálculos exponenciales como ocurrían en este caso. Por tanto se llego a la conclusión de que era un planteamiento erróneo y se modifico la traza de tal manera que respetara la restricción definida. La traza se modifico de la siguiente manera:

```
    pub fn build_quadratic_trace(a: BaseElement, b: BaseElement, n:
        usize) -> TraceTable<BaseElement> {
    let trace_width = 2;
    let mut trace = TraceTable::new(trace_width, n);
    trace.fill(
        |state| {
            state[0] = a; // El primer estado es 'a'
            state[1] = state[0]*state[0];     // b = a * a
        },
        |_, state| {
            state[0] = state[0];                // a se mantiene
                constante
            state[1] = state[0]*state[0];  // b = a^2. Si a no se
                cambia se mantiene constante
        },
    );
    trace
}
```

Este nuevo planteamiento si bien conceptualmente es correcto también produce errores. En este caso el error producido era *assertion 'left == right'failed: transition constraint degrees didn't match expected: [7] actual: [0]*. Este error quiere decir que Winterfell espera que la restricción de transición tenga grado 7, pero la verificación muestra que el grado calculado es 0. Un grado 0 en una restricción significa que la transición es trivial. En este caso se

considera la transición como trivial porque a y por tanto b permanecen constantes a lo largo de todos los pasos de la traza. La razón Winterfell espera un polinomio de grado 7 no está vinculada al calculo cuadrático que se realiza, si no que está vinculada al tamaño de la traza. Winterfell construye polinomios que interpolan los valores en la traza. Para una traza de tamaño 8, como es en este caso, los polinomios involucrados en la validación de la traza tienen un grado máximo 7. Debido a que la traza tiene tamaño 8, se espera un ciclo de 7 pasos que deba ser validado por un polinomio de grado 7.

Por tanto para evitar que el polinomio generado no sea trivial se plantea modificar la traza de tal manera que al siguiente valor de a en el ciclo se le sume 1, quedando tal que así:

```
    pub fn build_quadratic_trace(a: BaseElement, b: BaseElement, n:
        usize) -> TraceTable<BaseElement> {
    let trace_width = 2;
    let mut trace = TraceTable::new(trace_width, n);
    trace.fill(
        |state| {
            state[0] = a; // El primer estado es 'a'
            state[1] = state[0]*state[0];     // b = a * a
        },
        |_, state| {
            state[0] = state[0] + BaseElement::new(1);                //
                se suma 1 al valor anteior de a
            state[1] = state[0]*state[0];  // b = a^2. Si a no se
                cambia se mantiene constante
        },
    );
    trace
}
```

Con esta modificación se consigue generar una prueba correctamente. Conceptualmente es correcta pues la traza modificada sigue cumpliendo la restricción $a.a = b$ en cada paso de la misma. La relación cuadrática se mantiene al recalcular en cada paso b como a^2. La traza resultante sería:

$$[a_0, a_1, ..., a_n] = [2, 3, 4, ..., a_n]$$

$$[b_0, b_1, ..., b_n] = [2, 4, 9, 16, ..., a_n.a_n]$$

Teniendo en cuenta la traza modificada, posteriormente se ha definido el AIR para esta prueba atendiendo el ejemplo de Winterfell para saber como desarrollar un AIR así como la restricción definida. La restricción que evalúa cada transición de la traza sería la siguiente:

```
    fn evaluate_transition<E: FieldElement + From<Self::BaseField>>(
        &self,
        frame: &EvaluationFrame<E>,
        _periodic_values: &[E],
        result: &mut [E],
    ) {
        let current_a = frame.current()[0];
        let current_b = frame.current()[1];

        result[0] = current_a * current_a - current_b;  // Verifica que
            a^2 = b
```

```
11      }
```

Se comprueba posteriormente que el primer estado, state[0], de la traza sea igual a 0 al inicio del ciclo, y que el ultimo estado, state[1], en la ultima etapa del ciclo sea igual al valor self.result. Esto quiere decir que se comprueba que b tenga el valor esperado, $a_n.a_n$.

4.3.2. Verifier

En Solidity apenas hay existencia de librerías que estén altamente aceptadas que den soporte a Stark. En Rust y C++ si que existen librerías que permiten la integración de Stark como son *plonky3* [2] y *libSTARK* [3]. Solidity también presenta limitaciones a la hora de realizar operaciones criptográficas complejas y avanzadas como las utilizadas para verificar pruebas stark, lo cual incrementa la complejidad y añade complicaciones como un gasto excesivo de gas. Esto deja opciones limitadas para la verificación de una prueba STARK mediante un smart-contract de Solidity. Entre las distintas opciones posibles se consideran las siguientes opciones:

- Precompilado de verificación: Esta opción requería la implementación de la verificación mediante una biblioteca nativa como las mencionadas en Rust o C++ y llamar a esta biblioteca desde el contrato inteligente utilizando una precompilación, via una llamada externa.

- Implementación directa: Esta opción requeriría implementar las operaciones necesarias para verificar la prueba en Solidity. Entra las operaciones a implementar se puede incluir implementar operaciones de campo finito, hashing (como Blake3) y Merkle tree.

- Verificación off-chain: Esta opción permitiría verificar la prueba off-chain (fuera de la cadena) en un aplicación cliente. Una vez que la verificación es exitosa off-chain, se firmará el resultado utilizando una clave privada, como afirmación de que la prueba es válida. Se enviaría el resultado de la verificación, junto con la firma digital, a un contrato inteligente en la blockchain. Este contrato verifica la firma digital utilizando la clave pública asociada para asegurarse de que el resultado de la verificación no es manipulado.

- Verificación en Starknet y Cairo: Esta solución requeriría utilizar Starknet. Implicaría el desarrollo de un programa en Cairo que tomará los valores generados de la prueba y verificará la prueba STARK tras desplegarlo en StarkNet. El despliegue en StarkNet se podría realizar utilizando herramientas como StarkNet CLI, Hardhat, StarkNet.js o el plugin que incluye Remix para trabajar con StarkNet. El contrato de StarkNet se podría comunicar con el contrato de Solidity a partir una funcionalidad de comunicación entre Ethereum (L1) y StarkNet (L2) que permite enviar mensajes y datos entre contratos desplegados en ambas redes.

- Conversión de las pruebas stark a snark: Esta opción no es practica debida a las diferencias conceptuales entre ambos tipos de zkp.

Una vez expuestas las distintas alternativas se ha decidido realizar la verificación off-chain, con la herramienta Winterfell. Winterfell ofrece funciones que permite tanto generar

[2] `https://github.com/Plonky3/Plonky3`
[3] `https://github.com/elibensasson/libSTARK`

pruebas como verificarlas. Las razones de esta elección atienden primero a la falta de ejemplos desarrollados existentes para implementar un smart contract de Cairo que verifique una prueba STARK. Una implementación directa en Solidity implica implementar cálculos criptográficos complejos que Solidity no es capaz de soportar. El otro motivo de esta elección es la existencia de un verifier incorporado en la herramienta y librería Winterfell. Al aprovechar la herramienta Winterfell se permite una verificación más sencilla de las pruebas.

Se plantea un escenario donde se ha generado una prueba con Winterfell y se verifica en un entorno controlado. Una vez que la verificación es exitosa, el verifier firmará un hash de la prueba utilizando una clave privada. Estando el hash de la prueba como la firma se envían al contrato inteligente en la blockchain. Dicho contrato se encargará de verificar la autenticidad de la firma utilizando la clave pública del verifier.

El primer paso en este escenario, evidentemente, es verificar la prueba con Winterfell. Una vez verificada la prueba se procede a calcular un hash criptográfico de la misma, utilizando el algoritmo BLAKE3. Dicho hash actúa como una huella digital de la prueba, representando de manera única su contenido. La utilización de blake3 puede garantizar una computación rápida del hash. Se garantiza seguridad y resistencia a colisiones, algo fundamental para garantizar la integridad de la prueba. Para asegurar la autenticidad del hash, este se firma con el algoritmo ECDSA (Elliptic Curve Digital Signature Algorithm). La biblioteca k256 maneja operaciones criptográficas asociadas a curvas elípticas, por lo que es la mejor opción para trabajar con dicho algoritmo.

Posteriormente se ha generado un par de claves publica y privada. La clave privada (secret_key) se genera a partir de una secuencia de 32 bytes. Esta clave es la que se utiliza en el proceso de firma del hash de la prueba. Es especialmente vulnerable, porque si otro usuario tuviera acceso a la clave podría firmar pruebas de manera fraudulenta. La otra clave generada es la clave pública (public_key), que deriva de la clave privada. Esta clave se pasara al smart contract y se utilizará en el proceso de verificación. Con la clave privada ya generada, se procede a firmar el hash de la prueba. La firma digital es una prueba criptográfica de que el hash, y por ende la prueba STARK, ha sido generada y verificada por un usuario o entidad en posesión de la clave privada. Cualquier usuario o entidad en posesión de la clave pública puede comprobar que la prueba es valida.

La firma resultante es una secuencia de 65 bytes. En la secuencia se incluyen los parámetros r, s, y v. Estos parámetros son críticos para la verificación en un smart contract de Solidity. Los valores de r y s son cada uno de 32 bytes mientras que el valor de v es el ultimo byte de la secuencia. Dichos valores se definen como:

- El valor r representa la coordenada x de la curva elíptica. Tiene importancia a la hora de verificar la curva, pues dicho valor se deriva de la curva de la curva elíptica y la clave privada y la clave privada, siendo así un valor único para cada firma.

- El valor s es el segundo valor de la curva elíptica. Se calcula a partir del hash, la clave privada y el valor r. Junto a r verifica la integridad de la firma, así como su autenticidad.

- El parámetro v es un valor adicional, conocido como identificador de recuperación, y como se ha mencionado es el ultimo byte de la firma. Solo puede tomar como valor

27 (0x1b) o 28 (0x1c), . A partir de r y s se pueden calcular diferentes puntos de la curva. La función de este ultimo byte permite indicar cual de los puntos calculados, a partir de r y s, se va a utilizar.

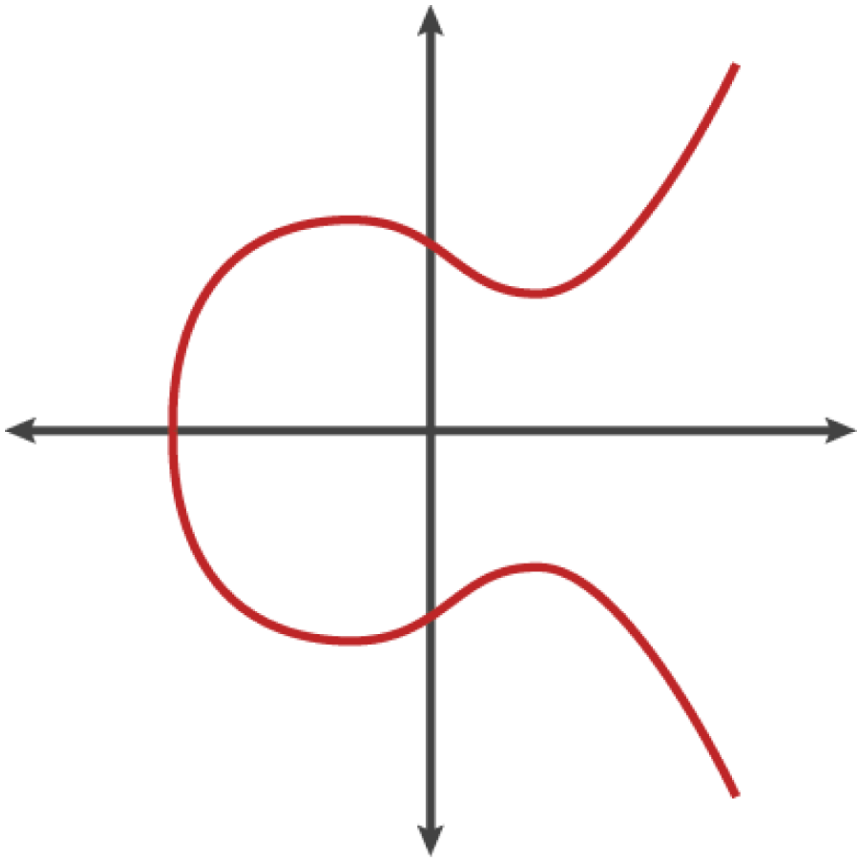

Figura 4.3: Curva Elíptica

Posteriormente, para asegurar que la firma generada sea interpretable con Ethereum se genera otro hash con el algoritmo Keccak256. El algoritmo Keccak, de la familia SHA3, es un algoritmo criptográfico ampliamente utilizado en Ethereum. Adicionalmente, se agrega un prefijo específico para Ethereum `"\x19Ethereum Signed Message:\n32"` al hash calculado con Keccak256.

Este es un enfoque seguro siempre que la clave privada del verifier se mantenga segura, la verificación offchain se realice en un entorno controlado y el contrato inteligente no presente vulnerabilidades y sea auditado. Permite de manera sencilla la verificación en Ethereum al asegurar que solo las pruebas válidas son aceptadas y confirmadas en el smart contract.

Antes de explicar el smart contract es relevante explicar los conflictos y problemas enfrentados a la hora de desarrollar el verificador off-chain. Uno de los primeros conflictos surgió cuando la firma generada por k256::ecdsa::Signature producía tan solo 64 bytes, mientras que se esperan 65 bytes por Ethereum. Es decir, no se estaba obteniendo de manera correcta el byte v. Para solucionarlo, se optó por trabajar con k256::ecdsa::recoverable::Signature, que utiliza firmas recuperables. Gracias a ello se puede obtener los valores de r y v de manera automática. Debido a la falta de métodos para recuperar el byte v, se optó por calcularlo

manualmente. Para ello se comparó la clave pública original con la recuperada para decidir el valor de v (27 o 28).

El segundo conflicto importante se debió a los distintos algoritmos de hashing utilizados. Primeramente se calcularon hashes con Blake3 y Keccak256 para la compatibilidad de los mismos con Ethereum. La integración de estos dos algoritmos y su aplicación en las firmas generó confusión respecto a qué hash debía firmarse y cómo este debía procesarse posteriormente. Se optó por diferenciar claramente los hashes producidos por Blake3 y los producidos por Keccak256. Esto se realizo marcando a estos últimos, tanto en los nombres de las variables como en la representación por pantalla, como Ethereum. De este modo quedaba delimitado el uso de Blake3 para uso interno y el uso de Keccak para la firma ECDSA, permitiendo así compatibilidad con Ethereum y mayor claridad.

El smart contract desarrollado para realizar la verificación, debido a las limitaciones expuestas de Solidity, tiene como objetivo verificar la validez de la firma ECDSA. Como se ha visto, se ha realizado la verificación de la prueba offchain y se han generado un hash para la prueba que se ha firmado utilizando ECDSA. Es crucial en escenarios donde se requiere confianza en pruebas generadas y verificadas fuera de la cadena, pero cuya validez debe ser asegurada dentro de un entorno blockchain. El principal objetivo de este smart contract es verificar que una firma ECDSA corresponde al hash de la prueba generada por el prover. Tambien debe verificar que dicha firma fue realiza por la clave privada asociada a la clave publica conocida. En dicho contrato se utilizará una libreria que proporciona funciones para la verificacion de firmas ECDSA.

Al desplegar el contrato, se debe proporcionar el hash de la prueba (expectedProofHash) y la clave pública (publicKey) que se ha usado para firmar las pruebas. La verificación se realiza a través de la función verifySignature. Se definen una serie de constantes que son fundamentales para la verificación de una firma ECDSA. Primeramente se verifica el hash de la prueba, para ello se compara el hash proporcionado en el despliegue del contrato con otro hash pasado como parámetro de la transacción. Si ambos no coinciden, la verificación falla, emitiendo un mensaje de depuración indicando la discrepancia entre los dos hashes. Dicha transacción se revertirá y se volverá al estado inicial. Posteriormente se compara la clave publica almacenada en el contrato con la clave publica proporcionada en la transacción. De igual manera, si ambas no coinciden entonces la verificación fallará, se emitirá un mensaje de depuración y se revertirá la transacción. La función convierte el hash proporcionado en un formato compatible con Ethereum. Posteriormente, se llama a la función verify, donde se interactúa con la librería EllipticCurve. En esta función se divide la firma en los valores r, s y v mencionados, que son los valores necesarios para la verificación. Obtenidos dichos valores se calcula el inverso el inverso modular de s con respecto a la constante N. La verificación de firmas ECDSA depende de la multiplicación de valores con el inverso de s. Se calculan los valores de $u1$, que es la multiplicación del hash por inverso modular calculado anteriormente y modulo N, y de $u2$, que es la multiplicación de r por el inverso modular y modulo N. Si los valores de $u1$ y $u2$ son son distintos de cero, se considerará la firma ECDSA como valida. Es una simplificación de una verificación completa, pudiéndose pulir en futuro.

