

# **Seguridad en el enrutamiento utilizando tecnología Blockchain**

Maestría en Redes de Datos

**Marcelo Alberto Gómez**

**Directora:** Dra. Patricia Bazán

**Co-Director:** Nicolás del Río

**Asesor Científico:** Ing. Miguel A. Morandi



Facultad de Informática  
Universidad Nacional de La Plata



# Índice general

<b>I INTRODUCCIÓN</b>	<b>10</b>
<b>Resumen</b>	<b>1</b>
<b>Abstract</b>	<b>1</b>
<b>1 Introducción</b>	<b>2</b>
1.1 Estructura . . . . .	2
1.2 Objetivos . . . . .	2
1.3 Motivación . . . . .	3
<b>2 Estado del Arte</b>	<b>4</b>
2.1 Gobernanza y gestión de información de ruteo global . . . . .	4
2.1.1 Historia de los RIRs . . . . .	4
2.1.2 Historia de los IRR . . . . .	7
2.1.3 Mecanismos actuales de seguridad en el ruteo . . . . .	8
2.2 Uso de la tecnología blockchain para validar la información de ruteo . . . . .	10
2.2.1 Antecedentes . . . . .	12
<b>II MARCO TEÓRICO</b>	<b>14</b>
<b>3 Recursos de internet</b>	<b>15</b>
3.1 Sistemas Autónomos . . . . .	15
3.2 Números IPv4 e IPv6 . . . . .	16
3.3 Administración global de los recursos . . . . .	17
<b>4 Conceptos de BGP</b>	<b>19</b>
4.1 Introducción a BGP . . . . .	19
4.2 Características de BGP . . . . .	20
4.3 Principales tipos de mensaje BGP . . . . .	21
4.4 Principales atributos de BGP . . . . .	21
4.5 Criterios de selección de rutas . . . . .	23
<b>5 Seguridad de ruteo en Internet</b>	<b>24</b>
5.1 Secuestro de rutas - Route Hijacking . . . . .	24
5.1.1 Antecedentes de secuestros de rutas . . . . .	24
5.1.1.1 2008 - Denegación de servicio, Caso Pakistan - Youtube . . . . .	24
5.1.1.2 2014 - Secuestro de BGP para obtener beneficios de criptomonedas . . . . .	25
5.2 Fuga de ruta - Route Leak . . . . .	25
5.2.1 Clasificación de Fugas de Ruta . . . . .	26

5.2.2	Propuesta para solucionar los problemas de fugas de ruta: RFC 9234. . . . .	27
5.2.2.1	Introducción . . . . .	27
5.2.2.2	Relaciones entre peers . . . . .	27
5.2.2.3	Corrección de roles . . . . .	28
5.2.2.4	Atributo BGP solo para el cliente (OTC) . . . . .	28
5.3	RPKI . . . . .	29
5.3.1	Introducción a RPKI . . . . .	29
5.3.2	Validación del origen de la ruta . . . . .	30
5.4	MANRS . . . . .	32
5.5	BGPSEC . . . . .	33
5.6	ASPA - Autonomous System Path Authorization . . . . .	34
5.6.1	Procedimiento de verificación cliente-proveedor . . . . .	35
5.6.2	Procedimiento de verificación de AS_PATH . . . . .	35
<b>6</b>	<b>Conceptos de Sistemas Distribuidos</b>	<b>37</b>
6.1	Definición de Sistemas distribuidos . . . . .	37
6.2	Problema de los generales bizantinos . . . . .	38
6.3	Teorema CAP . . . . .	39
<b>7</b>	<b>Conceptos de Criptografía</b>	<b>41</b>
7.1	Introducción . . . . .	41
7.2	Servicios de la criptografía . . . . .	41
7.3	Primitivas criptográficas . . . . .	42
7.3.1	Primitivas criptográficas sin clave . . . . .	43
7.3.1.1	Secuencias aleatorias . . . . .	43
7.3.1.2	Funciones hash . . . . .	43
7.3.2	Criptografía simétrica . . . . .	45
7.3.3	Criptografía de clave asimétrica . . . . .	45
7.4	Firma digital . . . . .	47
7.5	Árbol de Merkle . . . . .	48
7.6	Árbol Patricia Trie . . . . .	49
<b>8</b>	<b>Conceptos de Blockchain</b>	<b>51</b>
8.1	Definición de blockchain . . . . .	51
8.2	Componentes de una Blockchain . . . . .	51
8.3	Arquitectura de una blockchain . . . . .	52
8.4	Clasificación de una blockchain . . . . .	53
8.4.1	Blockchain pública . . . . .	54
8.4.2	Blockchain privadas o permissionadas . . . . .	55
8.4.3	Blockchain público-permissionadas . . . . .	55
8.5	Descentralización . . . . .	56
8.6	Mecanismos de consenso . . . . .	57

8.6.1	PBFT . . . . .	58
8.6.2	RAFT . . . . .	59
8.6.3	Proof of Work (PoW) ó Prueba de trabajo . . . . .	60
8.6.4	Proof of Stake (PoS) ó Prueba de Participación . . . . .	61
8.6.5	Committee-based PoS . . . . .	62
8.6.6	DPoS - Delegated PoS . . . . .	62
8.6.7	Proof of Authority ó Prueba de Autoridad . . . . .	63
8.6.8	Tower Byzantine Fault Tolerance - Tower BFT . . . . .	64
8.7	Escalabilidad de una blockchain . . . . .	64
<b>9</b>	<b>Plataformas blockchain consideradas</b>	<b>66</b>
9.1	Bitcoin . . . . .	66
9.1.1	Transacciones en la red Bitcoin . . . . .	66
9.1.2	Red Bitcoin . . . . .	67
9.1.3	RSK: Contratos inteligentes sobre la red Bitcoin . . . . .	67
9.2	Ethereum . . . . .	68
9.2.1	¿Qué es Ethereum? . . . . .	68
9.2.2	Características de la red Ethereum . . . . .	68
9.2.3	Historia de la red Ethereum . . . . .	69
9.2.4	Ecosistema de la red Ethereum . . . . .	70
9.2.4.1	Claves y direcciones . . . . .	71
9.2.4.2	Cuentas (Addresses) . . . . .	72
9.2.4.3	Abstracción de cuentas ERC-4337 . . . . .	73
9.2.4.4	Contratos inteligentes . . . . .	73
9.2.4.5	Éter criptomoneda/tokens: . . . . .	74
9.2.4.6	Transacciones y mensajes . . . . .	74
9.2.4.7	Tarifas y gas . . . . .	75
9.2.4.8	EVM . . . . .	76
9.3	Escalabilidad de la Blockchain Ethereum . . . . .	77
9.3.1	Soluciones de escalado on-chain . . . . .	77
9.3.2	Soluciones de escalado off-chain . . . . .	77
9.3.2.1	Sidechain o cadena lateral . . . . .	77
9.3.2.2	Soluciones de escalado de capa 2 . . . . .	78
9.4	Arbitrum . . . . .	79
<b>III</b>	<b>DESARROLLO DEL TRABAJO</b>	<b>81</b>
<b>10</b>	<b>Desarrollo de la solución propuesta</b>	<b>82</b>
10.1	Introducción . . . . .	82
10.2	Arquitectura de la solución propuesta . . . . .	82
10.2.1	Aportes realizados al contrato inteligente . . . . .	83

10.2.1.1	Soporte IPv6 . . . . .	83
10.2.1.2	Utilización de funciones para verificación de firmas realizadas offchain . . . . .	83
10.2.1.3	Ataques de repetición de firmas (Signature replay attacks) . . . .	84
10.2.1.4	Heartbeat function . . . . .	84
10.2.1.5	Migración a Solidity v0.8.18 . . . . .	84
10.2.1.6	Verificación del AS_PATH . . . . .	85
10.3	Despliegue del contrato inteligente . . . . .	85
10.4	Asignación y delegación de recursos de Internet mediante blockchain . . . . .	86
10.4.1	Secuencia de pasos para dar de alta un SA . . . . .	86
10.4.2	Secuencia de pasos para asignar el prefijo IPv6 al SA . . . . .	88
10.4.3	Los SAs indican mediante transacciones cuales son sus SAs vecinos . . . .	91
10.4.4	Conclusiones del trabajo experimental con Remix . . . . .	92
10.5	Despliegue del contrato inteligente con Hardhat . . . . .	92
10.6	Prueba del contrato inteligente . . . . .	97
10.7	Reporte de gas utilizado . . . . .	97
<b>11</b>	<b>Validación de la solución</b>	<b>100</b>
11.1	Introducción . . . . .	100
11.2	Entorno de software utilizado para la realización de pruebas . . . . .	100
11.3	Emulación de ataque de Route Hijacking exitoso . . . . .	103
11.4	Emulación de ataque de Route Hijacking evitado mediante el uso de tecnología blockchain . . . . .	105
<b>12</b>	<b>Conclusiones</b>	<b>108</b>
<b>13</b>	<b>Futuras líneas de investigación</b>	<b>110</b>
13.1	Información adicional on-chain . . . . .	110
13.2	Soporte dual stack IPv6/IPv4 . . . . .	110
13.3	Verificación del AS-PATH mediante el método ASPA . . . . .	110
13.4	Pruebas de la solución en redes público-permisionadas . . . . .	110
<b>14</b>	<b>Anexos</b>	<b>111</b>
14.1	Contrato inteligente IANA.sol . . . . .	111
14.2	Archivo de configuración de Hardhat . . . . .	121
14.3	Scripts para el deploy del contrato inteligente . . . . .	122
14.4	Scripts para testing del contrato inteligente . . . . .	129
14.5	Script Python para validación de paquetes UPDATE de BGP . . . . .	134
14.6	Servidores WEB en NodeJS en SA64503 . . . . .	142
14.7	Servidores WEB en NodeJS en sistema autónomo atacante SA64504 . . . . .	142
14.8	Configuración del R1 . . . . .	143
14.9	Configuración del cliente docker C1 . . . . .	145

14.10 Configuración del router R1-2 . . . . .	145
14.11 Configuración del router R2 . . . . .	145
14.12 Configuración del router R3 . . . . .	146
14.13 Configuración del router R3-2 . . . . .	146
14.14 Configuración del router R4 . . . . .	147
14.15 Configuración del router R4-2 . . . . .	147

# Índice de códigos

10.1	.env	93
14.1	IANA.sol	111
14.2	hardhat.config.js	121
14.3	deploy.js	122
14.4	start_r1.js	122
14.5	start_r2.js	124
14.6	start_r3.js	126
14.7	start_r4.js	128
14.8	iana_test.js	129



# Índice de figuras

2.1	Esquema jerárquico en la asignación de recursos de Internet [3] . . . . .	7
3.1	ASNs de propósito especial [37] . . . . .	16
4.1	Topología de red IPv6 . . . . .	22
5.1	Cadena de confianza de RPKI [45] . . . . .	29
5.2	Cadena de confianza de LACNIC - RPKI [42] . . . . .	30
5.3	Recuperación y validación de datos RPKI [45] . . . . .	31
5.4	Recuperación y validación de datos RPKI [51] . . . . .	33
5.5	ASPA - Formato de objetos firmados [55] . . . . .	34
5.6	ASPA - Procedimiento de verificación [56] . . . . .	35
6.1	Teorema CAP, ó Conjetura de Brewer[62] . . . . .	40
7.1	Taxonomía de las primitivas criptográficas [64] . . . . .	42
7.2	Criptografía simétrica, donde dos entidades comparten una sola clave de cifrado/-descifrado.[64] . . . . .	46
7.3	Criptografía asimétrica. Usando la clave pública de una persona, es posible cifrar un mensaje para que solo la persona con la correspondiente clave privada pueda descifrarlo.[68] . . . . .	46
7.4	Firma digital. Usando una clave privada, se puede crear una firma digital para que cualquier persona con la clave pública correspondiente pueda verificar que el mensaje fue creado por el propietario de la clave privada y no se modificó desde entonces.[68] . . . . .	48
7.5	Merkle Tree[71] . . . . .	48
7.6	Merkle Tree[71] . . . . .	50
8.1	Tipos de redes blockchain.[74] . . . . .	54
8.2	Diferentes tipos de red (centralizada, distribuida y descentralizada)[59] . . . . .	56
8.3	Dimensiones de la descentralización[76] . . . . .	57
8.4	Consenso PBFT. Diagrama de flujo[80]. . . . .	59
8.5	Diagrama de la cadena Bitcoin. Se observa el valor de nonce que se utiliza para minar un bloque y como cada bloque posee en su interior el hash del bloque anterior[82]. . . . .	61
9.1	Diagrama simplificado de un cliente de consenso y ejecución acoplados.[90] . . . . .	71
9.2	Diagrama simplificado de un cliente de consenso y ejecución acoplados.[91] . . . . .	72
9.3	Diagrama simplificado de un cliente de consenso y ejecución acoplados.[92] . . . . .	73
10.1	Topología de red IPv6 - Emulación de secuestro de rutas . . . . .	83
10.2	Solicitud de firma del SA 64501 . . . . .	87
10.3	Ejecución de test utilizando el plugin hardhat-gas-reporter . . . . .	99

11.1	Red de prueba IPv6 - Emulación de secuestro de rutas . . . . .	100
11.2	Topología de red IPv6 - Emulación de un secuestro de rutas en GNS3 . . . . .	101
11.3	Ejecución del comando <b>show bgp ipv6 summary</b> antes del ataque de BGP hijacking	103
11.4	Ejecución del comando <b>show bgp ipv6 unicast 2001:db8:300::/40</b> antes del ataque de BGP hijacking . . . . .	103
11.5	Ejecución del comando <b>curl [2001:db8:300::aa]</b> desde el cliente docker C1 antes del ataque de BGP hijacking . . . . .	103
11.6	Ejecución del comando <b>show bgp ipv6 summary</b> después del ataque de BGP hijacking . . . . .	104
11.7	Ejecución del comando <b>show bgp ipv6 unicast 2001:db8:300::/40</b> después del ataque de BGP hijacking . . . . .	104
11.8	Ejecución del comando <b>curl [2001:db8:300::aa]</b> desde el cliente docker C1 después del ataque de BGP hijacking. El SA65504 logra apropiarse del prefijo. . . . .	105
11.9	Ejecución del comando <b>show bgp ipv6 summary</b> antes del ataque de BGP hijacking	105
11.10	Ejecución del comando <b>show bgp ipv6 unicast 2001:db8:300::/40</b> antes del ataque de BGP hijacking . . . . .	105
11.11	Ejecución de script de validación en Python . . . . .	106
11.12	Ejecución del comando <b>curl [2001:db8:300::aa]</b> desde el cliente docker C1 antes del ataque de BGP hijacking . . . . .	106
11.13	Ejecución del comando <b>show bgp ipv6 unicast 2001:db8:300::/40</b> después del ataque de BGP hijacking . . . . .	106
11.14	Ejecución del comando <b>curl [2001:db8:300::aa]</b> desde el cliente docker C1 después del ataque de BGP hijacking. El mismo no tuvo éxito. . . . .	107
14.1	GNS3 - Configuración de las interfaces de red del contenedor Docker . . . . .	145

## **Parte I**

# **INTRODUCCIÓN**

## **Resumen**

Para realizar un ataque de route hijacking no se saca provecho a ninguna vulnerabilidad o falla de protocolo, sino que se explota que la arquitectura de BGP está basada en la confianza mutua. Por este motivo es que estos ataques son tan antiguos como el mismo protocolo, y en la actualidad estos fallos se siguen produciendo y se continúa investigando cual es la mejor estrategia para brindar seguridad al ruteo en Internet. Las nuevas soluciones, como RPKI, están generando riesgos debido a la centralización de la autoridad de enrutamiento, las mismas no se basan en la confianza mutua, pero dependen de la confianza en la autoridad. Con este trabajo se busca implementar un modelo menos centralizado, basado en blockchain, donde mediante el despliegue de una aplicación específica para la asignación de recursos de Internet en la red Ethereum Sepolia, y la posterior utilización de esa información almacenada, se demuestra como brindar seguridad al protocolo BGP. ...

## **Abstract**

Route hijacking attacks exploit the mutual trust that BGP architecture is based on, rather than any vulnerabilities or protocol flaws. For this reason, these attacks are as old as the protocol itself, and today these failures continue to occur and research continues on what is the best strategy to provide security for routing on the Internet. New solutions, such as RPKI, are generating risks due to the centralization of the routing authority, they are not based on mutual trust, but depend on trust in the authority. This work seeks to implement a less centralized model, based on blockchain technology, where security is provided to the BGP protocol through the deployment of a specific application for the allocation of Internet resources in the Ethereum Sepolia network, and the subsequent use of this stored information.

# Introducción

En este capítulo introductorio, se ofrece una panorámica de lo que conforma la tesis, brindando una visión general de la estructura del trabajo y de los objetivos que se pretenden alcanzar con la investigación realizada. Además, se expondrá la motivación que me llevó a profundizar en el estudio de la asignación de recursos y la seguridad del ruteo externo en Internet.

## 1.1 Estructura

Esta tesis está conformada por tres secciones, en el capítulo uno de la primera sección se comienza describiendo un breve resumen, objetivos y motivación del presente trabajo. En el capítulo dos se detalla el estado del arte y antecedentes sobre la asignación de recursos y el ruteo en Internet.

En la sección dos se explica todo el marco teórico en el cual se abarcan los conceptos que sirven de base para comprender la temática investigada. El marco teórico incluye conceptos de Sistemas Autónomos, números IP, ruteo externo con BGP y seguridad en el ruteo, con el fin de entender la problemática planteada. A posteriori se tratan conceptos criptográficos esenciales para poder profundizar en los principios de funcionamiento de la tecnología blockchain, los contratos inteligentes y las aplicaciones descentralizadas.

En la tercera y última sección se presenta el desarrollo de la solución propuesta y la validación de la solución, junto a las conclusiones y futuras líneas de investigación.

## 1.2 Objetivos

El objetivo general de esta tesis es analizar las propiedades de blockchain y su aplicación específica en la asignación y delegación de recursos de la infraestructura de Internet, así como también en la seguridad de ruteo externo BGP<sup>1</sup> contra ataques de Route Hijacking<sup>2</sup> o similares.

La hipótesis que se plantea, es que al utilizar blockchain para almacenar las asignaciones y delegaciones de recursos de Internet, se facilita el despliegue de un sistema de enrutamiento seguro entre sistemas autónomos. Adicionalmente proporciona un modelo de confianza flexible, equilibra el poder entre los actores participantes, simplifica la administración, brinda un sistema altamente auditable y resuelve el consenso distribuido otorgando confianza entre sus componentes sin el uso de certificados digitales ni control centralizado.

En el Internet-Draft “An analysis of the applicability of blockchain to secure IP addresses allocation, delegation and bindings” [1] se deja en evidencia la aplicabilidad de esta tecnología cuando compara algunas características fundamentales que los prefijos IPs comparten con las monedas o

---

<sup>1</sup>BGP: El protocolo de puerta de enlace de frontera (Border Gateway Protocol), es un protocolo mediante el cual se intercambia información de encaminamiento entre sistemas autónomos y se garantiza una elección de rutas libres de bucles.

<sup>2</sup>Route Hijacking (secuestro de rutas): Es cuando un operador de red se hace pasar por otro, al anunciar a Internet prefijos que no está autorizado. Este anuncio indebido puede ser intencional o por error en la operación. Los paquetes se envían al lugar equivocado y pueden causar ataques de denegación de servicio (DoS) o interceptación de tráfico.

activos que encontramos en cualquier blockchain. Tanto los prefijos IP como las criptomonedas se pueden asignar sin ambigüedad a las entidades y no pueden pertenecer a dos participantes al mismo tiempo (los prefijos se asignan a los SAs<sup>3</sup> y las monedas a los usuarios o direcciones). Los prefijos se pueden delegar entre SAs, de igual modo que las criptomonedas se pueden transferir de una dirección a otra. Y por último, ambos se pueden dividir hasta cierto límite.

Como aporte secundario se buscará validar la arquitectura implementando un prototipo funcional sobre una red Ethereum<sup>4</sup> o similar, que incluya los contratos inteligentes que realizarán las operaciones necesarias para la gestión de los recursos de Internet (y un cliente para interactuar con los contratos implementados).

### **1.3 Motivación**

El enrutamiento en Internet ha sido un problema crítico desde hace mucho tiempo, y la seguridad del protocolo BGP es un problema antiguo que nunca se ha podido resolver debido a que el problema raíz se basa en la confianza que debe existir entre los sistemas autónomos (SAs). Sin embargo, por primera vez, la tecnología blockchain presenta una oportunidad para abordar este problema. Al no requerir de la confianza mutua, blockchain puede proporcionar una mayor seguridad en el enrutamiento de Internet. Por lo tanto, la motivación principal de este trabajo de tesis de maestría es utilizar la tecnología blockchain para solucionar problemas del mundo real que no sean de índole financiera, el caso de uso más común, y proponer un modelo basado en blockchain para mejorar la seguridad del protocolo BGP y prevenir los ataques de route hijacking.

Con esto, se espera contribuir a la investigación en seguridad del enrutamiento en Internet y ofrecer una solución innovadora a un problema crítico que afecta a la confiabilidad y disponibilidad de los servicios en internet en el mundo.

---

<sup>3</sup>SAs: Un Sistema Autónomo es un grupo de redes IP que poseen una política de ruteo propia e independiente. Desde Internet sólo podrá verse aquella información de ruteo que el Sistema Autónomo u organización quiera mostrar, y nada de la complejidad del ruteo interno. Internet es una gran interconexión de Sistemas Autónomos.

<sup>4</sup>Ethereum: Es una plataforma open source, descentralizada que permite la creación de acuerdos de contratos inteligentes entre pares, basada en el modelo blockchain.