4.4. Scripts de ejecuccion de las pruebas

Para la ejecución de las pruebas se han desarrollado una serie de scripts con la intención de automatizar los procesos de generación y verificación de las pruebas. En un script se puede medir de manera más sencilla el tiempo transcurridos en ms tanto en la generación como en la verificación de una prueba. También se puede medir medir más fácilmente con un script el tamaño de la prueba en bytes.

4.4.1. Prototipo Plonk

Para la ejecución del prototipo Plonk se tiene un script que se encarga de generar la prueba y el smart contract con el verifier de manera automática.

Para permitir que los datos de entrada a y b sean proporcionados por el usuario en tiempo de ejecución se ha considerado utilizar el modulo readline de Node.js. Dicho modulo permite interactuar con el usuario mediante consola, solicitando datos y capturando la entrada.

```
1  const readline = require('readline');
2
3  const rl = readline.createInterface({
4    input: process.stdin,
5    output: process.stdout
6  });
7
8  const input = await new Promise((resolve) => {
9    rl.question('Enter value for a: ', (a) => {
10     rl.question('Enter value for b: ', (b) => {
11       resolve({ a: Number(a), b: Number(b) });
12       rl.close(); // Se cierra la interfaz
13     });
14   });
15 });
```

Sin embargo este modo da un error *Cannot read properties of undefined (reading 'createInterface')*. Este error se debe a que se esta utilizando Remix IDE, por lo tanto la operación readline no funcionará correctamente. Esto es debido a que es específica para Node.js en entornos de línea de comandos. Por lo tanto se busca una alternativa, que es utilizar la instrucción prompt().

```
1  const a = Number(prompt('Enter value for a: '));
2  const b = Number(prompt('Enter value for b: '));
3  const input = { a, b };
```

Esta alternativa tampoco funciona correctamente por la misma razón, al utilizar Remix IDE en vez de Node.js el funcionamiento de ambos métodos es erróneo. Es por ello que se toma la decisión de establecer los valores de a y b de manera manual dentro del script, lo cual implicaría modificar el script entre ejecuciones de las pruebas para cambiar los valores de entrada.

Para calcular el tiempo en el que tarda en generarse la prueba se han utilizado utilizado la siguiente función:

```
1  const start = performance.now(); // Inicio de la medicion
```

```
2
3
4       const end = performance.now(); // Fin de la medicion
5       const timeTaken = end - start; // Tiempo transcurrido en
            milisegundos
```

Estas expresiones se situarán entre las funciones encargadas de generar la prueba, y devuelven el tiempo transcurrido en milisegundos para la generación de la prueba.

Se ha importado la librería ethers, que permite interactuar con la blockchain de Ethereum. Esta librería se utiliza esencialmente para manipular y formatear los datos de la prueba en formato hexadecimal. Se ha definido un objeto logger para registrar mensajes en la consola. Dicho objeto tiene tres métodos distintos (info, debug y error), permitiendo mostrar mensajes de log, depuración y error. La lógica principal del script se halla dentro de una función asíncrona que utiliza un bloque try catch para manejar errores. Como se ha comentado anteriormente, los datos de entrada se definen de manera manual. Dichos valores de entrada se utilizaran para generar la prueba. El circuito PLONK se compila utilizando la API de Remix a traves de una llamada remix.call. Del mismo modo se leen los ficheros específicos para el proceso de generación, entre los que se encuentran un fichero .wasm, que contiene la lógica del circuito, y zkey_final, que contiene la clave de verificación final en formato JSON. Se crea un testigo utilizando los valores de entrada a y b y el fichero .wasm. El testigo es un conjunto de datos intermedios utilizado durante el proceso de generación. Con dicho testigo y la clave de verificación se genera la prueba zk-SNARK y sus señales públicas gracias la función snarkjs.plonk.prove(), que devuelve un objeto proof que se mostrará por pantalla. Se utiliza una clave, verification_key.json, para verificar que es una prueba valida.

Por último se genera un smart contract en Solidity que se encargará de verificar la prueba zk-SNARK en Ethereum. Se crea a partir de la plantilla plonk_verifier.sol.ejs. La prueba y sus entradas se almacenan en un fichero inputs.json que se utilizara después en el siguiente script.

Existe un segundo script encargado de automatizar el despliegue del smart contract encargado de verificar las pruebas generadas por Plonk. El script realiza el despliegue y la interacción con contratos inteligentes utilizando la biblioteca Web3.js. Comienza cargando los metadatos en JSON y datos de prueba necesarios para interactuar con los contratos inteligentes, así como la prueba que se utilizará en la verificación. El script verifica la disponibilidad de Web3 en el entorno web. Si está disponible, entonces se procede a crear una instancia de Web3 utilizando el proveedor. Esta instancia bien puede ser Metamask para desplegar los contratos en IOTA o bien la EVM presente en Remix IDE. En el caso de que el proveedor de Web3 es MetaMask, el script solicita acceso a las cuentas del usuario. Una vez obtenidas, el script obtiene las cuentas disponibles llamando web3.eth.getAccounts(). La primera cuenta obtenida será utilizada para desplegar el contrato inteligente. El script interactúa con la API de Remix para leer los archivos de contratos inteligentes escritos en Solidity. Una vez leídos los contratos, estos se compilan de manera secuencial utilizanado la API de Remix y se extrae el ABI (Application Binary Interface) y el bytecode del contrato. Se despliega el contrato con el bytecode compilado y los argumentos del constructor como parámetros. Finalmente, se llama a la función verifySignature y se muestra por pantalla el tiempo que tarda en ejecutarse la transacción y el gas utilizado en dicha transacción

4.4.2. Prototipo Groth16

Para el prototipo Groth16 se ha desarrollado un script en JavaScript que automatiza la generación de una prueba con Zokrates. Este script replica el proceso de generación de una prueba que realiza Zokrates. Para ello se utiliza la librería de Zokrates, la cual se carga desde un CDN (Content Delivery Network). Cuando la librería se carga (script.onload), se inicializa ZoKrates llamando a zokrates.initialize(), lo que devuelve un zokratesProvider para interactuar con la API de ZoKrates. Se define el circuito que se va a utilizar, es decir, el fichero .zok que se ha definido en 4.1.1. Se escribe directamente en el código porque se producían errores el entorno web de Remix IDE a la hora de leer un fichero. El circuito se compila y se devuelven los denominados artifacts, que incluyen el programa y otros datos relevantes en el proceso. Una vez obtenidos estos ficheros, se generan los testigos o witness y se introducen manualmente los valores para a y para b. Estos valores se deberán cambiar si se quiere generar una prueba con unos valores de entrada distintos. Posteriormente se ejecuta el proceso de setup, del que se obtienen un par de claves, clave publica y clave de verificación, necesarias para la generación de la prueba. Se genera la prueba, la cual se muestra por pantalla con su tamaño en bytes indicado, y se exporta el contrato inteligente generado automáticamente en un fichero .sol, es decir, un fichero Solidity. Dicho contrato, como se ha comprobado anteriormente, contiene el verificador.

El script de despliegue es muy similar al caso anterior, solamente cambiando los ficheros que se importan, la llamada a la función que verifica y los parámetros que se pasa a dicha función.

4.4.3. Prototipo STARK

Solamente se ha desarrollado un script de despliegue muy similar al de Plonk y Groth16. No se ha automatizado la generación de pruebas debido a que esta, junto a la primera etapa de verificación, se genera off-chain y en un entorno controlado. Es por ello que no se integra en el proyecto en el entorno web de Remix IDE. Para que el script de despliegue funcione satisfactoriamente se decide introducir de manera manual los valores obtenidos durante la generación de la prueba. Para verificar otras pruebas distintas sirve con modificar los valores de las variables calculatedProofHash, signature y publicKeyHash por sus valores correspondientes.

Capítulo 5

Métricas

En este capitulo se detallan las métricas que se utilizarán para el análisis comparativos de los tres prototipos. Para realizar el análisis comparativo entre los distintos tipos de ZKP. Tras investigar entre la literatura existente en torno a trabajos similares de comparativas entre distintos tipos de ZKP. También se ha tenido en cuenta literatura existente sobre métricas de análisis de software y en concreto de Solidity.

El análisis comparativo se enfocará en la evaluación de eficiencia, escalabilidad y complejidad de los tres prototipos implementados, con el objetivo de determinar cuál es el más viable. Es por ello que se han determinado las siguientes métricas.

5.1. Gas consumido

La primera métrica seleccionada es el gas consumido en cada prototipo. El gas en Ethereum es la unidad de medida que mide el esfuerzo computacional necesario para ejecutar operaciones en Ethereum [22]. El consumo de gas hace relación al coste económico que conlleva la ejecución de un contrato inteligente en la red Ethereum. Por esta métrica es esencial en un análisis de eficiencia. Para comparar los distintos prototipos se registra el consumo en la etapa de verificación de la prueba. Esto se debe a que las pruebas se generan fuera de la cadena, y solamente se verifican mediante un smart contract implementado en Solidity y desplegado en Ethereum. Esta forma de medición puede ayudar a identificar qué prototipo es más económico a la hora de verificar una prueba y cuál requiere mayor coste económico.

5.2. Tamaño en bytes

La segunda métrica considerada en el tamaño en bytes de la prueba generada por cada prototipo. El tamaño de una prueba puede afectar notoriamente al almacenamiento en un nodo de la blockchain, así como a la comunicación entre distintos nodos. La medición se realizará durante la etapa de generación de pruebas. Si una prueba tiene un menor tamaño, ésta permitirá una mayor escalabilidad y menores costes a la hora de verificarla en la red. Por el contrario si una prueba tiene un tamaño elevado, ésta disminuirá su escalabilidad y aumentará su coste a la hora de verificarla.

5.3. Tiempo de ejecución

La tercera métrica elegida es el tiempo de ejecución del prototipo, que será medido en milisegundos. El tiempo de ejecución se puede dividir entre el tiempo que tarda el prover en generar la prueba y el tiempo que tarda el verifier en verificar la prueba. Es por ellos que se puede deducir que:

$$tiempoEjecucion = tiempoGeneracin + tiempoVerificacin$$

El tiempo de generación de la prueba afecta directamente a la latencia del sistema. Este aspecto es crítico en aplicaciones donde se requiere una alta velocidad de procesamiento, o en aplicaciones que utilicen un gran volumen de datos. Por otra parte, el tiempo de verificación es importante, pues una verificación rápida es vital en entornos descentralizados. En los entornos centralizados se busca que la computación eficiente se equilibre con la seguridad y la integridad de la red blockchain.

Cabe destacar que todas las mediciones relativas a las anteriores métricas se realizarán 20 veces, con el objetivo de obtener diferentes condiciones de carga de la red Ethereum. De este modo se obtiene un análisis más completo y preciso. Se calcularán las medias aritméticas y desviaciones típicas de cada dato, así como se aportarán gráficas que apoyen visualmente el análisis.

5.4. Métricas relativas a complejidad

Por último, se pretende medir la complejidad del código de los smart contracts de Solidity. Para ello se utilizaran las métricas proporcionadas por la herramienta Solidity Metrics, entre las que se destacan las siguientes:

- Source Lines (Lines, nLines y nSLOC): Esta métrica se refiere al tamaño del código dividido y atendiendo a tres variantes interrelacionadas entre sí. La variable lines se refiere al número total de líneas en el código, incluyendo comentarios, espacios en blanco y código ejecutable. La variable nLines hace referencia al numero de líneas de código, pero en este caso excluyendo comentarios y espacios en blanco. Y por último, la variable nSLOC indica las líneas de código significativas, es decir, líneas que contienen instrucciones ejecutables. Un número alto de líneas de código implica un código más extenso, y probablemente más complejo y difícil de mantener.
- Complex. Score: Esta métrica se refiere a la complejidad ciclomática. La complejidad ciclomática mide el número de decisiones lógicas (if, bucles, etc.) que contiene un contrato inteligente. Esta métrica refleja cuántos caminos independientes hay en el flujo de ejecución del contrato. Una puntuación alta indica que el código es más complejo, llegando a provocar errores indeseados, además de dificultades en pruebas y mantenimiento.
- Funciones Expuestas: Estas métrica representa cuantas funciones están declaradas como public, external, internal y private. Tanto las funciones declaradas como public, como en las funciones declaradas como external, son funciones expuestas para ser llamadas desde fuera del contrato, pudiendo ser más vulnerables. En el caso de las funciones declaradas como internal o private se llaman dentro del contrato o por contratos heredados. Las funciones publicas y externas implican mayores riesgos, afectando a la complejidad y mantenibilidad del código.

Capítulo 6

Resultados

En este capítulo se comentarán los resultados obtenidos tras haber realizado las pruebas con los tres prototipos. Inicialmente se ha realizado el primer escenario de las pruebas donde se generan las pruebas durante un número total de 20 veces. En este primer escenario las pruebas se han generado atendiendo a la expresión mencionada en el capitulo 4. Los valores de entrada para generar la prueba serán constantes y serán $a = 3$ y $b = 9$.

Para analizar los resultados obtenidos, atendiendo a las métricas explicadas en el capitulo 5, se realiza un análisis estadístico. Se ha determinado calcular las siguientes variables estadísticas:

- Media aritmética: Es el valor promedio del conjunto de datos a estudiar. Proporciona una idea general del centro de los datos.
- Mediana: Es el punto en el que la mitad de los datos son menores y la otra mitad son mayores. Proporciona una visión más clara de la existencia de valores atípicos o outliers que pueden distorsionar la media
- Desviación estándar: Mide la cantidad de variación de un conjunto de datos con respecto a la media. Un valor bajo indica que los datos tienden a estar cerca de la media, mientras que un valor alto indica que los datos están más dispersos.
- Varianza: Es el cuadrado de la desviación estándar. Siendo similar a la desviación estándar, son dos variables interrelacionadas, por lo que un valor alto indica que hay mayor diferencia entre los valores y su media.
- Máximo y mínimo: Son el valor más alto y bajo respectivamente del conjunto de datos. Permite complementar la información sobre los valores atípicos y outliers.
- Rango: Es la diferencia entre mínimo y máximo. Un mayor rango puede indicar un conjunto de datos con valores más disperso.
- Coeficiente de variación: El coeficiente de variación es la relación entre la desviación estándar y la media. Es útil para comparar la variabilidad entre conjuntos de datos que tienen diferentes unidades o medias.
- Percentiles (25 %, 50 %, 75 %): Los percentiles dividen el conjunto de datos en segmentos. Ayudan a comprender la distribución de los datos.

- Boxplot: Es un gráfico que muestra la distribución de los datos por medio de sus cuartiles, resaltando la mediana y los valores atípicos.
- Coeficiente de correlación: Mide la relación entre dos variables. Un valor cercano a 1 indica una correlación positiva fuerte, mientras que un valor cercano a -1 indica una correlación negativa fuerte.
- Gráfico de dispersión: Permite observar de manera gráfica si existe una relación entre dos variables distintas.

Es necesario comentar que se están comparando esencialmente dos tipos de algoritmos. Se comparan Snark y Stark, los cuales están explicados en las secciones 2.4.2 y 2.4.6. Groth16 y Plonk son dos tipos de algoritmos snark distintos, explicados ambos en las secciones 2.4.3 y 2.4.4.

6.1. Tiempo de generación de una prueba

Se han calculados las anteriores expresiones estadísticas primeramente con el tiempo de generación. Se han obtenido los siguientes resultados en la tabla 6.1. De entre los tres

	Groth16	**Plonk**	**Stark**
Media	218,94	508,27	3,38
Mediana	212,30	714,10	3,27
Desviación Estándar	52,33	65,82	1,04
Varianza	2601,75	4115,30	1,03
Máximo	425,90	714,10	6,96
Mínimo	177,60	422,40	2,41
Rango	248,30	219,70	4,55
Coeficiente de Variación	0,24	0,13	0,31
Percentil 25	189,83	470,30	2,72
Percentil 50	212,3	486,70	3,29
Percentil 75	223,23	537,90	3,44

Tabla 6.1: Tiempo de generación (ms)

prototipos Stark tiene claramente el menor tiempo promedio, lo cual sugiere que es el tipo de ZKP más rápido en generar una prueba ZKP. Stark tiene una diferencia de 2 ordenes de magnitud con respecto a Groth16 y Plonk, que es una diferencia bastante considerable. Por su parte, los zkp de tipo Plonk y Groth 16 son más lentos. Pero según lo observado Groth es considerablemente más rápido que Plonk. La mediana es mayor en el caso de Plonk con respecto a la media, lo cual sugiere una asimetría en la distribución de todo el conjunto de valores y la presencia de outliers en el conjunto.

Groth16 y Plonk tienen una desviación estándar considerablemente mayor, 1 orden de magnitud superior, en comparación con la desviación estándar de Stark, significando esto que en ambos casos que los tiempos de generación pueden variar bastante. Por su parte Stark tiene una desviación estándar bastante baja lo que indica que la dispersión es menor y que los tiempos de generación son mas consistentes.