# Estado del Arte

Aunque todavía existe una percepción generalizada acerca de la naturaleza descentralizada de Internet, en realidad la gestión de las funciones claves de internet (Arquitectura de direccionamiento IP, DNS<sup>1</sup> y Enrutamiento) es en parte jerárquica y centralizada si consideramos el plano de control de los sistemas, ya que los datos suelen estar descentralizados.

El objetivo propuesto es plantear un mecanismo de seguridad en el enrutamiento utilizando la tecnología blockchain, y para ello se van a analizar las últimas investigaciones que se han llevado a cabo acerca de cómo las propiedades de blockchain pueden mejorar y descentralizar la gestión de algunos de estos sistemas claves que componen Internet. Específicamente se revisa su aplicación en la asignación y delegación de recursos de la infraestructura de Internet, así como también en la seguridad de ruteo externo BGP<sup>2</sup> contra ataques de Route Hijacking<sup>3</sup> o similares.

La evolución en la gestión de los recursos que forman la infraestructura de internet ha sido investigada y documentada por la comunidad científica a lo largo del tiempo. El análisis del estado del arte acá realizado se agrupa en dos partes, primero se exploran las evoluciones que tuvieron lugar en la administración de los recursos que componen la red, y posteriormente se avanza sobre los progresos que ha tenido la seguridad en el enrutamiento dinámico global.

## 2.1 Gobernanza y gestión de información de ruteo global

El enfoque de esta sección es proporcionar una visión integral sobre la gobernanza y la gestión de información de ruteo global, componentes esenciales en la estructura actual de Internet. Se realizará una revisión histórica y contextual de las entidades que han jugado un rol crucial en la administración y seguridad de Internet: los Registros Regionales de Internet (RIRs) y los Registros de Ruteo de Internet (IRRs). Adicionalmente, se explorarán los mecanismos actuales empleados para brindar seguridad en el ruteo externo, destacando su relevancia y desafíos en el entorno de Internet actual. Con esto se busca entender cómo estas entidades y mecanismos han moldeado el panorama actual de la asignación de recursos y ruteo en Internet, y cómo pueden ser mejorados con la integración de la tecnología blockchain.

### 2.1.1 Historia de los RIRs

Desde el comienzo de la Internet, todos los dispositivos necesitaron de una dirección IP para poder conectarse entre sí, un número único que identifique el dispositivo y le permita ser localizado por los demás. Para garantizar la unicidad de las direcciones, estas se deben asignar y registrar

---

<sup>1</sup>DNS: Es un sistema de nomenclatura jerárquico descentralizado para dispositivos conectados a redes IP que permite asociar información variada con nombres de dominio asignados a cada uno de los participantes.

<sup>2</sup>BGP: El protocolo de puerta de enlace de frontera (Border Gateway Protocol), es un protocolo mediante el cual se intercambia información de encaminamiento entre sistemas autónomos y se garantiza una elección de rutas libres de bucles

<sup>3</sup>Route Hijacking (secuestro de rutas): Es cuando un operador de red se hace pasar por otro, al anunciar a Internet prefijos que no está autorizado. Este anuncio indebido puede ser intencional o por error en la operación. Los paquetes se envían al lugar equivocado y pueden causar ataques de denegación de servicio (DoS) o interceptación de tráfico.

de forma organizada. Estas formas, en que se registra la asignación de las direcciones, fueron evolucionando con el tiempo. Desde sus inicios cuando Jon Postel llevaba el registro manualmente en su libreta, que luego se formalizó en la creación del IANA<sup>4</sup>, hasta la actualidad donde a raíz del crecimiento exponencial de Internet en todo el mundo fue necesaria la creación progresiva de diferentes Registros Regionales de Internet (RIRs<sup>5</sup>) que supervisen que la asignación de los recursos en las diferentes regiones se ajusten a las necesidades de cada región [2]

Desde los inicios de Internet los sucesos más destacados que se fueron produciendo son:

- En 1969, el año en que nació ARPANET<sup>6</sup>, Steve Crocker, Jon Postel y Vint Cerf eran estudiantes de posgrado en la Universidad de California en Los Ángeles (UCLA). Los tres habían trabajado desarrollando los protocolos de la red precursora de internet.
- En diciembre de 1972, con el acuerdo de la comunidad de investigadores, Postel se convierte en la autoridad central de facto para la asignación y seguimiento de identificadores. Con el tiempo, también se convierte en el editor de RFC<sup>7</sup> y mantiene la lista oficial de nombres y direcciones de host. Realizaba un seguimiento manual de las direcciones en una libreta, al estilo de una guía telefónica.
- En marzo de 1976, el Dr. Postel, llevándose su trabajo voluntario como coordinador de números de Internet con él, se une al Instituto de Ciencias de la Información (ISI) de la Universidad del Sur de California (USC).
- En septiembre de 1981 se presentó el RFC 791 que desarrollaba las especificidades del protocolo de Internet. Este fue el comienzo formal del espacio de direcciones de Internet. La RFC 790, presentada en el mismo momento, documenta las asignaciones de las primeras direcciones IP.
- Comenzando el año 1988 se funda el IANA, mediante un contrato entre el DARPA<sup>8</sup> y el ISI. El término IANA se acuña durante la transición de ARPANET a Internet, y se menciona por primera vez en diciembre de 1988, en la RFC1083, donde Joyce Reynolds se nombra como el punto de contacto de IANA y Jon Postel es tanto “Arquitecto adjunto de Internet” y “Editor de RFC”.
- Con la expansión de Internet y teniendo en cuenta que las direcciones IP y los ASN<sup>9</sup> son

---

<sup>4</sup>Internet Assigned Numbers Authority: Es la entidad que supervisa la asignación global de direcciones IP, sistemas autónomos, servidores raíz de nombres de dominio DNS y otros recursos relativos a los protocolos de Internet. Actualmente es un departamento operado por ICANN.

<sup>5</sup>Un Registro Regional de Internet, en inglés Regional Internet Registry (RIR), es una organización que supervisa la asignación y el registro de recursos de números de Internet dentro de una región particular del mundo. Los recursos incluyen direcciones IP (tanto IPv4 como IPv6) y números de sistemas autónomos (para su uso en encaminamiento BGP)

<sup>6</sup>ARPANET fue una red de computadoras creada por encargo del Departamento de Defensa de los Estados Unidos (DOD) para utilizarla como medio de comunicación entre las diferentes instituciones académicas y estatales.

<sup>7</sup>Los Request for Comments, más conocidos por sus siglas RFC, son una serie de publicaciones del grupo de trabajo de ingeniería de internet que describen diversos aspectos del funcionamiento de Internet y otras redes de computadoras, como protocolos, procedimientos, etc. y comentarios e ideas sobre estos.

<sup>8</sup>DARPA: Es la Agencia de Proyectos de Investigación Avanzados de Defensa del Departamento de Defensa de Estados Unidos, responsable del desarrollo de nuevas tecnologías para uso militar.

<sup>9</sup>Un número de sistema autónomo (o ASN) se asigna a cada SA, el que lo identifica de manera única a sus redes



recursos finitos, aparece la necesidad de que exista un manejo neutral y efectivo de estos recursos. Con el fin de asegurar la distribución justa e igualitaria, así como para prevenir el acaparamiento, surgieron los RIRs, organizaciones encargadas de administrar los recursos a nivel regional.

- En 1992 comienza a operar en Estados Unidos el Defence Data Network NIC, que luego se transformaría en InterNic<sup>10</sup> administrando toda América y el sur de África. RIPE NCC<sup>11</sup> inició sus operaciones en abril de ese año en Ámsterdam, y su región de servicio está formada por países de Europa, Oriente Medio y partes de Asia Central.
- En 1993 se crea APNIC<sup>12</sup>, que pasaría a administrar las direcciones IP de Asia y la Región Pacífica.
- En 1997 en Estados Unidos se crea ARIN<sup>13</sup> tomando las tareas de Internic para la región de América Anglosajona, varias islas de los océanos Pacífico y Atlántico.
- Diciembre de 1998 la Universidad de California Meridional (USC) comienza la ejecución de un contrato de transferencia con la corporación ICANN<sup>14</sup>.
- En 1998 APNIC muda su sede a Australia, y se crea el ICANN, una corporación privada sin fines de lucro que se estableció principalmente para administrar la asignación global de direcciones IP, SAs<sup>15</sup>, y servidores raíz de DNS.
- En el 2002 el ICANN da la autorización final a LACNIC<sup>16</sup> para la administración de las direcciones IP de América Latina y el Caribe, quedando entonces ARIN administrando Estados Unidos, Canadá y Sud África.
- En el mes de abril del 2005 ICANN autoriza formalmente la operación de AFRINIC<sup>17</sup> el que tendrá la responsabilidad de la administración de las direcciones IP de toda África.

La relación que existe entre los RIRs y la ICANN es la siguiente: La ICANN delega los recursos de Internet a los RIR, y a su vez los RIR siguen sus políticas regionales para una posterior subdelegación de recursos a sus clientes, que incluyen ISPs y organizaciones para uso propio. En la figura 2.1 se observan algunos casos en que los RIRs ponen a disposición de registros nacionales (NIR) y locales (LIR) los bloques de direcciones más pequeños que se conceden posteriormente a los ISPs para que sean asignados a los usuarios finales.

---

dentro de Internet.

<sup>10</sup>InterNic: Internet Network Information Center

<sup>11</sup>Réseaux IP Européens Network Coordination Centre

<sup>12</sup>Asia Pacifico Network Information Center.

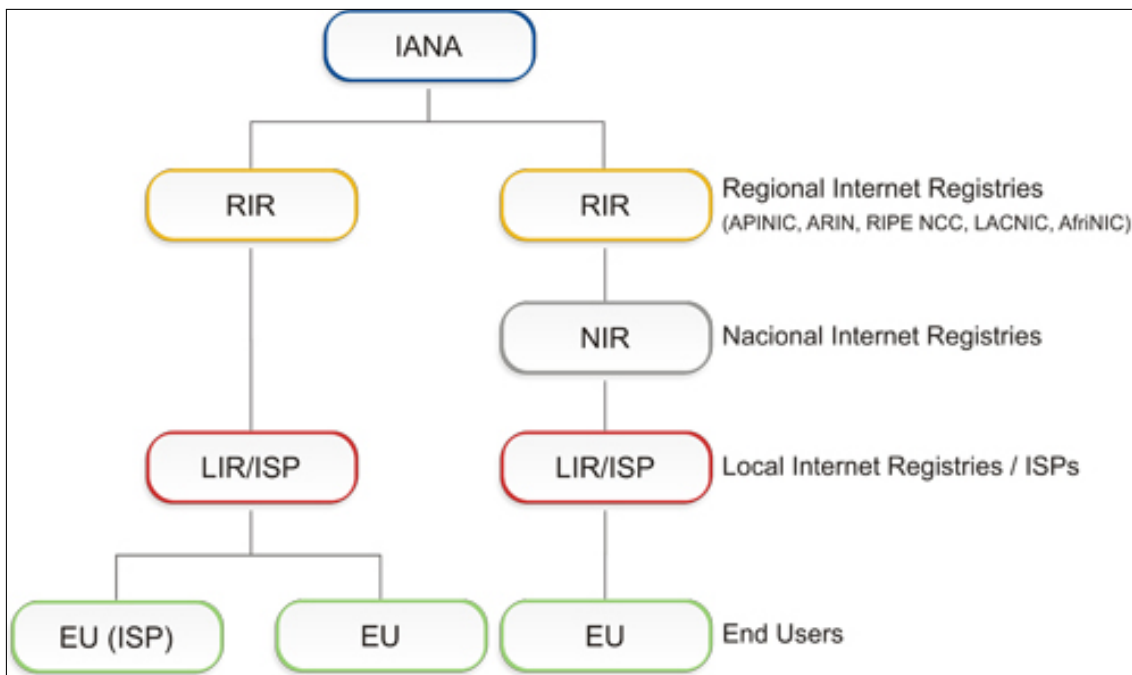
<sup>13</sup>American Registry for Internet Numbers.

<sup>14</sup>Internet Corporation for Assigned Names and Numbers

<sup>15</sup>Un Sistema Autónomo es un grupo de redes IP que poseen una política de ruteo propia e independiente. Desde Internet sólo podrá verse aquella información de ruteo que el Sistema Autónomo u organización quiera mostrar, y nada de la complejidad del ruteo interno. Internet es una gran interconexión de Sistemas Autónomos.

<sup>16</sup>LACNIC. Latin American and Caribbean Internet Address Registry.

<sup>17</sup>AFRINIC. African Network Information Centre.



**Figura 2.1:** Esquema jerárquico en la asignación de recursos de Internet [3]

Se hace aquí una distinción entre delegación (o distribución) y asignación de direcciones IP. Las direcciones son distribuidas a los NIRs e ISPs para que a su vez sean asignadas a sus usuarios finales[3].

Colectivamente, los RIR participan en la Number Resource Organization (NRO), formada como una entidad para representar sus intereses colectivos, llevar a cabo actividades conjuntas y coordinar las actividades de los RIR globalmente. [4]

### 2.1.2 Historia de los IRR

Si bien los RIRs realizan la asignación de los recursos, el protocolo BGP no impide que un administrador de red anuncie intencional o accidentalmente, un prefijo aleatorio que no le corresponde desde su SA<sup>18</sup>, independientemente de sus derechos para anunciar esa ruta. A raíz de este inconveniente, se originan los IRR<sup>19</sup> a mediados de la década del 90, los cuales representan un registro distribuido con la información de políticas de ruteo de los SAs. Esta información posibilita un mecanismo “fuera de banda” que permite a los ISPs diseñar los filtros necesarios para validar los mensajes UPDATE de BGP. Existen muchos IRR hoy en día, algunos administrados por los RIRs, como el de RIPE NCC ó el de LACNIC, y otros administrados por redes de investigación como el servicio RADb<sup>20</sup>.

<sup>18</sup>SA: Un Sistema Autónomo es un grupo de redes IP que poseen una política de ruteo propia e independiente. Desde Internet sólo podrá verse aquella información de ruteo que el Sistema Autónomo u organización quiera mostrar, y nada de la complejidad del ruteo interno. Internet es una gran interconexión de Sistemas Autónomos.

<sup>19</sup>Internet Routing Registries: El IRR proporciona un mecanismo para validar el contenido de los mensajes UPDATE de BGP o mapear un ASN de origen a una lista de prefijos.

<sup>20</sup>La base de datos de activos de enrutamiento RADb es operada por Merit Networks, una organización sin fines de lucro gobernada por las universidades públicas de Michigan <https://www.radb.net/>.

Las políticas de enrutamiento son los objetivos de ruteo del administrador del SA, que controla qué prefijos se adoptan y qué prefijos se propagan a los SAs vecinos. Estos objetivos se almacenan como registros de ruteo, que junto a los demás registros de los demás SAs, comprenden una vista global compartida de la información de la política de enrutamiento de Internet.

Los operadores de las redes comparten voluntariamente la información de su política de ruteo, la cual generalmente incluye datos como el contacto del operador, la rutas, el ASN, el enrutador y otros objetos de datos en un formato semi-estandarizado llamado RPSL<sup>21</sup>.

En el artículo “Internet routing registries, data governance, and security” [5] se analiza la carga, actualización, y exactitud de los datos de las políticas de enrutamiento que se alojan en los IRR. Los autores buscan solucionar los problemas de la seguridad en el ruteo, estudiando los incentivos de los diversos actores en el problema de gobernanza de los datos de enrutamiento. Los principales incentivos a no participar de parte de los ISP que se mencionan son: algunos operadores se muestran reacios a publicar políticas de enrutamiento porque pueden revelar sus relaciones comerciales; y hay incentivos débiles para eliminar objetos obsoletos, ya que implica trabajo y ningún beneficio inmediato para el que lo elimina.

La complejidad radica en que las políticas de ruteo se generan en forma distribuida y la información se encuentra descentralizada. El rol que cumplen los IRR hoy, es en parte, centralizar esa información para su posterior consulta. En el sitio web que coordina los IRR<sup>22</sup> se listan 25 IRRs, los cuales intercambian información entre ellos para facilitar la disponibilidad de estos registros al público. Aunque replicar las bases de datos entre los IRR mejora la disponibilidad de la información, no asegura una visión global consistente de las políticas de ruteo.

En los últimos años los investigadores han observado como la información proporcionada por los IRR no es del todo confiable y buscan estrategias para actualizar las bases del IRR desde otras fuentes de información más precisas. La tendencia actual de los IRR busca confiar en gobernanzas de datos más centralizadas y jerárquicas, basadas en los RIR, para autorizar y validar los recursos. El reciente IRR de LACNIC usa datos del repositorio WHOIS, complementado con información confiable proveniente de RPKI<sup>23</sup>, y datos generados por los operadores (AS-SET) [6].

### **2.1.3 Mecanismos actuales de seguridad en el ruteo**

El protocolo BGP es el que hace posible el ruteo en Internet, el mismo permite la comunicación entre los diferentes SAs (sistemas autónomos) mediante el intercambio o publicación de información de ruteo. BGP fue diseñado inicialmente sin tener en cuenta, de forma inherente, el aspecto de la seguridad. La misma fue dejada de lado priorizando la simplicidad y la robustez. Sin embargo, actualmente la seguridad es vital para el normal funcionamiento de la red.

En el año 2022, solo en el cuarto trimestre(Q4) existieron 9956 organizaciones que tomaron con-

---

<sup>21</sup>RPSL (Routing Policy Specification Language): Es un lenguaje de uso común por los ISP para describir sus políticas de enrutamiento.

<sup>22</sup>[www.irr.net](http://www.irr.net), el sitio es mantenido por Merit Network.

<sup>23</sup>RPKI es un marco de infraestructura de clave pública especializado, diseñado para proteger la infraestructura de enrutamiento de Internet y ayudar a los proveedores de Internet a verificar el derecho de uso de un recurso.

trol ilegalmente de rutas BGP pertenecientes a otras organizaciones [7]. Los incidentes de ruteo más comunes son: Route Hijacking, Route Leak<sup>24</sup>, PATH<sup>25</sup> o IP spoofing, y los mismos pueden ocasionar grandes pérdidas económicas para los diferentes integrantes que forman la red, además de los daños a la reputación y el robo de datos.

Algunos de los casos más difundidos por los medios son:

Pakistan Telecom - YouTube, en el 2008 [8].

Hacker Redirects Traffic From 19 Internet Providers to Steal Bitcoins [9].

Massive BGP hijack by AS12389 (Rostelecom) en Abril de 2020 [10].

Dada la gravedad de estos ataques, el IETF (Internet Engineering Task Force o Grupo de trabajo de Ingeniería de Internet) diseñó en 2012 una solución para la seguridad de ruteo entre sistemas autónomos por medio del RPKI (Resource Public Key Infrastructure, RFC 6480 [11]). Esta solución es un mecanismo que permite validar el origen de los prefijos. La información de los propietarios de los prefijos IP, los números de sistemas autónomos (ASN), y los ROA<sup>26</sup> se almacenan en un repositorio públicamente accesible.

Actualmente la seguridad de BGP se basa en una configuración cuidadosa y en la confianza entre los operadores de red, donde a través de mecanismos fuera de banda se dicen entre sí que prefijos anunciar. El trabajo conjunto, el compromiso de los operadores y de las instituciones para cumplir MANRS<sup>27</sup> encamina a disminuir gradualmente los ataques al ruteo [12].

La implementación de RPKI representa un gran progreso en seguridad de ruteo y presenta varias ventajas: es una implementación liviana, que valida el origen de la ruta y no requiere modificar los paquetes UPDATE de BGP. Desafortunadamente, el despliegue global del RPKI es más lento de lo esperado con solo el 18% del total de bloques /24 de direcciones IPv4 de los cinco RIRs [13].

Una de las razones que pueden explicar el lento despliegue de RPKI es que las Autoridades de Certificación (CA) tienen el control final de los recursos, y esto podría ser utilizado para dar de baja los prefijos de IP unilateralmente.

Este tipo de riesgos que surgen en las arquitecturas centralizadas, además de proveer poca transparencia, se presenta como potencial vector de ataque. En el paper “On the Risk of Misbehaving RPKI Authorities” [14] los autores presentan los inconvenientes de ruteo que pueden surgir cuando se cambia el modelo de amenaza, en vez de suponer que las autoridades de la PKI son confiables y la ruta está bajo ataque, discute los riesgos que aparecen cuando las autoridades no son 100% confiables (por errores de configuración, o por coerción legal por parte de actores patrocinados por estados que buscan imponer censura, control de información o vigilancia). Dado que las direccio-

---

<sup>24</sup>Route Leak (fuga de ruta): Es la propagación de anuncios de prefijos más allá de su alcance previsto. Es decir, a un operador de red con múltiples proveedores ascendentes, se le “escapa” una ruta BGP aprendida hacia otro SA.

<sup>25</sup>PATH (Ataques contra el camino): Son técnicas que modifican el AS\_PATH, como insertar en el mensaje UPDATE no solo el propio SA sino más números de SAs, u originar una ruta para determinado prefijo pero utilizando un número de SA que no es el que corresponde que origine rutas para ese prefijo.

<sup>26</sup>ROA: Del inglés Route Origin Authorizations, son objetos firmados digitalmente, que proveen una autorización explícita por parte del poseedor de un prefijo permitiendo a un ASN originar rutas de ese rango.

<sup>27</sup>MANRS: Normas Mutuamente Acordadas para Enrutamiento en Internet ( <https://www.manrs.org> )

nes IP son un activo crítico de los ISP, a estos les gustaría tener un mayor grado de control sobre ellas, pero sin perder la seguridad de estar certificado por una CA, es decir, poder equilibrado entre los usuarios y la CA [15].