La varianza en el caso de Stark es considerablemente inferior a los otros dos tipos de ZKP. Esto quiere decir que los tiempos de generación de Plonk varían menos que en los

otros dos casos. Plonk presenta la mayor varianza de las tres evaluadas, sugiriendo así que que los tiempos de generación son los más dispersos. Esto cuadra con la interpretación de la mediana y la presencia de outliers. Respecto a máximos, mínimos y rangos se observa que Plonk tiene esos valores más altos, lo cual concuerda con su alta varianza.

Plonk tiene el coeficiente de variación más bajo de los tres tipos de zkp. Dado que cuanto menor es el coeficiente de variación los tiempos de generación serán más consistentes. Esto implica que Plonk, a pesar de la alta desviación estándar y varianza, la variabilidad de los tiempos de generación es baja.

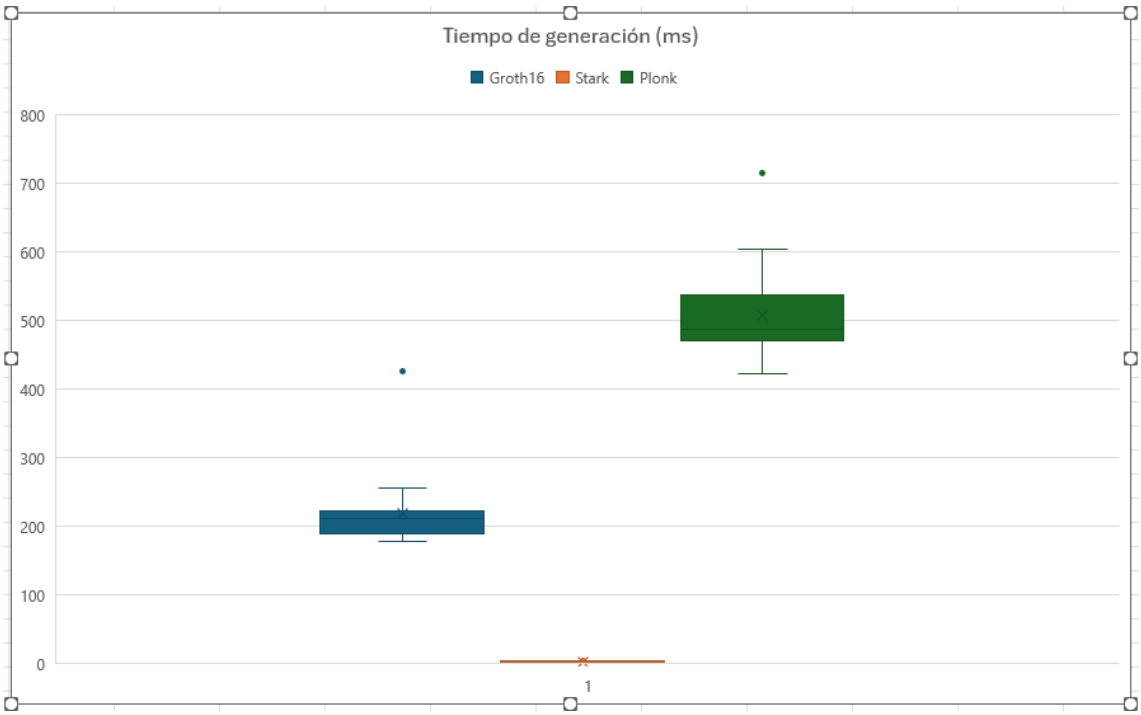

Figura 6.1: Boxplot del tiempo de generación

En la figura 6.1 se muestra el boxplot del tiempo de generación. En el se muestra como las pruebas Stark tiene una caja muy pequeña apenas perceptible en la gráfica respecto a Plonk y Groth16. Comparando la gráfica y los datos obtenidos, se concluye que el tiempo de generación de Stark tiene una variabilidad moderado, con un tiempo de generación pequeño con respecto a sus pares, concentrado en torno a los 3 ms, y la presencia de un outlier (6,963 ms) que indica que el tiempo de generación ocasionalmente puede ser mayor. Groth16 muestra una caja con el rango intercuartílico, tomando valores, aproximadamente, entre 190 ms y 225 ms. Esto indica una variabilidad moderada y que los valores están concentrados. Existe la presencia de un outlier alrededor de 425 ms, lo que indica que el tiempo de generación puede ser significativamente más alto. Por su parte, Plonk muestra una caja amplia cuyo calor intercuartílico va, aproximadamente, de 470 ms hasta 540 ms. Esto indica una mayor variabilidad en el tiempo de generación. Existe un valor outlier de 714 ms, esto quiere decir que ocasionalmente los tiempos de generación que son considerablemente más altos.

En resumen, Stark es el más rápido y consistente de los tres, presentando tiempos de generación mucho más bajos y una varianza mínima. Groth es más rápido y consistente

en comparación con Plonk aún teniendo una dispersión elevada. Plonk tiene el tiempo de generación más alto, una gran dispersión y presenta varios valores atípicos.

6.2. Tiempo de verificación de una prueba

El tiempo de verificación de una prueba es el tiempo que tarda en ejecutarse la transacción de la función que verifica la prueba en smart contract de cada prototipo. El tiempo de verificación en Stark, debido a las características del verifier explicadas en el capitulo 4, se considera:

$$tiempoVerificacion = tiempoVerificacionOffchain + tiempoVerificacionSmartContract$$

6.2.1. Tiempo de verificación de una prueba en Ethereum

	Groth16	**Plonk**	**Stark**
Media	402,19	295,56	11335,59
Mediana	354,7	282,5	11264,64
Desviación Estándar	126,98	73,94	1035,52
Varianza	15316,93	5194,44	1018693,28
Máximo	704,5	456,6	14182,44
Mínimo	276,4	205,6	8636,95
Rango	428,1	251	5545,49
Coeficiente de Variación	0,09	0,25	0.09
Percentil 25	306,75	242,27	11181,32
Percentil 50	354,7	282,5	11264,64
Percentil 75	503,37	317,27	11589,7

Tabla 6.2: Tamaño de verificación de una prueba en Ethereum (ms)

En la tabla 6.2 se muestra el tiempo de verificación de una prueba cuyo smart contract con el verifier está desplegado en Ethereum. Según lo mostrado en la tabla Plonk tiene el menor tiempo de verificación. El tiempo promedio de verificación de Stark es muy elevado, aunque hay que tener en cuenta que es la suma de la verificación on-chain y off-chain.

Plonk y Groth16 tienen medianas más bajas que sus respectivas medias, lo que sugiere que la existencia de outliers, es decir, que alguna verificación haya tardado más tiempo que el promedio. Por su parte, a pesar de tener un tiempo considerablemente mayor, la media y mediana están cercanas, lo que indica una distribución más uniforme de los datos.

Stark tiene una desviación estándar más alta, lo cual refleja una mayor variabilidad en el tiempo de verificación. De igual manera Groth16 también muestra una variabilidad elevada, mientras que Plonk tiene la desviación estándar más baja, indicando una variabilidad menor y tiempos de verificación más consistentes. Stark tiene una varianza mayor a sus pares, lo que indica una dispersión alta entre sus tiempos de verificación. Plonk tiene el rango más corto, lo que refuerza la consistencia y poca variabilidad entre los tiempos. Mientras que Stark tiene un rango mucho mayor, que indica que sus tiempos de verificación son muy variables. El coeficiente de variación en Stark es el más bajo de todos, dando pie a posibles contradicciones con su alta variabilidad, sin embargo esto es debido a la elevada media que tiene. Groth16 y Plonk tienen un coeficiente de variación mas alto que en el

caso de Stark, indicando así una mayor dispersión.

Proof Verification Time Off-chain (ms)	Proof Verification Eth (ms)
11,645	14.170,8
12,559	11.620,6
15,519	10.560,4
13,361	11.204,6
17,328	11.237,8
12,332	11401,8
12,333	9903,2
12,38	12326
15,493	11417,4
12,835	11085,6
12,178	11434,02
12,86	11261,3
14,376	11191
12,301	11219,5
12,919	11446,4
13,454	8623,5
12,424	11.169
13,197	11168,7
12,172	12299,6
11,959	11707,3

Figura 6.2: Resultados de tiempo de verificación de Stark en Ethereum

Una vez analizado que representa cada resultado, se pueden apreciar contradicciones respecto a la variabilidad de Stark. Dicha contradicción apreciada surge principalmente por la relación entre su coeficiente de variación y su elevada media. Cuando el coeficiente de variación es bajo, generalmente significa que los datos son relativamente consistentes en comparación con la media. Parece que Stark es más consistente debido al bajo coeficiente de variación, pero en realidad tiene tiempos de verificación muy variables. Comparando con los resultados mostrados en la figura 6.2 se puede comprobar cómo los tiempos obtenidos durante la verificación off-chain son más homogéneos y presentan una menor variabilidad. Esto se debe a que realizar una verificación off-chain en un entorno más controlado, con menos factores externos que puedan influir en los tiempos de ejecución como la carga de la red. Por su parte, el tiempo medido en Ethereum es mucho más heterogéneo y presenta mayor variabilidad. Esto se puede deber a la naturaleza más volátil de la blockchain, donde pueden interferir mayor cantidad de componentes externos. Si bien el tiempo off-chain es más consistente, y predecible por tanto, cuando se combina con el tiempo on-chain este pasa a ser afectado por el tiempo altamente variable on-chain. Hay que tener en cuenta el tiempo de verificación off-chain para evitar inconsistencias en el análisis.

En la figura 6.3 se muestra el boxplot del tiempo de verificación de Ethereum. Las cajas de color azul oscuro (Tiempo de verificación off-chain), color naranja (Tiempo de Verificación on-chain) y de color verde (Tiempo total) hacen referencia a los tiempos medidos

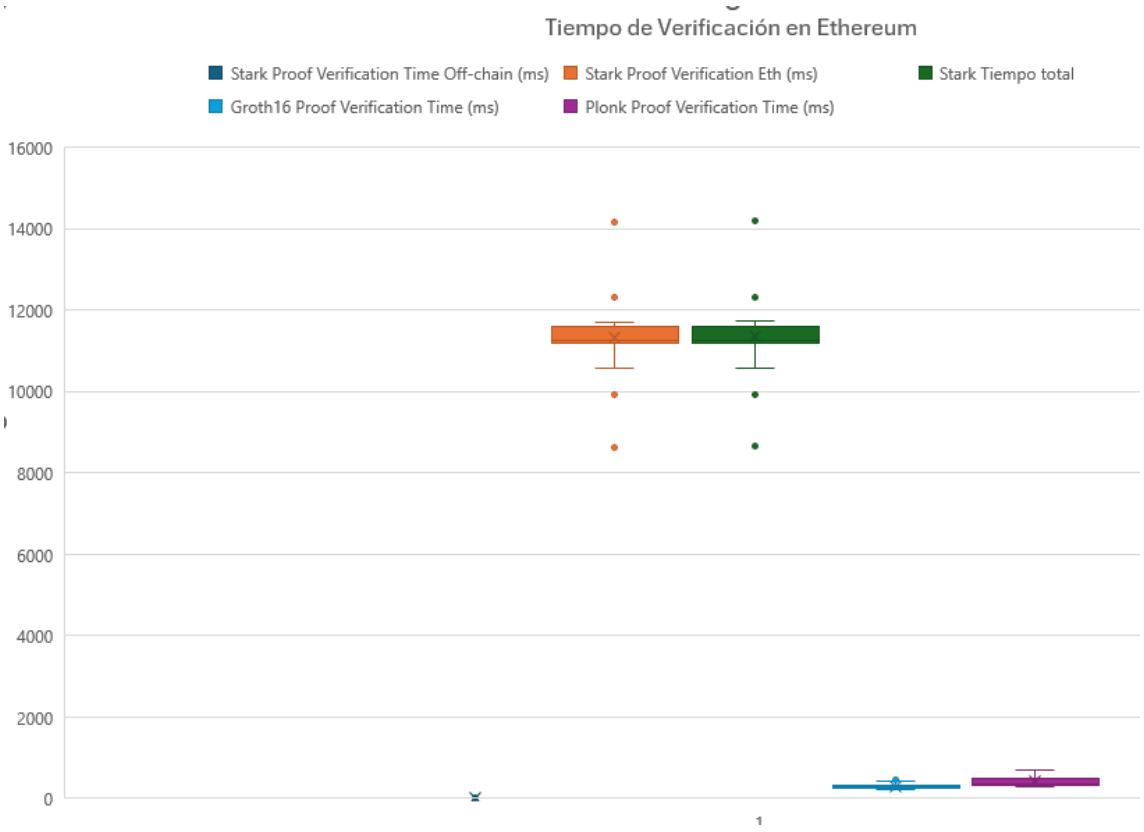

Figura 6.3: Boxplot del tiempo de verificación de Ethereum

en Ethereum para Stark. Por su parte la caja azul claro hace referencia a Groth y la caja violeta hace referencia a Plonk. En él se muestran los tiempos de verificación de los tres prototipos, y en concreto en el caso de Stark se muestran la división de tiempos off-chain y on-chain. Comparando en los datos originales, mostrados en la figura 6.4, el tiempo de verificación off-chain en Stark apenas muestra una línea sin caja ni bigotes aparentes. Al comparar con los resultados, esto indica que los tiempos de verificación son homogéneos, con un valor bajo y poca variabilidad. En el caso de el tiempo de verificación on-chain para Stark, y por consecuente, el total de la suma de ambos tiempos para el caso de Stark, muestran ambos una caja con una gran dispersión. En ambos casos, la caja muestra un rango intercuartílico, indicando así que el tiempo medido sufre una gran variabilidad. Se observan outliers por arriba y por abajo, lo que sugiere que el tiempo de verificación puede ser considerablemente mayor o menor que el tiempo promedio. Esta gran variabilidad se puede interpretar que el tiempo de verificación on-chain es más susceptible a la red Ethereum, impactando significativamente en una gran variabilidad en el tiempo total. En el caso de Groth16 (azul claro) y Plonk (morado), se observa como tienen cajas mucho menores que en el caso anterior, indicando un tiempo de verificación más homogéneo y variabilidad pequeña. En ambos casos, no se aprecian outliers a simple vista, sugiriendo que los tiempos tienen poca variabilidad entre si, pero al comprar con los datos recolectados se aprecia una cierta variabilidad.

Aunque Groth16 y Ethereum se hayan desplegado también en Ethereum, las diferencias con Stark podrían deberse a las diferencias entre algoritmos, pues ambos casos tienen procedimientos de verificación más optimizados para la ejecución en la EVM, así como

Stark			Groth15	Plonk
Proof Verification Time Off-chain (ms)	Proof Verification Eth (ms)	Tiempo total	Proof Verification Time (ms)	Proof Verification Time (ms)
11,645	14.170,8	14.182,4	421,9	342,1999999
12,559	11.620,6	11.633,2	257	609,1
15,519	10.560,4	10.575,9	221,1	276,4
13,361	11.204,6	11.218,0	249,4	385,6
17,328	11.237,8	11.255,1	308,8	304,3
12,332	11401,8	11.414,1	239,9	356,1
12,333	9903,2	9.915,5	205,6	305,1999999
12,38	12326	12.338,4	454,3	317,4
15,493	11417,4	11.432,9	287	520,9000001
12,835	11085,6	11.098,4	308,1	450,8000001
12,178	11434,02	11.446,2	320,1	372,8000001
12,86	11261,3	11.274,2	288,6	296,1
14,376	11191	11.205,4	456,6	353,3000001
12,301	11219,5	11.231,8	343	559,8
12,919	11446,4	11.459,3	226,2	311,4
13,454	8623,5	8.637,0	295,2	295,5
12,424	11.169	11.181,1	273,5	363,6
13,197	11168,7	11.181,9	278	316,8000001
12,172	12299,6	12.311,8	216,1	602
11,959	11707,3	11.719,3	260,7	704,5

Figura 6.4: Resultados de tiempo de verificación

también tienen más herramientas a su disposición. Un procedimiento de verificación más optimizado resulta en menor sensibilidad a las fluctuaciones de la red Ethereum y unos tiempos de verificación más consistentes. El tamaño de la prueba, el gas consumido y el estado de la red durante la ejecución de las transacciones también son motivos de peso para explicar las diferencias entre Stark y Groth16 y Plonk.