Además de los inconvenientes de la centralización, el despliegue de RPKI a un 100% de la red, no representa una cura para todos los males, ya que esta seguirá siendo vulnerable a ciertos tipos de ataques, como el “one hop hijacking”<sup>28</sup> [16]. RPKI solo provee validación de origen, por esto la validación de camino se ha vuelto una característica deseable. Una PKI es el requisito previo de otras propuestas más avanzadas que se han propuesto para estandarizar la validación del camino para BGP, algunas de ellas son BGPSEC[17], S-BGP[18], soBGP[19].

Estas propuestas requieren modificar la estructura de los mensajes de BGP (coexistiendo con la versión estándar), y la utilización de criptografía en línea, incrementando significativamente los recursos necesarios en los equipos de ruteo.

A pesar de que la especificación de BGPSEC está completa [RFC 8205], [RFC 8207], [RFC 8208], [RFC 8210] y existen implementaciones de código abierto disponibles, todavía no hay implementaciones comerciales de parte de los proveedores de enrutadores [20]. Esto hace suponer que la implementación de la validación de camino en el mundo real puede resultar limitada en el futuro previsible.

Existen investigaciones que indican que dadas las políticas de enrutamiento más frecuentes de los operadores, un despliegue parcial de BGPSEC, solo puede proporcionar mejoras marginales a la seguridad sobre lo que ya es posible con RPKI desplegado al 100%, debido a que RPKI aborda una gran parte de la superficie del problema [21].

Actualmente los desafíos están en proponer un sistema que resuelva estos inconvenientes sin centralizar el poder y la gobernanza de la información.

## **2.2 Uso de la tecnología blockchain para validar la información de ruteo**

La naturaleza distribuida de las políticas de ruteo nos permite pensar un modelo basado en blockchain para resolver la problemática de la seguridad de enrutamiento. En la tesis “Certificados Digitales: De una Arquitectura Jerárquica y Centralizada a Una Distribuida y Descentralizada” [22] se plantea una reingeniería y rediseño de la Arquitectura de Certificados Digitales hacia un modelo descentralizado y distribuido utilizando blockchain<sup>29</sup> o la cadena de bloques como mecanismo de validación, sincronización y administración del ciclo de vida de esta arquitectura. Los autores plantean que, para lograr el objetivo de la descentralización, desde un punto de vista técnico, se requiere la colaboración de distintas entidades que asuman aunadas ese rol de organismo central multifacético.

En un principio la aplicación de blockchain (con Bitcoin) era exclusiva de los sistemas financieros.

<sup>28</sup>One hop hijacking o path-shortening attack: El ataque de “acortamiento de ruta” se produce cuando un atacante anuncia una ruta falsa corta a un prefijo que termina en el SA de origen autorizado.

<sup>29</sup>Blockchain: Es una estructura de datos segura y compartida globalmente, que mantiene un registro único, consensado e inmutable de transacciones.

Posteriormente se fueron explorando otros campos de aplicación para su utilización, como la identidad digital, autenticación y la seguridad. Al incorporar blockchain en las soluciones de seguridad de enrutamiento, se logra descartar la estructura 100% jerárquica en la que se basa hoy en día la infraestructura RPKI. Mediante contratos inteligentes se establece el tipo de relación que existirá entre las autoridades (IANA, RIR) y los usuarios finales o ISPs, logrando una mayor transparencia y confiabilidad en la delegación de recursos de internet, eliminando los riesgos que existen con una entidad central de gestión.

La tecnología blockchain y los contratos inteligentes facilitan la creación de una fuente de datos inmutable y descentralizada con la información necesaria para agilizar el mecanismo por el cual los routers realicen la validación, tanto del origen como del AS\_PATH<sup>30</sup>. De esta manera, nos proporcionan actualmente los medios para intercambiar la información necesaria que permita verificar la validez de los mensajes enviados entre los vecinos BGP de forma confiable.

En el artículo InBlock “A Distributed Autonomous Organization for Internet Address Management” [23] los autores proponen una gestión descentralizada de direcciones IPv6 mediante un conjunto de contratos inteligentes sobre la blockchain Ethereum, que implementan todas las funciones necesarias para la gestión de un conjunto global de direcciones sin ninguna intervención humana. En esta investigación se proponen a limitar la acción humana solamente a la definición inicial de algunas reglas de asignación antes de desplegar el contrato en la blockchain, principalmente definir el tamaño del bloque de prueba (por ej.: /20). Luego mediante un mecanismo de tarifas, dado que el recurso de IPv6 es prácticamente ilimitado, se pretende lograr la conservación y la equidad en la distribución de los recursos.

Otra perspectiva es suponer que con el capital financiero que poseen las grandes corporaciones tecnológicas, quizás las tarifas no sean suficientes para prevenir la monopolización de los recursos por parte de las mismas. Para lograr la equidad en la distribución de recursos tenemos que definir pautas, que hoy en día surgen de foros y reuniones técnicas abiertas a toda la comunidad de internet, que se llevan a cabo mediante el esfuerzo y la participación de los RIRs en pos del desarrollo de internet en cada región. Respecto a estas pautas o políticas las opciones son; o se definen reglas bien específicas para poder eliminar la intervención humana del proceso de asignación (solo se pueden cambiar previo al despliegue del contrato), o se definen reglas más generales, y flexibles, manteniendo una mínima participación humana que interactúe con el contrato inteligente conducido por las políticas de asignación de cada RIR.

De cualquier forma se requiere de cierta participación humana, aunque la tecnología blockchain se beneficia significativamente de la inmutabilidad, se necesita un cierto grado de mutabilidad para corregir errores y posibles mejoras o modificaciones en el proceso a desarrollar. Las actualizaciones o upgrades resuelven esta aparente contradicción al proporcionar un mecanismo de actualización simple y robusto para los contratos inteligentes. Estos upgrades pueden ser controlados por cualquier tipo de gobernanza, ya sea una billetera multi-sig o una DAO compleja[24] [25].

---

<sup>30</sup>AS\_PATH: Atributo de BGP que identifica los sistemas autónomos (AS) a través de los cuales ha pasado el mensaje UPDATE. Enumera en orden inverso los AS atravesados por un prefijo, con el último AS colocado al principio de la lista. El propósito principal de AS\_PATH es proporcionar prevención de loops durante el enrutamiento entre AS.

Al diseñar la solución de seguridad en ruteo con blockchain surgen algunos interrogantes relacionados al manejo de las direcciones IP. En los orígenes de Internet solo existían las direcciones IPv4, que tienen una extensión de 32 bits, lo que permitía contar con aproximadamente 4.3 mil millones de direcciones. Con la explosión mundial de usuarios a mediados de la década de 1990, se comenzó a impulsar el uso del protocolo IPv6 con una extensión de 128 bits, multiplicándose exponencialmente la cantidad de direcciones IP disponibles. Teniendo en cuenta que solo una ínfima parte del espacio total posible de IPv6 ha sido asignado a los RIRs, que las direcciones IPv4 se encuentran prácticamente agotadas, y que el avance de IoT acelerará el despliegue de IPv6 (y el fin de ciclo de vida de IPv4), tiene sentido enfocar el diseño de la solución blockchain para seguridad de ruteo exclusivamente en IPv6, aunque si se requiere también se podría implementar en redes IPv4. [26] [27]

Otro aspecto a tener en cuenta es el bloque mínimo que se recomienda publicar por BGP. En IPv4 es un /24, y en IPv6 es un /48. Los RIR no asignan prefijos IPv6 más largos a /48 y la mayoría de los operadores suelen filtrar los prefijos mayores a estos. Estos detalles o especificaciones se deben tener en cuenta a la hora de diseñar, desarrollar y desplegar los contratos inteligentes. Se debe considerar que estos valores pueden variar en el futuro [28], y estas modificaciones que se pueden dar en las políticas se deben analizar profundamente, para determinar el nivel de detalle o abstracción que debe manejar el contrato inteligente.

### 2.2.1 Antecedentes

Se han realizado investigaciones previas sobre el tema, y las mismas tienen enfoques muy diversos, incluso en los tipos de blockchain y algoritmos de consenso a utilizar. Algunos autores se inclinan a soluciones con blockchain a medida o permissionadas, las más destacadas son:

- Dmitri Tsumak, en el paper “Securing BGP using blockchain technology” [29], plantea una solución que utiliza la plataforma blockchain Quorum.
- En el paper “IPchain: Securing IP Prefix Allocation and Delegation with Blockchain” [15] los autores plantean el desarrollo de una blockchain que haga uso del algoritmo de consenso Proof of Stake.
- En el paper “Routechain: Towards blockchain-based secure and efficient bgp routing” [30], los autores plantean un diseño con Clique, una modificación de PoA<sup>31</sup> (Proof of Authority). La solución involucra a todos los SAs divididos en subgrupos, y cada subgrupo comparte una blockchain. Los subgrupos se construyen en función de su proximidad geográfica para reducir los retrasos de propagación, mejorar la escalabilidad y lograr un consenso más rápido.

Si se profundiza en los detalles de estas soluciones a medida (propias o basadas en blockchain privadas) podemos observar que diseñar un protocolo de consenso distribuido entre los SAs, es más sencillo que diseñar uno para una moneda seudónima como Bitcoin, o una

---

<sup>31</sup>Proof of Authority (PoA) es una familia de algoritmos de consenso para blockchain permissionada cuya relevancia se debe a aumentos de rendimiento con respecto a los algoritmos BFT típicos; esto resulta debido a intercambios de mensajes más ligeros. PoA se propuso originalmente como parte del ecosistema Ethereum para redes privadas y se implementó en los clientes Aura y Clique.

red como Ethereum. Esto se debe a que las entidades que tienen que lograr un consenso son bien conocidas de antemano. A pesar de esto algunos autores prefieren hacer uso de una infraestructura más genérica, como Ethereum.

Las siguientes investigaciones proponen soluciones que se montan sobre Ethereum:

- En el paper “BGPcoin: Blockchain-Based Internet Number Resource Authority and BGP Security Solution” [31], Qianqian Xing propone un prototipo funcional realizando una demostración sobre la red de pruebas oficial de Ethereum Ropsten.
- En el paper “A Distributed Autonomous Organization for Internet Address Management” [23] los autores proponen una DAO, un conjunto de contratos inteligentes sobre Ethereum.

La ventaja de las soluciones basadas en Ethereum es que aprovechan el diseño en capas, y logran independizar el desarrollo de la solución basada en los contratos inteligentes, de los avances en la blockchain utilizada. Esto permite la evolución de la blockchain sin afectar el contrato inteligente desplegado [23].

Este enfoque parece más acertado que las blockchains a medida, ya que con la transición a Proof of Stake y la implementación de la Beacon Chain se ha logrado acercarse al objetivo de tener una red escalable y segura con adopción masiva. Además, la implementación de soluciones como “zero-knowledge (ZK) rollups” y “optimistic rollups” han contribuido a mejorar aún más la escalabilidad de la red [32] [33].