6.2.2. Tiempo de verificación de una prueba en IOTA

	Groth16	**Plonk**	**Stark**
Media	7484,07	7540,13	6179,01
Mediana	7480,35	7117,3	6362,6
Desviación Estándar	801,68	1555,23	491,03
Varianza	610549,35	2297789,19	229058,29
Máximo	9226,7	12658,2	7099,2
Mínimo	5586	5952,1	5309,5
Rango	3640,7	6706,1	1789,7
Coeficiente de Variación	0,11	0,21	0,08
Percentil 25	7026,93	6636,93	5824,05
Percentil 50	7480,35	7117,3	6074,1
Percentil 75	8111,93	7674,83	6505,38

Tabla 6.3: Tamaño de verificación de una prueba en IOTA (ms)

En la tabla 6.3 se muestran los datos recogidos del tiempo de verificación en IOTA. Del mismo modo que en Ethereum, se mide lo que tarda en procesarse la transacción de la función de verificación de cada smart contract. Según se observa en los resultados, Stark es el más rápido en promedio, y las diferencias entre los tres no son elevadas, siendo Plonk el más lento de los tres. La mediana en el caso de Groth16 y Stark está bastante cercana a la media, lo que indica una distribución más simétrica y homogénea. Sin embargo, en el caso de Plonk, la mediana es significativamente más baja que la media. Esto quiere decir que algunos tiempos de verificación son mucho más altos, debido a outliers o variaciones. Stark tiene la desviación estándar más baja, por lo que los tiempos de verificación son los más consistentes. Groth16 la mayor desviación estándar, indicando más variabilidad, y Plonk

Figura 6.5: Boxplot tiempos de verificación de tiempos en IOTA

presenta una variabilidad intermedia, a pesar de la presencia de outliers. Plonk tiene la mayor varianza, lo que sugiere que, en ocasiones, los tiempos de verificación pueden ser considerablemente mayores. Stark tiene el rango más bajo, reforzando así la idea de que tiene la menor variabilidad y la mayor consistencia.

En el gráfico 6.5 se muestra el boxplot de los tiempos de verificación en IOTA. Este gráfico esta en concordancia con los resultados obtenidos, mostrando en él a Stark como la prueba que menor variabilidad ofrece y cuyos valores son más consistentes. En el caso de Plonk y Groth16 muestran mayor variabilidad. Plonk presenta dos outliers con valores muy superiores al promedio, lo que indica que en algunas ejecuciones, el tiempo de verificación de Plonk fue significativamente más alto que el promedio. Los bigotes y la caja son extensos, lo que sugiere que los tiempos de verificación pueden extenderse considerablemente, pero siguen dentro de un rango moderado. Por su parte, Groth16, al tener una caja y bigotes muy extensos, lo que indica que presenta una gran variabilidad, en la que los tiempos de verificación pueden variar significativamente. No se observan outliers, lo que indica una mayor predictibilidad de los valores a pesar de su variabilidad.

En general, se observan que los tres tipos de pruebas tienen una gran variabilidad en los tiempos de verificación, lo cual puede estar provocado por la latencia de la red IOTA. A pesar de estas variaciones, Groth16 y Stark han mostrado una mayor consistencia, lo que podría indicar que son menos sensibles a las variaciones en la latencia de la red.

A pesar de todo, los tres tipos de prueba tienen una variabilidad bastante mayor que los tiempos de verificación en Ethereum. Esto se puede interpretar como una consecuencia

de la utilización de Metamask como puente entre el entorno Remix IDE y la red de IOTA.

6.3. Tamaño de una prueba

En el caso estudiado, donde las entradas para las pruebas son constantes el tamaño se mantiene constante en los casos de Groth16 y Stark, mientras que Plonk presenta ligeras variaciones. Las variaciones presentadas por Plonk, tal y como se puede ver en la tabla 6.4,

	Groth16	**Plonk**	**Stark**
Media	765	2009,1	3071
Mediana	765	2099	3071
Desviación Estándar	0	2,40	0
Varianza	0	5,49	0
Máximo	765	2103	3071
Mínimo	765	2095	3071
Rango	0	8	0
Coeficiente de Variación	0	0,001	0
Percentil 25	765	2097,25	3071
Percentil 50	765	2097,25	3071
Percentil 75	765	2097,25	3071

Tabla 6.4: Tamaño de la prueba (bytes)

son ligeras y no tienen un fuerte impacto en la variabilidad. Las pruebas Groth y Stark se mantienen constantes, lo que puede ser beneficioso a la hora de transmitir y almacenar la prueba. Las pruebas Stark son las que mayor tamaño tienen pero a su vez no tiene variabilidad. Groth16 por su parte tiene las pruebas más pequeñas y también no presenta variabilidad.

6.4. Gas usado

6.4.1. Gas usado en Ethereum

De igual manera que con el tamaño de la prueba, el gas usado en la verificación en Ethereum es constante en los casos de Groth16 y Stark, pero en Plonk presenta variaciones. Se ha medido el gas que se consume en la transacción realizada con la función encargada de verificar la prueba en cada contrato. En la tabla 6.5 se muestran los resultados que se han obtenido al medir el gas consumido en Ethereum Según se observa, Stark es el que consume menos gas de promedio y Plonk es el que mas gas consume de promedio. En Groth16 y Stark los valores de la mediana son iguales a los valores de la media, lo cual indica que el consumo de gas se ha mantenido constante a lo largo del tiempo. Plonk presenta una mediana cercana a la media, lo que indica que hay variaciones en el consumo de gas aunque este se mantiene estable. Plonk tiene una pequeña desviación estándar, indicando así que las variaciones producidas son mínimas. De igual manera sucede con la varianza, con un valor no elevado, y con el coeficiente de variación, con un valor muy bajo, ambas indican que Plonk sufre variaciones en el consumo de gas pero no es apenas perceptible con respecto al consumo promedio.

	Groth16	Plonk	Stark
Media	268324	319024,1	210348
Mediana	268324	318831	210348
Desviación Estándar	0	1069,84	0
Varianza	0	1087320,99	0
Máximo	268324	321158	210348
Mínimo	268324	317258	210348
Rango	0	3900	0
Coeficiente de Variación	0	0,003	0
Percentil 25	268324	318257	210348
Percentil 50	268324	318831	210348
Percentil 75	268324	317271,5	210348

Tabla 6.5: Gas usado en Ethereum

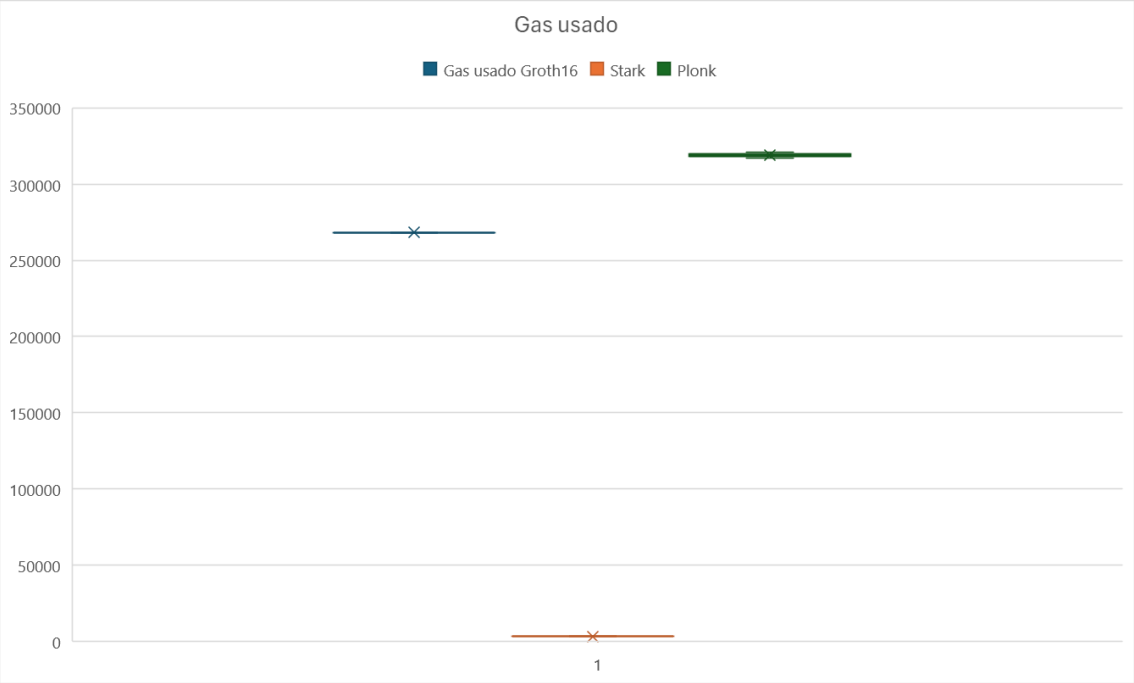

Figura 6.6: Boxplot Gas Usado en Ethereum

En el gráfico 6.6 se muestra como las pruebas Stark y Groth1,6 al tener ambos una linea sin la presencia una caja, se intuye efectivamente tienen un consumo constante de gas. En cambio Plonk muestra una caja con bigotes, siendo la caja en el rango intercuartílico, es decir, los percentiles 25, 50 y 75. La linea horizontal representa la mediana. La caja es pequeña y apenas se muestran variaciones ni outliers. Por tanto se intuye que Plonk presenta cierta variabilidad pero es bastante constante en el consumo de gas.

6.4.2. Gas usado en IOTA

Con respecto al consumo de gas medido al ejecutar las mismas transacciones en IOTA se han obtenido los resultados mostrados en la tabla 6.6. Según se observa, Stark es el más más eficiente en promedio en términos de gas usado. En el caso de Groth16 y Stark la

	Groth16	Plonk	Stark
Media	261824	312274,05	178911
Mediana	261824	312264,5	178911
Desviación Estándar	0	1193,66	0
Varianza	0	1353577,45	0
Máximo	261824	314436	178911
Mínimo	261824	310242	178911
Rango	0	4194	0
Coeficiente de Variación	0	0,004	0
Percentil 25	261824	311196	178911
Percentil 50	261824	312264,5	178911
Percentil 75	261824	313351	178911

Tabla 6.6: Gas usado en IOTA

mediana es igual a la media, lo que indica que el consumo de gas es constante durante la ejecución. En el caso de Plonk, la mediana está muy cerca de la media, lo que indica que el conjunto de datos es bastante homogéneo presentando solo una ligera variabilidad. La desviación estándar de Plonk indica que hay una ligera variabilidad en el consumo de gas. De igual manera, la varianza representa la misma situación. El coeficiente de variación de Plonk es muy bajo, sugiriendo que, a pesar de la variación presente, esta es insignificante en relación con el consumo medio de gas.

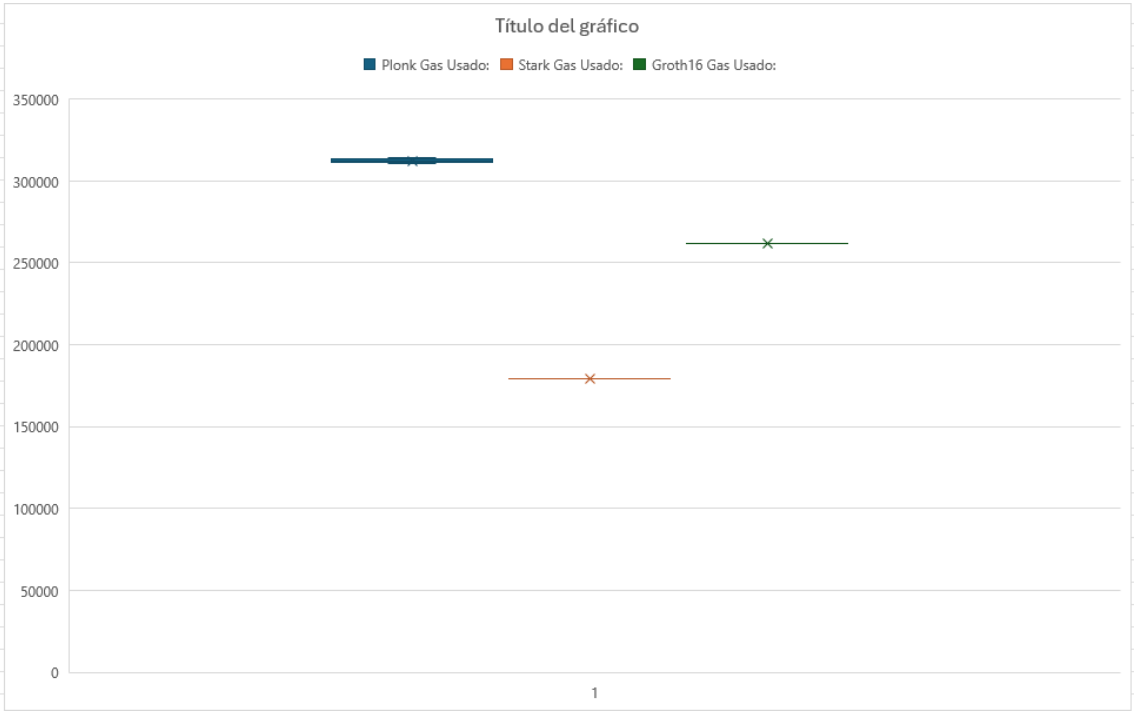

Figura 6.7: Boxplot Gas usado en IOTA

En la figura 6.7 representa el boxplot del gas consumido en IOTA. Los resultados obtenidos en la tabla 6.6 son consistentes con el boxplot. En él se demuestra que tanto Groth16 como Stark son dos tipos de prueba zkp cuyo consumo de gas es constante. Y

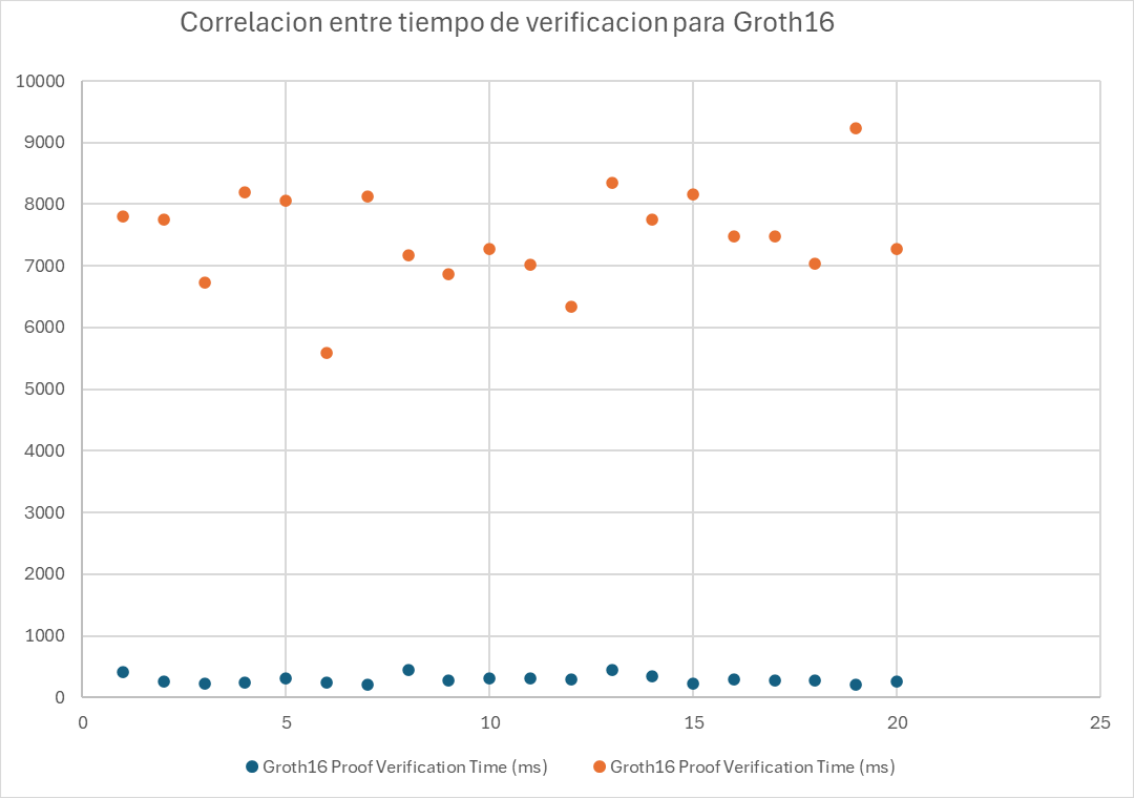

Figura 6.8: Correlación entre los tiempos de verificación entre Ethereum e IOTA en Groth16

en el caso de Plonk se muestra una caja muy pequeña, apenas imperceptible, sin apenas outliers. Esto sugiere que el el consumo de gas para Plonk es bastante consistente, sin apenas variaciones.

6.5. Correlación entre los tiempos de verificación en IOTA y Ethereum

En la figura 6.8 se muestra la correlación entre los tiempos de verificación de Groth16 en Ethereum e IOTA. Los puntos azules representan el tiempo obtenido en Ethereum, mientras que los naranja representan los tiempos obtenidos con IOTA. Según se muestra, los tiempos de Ethereum se concentran en la parte baja de la gráfica, por lo que el tiempo de verificación en Ethereum para Groth16 es significativamente más rápido y mucho más consistentes, sin mucha variación. Los tiempo de IOTA están esparcidos entre 6000 y 9000 ms. Esto sugiere que el tiempo de verificación de Groth16 en IOTA es mucho mayor que en Ethereum y presenta una mayor variabilidad. Esto puede deberse a la infraestructura utilizada o la latencia añadida por el uso de Metamask y otras capas de red en IOTA. No se aprecia de manera visible una correlación entre los tiempos de verificación en ambas plataformas para Groth16.