Una de las investigaciones más destacadas relacionadas con este enfoque es la implementación del proceso llamado Sharding, que divide la carga de la red en sub-blockchains llamadas shards. Esto significa que en lugar de que todos los nodos tengan que almacenar y procesar todas las transacciones de la red, cada nodo sólo tiene que hacerlo con las transacciones de su propio shard. De esta manera, se busca mejorar la eficiencia y escalabilidad de la red al reducir la cantidad de información que cada nodo tiene que procesar. Para llevar a cabo esta fragmentación de la red, se realiza una actualización en varias fases. La primera de ellas es la llamada fase cero, que incluye la implementación de la Beacon Chain. Esta cadena sirve como capa de consenso y contiene toda la lógica necesaria para mantener los shards seguros y sincronizados. De esta manera, se garantiza que cada shard tenga una copia consistente y actualizada del estado de la red.[34] [35]

Las acciones para lograr consenso en un gran número de participantes requieren de un gran esfuerzo y mucho dinero en investigación y desarrollo, por lo que parece muy razonable montar la solución sobre una red existente (Ej.: Ethereum). Al delegar la responsabilidad del consenso a una red como Ethereum y desacoplar las funciones de la solución de seguridad en el enrutamiento del diseño propio de la blockchain, se obtienen las siguientes ventajas:

Independencia con el diseño, desarrollo, evolución de la blockchain y del algoritmo de consenso.

Rápido despliegue y un ahorro en el coste de mantenimiento.

Se logra una mayor descentralización lógica, que si se confía en blockchains no públicas para el despliegue de la solución.



## **Parte II**

# **MARCO TEÓRICO**

# Recursos de internet

En este capítulo se presentan los recursos de Internet, sistemas autónomos(SAs) y números IPv4 e IPv6, que hacen posible la conectividad y el intercambio de información a través de la red. Además se explica como es el sistema de gobernanza de los mismos, lo cual es fundamental para comprender cómo es hoy en día el funcionamiento de Internet.

## 3.1 Sistemas Autónomos

Un Sistema Autónomo representa un conjunto de redes bajo la misma administración, tanto desde el punto de vista técnico como legal, aunque puede ser gestionada por más de un operador de red y tener por debajo otras redes dependientes administradas por otras organizaciones. Esta administración se encarga de definir una única política de ruteo administrativamente unificada, o sea un dominio de ruteo independiente. Se debe definir claramente qué áreas desea conectar a Internet, qué dispositivos serán visibles desde la red de redes, con qué protocolos se comunicará cada dispositivo de la red, qué bloques de direcciones IP podrán ver las demás organizaciones, entre otros parámetros de la red. Los Sistemas Autónomos normalmente utilizan uno o más bloques de direcciones IP que le han sido asignados por un RIR (Regional Internet Registry) o NIR (National Internet Registry), es decir, utilizan bloques de direcciones IP propios y se vinculan a más de un sistema autónomo utilizando el protocolo de enrutamiento dinámico de Internet: BGP (Border Gateway Protocol).

Los Sistemas Autónomos se identifican con un número de AS o ASN que los identifica de forma unívoca en el sistema de enrutamiento BGP. Este número de AS es un número de 16 bits o de 32 bits, y debe ser distribuido por un RIR o un NIR. El pool central (ó sea, el conjunto centralizado) de ASNs lo tiene IANA, y al igual que con las direcciones IP, ésta los asigna a los diferentes RIRs para que puedan ser reasignados en cada región.

Los números de sistemas autónomos de 16 bits fueron definidos en la RFC 1930[36] y se utilizará para su identificación números enteros del 0 al 65535. Igualmente, los números de sistemas autónomos de 32 bits fueron definidos por la RFC 4893 y se utilizarán para su identificación números enteros del 0 al 4294967295. Utilizando en ambos casos la representación textual del valor decimal “asplain” definida en el RFC 5396. Los números de sistemas autónomos se encuentran clasificados según sus usos y cómo han sido asignados. El rango comprendido entre el número 64496 y el número 64511 (de 16 bits) y el rango comprendido entre los números 65536 y 65551 (de 32 bits), han sido definidos para uso en documentación y ejemplos. La tabla muestra cómo es la subasignación que se realiza sobre los diferentes ASNs (3.1)

AS Number	Reason for Reservation	Reference
0	Reserved by <a href="#">[RFC7607]</a>	<a href="#">[RFC7607]</a>
112	Used by the AS112 project to sink misdirected DNS queries; see <a href="#">[RFC7534]</a>	<a href="#">[RFC7534]</a>
23456	AS_TRANS; reserved by <a href="#">[RFC6793]</a>	<a href="#">[RFC6793]</a>
64496-64511	For documentation and sample code; reserved by <a href="#">[RFC5398]</a>	<a href="#">[RFC5398]</a>
64512-65534	For private use; reserved by <a href="#">[RFC6996]</a>	<a href="#">[RFC6996]</a>
65535	Reserved by <a href="#">[RFC7300]</a>	<a href="#">[RFC7300]</a>
65536-65551	For documentation and sample code; reserved by <a href="#">[RFC5398]</a>	<a href="#">[RFC5398]</a>
4200000000-4294967294	For private use; reserved by <a href="#">[RFC6996]</a>	<a href="#">[RFC6996]</a>
4294967295	Reserved by <a href="#">[RFC7300]</a>	<a href="#">[RFC7300]</a>

Figura 3.1: ASNs de propósito especial [37]

Hoy en día Internet es una gran red de redes, formada por la operación conjunta de miles de SAs de diferentes instituciones con funciones distintas: proveedores de acceso, proveedores de contenido, universidades, empresas usuarias de Internet, organismos gubernamentales, etc. Normalmente, cada AS tiene un mapa de todo Internet y sabe cuáles bloques de direcciones IP están vinculados a cada uno de los otros AS en la red, así como el mejor camino para llegar a cada destino en Internet. Este mapa es construido de forma dinámica por BGP.

Para un proveedor de Internet, no ser un sistema autónomo y depender de su proveedor de tránsito IP es una enorme limitación. Un proveedor que es un sistema autónomo puede lograr redundancia con múltiples proveedores de tránsito, y mejorar su interconexión con el resto de la Internet. También puede cambiar de proveedor de tránsito sin tener que preocuparse por el tamaño del bloque que ofrece el nuevo proveedor ni por reenumerar los dispositivos de su red. Además de realizar acuerdos de intercambio de tráfico (peering) con otros sistemas autónomos, puede participar en puntos de intercambio de tráfico de Internet<sup>1</sup>(IXP).

### 3.2 Números IPv4 e IPv6

El protocolo de Internet o IP (Internet Protocol) es la tecnología o conjunto de reglas de comunicación, que permite que todas las diferentes redes, que forman la red global, se pueden comunicar entre sí. El Protocolo de Internet especifica que cada dispositivo de la red global necesita un identificador numérico único, conocido como dirección IP, que permita encontrarlo sin posibilidad de error o confusión. Para que los paquetes de datos se envíen correctamente de una red a otra, desde el dispositivo de origen hasta su destino final, es fundamental que las direcciones IP no se repitan, y que no existan dudas sobre el organismo que tiene permitido utilizar esa numeración. Esta es la razón por la cual este recurso se administra globalmente de forma controlada y organizada. A su vez los proveedores de Internet y otras redes también deben distribuir las direcciones IP a los usuarios de forma planificada, garantizando su unicidad, documentando todo, preservando los recursos y permitiendo que los protocolos de enrutamiento las utilicen de forma optimizada.

<sup>1</sup>IXPs (Internet eXchanges Points): A través de un IXP, se produce el intercambio de tráfico entre las redes de diversas entidades (operadores, proveedores de acceso, organismos de gobierno, entidades académicas) con el objetivo de eficientizar el ruteo de Internet, mejorando la calidad en las comunicaciones y reduciendo los costos de interconexión.

Hoy en día, en Internet se utilizan dos versiones del protocolo IP. La versión legada, que se utiliza desde 1983 y sigue siendo la más usada a nivel mundial, llamada IPv4. Y la versión actual, cuyo uso en la red está aumentando rápidamente: IPv6.

IPv4 especifica un espacio de 32 bits para las direcciones, lo que significa que hay  $2^{32}$  direcciones posibles. Esto hace que IPv4 pueda direccionar poco más de 4 mil millones de dispositivos en la red. Casi la totalidad de las direcciones IPv4 ya se han distribuido a alguna empresa o institución. En toda América Latina, un proveedor de Internet u otro tipo de institución ya no puede obtener bloques adicionales de direcciones IPv4. Solo las entidades que nunca antes obtuvieron bloques IPv4 pueden obtener estos bloques y, aún así, pueden obtener como máximo un bloque /22. La reserva de direcciones IPv4 destinada a nuevos entrantes en la región de América Latina se está agotando rápidamente. Muy pronto no será posible obtener bloques de direcciones IPv4, ni siquiera para los nuevos entrantes.

En cambio con IPv6, las direcciones IP dejan de ser un recurso escaso y pasan a ser un recurso abundante. IPv6 reserva un espacio de 128 bits para las direcciones. En la distribución de direcciones IPv6 normalmente no se consideran las direcciones individuales, sino grandes bloques de direcciones con los cuales se pueden numerar una gran cantidad de dispositivos en grandes redes. Al planificar las redes, la preservación de los recursos pasa a ser menos importante que una buena documentación, organización y el uso optimizado de los protocolos de enrutamiento. El uso de IPv6 está creciendo rápidamente en la Internet global y es uno de los factores que permitirá la continuidad de su crecimiento, la inclusión digital, Internet de las cosas y otras innovaciones. Es fundamental que todos los proveedores de Internet implementen IPv6 en sus redes. Actualmente, esto se hace de forma paralela al uso de IPv4, es decir, los usuarios y dispositivos deben utilizar ambos tipos de direcciones. Sin embargo, en un futuro próximo solo se utilizará IPv6.

### **3.3 Administración global de los recursos**

Las direcciones IP y los ASN son administrados por organizaciones que operan de forma jerárquica. Quien gestiona globalmente los bloques de direcciones IP y los ASN es la IANA (Internet Assigned Numbers Authority), es decir, la Autoridad de Números Asignados en Internet. La IANA es operada por una organización sin fines de lucro llamada PTI (Public Technical Identifiers), filial de ICANN (The Internet Corporation for Assigned Names and Numbers) [38].

La IANA es el stock central de bloques IP y ASN. Distribuye grandes bloques de direcciones IP y ASN a organizaciones regionales llamados Registros Regionales de Internet. Existen cinco RIRs y cada uno de ellos es responsable de administrar los bloques de direcciones IP y los ASN en una región determinada:

ARIN: Estados Unidos, Canadá y algunas islas del Caribe

RIPE NCC: Europa principalmente, pero también incluye parte de Asia

APNIC: Asia y el Pacífico (Oceanía)

AFRINIC: África

## LACNIC: América Latina y Caribe

En algunos países hay organizaciones nacionales responsables de gestionar las direcciones IP y los ASN. En este contexto, se denominan NIR (National Internet Registries) o Registros Nacionales de Internet. Los cinco RIRs componen una organización llamada NRO (Number Resource Organization), a través de la cual coordinan acciones conjuntas y divulgan estadísticas sobre la distribución de los recursos. Los requerimientos para obtener recursos propios de internet son bastante homogéneos entre los diferentes RIRs y están basados en algunos puntos de la RFC-1930[36]. Estos requerimientos para LACNIC[39] son:

- Las políticas (reglas) actuales sólo permiten distribuir bloques IPv4 a los nuevos entrantes.
- La solicitud de IPv4 ingresará a la lista de espera una vez que tenga recursos IPv6 asignados.
- Debe estar legalmente constituido dentro del área de cobertura de LACNIC.
- Podrá solicitar bloques IPv4 desde un /24 hasta un /22, debiendo justificar el uso inmediato del 25% de las direcciones y el uso del 50% hasta en un año.
- Devolver al upstream provider todo el espacio IPv4 recibido de ellos en un plazo no mayor a 12 meses.
- Junto con la asignación del bloque IPv4 también debe solicitar la asignación de un bloque de direcciones IPv6.