En la figura 6.9 se muestra la correlación entre los tiempos de verificación de Plonk en Ethereum e IOTA. De igual manera que en la gráfica anterior, los tiempos en Ethereum se concentran en la parte baja de la gráfica, indicando que Plonk es más rápido y más consistente. Los tiempos de IOTA son mas variables y elevados que para Ethereum. Tampoco

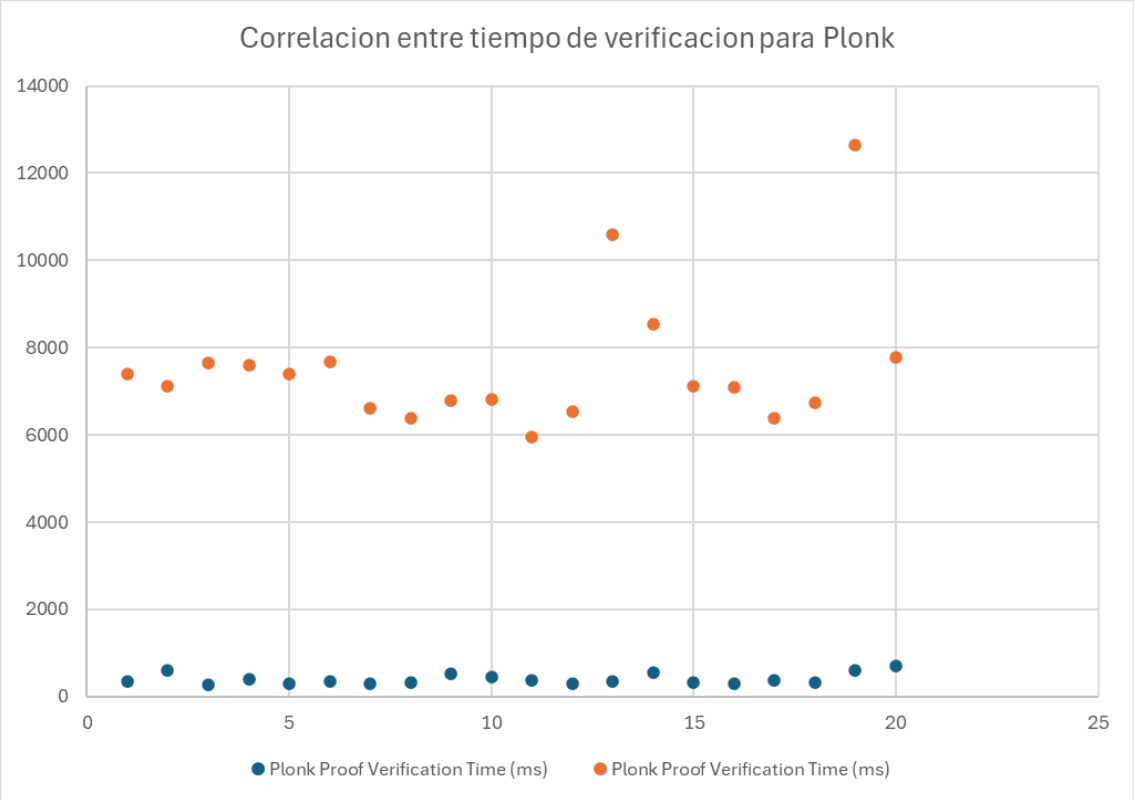

Figura 6.9: Correlación entre los tiempos de verificación entre Ethereum e IOTA en Plonk

se observa un patrón que indique correlación evidente entre los tiempos de verificación de Plonk en ambas plataformas.

En la figura 6.10 de igual manera la correlación entre los tiempos de verificación de Stark en Ethereum e IOTA. En este caso, la situación cambia y se encuentra que el tiempo total de verificación en Ethereum toma valores altos (entre 10000 y 14000 ms), mientras que los tiempo de verificación en IOTA se mantienen más bajos y estables (alrededor de 6000 ms), aunque ambos presentan mayor variabilidad que en los algoritmos Snark estudiados. Repetidamente no parece haber una correlación visible entre las plataformas, ya que los puntos no siguen un patrón conjunto. Los tiempos en IOTA son variables y no dependen de los tiempos observados en Ethereum.

En ninguno de los casos parece haber una correlación clara entre los tiempos de verificación en Ethereum e IOTA para los tres algoritmos (Groth16, Plonk, y Stark). Los tiempos en Ethereum tienden a ser tiempos más bajos y consistentes, sin apenas variabilidad, mientras que los tiempos en IOTA son más elevados y variables. No se muestra un patrón que sugiera dependencia entre ambos conjuntos de datos. Esto indica que el rendimiento en la verificación en una plataforma no afecta a la verificación en la otra plataforma.

6.6. Complejidad de los contratos inteligentes.

Una vez examinadas las métricas referidas a la ejecución de las pruebas, es decir, generación y verificación tanto en Ethereum como en IOTA. Se va a proceder a medir la

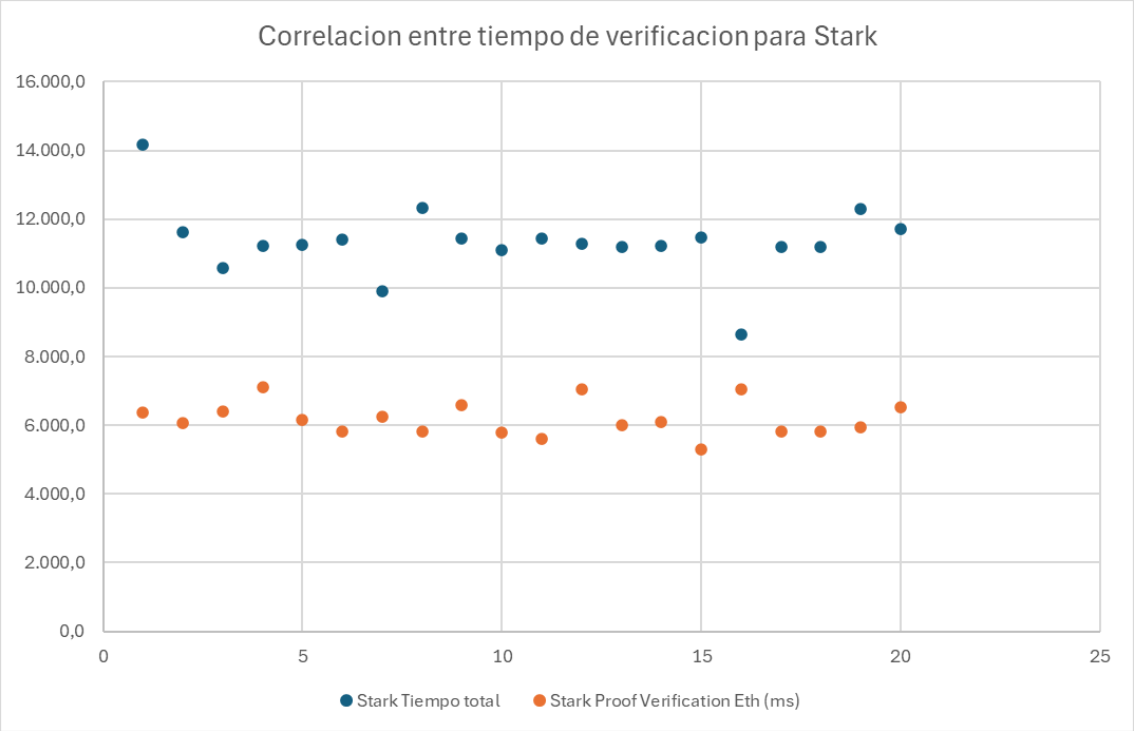

Figura 6.10: Correlación entre los tiempos de verificación entre Ethereum e IOTA en Stark

complejidad de los smart contracts desarrollados en Solidity.

Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
	c:\Users\Lenovo\OneDrive\MASTER\IOT\Trabajo Fin de Master (TFM)\Codigo\default\workspace (3)\verifier.sol	2		197	186	157	25	220	
	c:\Users\Lenovo\OneDrive\MASTER\IOT\Trabajo Fin de Master (TFM)\Codigo\default\workspace (3)\contracts\zkp\verifier_stark.sol	1		85	81	58	3	57	
	c:\Users\Lenovo\OneDrive\MASTER\IOT\Trabajo Fin de Master (TFM)\Codigo\default\workspace (3)\contracts\zkp\verifier_plonk.sol	1		50	46	27	7	12	
	c:\Users\Lenovo\OneDrive\MASTER\IOT\Trabajo Fin de Master (TFM)\Codigo\default\workspace (3)\contracts\zkp\verifier_groth16.sol	1		47	43	26	6	11	
	c:\Users\Lenovo\OneDrive\MASTER\IOT\Trabajo Fin de Master (TFM)\Codigo\default\workspace (3)\contracts\zkp\EllipticCurve.sol	1		446	347	185	131	215	
	c:\Users\Lenovo\OneDrive\MASTER\IOT\Trabajo Fin de Master (TFM)\Codigo\default\workspace (3)\zk\verifier.sol	2		202	191	162	25	231	
	c:\Users\Lenovo\OneDrive\MASTER\IOT\Trabajo Fin de Master (TFM)\Codigo\default\workspace (3)\circuits\zk\groth16\zk\build\zk_verifier.sol	1		170	170	97	35	468	
	c:\Users\Lenovo\OneDrive\MASTER\IOT\Trabajo Fin de Master (TFM)\Codigo\default\workspace (3)\circuits\zk\plonk\zk\build\zk_verifier.sol	1		697	697	515	67	2801	
	c:\Users\Lenovo\OneDrive\MASTER\IOT\Trabajo Fin de Master (TFM)\Codigo\default\workspace (3)\zk\build\groth16\zk_verifier.sol	1		170	170	97	35	468	
	c:\Users\Lenovo\OneDrive\MASTER\IOT\Trabajo Fin de Master (TFM)\Codigo\default\workspace (3)\zk\build\plonk\zk_verifier.sol	1		697	697	515	67	2801	
	c:\Users\Lenovo\OneDrive\MASTER\IOT\Trabajo Fin de Master (TFM)\Codigo\default\workspace (3).deps\remix-tests\remix_tests.sol	1		225	225	164	29	84	
	c:\Users\Lenovo\OneDrive\MASTER\IOT\Trabajo Fin de Master (TFM)\Codigo\default\workspace (3).deps\remix-tests\remix_accounts.sol	1		39	39	22	1	3	
	c:\Users\Lenovo\OneDrive\MASTER\IOT\Trabajo Fin de Master (TFM)\Codigo\default\workspace (3).deps\npm\hardhat\console.sol	1		1552	1550	1165	2	784	
	Totals	**15**		**4577**	**4442**	**3190**	**433**	**8155**	

Figura 6.11: Resultado de Solidity Metrics

Gracias a la herramienta Solidity Metrics, se ha obtenido un resumen de las métricas de complejidad que se muestran en la figura 6.11. En dicha imagen se muestra una tabla con una visión detallada del tamaño, complejidad y otros aspectos clave del código fuente de contratos inteligentes en Solidity. En total se ha obtenido los siguientes resultados:

- Lines (4577): Total de líneas en todos los archivos combinados.
- nLines (4442): Total de líneas no vacías en todos los archivos.
- nSLOC (3190): Total de líneas de código fuente.
- Comment Lines (433): Total de líneas de comentarios.
- Complexity Score (8155): Suma total de la puntuación de complejidad.

Analizando los resultados obtenidos en cada contrato inteligente se desglosan los siguientes resultados.

- El fichero verifier.sol, que es el contrato generado automáticamente por Zokrates, tiene dos contratos lógicos y un tamaño relativamente moderado con 157 líneas de código fuente. La complejidad es de 220, lo que indica que el código tiene una complejidad media.
- El fichero verifier_stark.sol, que es el contrato desarrollado para verificar una prueba Stark, tiene un tamaño pequeño con 58 líneas de código fuente y una puntuación de complejidad baja (57). Esto sugiere que es relativamente sencillo y fácil de mantener.
- El fichero verifier_plonk.sol, que es el contrato desarrollado para verificar una prueba Plonk, tiene un tamaño pequeño con 27 líneas de código fuente y una puntuación de complejidad baja (12).
- El fichero verifier_groth16.sol, que es el contrato desarrollado para verificar una prueba Groth16, tiene un tamaño pequeño con 26 líneas de código fuente y una puntuación de complejidad baja (11).
- EL fichero EllipticCurve.sol es considerablemente más grande con 185 líneas de código fuente y una puntuación de complejidad más elevada, de 215.
- El fichero zk/verifier.sol, es un fichero generado automáticamente por el script de generación de Groth16, es similar al primer contrato estudiado. Al igual que dicho contrato, este tiene 2 contratos lógicos, un tamaño moderado con 165 líneas y una complejidad relativamente alta (231)
- El fichero circuits/zk/groth16/zk/build/zk_verifier.sol, tiene un tamaño no elevado, de 97 líneas de código fuente, y una complejidad elevada (468) con respecto a su tamaño. Es el smart contract que se ha generado automáticamente para Groth 16 utilizando el script de generación que basado en la herramienta Circom, es decir, este contrato es un resto de pruebas anteriores.
- El fichero circuits/zk/plonk/zk/build/zk_verifier.sol, tiene un tamaño más elevado con 515 líneas de código, y una complejidad muy elevada marcando el máximo, con una puntuación de 2801. Es el contrato que se genera con el script de Plonk.
- El fichero /zk/build/groth16/zk_verifier.sol es el mismo que el fichero circuits/zk/-groth16/zk/build/zk_verifier.sol.
- El fichero /zk/build/plonk/zk_verifier.sol es el fichero generado automáticamente por la herramienta Circom, y es el mismo fichero que circuits/zk/groth16/zk/build/zk_verifier.sol.

- Los ficheros remix_tests.sol, remix_accounts.sol y console.sol son contratos auxiliares. El primero tiene un tamaño moderado, con 164 líneas de código, y una complejidad baja, con una puntuación de 84, Por su parte, el segundo contrato es un archivo pequeño y muy sencillo con solo 22 líneas de código fuente y una complejidad casi nula (3). Por ultimo, el tercer contrato tiene un tamaño grande con 1165 líneas de código fuente y una complejidad elevada de 784

Es importante anotar que existen ficheros repetidos. Esto es debido a que las herramientas Circom y Zokrates generaban automáticamente los contratos inteligentes con el verificador en un directorio distinto a los contratos inteligentes generados por los scripts. También se probó a generar pruebas Groth16 con Circom. Se ha decidido mantener dichos contratos con el objetivo de comparar si los contratos inteligentes generados por las herramientas eran iguales que los generados por los scripts. Y en efecto, se comprueba como los ficheros generados por las herramientas son idénticos a los generados por los scripts, pues se mantienen todas las métricas de complejidad con los mismos valores. Cabe observar que el contrato para el caso de Groth16 generado por Zokrates es mas eficiente que el generado por Circom, al tener la mitad de complejidad a pesar de tener un mayor numero de líneas de código con respecto al generado por Circom.

De entre el total de contratos se extrae que 6 de ellos son librerías y el resto son contratos que se puede interactuar con dichas librerías. Las librerías mencionadas son aquellas que definen todas las funciones criptográficas y matemáticas encargadas de realizar la verificación de cada algoritmo, es por ello que la complejidad obtenida en dichas librerías es mayor. Se observa que en el total no hay ninguna función payable, y hay 32 funciones públicas. Hay 448 funciones cuya vista es interna, siendo 404 de ellas pure y 23 view. Esto quiere decir que se puede mostrar su resultado al usuario. De las variables de estado se puede extraer que 14 son publicas, de un total de 208 variables. Se utilizan en los contratos funciones hash y ensamblador. En los contratos que se utiliza ensamblador suelen ser los que más complejidad presentan.

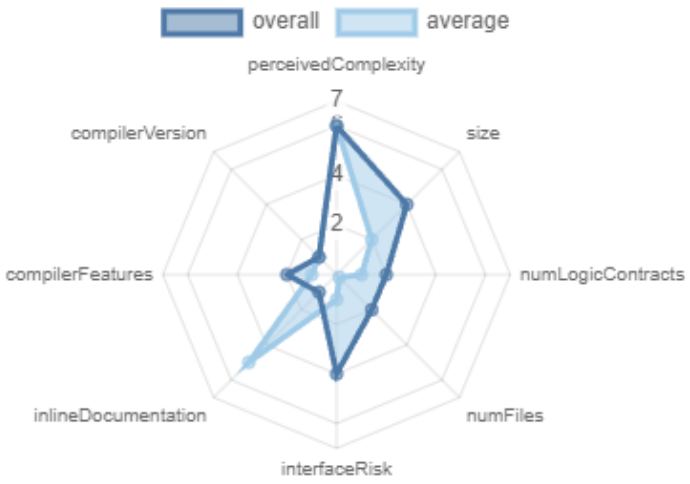

Figura 6.12: Gráfico de Radar: Complejidad percibida

En el gráfico 6.12 se compara la complejidad percibida con otros factores como el tamaño (size), la versión del compilador (compilerVersion), el número de contratos lógicos (numLogicContracts), el riesgo de la interfaz (interfaceRisk), la documentación en línea

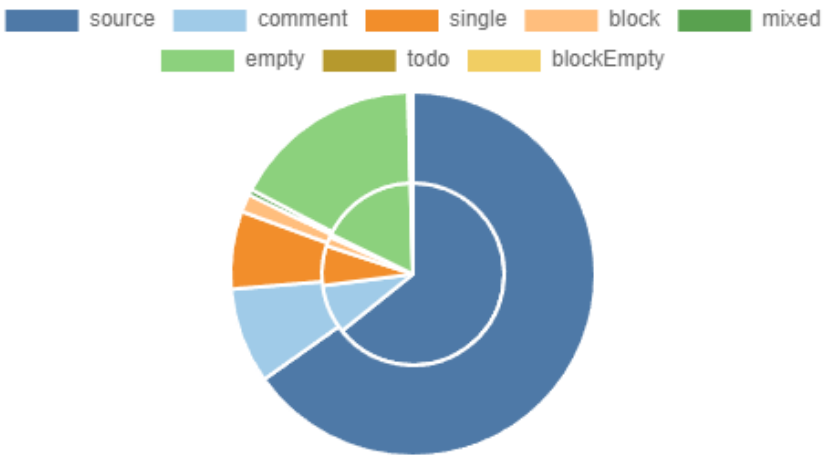

Figura 6.13: Source Lines (sloc vs. nsloc)

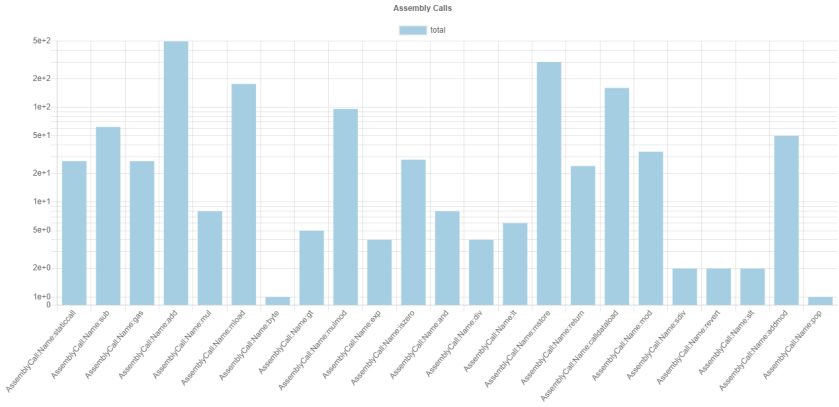

Figura 6.14: Assembly Calls

(inlineDocumentation), y las características del compilador (compilerFeatures). En este gráfico se indica que el código es percibido como complejo debido al tamaño total, numero de contratos lógicos presentes. Se indica una documentación escasa lo cual puede perjudicar en el mantenimiento del código. Se aprecia un riesgo bajo con respecto a las interfaces.

En el gráfico 6.12 se muestra la distribución de diferentes tipos de líneas de código dentro del proyecto. La mayor parte de lines son Source, es decir, código fuente. Es un resultado esperable en un proyecto en desarrollo. Hay una cantidad considerable de comentarios, pero pudiendo ser mejorada la documentación para mayor claridad y comprensión del código.

El gráfico 6.14 muestra la frecuencia de llamadas a diferentes operaciones de ensamblador (assembly calls) dentro del código. Las operaciones Mstore, Add, Mul y Exp son las más frecuentes y sugiere un uso intensivo de cálculos matemáticos y almacenamiento en memoria. Es lógico sabiendo que para verificar una prueba zkp se realizan cálculos criptográficos avanzados. Las operaciones Staticcall y Sub tienen también una frecuencia alta. Esto implica en una implementación compleja con llamadas externas. Las funciones Revert y Return se utilizan con menor frecuencia debido a que son funciones de control de flujo.

El gráfico 6.15 muestra la frecuencia de las diferentes llamadas a funciones dentro del

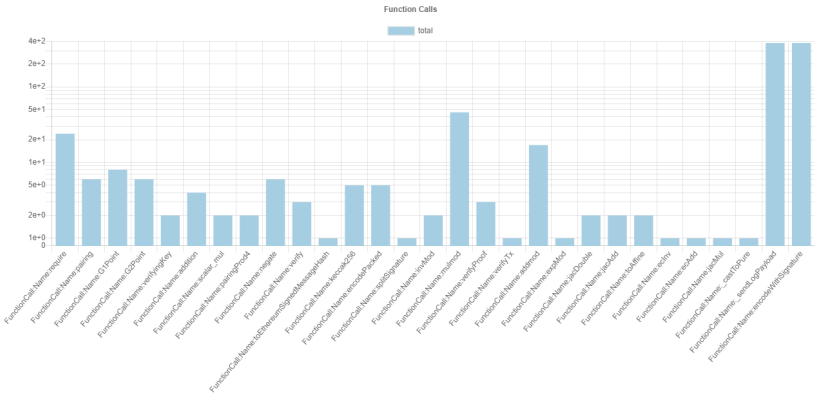

Figura 6.15: Function Calls

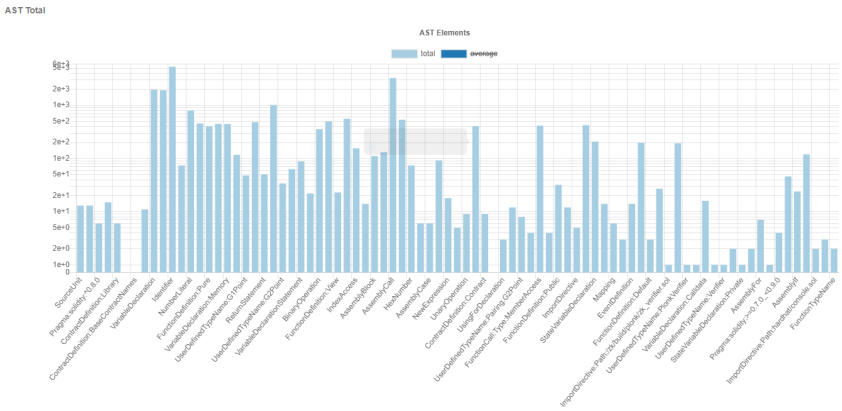

Figura 6.16: AST Elements

código. Las llamadas relativas a las funciones encargadas de la verificación y receive son las más frecuentes. Tiene lógica en contexto donde se verifican pruebas criptográficas. Llamadas a otras funciones como add, sub, mul reflejan una variedad de operaciones matemáticas y lógicas, utilizadas para verificar las pruebas zkp. Se percibe diversidad en las llamadas a funciones, sugiriendo una lógica distribuida entre los diferentes contratos.

El gráfico 6.16 describe la cantidad de elementos del Árbol de Sintaxis Abstracta (AST) en el código, como pueden ser declaraciones de variables, identificadores, declaraciones de funciones, etc. Identifier y VariableDeclaration destacan sobre el resto, algo común en un programa que maneja múltiples variables y parámetros. AssemblyBlock y AssemblyCall tiene una presencia considerable, lo que concuerda con el uso del ensamblador en varios de los smart contracts. También destaca FunctionDefinition, reforzando la modularidad del proyecto.

El ultimo gráfico 6.17 resume la distribución total de elementos del AST y va ligado al gráfico 6.16. AST Statements son la mayoría de elementos en el gráfico, algo esperable al incluir todas las declaraciones del código. StateVars stateVarsPublic tienen también una presencia considerable, significando esto que la blockchain manejará una gran cantidad de datos. ContractDefinitions y functionsPublic también son importantes, pues reflejan la estructura del proyecto con la definición de múltiples contratos y funciones publicas para interactuar con dichos contratos.

Los resultados devueltos por Solidity Metrics revelan que, aunque algunos archivos in-

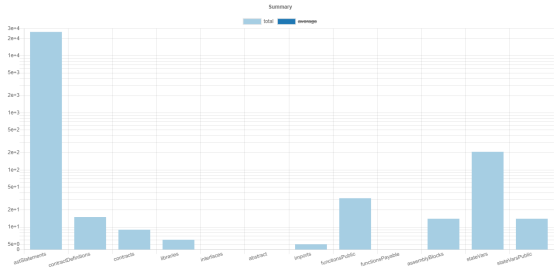

Figura 6.17: Total elementos AST

dividuales presentan una complejidad relativamente baja, hay otros con una complejidad considerablemente alta. En particular, el fichero zk_verifier.sol. Este fichero es el generado automáticamente por la herramienta Plonk. Aparece dos veces en el sistema de ficheros, esto se debe a que el plugin de Circom genera el fichero en un directorio distinto a cuando se genera por medio del script de generación. Los archivos que tienen mayor numero de SLOC son aquellos ficheros con mayor complejidad, esto sugiere que estos contratos inteligentes tienen una lógica compleja que podría aumentar el riesgo de errores o vulnerabilidades.

Poniendo en contexto estos resultados, los smart contracts desarrollados presentan una baja complejidad, pero los generados automáticamente y las librerías utilizadas aportan complejidad al total del proyecto. Los contratos desarrollados interactúan con estos segundos, llevando a cabo estos el total de la labor de verificación, utilizando funciones matemáticas y llamadas a ensamblador para verificar las pruebas zkp.

Relacionando los resultados obtenidos por Solidity Metrics con los algoritmos estudiados se observan diferencias significativas en términos de tamaño, complejidad del código, y funcionalidad. El contrato que incluye el verifier para Stark es pequeño, presentando solo 58 líneas de código y una baja puntuación de complejidad (57). Esto implica, que a pesar de la complejidad propia del algoritmo Stark, es un contrato simple y fácil de mantener. A pesar de que Stark maneja pruebas más grandes y tiempos de verificación más largos, su contrato inteligente es poco complejo, lo cuál se debe a que la gran parte del trabajo de verificación se realiza off-chain, dejando al contrato inteligente con una funcionalidad más reducida.

El contrato encargado del verifier de Plonk tiene solamente 27 líneas de código y una puntuación de complejidad muy baja (12). Sin embargo, el contrato generado por la herramienta Circom tiene una complejidad muy alta con 515 líneas de código y una puntuación de complejidad de 2801, representando el mayor nivel de complejidad observado en este análisis. Plonk, por tanto, presenta la mayor complejidad de entre los tres algoritmos.

De manera similar al otro Snark, Groth16 en el contrato encargado del verifier presenta un complejidad muy baja, con 26 líneas de código y una puntuación de complejidad de 11. El contrato generado por Zokrates presenta una complejidad mayor al anterior, con una puntuación de 231, pero moderada en general. Aunque Groth16 es ser más eficiente en cuanto a los tiempos de verificación y el tamaño de las pruebas, introduce mayor complejidad al generar automáticamente los contratos.

Estos resultados para ambos Snark sugieren que la automatización del proceso de ge-

neración, si bien es útil, puede incrementar la complejidad del código si no se optimiza adecuadamente. De los tres algoritmos, Stark es el que presenta la menor complejidad, lo que lo hace más atractivo en términos de mantenimiento y gestión. Stark sería el algoritmo ideal si se prioriza la sencillez y baja complejidad. Groth16 aún siendo eficiente y sin presentar una gran complejidad requiere de optimización en los contratos generados automáticamente. Plonk ofrece flexibilidad, pero la alta complejidad de los contratos generados automáticamente debe ser gestionada de manera cuidadosa y optimizada de manera correcta.

Capítulo 7

Conclusiones y Trabajo Futuro

Las pruebas de conocimiento cero son un tecnología novedosa y aunque compleja en su funcionamiento interno, poseen potencial para mejorar la privacidad en ámbitos como la identidad digital. Dentro de las pruebas de conocimiento se han estudiado dos tipos de ellas, Snark y Stark.

Las pruebas Stark son mas eficientes en su generación debido a su menor tamaño y por ello consumen menos gas. Proporciona tiempos de generación muy consistentes, por lo que son ideales para contextos donde se necesiten pruebas reducidas y en aquellos escenarios donde las transacciones son más restringidas en términos económicas. Las pruebas de de tipo Snark en cambio son mejor elección si se decide priorizar la rapidez en la verificación.

Los altos tiempos de verificación se deben al proceso de verificación escogido, el cual estaba poco optimizado para Ethereum provocando en un tiempo de verificación mayor y que las transacciones en Ethereum encargadas de la verificación puedan verse afectadas por la latencia de la red.

Dentro de las pruebas Stark se han estudiado dos algoritmos distintos, Groth16 y Plonk. En el caso de Groth16, es la mejor opción en escenarios donde la predecibilidad en el tiempo de generación verificación sea crucial. Por su parte, Plonk es preferible para aplicaciones más complejos, pues ofrece un soporte mejor para circuitos más complejos. Sin embargo en contextos donde se priorice la consistencia en los tiempos de verificación y generación no es recomendable pues posee una mayor variabilidad.

Tanto Plonk y Groth16, debido a la existencia de herramientas y librerías que permiten la generación y verificación de las pruebas, son algoritmos atractivos y útiles en escenarios altamente optimizados para cada prueba.

Las pruebas criptográficas se han verificado en contratos inteligentes desplegados tanto en Ethereum como en IOTA. Ethereum es una plataforma más asentada y con una alta descentralización, con mayor soporte de la comunidad y múltiples herramientas que permiten su utilización. Según lo observado presenta un mayor consumo de gas y altas latencias que puedan afectar a las transacciones. Por tanto Ethereum es una mejor opción para aquellas aplicaciones que requieran de mayo soporte, o para cuando haya requisitos de ser altamente descentralizado.

En el caso de IOTA se observa que es una plataforma con menos consumo de gas, lo que implica una mayor eficiencia a la hora de gestionar transacciones. Sin embargo esto puede entrar en conflicto con los altos tiempos de verificación observados. Esos tiempos elevados se han producido previsiblemente por la utilización de Metamask como puente entre el entorno utilizado y la red de IOTA. La utilización de Metamask puede haber producido latencias adicionales debido a la necesidad de intermediación y la potencial ineficiencia en la comunicación entre Metamask y la red IOTA. Además, la latencia de la red y la configuración específica del entorno pueden afectar en los tiempos de verificación en situaciones de alta demanda de la red. Por lo tanto, aunque IOTA presenta ventajas en términos de eficiencia, es necesario optimizar la infraestructura existente y considerar alternativas para reducir la latencia y mejorar los tiempos de verificación. Entonces IOTA es una buena opción en aplicaciones que requieren una alta escalabilidad y que precisen de bajos costos. Se observa que tanto IOTA como las pruebas Stark son tecnologías más recientes, por lo que no hay herramientas optimizadas para ambas. A pesar de dichas limitaciones, se observan resultados aceptables que permiten el desarrollo posterior.

Debido a que la inspiración de este proyecto se produjo gracias a las identidades digitales, en dicho contexto se pueden observar tres escenarios distintos para cada tipo de prueba:

- Plonk junto a IOTA es una combinación óptima para aplicaciones que requieran de gran complejidad y escalabilidad, donde los costos y la flexibilidad son prioridades. Se debe tener en cuenta en este escenario la variación en la verificación.
- Si se prioriza la predecibilidad en el tiempo de verificación entonces Groth16 junto a Ethereum es una opción razonable.
- Para aplicaciones de identidad digital que prioricen la seguridad, consistencia, descentralización, y rapidez en la verificación entonces Stark junto a IOTA es la mejor opción.

De estos escenarios se entiende que para una aplicación de identidad digital la última opción es la mejor.

Como posibles lineas de trabajo en un futuro se plantean:

- Diseñar otro entorno de pruebas en el que cada prueba generada tenga distintos valores para comprobar cuál es el comportamiento de los tres prototipos al tomar distintas entradas.
- Investigación sobre snark-recursivos y bulletproofs.
- Integración de un nodo wasp para comprobar el rendimiento de IOTA.
- Optimización del proceso de verificación de las pruebas Stark.
- Desarrollo de contratos inteligentes en Cairo con un verifier para pruebas Stark. De este modo se puede aprovechar la plataforma Starknet.
- Desarrollo de herramientas y librerías que permitan la generación y verificación de las pruebas Stark en Ethereum.
- Desarrollo de una aplicación de identidad digital, teniendo en cuenta las pruebas y plataformas estudiadas.