Si los recursos no se utilizan (las IPs no aparecen en la tabla de enrutamiento global) o si las reglas para su uso no se respetan (incumplimiento de las políticas de LACNIC ó del acuerdo de servicios), la distribución puede ser revocada. Otros motivos son que la organización ya no exista o que las direcciones sean devueltas voluntariamente por una organización que ya no las utiliza. En todos estos casos, los recursos vuelven a la reserva de LACNIC y se pueden distribuir a otra organización de acuerdo con las reglas vigentes, organizaciones que no posean recursos IPV4 siendo la asignación máxima un bloque /22.

# Conceptos de BGP

El protocolo de puerta de enlace de frontera o BGP (del inglés Border Gateway Protocol) es uno de los protocolos de enrutamiento más importantes en el ámbito de Internet. BGP es utilizado por los proveedores de servicios de Internet (ISP) para intercambiar información de enrutamiento entre ellos y para tomar decisiones sobre el mejor camino para enrutar el tráfico de Internet. En este capítulo, se presentarán los conceptos fundamentales de BGP, comenzando con una introducción al protocolo y sus características, seguido por una descripción de los principales tipos de mensaje BGP, los atributos que se utilizan para tomar decisiones de enrutamiento, y finalizando con los criterios de selección de rutas utilizados por los routers BGP.

## 4.1 Introducción a BGP

Los protocolos de ruteo se clasifican en dos tipos: los protocolos de ruteo interno (iGP o interior Gateway Protocol) que se utilizan en redes LAN, como RIP y OSPF; y los protocolos de ruteo externo (eGP o exterior Gateway Protocol), que se utilizan en redes tipo WAN. En esta sección se mencionarán los protocolos de ruteo externo empleados en el pasado, que dieron lugar al nacimiento de BGP v4. En los años 80, cuando nació la pila de protocolos TCP/IP, el tamaño de Internet era considerablemente menor de lo que es hoy en día, pero de todos modos era necesario un protocolo que permitiera compartir la información de ruteo. En un principio se utilizó un protocolo sencillo llamado GGP (Gateway to Gateway Protocol) documentado en la RFC 823, este calculaba en forma distribuida la mejor ruta basado en la mínima cantidad de saltos. Cuando la red comenzó a crecer y GGP no podía escalar lo suficiente, se propone el protocolo EGP (Exterior Gateway Protocol) estandarizado en el año 1984 mediante la RFC 827, y luego actualizado por la RFC 904. En EGP es donde se introduce la noción de Sistema Autónomo, con el fin de cubrir las carencias de GGP se divide la red en grupos que participan en el enrutamiento presentándose con su propio número de sistema autónomo (ASN).

A medida que la red InterNet crecía y se interconectaban una mayor cantidad de SAs, la topología de la red comenzó a transformarse de una topología árbol a una topología mesh, y se comenzaron a divisar inconvenientes en la performance. La RFC 827 indicaba que la topología del set de SAs debía tener una estructura tipo árbol para poder prevenir los loops. Fue en ese entonces que resultó necesario crear un nuevo protocolo de ruteo que se adapte a las necesidades de crecimiento de la red. En 1989 debido a las limitaciones que tenía EGP, se introduce el protocolo BGP, estandarizado en la RFC 1105[40].

La primera versión del estándar definió muchos de los conceptos que hoy son utilizados por el actual protocolo BGP, como formato de los mensajes y mecanismo de comunicación entre pares. La versión actual de BGP es la 4 (BGP4) y se encuentra documentada en la RFC 1771 y RFC 4271 desde el año 1995 y 2006 respectivamente. BGP 4 es el protocolo de ruteo exterior que utiliza InterNet desde el año 1995. Las principales ventajas de la versión 4 sobre sus predecesoras fueron la posibilidad de manejar classless inter domain routing (CIDR o enrutamiento entre dominios

sin clases) y agregación con el fin de disminuir el tamaño de las tablas de ruteo. La agregación de rutas al reducir la cantidad de prefijos que se mantendrán y anunciarán reduce los requerimientos de hardware (CPU y memoria) de los routers, mejorando la escalabilidad y obteniendo más estabilidad en la red.

## 4.2 Características de BGP

BGP (Border Gateway Protocol ) es un protocolo de ruteo externo(EGP) utilizado para comunicar a los SAs que componen Internet. Un SA es un grupo de redes IP que poseen una política de ruteo propia e independiente y se identifican dentro de internet según su número de sistema autónomo (ASN). Desde Internet sólo podrá verse aquella información de ruteo que el SA u organización quiera anunciar, ocultando la complejidad del ruteo interno. La forma en que se lleva la información de rutas dentro del SA no es visible desde Internet, y se suele realizar con un protocolo de ruteo interno o IGP.

BGP no requiere conocimiento de toda la topología de la red y se clasifica como un protocolo de tipo vector de ruta (Path Vector Protocol). Esto significa que haciendo uso del AS\_PATH (un atributo BGP que enumera los SAs que se encuentran a lo largo de la ruta), tiene en cuenta el camino que hacen los paquetes para llegar a determinado destino y puede gestionar información de rutas dinámicas garantizando un camino libre de loops.

Si bien la métrica de BGP se basa en la cantidad de SAs que debe atravesar el paquete para llegar a la red destino, conocer los SAs que están involucrados en una ruta permite que un nodo tome decisiones sobre la ruta anunciada utilizando criterios que no sean simplemente el menor costo. BGP se clasifica como un protocolo dinámico que puede enrutar el tráfico, es decir, seleccionar la ruta, en función de las políticas de ruteo y/o las condiciones de la red.

Las políticas son implementadas en cada router y permiten tomar decisiones diferentes a las que podrían haberse aprendido por el protocolo. Mediante las políticas se puede manipular el ruteo y se pueden tomar decisiones acerca de dónde o hacia dónde transmitir el tráfico, por ejemplo, no enviar tráfico a través de una red u organización que no se considera confiable. Las políticas son implementadas a través de filtros, aceptando, rechazando o modificando rutas informadas por los vecinos de modo de manipular el flujo de los datos.

Para que un router pueda intercambiar información con otros, primero debe establecer una sesión con uno o más vecinos, a través de la cual intercambiarán mensajes. Dicha sesión debe ser configurada explícitamente en ambos extremos y permite mantener un link virtual entre 2 dispositivos, el cual permitirá no solo el intercambio de información, sino el monitoreo del estado de la red [41]. Para esto BGP emplea el puerto 179 de TCP, aprovecha el servicio de transporte confiable y elimina la necesidad de implementar por si mismo el manejo de la fragmentación de los datos, retransmisión, acuse de recibo, secuenciación, además del esquema de autenticación de TCP . Esto representa una gran diferencia respecto de otros protocolos de ruteo, lo cuales corren sobre IP o UDP.

### 4.3 Principales tipos de mensaje BGP

Una vez que se establece la sesión TCP, ambos extremos envían un mensaje llamado OPEN con información sobre la configuración y las capacidades de cada extremo, para negociar los parámetros que caractericen a esa sesión. Si la negociación tiene éxito el router BGP logra establecer una vecindad (o relación de peering) con el router del otro extremo. Posteriormente se envían mensajes de mantenimiento de la conexión (KEEPALIVE) regulares para mantener la integridad de la sesión, y comienza a enviar una copia de sus propias tablas y recibir la correspondiente información de su par a través de mensajes de actualización de rutas (mensaje UPDATE), en la medida en que las políticas configuradas para este par lo permitan. Seguidamente, el enrutador enviará solo mensajes keepalive y actualizaciones incrementales si hay algún cambio en la tabla de enrutamiento. Una sesión de BGP se cierra si se pierden tres keepalives consecutivos y no se reciben mensajes UPDATE. En caso de detectarse una caída sobre la sesión configurada, se podrá inferir que el par está presentando inconvenientes, lo que generará que deban actualizarse las tablas de ruteo para dejar de enviar información a través de él.

Es importante acotar que BGP actualiza a sus pares solo con las rutas que utiliza. En otras palabras, solo los mejores caminos se anuncian a sus pares. Cuando cambia la mejor ruta, se anuncia la nueva ruta, lo que permite a los compañeros saber si deben reemplazar la mejor ruta anterior por la nueva mejor ruta.

### 4.4 Principales atributos de BGP

BGP es un protocolo de ruteo que tiene la capacidad de tomar decisiones basándose en diferentes métricas las cuales son descritas por un conjunto de atributos. Estos atributos son parámetros preestablecidos que viajan de un router a otro junto con la información sobre los prefijos dentro del paquete UPDATE. Los mismos son manipulados por los administradores de redes y son tenidos en cuenta por la política de ruteo al momento de elegir la mejor ruta.

Los atributos principales son: Weight, Local Preference, Origin, AS\_PATH, Next\_Hop y Community.

Se desarrolla a continuación la explicación de los atributos AS\_PATH y Local Preference, que son los referidos en el desarrollo y validación de la solución de seguridad en el enrutamiento utilizando tecnología blockchain.