Chapter 8

Introduction

In recent years Blockchain has emerged as an innovative technology for building distributed systems. Blockchain is a technology based on a database shared and distributed among different nodes where the recorded information is stored in blocks linked together through cryptography, and validated in a decentralized way through a common protocol. In other words, there is no central authority controlling the network. The information is stored in the nodes of the network and an identical copy of the information stored in each node is maintained, thus guaranteeing integrity. Blockchain is a type of DLT (Distributed Ledger Technology), i.e. it is a shared, decentralized ledger whose history is immutable. Blockchain records and verifies all transactions made on a network. Blockchain can be considered as a database to which only new transactions can be added, but previous transactions cannot be modified or deleted. As the blocks are cryptographically linked, the generated blockchain is immutable, so the recorded transactions cannot be altered.

The best known application of blockchain is cryptocurrencies or digital currencies. This use case is intended to decentralize financial transactions without the need for a central authority such as a bank. The best known cryptocurrency is Bitcoin. Bitcoin is often used for international transfers, value reserves, online payments, investments, micro-payments, decentralized finance (DeFi), payments without a bank account, peer-to-peer exchange, etc. For example, Bitcoin is very useful in an online shopping scenario where Bitcoin payments are supported. A user will select the service or product he/she wishes to purchase and when it is time to pay, this user chooses Bitcoin as the payment method. The website displays a Bitcoin address, which is a unique string of characters where the payment should be sent, along with the exact amount of Bitcoin to be transferred. The user opens his digital wallet, pastes in the Bitcoin address. He then enters the exact amount of Bitcoin to send and confirms the transaction. This transaction is transmitted to the Bitcoin network, where it is verified by nodes and miners. When the transaction is verified the online shopping website receives notification that the payment has been successful.

But this is not the only existing application. Blockchain can be used to create smart contracts, electronic voting, supply chain management and digital identity management. For example, when applied to digital identity, it allows users to have control over their personal information, reduce the risk of fraud, and increase privacy.

The use of Blockchain technology has had a significant impact in the fields of the In-

ternet of Things, innovating in the field of distributed systems by offering a secure and decentralized structure. Blockchain relies on well-known cryptographic algorithms, such as SHA-1 or hashing. to form its blockchain structure. Each block contains the transaction history, which ensures the immutability and integrity of the data history. This approach allows the transactions made to be public and open. Blockchain offers an additional layer of security in distributed systems by allowing secure transactions without the need for intermediaries or a centralized structure. As a structure whose history is immutable, Blockchain allows users to have greater control thanks to the implementation of mechanisms that ensure that data can only be shared with the explicit consent of the user and that any transaction must be made transparent. Blockchain has driven research into new models such as ZKPs to ensure greater privacy in a transaction. These allow transactions to be validated without revealing sensitive details, enhancing privacy without sacrificing security.

Zero Knowledge Proof (ZKP) or zero knowledge proof is a method between two parties 'prover' (tester) and 'verifier' (verifier). It is an emerging technology, which allows one party to prove to another that a proof is true without revealing additional information. [16]. A practical use case that exemplifies how a zero-knowledge proof works is as follows. Imagine a car rental company, where the company requires the person to provide a copy of his driver's license and credit card to verify that he has a valid license and sufficient available credit. The person who wants to rent a vehicle pre-registers their identity, driver's license and credit card on a platform that supports Zero Knowledge Proof. When a vehicle rental is requested, the platform generates a ZKP test instead of sending personal information. With this zero knowledge proof in this scenario it allows to prove that the person has a valid driver's license and that the credit card has sufficient credit. The rental company receives this cryptographic evidence and verifies it without the need to view driver's license or credit card details. Once the ZKP proofs are verified, the rental company approves the rental request.

In the context of Blockchain, ZKP is used to increase transaction privacy by allowing network participants to verify the validity of a transaction without needing to know the specific details of the transaction. This approach is important in applications where privacy is critical, such as financial transactions or digital identity management. Zero-knowledge proofs are very useful tools in the design of secure protocols. However, the concept of proof of knowledge is very subtle and great care is needed to obtain a successful formalization.

8.1. Motivation

The Internet of Things master's degree provides IoT-related tools and competencies to students taking the master's degree. The Internet of Things (IoT) is a field composed of multiple global information systems. These systems are composed of massive heterogeneous and decentralized devices that can be identified, detected and processed using standardized and interoperable communication protocols.

The goal of this technology is to connect a wide variety of devices allowing to change the way the user interacts with the physical world. These devices can take advantage of the decentralized nature of blockchain by establishing trust, improving security and facilitating communication and data exchange. The transparency of blockchain's history offers advantages to IoT applications.

In the vast majority of cases IoT data is stored on different servers in the cloud, and processed in a distributed manner. This approach can lead to security breaches, making IoT devices susceptible to cyber-attacks. Decentralized systems can provide privacy and scalability, without the need to rely on a central authority or specific hardware. In this context, distributed ledger technologies (DLT) and similar technologies such as Blockchain can be important in the IoT environment.

The most important implementation of blockchain in IoT is the IOTA project. IOTA is a cryptocurrency that promises high scalability, with near-instantaneous transfers at zero cost, focused on Internet of Things solutions. Founded by the IOTA Foundation, the project seeks to solve some of the problems of traditional blockchain technologies, adapting specifically to the needs of IoT.

Currently blockchains face the challenges of being decentralized, secure and scalable at the same time. This dilemma is known as the Scalability Trilemma [66], which postulates that a software system can only have two of the three properties between decentralization, security and scalability. It is not absolute terms, because you can always have some of the three, but what Vitalik Buterin suggests is that in a blockchain system, such as those that will be studied, if you want to gain one of the three properties, you must lose another of the remaining ones. Traditional blockchains such as Bitcoin and Ethereum, for example, are decentralized and secure, but often struggle with scalability issues, resulting in high transaction fees and long processing times. However, there are solutions that have attempted to reach high heights in all three characteristics, even without fulfilling them 100%, such as sharding, sidechains or sidechains, Ethereum layer 2, Proof of Stake (PoS), Directed Acyclic Graphs (DAGs), new consensus algorithms or hybrid chains.

These design issues are particularly important in Blockchain, such as the presence of fees, high processing time and lack of scalability, which do not fit well in a heterogeneous device scenario such as those arising from the new economy emerging with IoT. In an IoT environment where a multitude of devices need to communicate and transact data efficiently and securely, the scalability limitations and costs associated with traditional blockchains are not viable. To mitigate these issues, IOTA created its own DLT, which is called Tanglesilvano2020iota. By eliminating miners and transaction fees, and allowing each transaction to validate at least two previous transactions, the Tangle provides a more efficient and scalable system for the IoT ecosystem, at the cost of losing decentralization. This loss of decentralization follows the Scalability Trilemma.

The presence of Blockchain has impacted the privacy of data used in an IoT environment enabling new applications, but also presenting new challenges in data management and data security. In the IoT context it is vital to have a system in place to ensure data integrity and privacy, as well as protection against unauthorized access. Digital identities therefore become essential in IoT, as they can ensure that only authorized devices access the network and can communicate with each other. Blockchain and IOTA provide a robust solution for digital identity management in IoT. The immutability and decentralization features ensure that identities cannot be forged or altered without detection. Each device can have its identity registered in a distributed block, enabling fast and reliable verification of its authenticity.

Ethereum is a decentralized, open source blockchain platform that enables the creation and execution of smart contracts. It was proposed in 2013 by programmer Vitalik Buterin

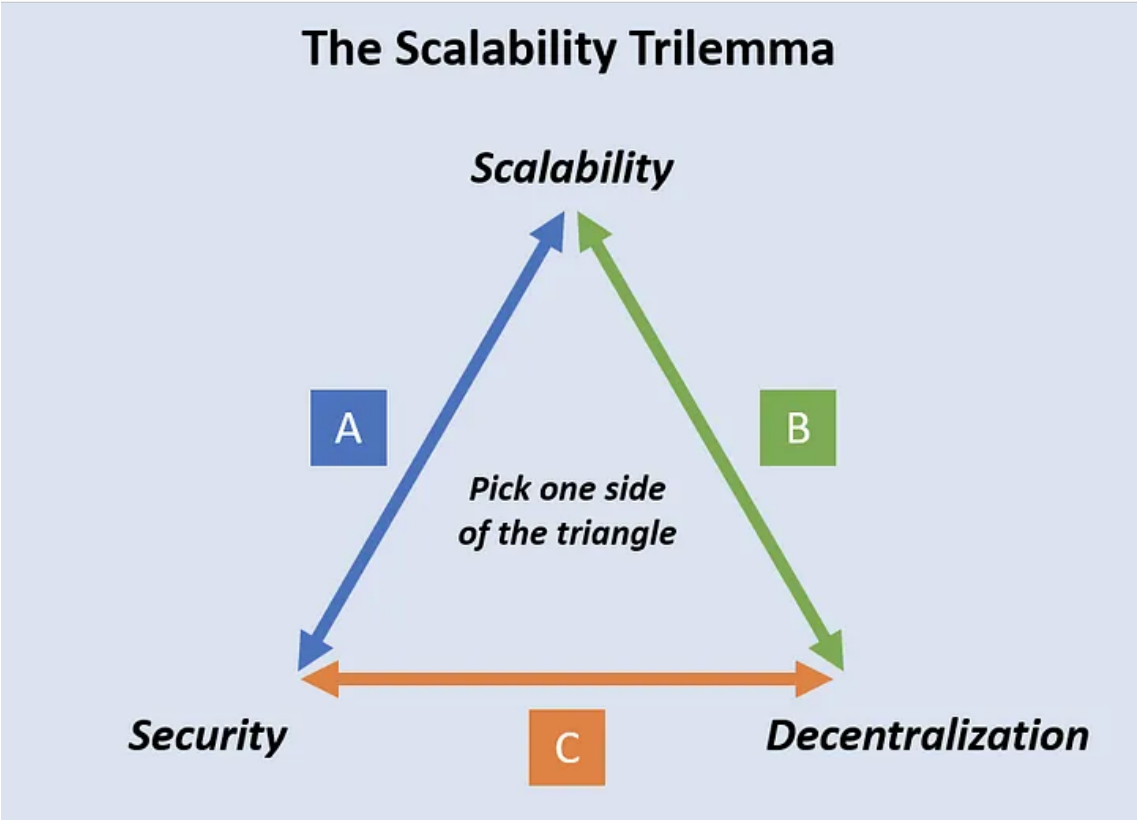

Figure 8.1: Scalability trilemma

and its development began in 2014, with its mainnet launched in July 2015. Ethereum's importance in the context of IoT and digital identity management lies in its ability to execute smart contracts, which are programs that run exactly as configured, without a central server, and without the possibility of disruption, censorship, fraud or third-party interference.

Smart contracts on Ethereum can automate and secure a variety of operations in an IoT environment, from identity management to transaction execution and data exchange between devices. Ethereum's flexibility and robustness make it an ideal platform for experimenting with technologies such as Zero-Knowledge Proof. These enable enhanced privacy and security by allowing verification of information without revealing the underlying data, which is crucial in handling sensitive data in IoT.

This master's thesis proposes a comparative study on the different types of ZKP implemented and integrated in both a blockchain such as Ethereum and IOTA's Tangle.

8.2. Objectives

Zero Knowledge Proofs have become a very novel area of study in the blockchain domain, specifically with respect to privacy and cryptography. The goal of this study is to analyze how ZKPs behave in both Ethereum and IOTA. The following set of development objectives are posed...:

beginitemizeOverall objectives including: Study of technologies related to Zero-

Knowledge Proof, use of comparison metrics to analyze ZKP on Ethereum and IOTA and the development of tests that enable such analysis. Specific Objectives:

- Development of 3 different ZKP prototypes capable of generating proofs.
- Development of smart contracts in Solidity that verify each generated test.
- Use of the selected metrics and perform the analysis based on these metrics. of the test scenario for benchmarking. In this scenario of should be deployed of each smart contract in Ethereum and IOTA. Also measurements should be performed and the process should be repeated 20 times.

Once all the necessary data has been obtained, we will proceed to analyze the obtained results considering the following research questions:

- Which type of ZKP is better, based on different metrics?
- Between Ethereum's blockchain and IOTA's Tangle, on which platform does ZKP perform better?
- What is the type of ZK whose proof weighs the least?

8.3. Plan of work

This work has had an approximate development of 6 months, from February to September, covering different stages that can be defined as follows:

- Phase of definition of the subject to deal with: In this phase we have considered which was an interesting use case in IOTA, choosing Digital Identities.
- Technology research phase: In order to work on digital identities we chose to research on ZKP. In this stage literature on IOTA and ZKP has been reviewed, researching on the different types of ZKP.
- Prototyping phase: In this phase we designed the different ZKP prototypes and the different smart contracts for Ethereum and IOTA to validate them.
- Test execution phase: Once the prototypes have been developed, the tests are performed by deploying the smart contracts with the verifiers on the Ethereum blockchain and the IOTA Tangle.
- Comparative analysis phase: In this phase, the analysis of the tests performed has been carried out taking into account a series of defined metrics, in order to compare the performance of the different types of ZKP chosen in Ethereum and IOTA.
- Drafting phase: In this stage the report was written, defining the necessary theoretical concepts and detailing the analysis performed as well as the conclusions obtained.

8.4. Structure of the document

The paper is structured in 7 chapters describing the development carried out on the study of zero-knowledge proofs and their integration in both Ethereum and IOTA. A brief summary of the content of each chapter is presented below:

- Chapter 1. Introduction: An introduction to ZKPs is given, establishing the motivations that make this topic relevant and novel. The objectives of the study are explained and an overview of the content of the document is presented.

- Chapter 2. Technologies and State of the Art: This chapter explores the theoretical foundations of the different technologies and tools used throughout the work. Concepts related to zero-knowledge proof, the cryptography used by the varieties of ZKP that are chosen, concepts about blockchain, Ethereum, IOTA and the tools used for the development of the work will be explored. A section of this chapter will develop the state of the art. It will explore existing works, projects and articles that serve as a basis and inspiration for the development of the work. It is the result of previous research work. Throughout this subchapter we offer the results of this research on Zero Knowledge Proof, on comparative studies between different types of ZKP and the metrics used.

- Chapter 4.Methodology: This chapter will detail the methodology followed in the process of the work. It describes how the different phases of the project were carried out, from research and development to testing and writing the report. It also includes the general diagram followed by the smart contracts and the tests carried out.

- Chapter 4. Development: This chapter details the three smart contracts developed, together with their tests. It explains how these contracts follow the general diagram described in the previous chapter and provides technical details on their implementation and operation.

- Chapter 5. Metrics: This chapter explains the metrics that will be used throughout the comparative analysis. Metrics will be used to compare test generation times, test verification times and gas consumed during the process. Metrics will also be defined to evaluate the complexity of the smart contracts developed.

- Chapter 6. Comparative analysis results: In this chapter an analysis will be performed comparing different types of ZKP implemented in both Ethereum and IOTA taking into account a series of metrics chosen to compare their performance. Subchapters detailing the results obtained based on various metrics chosen to evaluate their performance are included. In addition, the results obtained after performing the comparative analysis will be discussed and their implications analyzed.

- Chapter7. Conclusions and future work: This chapter presents the general conclusions drawn from the comparative analysis of ZKPs deployed on Ethereum and IOTA. It summarizes the key findings and answers the fulfillment of the initially stated objectives. Ideas and recommendations for future research that aims to expand on this research topic are also provided

Chapter 9

Conclusions and Future Work

Zero-knowledge proofs are a novel technology and although complex in their inner workings, they possess potential for improving privacy in areas such as digital identity. Two types of zero-knowledge proofs, Snark and Stark, have been studied.

Stark proofs are more efficient in their generation due to their smaller size and therefore consume less gas. They provide very consistent generation times, making them ideal for contexts where small tests are needed and in scenarios where transactions are more economically constrained. Snark type tests are a better choice if you decide to prioritize speed of verification.

The high verification times are due to the chosen verification process, which was poorly optimized for Ethereum leading to a longer verification time and that Ethereum transactions in charge of verification may be affected by network latency.

Within the Stark tests, two different algorithms have been studied, Groth16 and Plonk. In the case of Groth16, it is the best option in scenarios where predictability in the verification generation time is crucial. Plonk, on the other hand, is preferable for more complex applications, as it offers better support for more complex circuits. However, in contexts where consistency in verification and generation times is a priority, it is not recommended because it has a higher variability.

Both Plonk and Groth16, due to the existence of tools and libraries that allow the generation and verification of tests, are attractive and useful algorithms in highly optimized scenarios for each test.

Cryptographic proofs have been verified on smart contracts deployed on both Ethereum and IOTA. Ethereum is a more settled and highly decentralized platform, with greater community support and multiple tools that enable its use. As observed it presents higher gas consumption and high latencies that can affect transactions. Therefore Ethereum is a better choice for those applications that require more support, or for when there are requirements to be highly decentralized.

In the case of IOTA it is noted that it is a platform with less gas consumption, which implies greater efficiency when handling transactions. However this may conflict with the

observed high verification times. These high times have predictably been caused by the use of Metamask as a bridge between the environment used and the IOTA network. The use of Metamask may have resulted in additional latencies due to the need for intermediation and potential inefficiency in communication between Metamask and the IOTA network. In addition, network latency and environment-specific configuration may affect in verification times in situations of high network demand. Therefore, although IOTA has advantages in terms of efficiency, it is necessary to optimize the existing infrastructure and consider alternatives to reduce latency and improve verification times. So IOTA is a good option in applications that require high scalability and need low costs. It is noted that both IOTA and Stark testing are newer technologies, so there are no tools optimized for both. Despite these limitations, there are acceptable results that allow for further development.

Since the inspiration for this project came from digital identities, three different scenarios for each type of test can be observed in this context:

- Plonk together with IOTA is an optimal combination for applications requiring high complexity and scalability, where costs and flexibility are priorities. Verification variation must be taken into account in this scenario.
- If predictability in verification time is prioritized then Groth16 alongside Ethereum is a reasonable choice.
- For digital identity applications that prioritize security, consistency, decentralization, and speed in verification then Stark alongside IOTA is the best choice.

From these scenarios it is understood that for a digital identity application the last option is the best.

As possible lines of work in the future, the following are proposed:

- Design another test environment in which each generated test has different values to check the behavior of the three prototypes when taking different inputs.
- Research on snark-recursives and bulletproofs.
- Integration of a wasp node to test the performance of IOTA.
- Optimization of the Stark test verification process.
- Development of smart contracts in Cairo with a verifier for Stark tests. This way the Starknet platform can be leveraged.
- Development of tools and libraries that allow the generation and verification of Stark proofs in Ethereum.
- Development of a digital identity application, taking into account the proofs and platforms studied.

Bibliografía

[1] Md Akram y Anshuman Sen. «A case study Evaluation of Blockchain for digital identity verification and management in BFSI using Zero-Knowledge Proof». En: *2022 International Conference on Decision Aid Sciences and Applications (DASA)*. IEEE. 2022, págs. 1295-1299.

[2] Mays Alshaikhli et al. «Evolution of Internet of Things from blockchain to IOTA: A survey». En: *IEEE Access* 10 (2021), págs. 844-866.

[3] *Anatomy of a Smart Contract | IOTA Wiki*. en. Jun. de 2024. URL: https://wiki.iota.org/isc/explanations/smart-contract-anatomy/ (visitado 05-09-2024).

[4] *Architecture | MetaMask developer documentation*. en. URL: https://docs.metamask.io/wallet/concepts/architecture/ (visitado 05-09-2024).

[5] Alberto Ballesteros Rodríguez. «zk-SNARKs analysis and implementation on Ethereum». En: (2020).

[6] Eli Ben-Sasson, Lior Goldberg y David Levit. «Stark friendly hash–survey and recommendation». En: *Cryptology ePrint Archive* (2020).

[7] Eli Ben-Sasson et al. «Scalable, transparent, and post-quantum secure computational integrity». En: *Cryptology ePrint Archive* (2018).

[8] Benedikt Bünz et al. «Bulletproofs: Short proofs for confidential transactions and more». En: *2018 IEEE symposium on security and privacy (SP)*. IEEE. 2018, págs. 315-334.

[9] Vitalik Buterin et al. «A next-generation smart contract and decentralized application platform». En: *white paper* 3.37 (2014), págs. 2-1.

[10] *Capa 2*. es. URL: https://ethereum.org/es/layer-2/ (visitado 05-09-2024).

[11] Gerard Capdevila Solanich. «Análisis de IOTA y Ethereum para comunicación de dispositivos IoT». En: (2022).

[12] Thomas Chen et al. «A review of zk-SNARKs». En: *arXiv preprint arXiv:2202.06877* (2022).

[13] *Consensys/vscode-solidity-metrics*. original-date: 2019-05-18T09:48:33Z. Mayo de 2024. URL: https://github.com/Consensys/vscode-solidity-metrics (visitado 05-09-2024).

[14] *Data Structures | IOTA Wiki*. en. Ene. de 2024. URL: https://wiki.iota.org/learn/protocols/iota2.0/core-concepts/data-structures/ (visitado 05-09-2024).

[15] Alfredo De Santis, Silvio Micali y Giuseppe Persiano. «Non-interactive zero-knowledge proof systems». En: *Advances in Cryptology—CRYPTO'87: Proceedings 7*. Springer. 1988, págs. 52-72.

[16] Alfredo De Santis y Giuseppe Persiano. «Zero-knowledge proofs of knowledge without interaction». En: *Proceedings., 33rd Annual Symposium on Foundations of Computer Science*. IEEE Computer Society. 1992, págs. 427-436.

[17] *Deploy a Smart Contract | IOTA Wiki*. en. Jun. de 2024. URL: https://wiki.iota.org/isc/how-tos/deploy-a-smart-contract/ (visitado 05-09-2024).

[18] *EVM Quickstart Guide | IOTA Wiki*. en. Jun. de 2024. URL: https://wiki.iota.org/isc/getting-started/quick-start/ (visitado 05-09-2024).

[19] *facebook/winterfell*. original-date: 2021-04-23T19:20:43Z. Ago. de 2024. URL: https://github.com/facebook/winterfell (visitado 05-09-2024).

[20] Crypto Fairy. *Under the hood of zkSNARK Groth16 protocol (part 1)*. es. Sep. de 2023. URL: https://medium.com/coinmonks/under-the-hood-of-zksnark-groth16-protocol-2843b0d1558b (visitado 05-09-2024).

[21] Ariel Gabizon, Zachary J Williamson y Oana Ciobotaru. «Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge». En: *Cryptology ePrint Archive* (2019).

[22] *Gas y tarifas | ethereum.org*. URL: https://ethereum.org/es/developers/docs/gas/ (visitado 06-09-2024).

[23] *GitHub - facebook/winterfell: Un comprobador y verificador STARK para cálculos arbitrarios*. URL: https://github.com/facebook/winterfell (visitado 05-09-2024).

[24] Oded Goldreich y Hugo Krawczyk. «On the composition of zero-knowledge proof systems». En: *SIAM Journal on Computing* 25.1 (1996), págs. 169-192.

[25] Oded Goldreich y Yair Oren. «Definitions and properties of zero-knowledge proof systems». En: *Journal of Cryptology* 7.1 (1994), págs. 1-32.

[26] Shafi Goldwasser, Silvio Micali y Chales Rackoff. «The knowledge complexity of interactive proof-systems». En: *Providing sound foundations for cryptography: On the work of shafi goldwasser and silvio micali*. 2019, págs. 203-225.

[27] Yinjie Gong et al. «Analysis and comparison of the main zero-knowledge proof scheme». En: *2022 International Conference on Big Data, Information and Computer Network (BDICN)*. IEEE. 2022, págs. 366-372.

[28] Jens Groth. «On the size of pairing-based non-interactive arguments». En: *Advances in Cryptology–EUROCRYPT 2016: 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II 35*. Springer. 2016, págs. 305-326.

[29] Daira Hopwood et al. «Zcash protocol specification». En: *GitHub: San Francisco, CA, USA* 4.220 (2016), pág. 32.

[30] Giacomo Ibba et al. «A Curated Solidity Smart Contracts Repository of Metrics and Vulnerability». En: *Proceedings of the 20th International Conference on Predictive Models and Data Analytics in Software Engineering*. 2024, págs. 32-41.

[31] *iden3/circom*. original-date: 2021-10-11T07:55:24Z. Sep. de 2024. URL: https://github.com/iden3/circom (visitado 05-09-2024).

[32] *Introducción - ZoKrates*. URL: https://zokrates.github.io/ (visitado 05-09-2024).

[33] *Introducción a los contratos inteligentes*. es. URL: https://ethereum.org/es/developers/docs/smart-contracts/ (visitado 05-09-2024).

[34] *Introduction to Digital Autonomy | IOTA Wiki*. en. Dic. de 2023. URL: https://wiki.iota.org/learn/protocols/iota2.0/introduction-to-digital-autonomy/ (visitado 05-09-2024).

[35] Milidrag Ivkic. «A Comparison of a Smart Contract Implementation in Ethereum and IOTA». Tesis doct. University of Applied Sciences, 2023.

[36] Max Kobelt, Michael Sober y Stefan Schulte. «A Benchmark for Different Implementations of Zero-Knowledge Proof Systems». En: *2023 IEEE International Conference on Blockchain (Blockchain)*. IEEE. 2023, págs. 33-40.

[37] Ronald Mannak. *Comparing General Purpose zk-SNARKs*. es. Ago. de 2020. URL: https://medium.com/coinmonks/comparing-general-purpose-zk-snarks-51ce124c60bd (visitado 05-09-2024).

[38] *MetaMask/metamask-extension*. original-date: 2015-09-06T16:34:48Z. Sep. de 2024. URL: https://github.com/MetaMask/metamask-extension (visitado 05-09-2024).

[39] Austin Mohr. «A survey of zero-knowledge proofs with applications to cryptography». En: *Southern Illinois University, Carbondale* (2007), págs. 1-12.

[40] Jan Olav Moltu. «Utilizing the IOTA Smart Contract Platform in a Local Flexibility Market». Tesis de mtría. NTNU, 2023.

[41] Eduardo Morais et al. «A survey on zero knowledge range proofs and applications». En: *SN Applied Sciences* 1 (2019), págs. 1-17.

[42] *Máquina virtual de Ethereum (EVM)*. es. URL: https://ethereum.org/es/developers/docs/evm/ (visitado 05-09-2024).

[43] Satoshi Nakamoto. «Bitcoin: A peer-to-peer electronic cash system». En: (2008).

[44] Anca Nitulescu. «zk-SNARKs: A gentle introduction». En: *Ecole Normale Superieure* (2020).

[45] OKX. *Entendiendo el trilema de la blockchain: guía para principiantes*. es. URL: https://www.okx.com/es-la/learn/blockchain-trilemma-guide (visitado 20-09-2024).

[46] *Open sourcing Winterfell: A STARK prover and verifier*. en-US. Ago. de 2021. URL: https://engineering.fb.com/2021/08/04/open-source/winterfell/ (visitado 05-09-2024).

[47] *openzeppelin-contracts/contracts/utils/cryptography/ECDSA.sol at master · OpenZeppelin/openzeppelin-contracts*. es. URL: https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/utils/cryptography/ECDSA.sol (visitado 05-09-2024).

[48] Alexandre Miranda Pinto. «An introduction to the use of zk-SNARKs in blockchains». En: *Mathematical Research for Blockchain Economy: 1st International Conference MARBLE 2019, Santorini, Greece*. Springer. 2020, págs. 233-249.

[49] *Protect Yourself From Identity Theft By Using Zero-Knowledge Proof, Solidity, and Ethereum | HackerNoon*. URL: https://hackernoon.com/protect-yourself-from-identity-theft-by-using-zero-knowledge-proof-solidity-and-ethereum (visitado 05-09-2024).

[50] *Rollups de conocimiento cero*. es. URL: https://ethereum.org/es/developers/docs/scaling/zk-rollups/ (visitado 05-09-2024).

[51] *Rollups optimistas*. es. URL: https://ethereum.org/es/developers/docs/scaling/optimistic-rollups/ (visitado 05-09-2024).

[52] Uuna Saarela. «Practical Zero-Knowledge within the European Digital Identity Framework: Implementing Privacy-Preserving Identity Checks». En: (2024).

[53] Nathan Sealey, Adnan Aijaz y Ben Holden. «Iota tangle 2.0: Toward a scalable, decentralized, smart, and autonomous IoT ecosystem». En: *2022 International Conference on Smart Applications, Communications and Networking (SmartNets)*. IEEE. 2022, págs. 01-08.

[54] Blockchain Development Services. *Decoding ZK-STARK vs ZK-SNARK: An In-Depth Comparative Analysis*. es. Abr. de 2024. URL: https://medium.com/@pamelawatsona3/decoding-zk-stark-vs-zk-snark-an-in-depth-comparative-analysis-4908f775151f (visitado 05-09-2024).

[55] Bilal Shabandri y Piyush Maheshwari. «Enhancing IoT security and privacy using distributed ledgers with IOTA and the tangle». En: *2019 6th International conference on signal processing and integrated networks (SPIN)*. IEEE. 2019, págs. 1069-1075.

[56] Wellington Fernandes Silvano y Roderval Marcelino. «Iota Tangle: A cryptocurrency to communicate Internet-of-Things data». En: *Future generation computer systems* 112 (2020), págs. 307-319.

[57] *snarkjs/templates/verifier_plonk.sol.ejs at 2c570a815bcbba83e2d9dc0c41878c307dd7ceee · iden3/snarkjs*. es. URL: https://github.com/iden3/snarkjs/blob/2c570a815bcbba83e2d9dc0c418 templates/verifier_plonk.sol.ejs (visitado 05-09-2024).

[58] Montse Sorrius Martí. «Seguridad en la Internet de las cosas. Estudio de IOTA para el Internet of Things». En: (2018).

[59] *STARK*. en. URL: https://starkware.co/stark/ (visitado 05-09-2024).

[60] *Starknet Documentation*. es. URL: https://docs.cairo-lang.org/ (visitado 05-09-2024).

[61] *Starknet Documentation*. en. URL: https://docs.starknet.io/ (visitado 05-09-2024).

[62] Rob Stupay. *Running JS Async/Await scripts in Remix-IDE*. en. Dic. de 2020. URL: https://medium.com/remix-ide/running-js-async-await-scripts-in-remix-ide-3115b5dd7687 (visitado 05-09-2024).

[63] Xiaoqiang Sun et al. «A survey on zero-knowledge proof in blockchain». En: *IEEE network* 35.4 (2021), págs. 198-205.

[64] Don Tapscott, Alex Tapscott et al. «La revolución blockchain». En: *Descubre cómo esta nueva tecnología transformará la economía global. Ediciones Deusto* (2017).

[65] Laurence Tennant. «Improving the Anonymity of the IOTA Cryptocurrency». En: *Univ. Cambridge, Cambridge, UK, Tech. Rep* (2017), págs. 10-09.

[66] *The Limits to Blockchain Scalability*. URL: https://vitalik.eth.limo/general/2021/05/23/scaling.html (visitado 06-09-2024).

[67] turingcomplete stackexchange. *What does it mean that Ethereum is "turing complete"*. Mar. de 2016. URL: https://ethereum.stackexchange.com/questions/2464/what-does-it-mean-that-ethereum-is-turing-complete.

[68] *Welcome to Remix's documentation! — Remix - Ethereum IDE 1 documentation*. URL: https://remix-ide.readthedocs.io/en/latest/# (visitado 05-09-2024).

[69] Hongbo Wen et al. «Practical Security Analysis of Zero-Knowledge Proof Circuits.» En: *IACR Cryptol. ePrint Arch.* 2023 (2023), pág. 190.

[70] Hongbo Wen et al. «Practical Security Analysis of {Zero-Knowledge} Proof Circuits». En: *33rd USENIX Security Symposium (USENIX Security 24)*. 2024, págs. 1471-1487.

[71] *What is Zero-Knowledge Proof(Part 2): ZK-STARK | by Bitunix | Medium*. URL: https://bitunix.medium.com/what-is-zero-knowledge-proof-part-2-zk-stark-9e3e3f4ac3f6 (visitado 05-09-2024).

[72] Huixin Wu y Feng Wang. «A survey of noninteractive zero knowledge proof system and its applications». En: *The scientific world journal* 2014.1 (2014), pág. 560484.

[73] Xiaohui Yang y Wenjie Li. «A zero-knowledge-proof-based digital identity management scheme in blockchain». En: *Computers & Security* 99 (2020), pág. 102050.

[74] Cathie Yun. *Programmable Constraint Systems for Bulletproofs*. es. Feb. de 2019. URL: https://medium.com/interstellar/programmable-constraint-systems-for-bulletproofs-365b9feb92f7 (visitado 05-09-2024).

[75] Zibin Zheng et al. «Blockchain challenges and opportunities: A survey». En: *International journal of web and grid services* 14.4 (2018), págs. 352-375.

[76] *ZK Rollups in 2023: From SNARK to STARK and Beyond | HackerNoon*. es. URL: https://hackernoon.com/zk-rollups-in-2023-from-snark-to-stark-and-beyond?ref=hackernoon.com (visitado 05-09-2024).

[77] *ZK-STARKs: Crean confianza verificable, incluso frente a las computadoras cuánticas | por Adam Luciano | Coinmonks | Medium*. URL: https://medium.com/coinmonks/zk-starks-create-verifiable-trust-even-against-quantum-computers-dd9c6a2bb13d (visitado 05-09-2024).

[78] *Zokrates/ZoKrates*. original-date: 2017-10-23T08:18:23Z. Sep. de 2024. URL: https://github.com/Zokrates/ZoKrates (visitado 05-09-2024).

[79] *¿Qué es un Árbol Merkle?* es. Section: Blockchain. URL: https://academy.bit2me.com/que-es-un-arbol-merkle/ (visitado 05-09-2024).