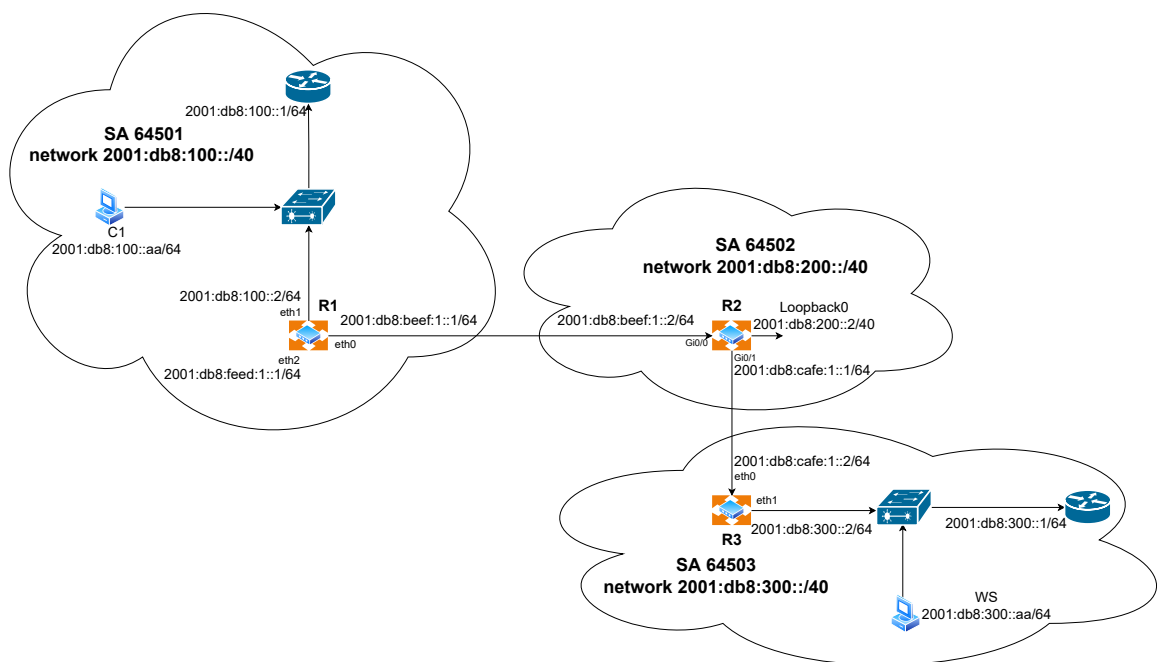
- AS\_PATH: Es un atributo Well-Known/Mandatory, esto significa que el mismo debe estar presente en forma obligatoria en el paquete UPDATE y debe ser reconocido por todas las implementaciones de BGP. Es obligatorio debido a que es el atributo clave para la elección del mejor camino, ya que guarda la secuencia de SAs que se atravesaron para llegar al AS del destino.

Cuando un anuncio de ruta pasa a través de un sistema autónomo, el número del SA es añadido a una lista ordenada de números de SA que el anuncio de ruta ha atravesado. La figura 4.1 muestra la situación en la cual el SA 64503 origina la ruta a 2001:db8:300::/40,



y anuncia la misma al SA 64502 con el atributo de AS\_PATH igual a 64503. El SA 64502 hará un anuncio de regreso a SA 64503 con el atributo AS\_PATH 64502,64503, y también hará un anuncio hacia SA 64501, también con el atributo AS\_PATH 64502,64503.

El SA 64503 rechazará la ruta 2001:db8:300::/40 cuando descubra su propio ASN en el AS\_PATH, de esta forma se utiliza el atributo para detectar loops en las rutas. En cambio el SA 64501 aprenderá la ruta y la instalará en la tabla de ruteo, dado que hasta ese momento tiene la lista AS\_PATH mas corta.



**Figura 4.1:** Topología de red IPv6

En el siguiente ejemplo se puede ver la salida del comando "show ip bgp", donde se puede observar el atributo AS\_PATH al final de cada fila:

- **Local Preference:** Cuanto mayor es el valor de este atributo, mayor es el grado de preferencia entre rutas a un mismo destino. Es un atributo local al sistema autónomo que no se propaga por eBGP, solo por iBGP. El atributo local preference asume por defecto el valor 100 y para configurarle otro valor se hace a través del comando "route-map". Se utiliza principalmente dentro de un AS para determinar la ruta principal hacia un determinado destino. El router BGP que recibe varias rutas hacia un destino específico elige la ruta con mayor preferencia local como la ruta principal.

Este atributo se tiene en cuenta antes que el atributo AS\_PATH al momento de seleccionar la mejor ruta hacia un destino y es el que se decidió utilizar en este desarrollo, para poder otorgarle más prioridad a los prefijos que se validen en la blockchain.

## 4.5 Criterios de selección de rutas

El proceso de decisión de encaminamiento de BGP respecto a otros procesos de decisión es más complejo, ya que no solo verifica la dirección de red de destino sino que también toma decisiones en base a los atributos de cada una de las rutas aprendidas.

Cuando un router BGP recibe un paquete para su encaminamiento, en caso de contar con más de una ruta, se seleccionará de acuerdo a la longitud del prefijo, eligiendo la menor de ellas (prefijo más largo) utilizando el valor de la máscara de red definida en la tabla de ruteo. Siempre son preferibles prefijos más largos antes que cortos en el reenvío de un paquete.

En caso de seguir teniendo varias alternativas para seleccionar, analizará el atributo LOCAL-PREF para dicha entrada y seleccionará la ruta que posea el número mayor. Si aún tuviera más de una ruta a dicho destino, analizará el atributo AS-PATH (lista de sistemas autónomos que debe atravesar) y seleccionará aquella ruta que posea un camino más corto. Para finalizar el proceso, si aún hubiera que tomar una decisión respecto a qué ruta utilizar, se seleccionará como ruta con mayor preferencia a aquella que haya sido aprendida a través de iGP y por último aquella que posea un atributo Multi-Exit-Discriminator (MED) más bajo. Una vez seleccionada la ruta, el paquete será conmutado al próximo salto de la red para que el proceso continúe hasta alcanzar el destino.

# Seguridad de ruteo en Internet

En la actualidad, existen varios riesgos de seguridad que pueden comprometer la seguridad del ruteo en Internet, como el secuestro de rutas, la fuga de rutas, y otros tipos de ataques malintencionados. Para abordar estos desafíos, se han desarrollado varias soluciones y normas de seguridad, como RPKI, BGPSEC, ASPA y MANRS. En este capítulo, se abordan en detalle estos riesgos de seguridad, además de los beneficios y limitaciones de las soluciones de seguridad existentes para proteger el ruteo en Internet.

## 5.1 Secuestro de rutas - Route Hijacking

BGP es el protocolo que se utiliza en Internet para la comunicación entre los SAs, permitiendo que la información de ruteo pueda fluir entre ellos. En los primeros días de la red Internet, la red era pequeña y gracias a una gran cadena de confianza que se establecía entre los operadores de las redes, el ruteo global se lleva a cabo de la mejor manera posible.

La confianza en Internet radica en que cada organización anuncie sólo sus propios prefijos, o los prefijos de las organizaciones a las que le da tránsito, y esto no está garantizado inherentemente por BGP.

Cuando se reciben rutas, se está afectando al tráfico saliente de la organización. Si las rutas que se reciben son incorrectas, los routers estarán encaminando los paquetes en forma errónea. En forma análoga, cuando se anuncian rutas, se logra afectar el tráfico entrante. Si las rutas que se anuncian a otros SAs no pertenecen a quien las anuncia, el tráfico que no es para la organización será atraído indebidamente hacia la misma.[42]

Se llama "secuestro de ruta" (o route hijacking) a la acción que realiza un SA al anunciar a Internet prefijos que no le corresponde anunciar, por no estar autorizado a hacerlo, ya sea porque no le pertenece o porque no es parte de los sistemas autónomos a los que le da tránsito. Este anuncio indebido puede ser intencional o por error en la operación.

Entre los motivos por los cuales se realizan los secuestros de rutas se encuentran los errores involuntarios de los operadores, los ataques de denegación de servicios (DoS), ataques "man in the middle" (MITM) y manipulación de rutas con fines económicos relacionados a robo o minería de criptomonedas.

### 5.1.1 Antecedentes de secuestros de rutas

#### 5.1.1.1 2008 - Denegación de servicio, Caso Pakistan - Youtube

Previo al incidente el AS36561 anunciaba la red 208.65.152.0/22, ambos pertenecientes a YouTube. El AS36561 también anunciaba otros prefijos, pero no están involucrados en el evento.

El domingo 24 de febrero de 2008, Pakistan Telecom (AS17557), ante una orden del gobierno de bloquear el acceso a YouTube, inició un anuncio no autorizado del prefijo de red 208.65.153.0/24.

Uno de los proveedores upstream de Pakistan Telecom, PCCW Global (AS3491) envió este anuncio al resto de Internet. Los enrutadores de todo el mundo reciben el anuncio y el tráfico de YouTube se redirige a Pakistán, lo que resultó en un secuestro de ruta generando un bloqueo de YouTube a nivel global durante aproximadamente dos horas.

Luego el AS36561(YouTube) comienza a anunciar 208.65.153.0/24. Con dos prefijos idénticos en el sistema de enrutamiento, las reglas de la política BGP, como preferir el AS\_PATH más corto, determinan que ruta se elige. Esto significa que AS17557 (Pakistan Telecom) sigue atrayendo parte del tráfico de YouTube.

Finalmente, el AS36561(YouTube) anuncia los prefijos 208.65.153.128/25 y 208.65.153.0/25. Debido a la regla de coincidencia de longitud máxima de prefijo (LPM), cada enrutador que recibió estos anuncios comenzó a enviar nuevamente el tráfico a YouTube [43].

Si en el momento que se produjo este secuestro de ruta PCCW Global hubiera tenido la infraestructura para implementar la validación de origen (validar el SA que origina los anuncios), podemos afirmar que el secuestro de ruta no hubiera sido posible.

#### **5.1.1.2 2014 - Secuestro de BGP para obtener beneficios de criptomonedas**

Entre febrero y mayo de 2014, el equipo de investigación de Dell SecureWorks Counter Threat Unit(CTU) descubrió una entidad desconocida que secuestraba repetidamente el tráfico destinado a ciertas redes pertenecientes a Amazon, Digital Ocean, OVH y otras grandes empresas de alojamiento. Modificaron el tráfico, publicando prefijos más específicos. En total se documentaron 51 redes comprometidas de 19 ISPs diferentes.

Para obtener un flujo de ingresos más confiable en comparación con la minería en solitario, la mayoría de los mineros de Bitcoin hoy minan a través de grupos de minería conocidos como pools. Esta opción es posible gracias al protocolo de minería de código abierto Stratum, utilizado actualmente por casi todos los pools de minería. Los mineros comienzan el proceso de minería contactando a un pool server, que envía información al minero, rastrea el trabajo del mismo y paga las recompensas en consecuencia.

El secuestrador redirigió las conexiones de los mineros de criptomonedas a un pool de minería controlado por los secuestradores. Una vez conectados a esta dirección IP, los mineros continuaron recibiendo tareas pero ya no recibieron las recompensas por su esfuerzos. El atacante recolectó las ganancias de los mineros, ganando un estimado de USD 83.000 en poco más de cuatro meses [9].

## **5.2 Fuga de ruta - Route Leak**

Si bien la “fuga de ruta” era un fenómeno conocido que causaba interrupciones significativas en el enrutamiento de Internet, previo a la publicación de la RFC 7908[44], era un término vago que se utilizaba de diversas formas por diferentes investigadores. Recién en el año 2016, la RFC proporcionó una definición técnica del problema:

”Una fuga de ruta es la propagación de anuncios de enrutamiento más allá de su alcance previsto.

Es decir, un anuncio de un sistema autónomo (AS) de una ruta BGP aprendida a otro AS infringe las políticas previstas del receptor, el remitente y/o uno de los AS predecesores a lo largo del AS-PATH”.

El alcance previsto generalmente se define mediante un conjunto de políticas de redistribución/filtrado local distribuidas entre los AS involucrados. A menudo, estas políticas previstas se definen en términos de la relación comercial de peering entre AS (por ejemplo cliente, proveedor de tránsito ó peer).

Un ejemplo de una fuga de ruta es cuando a un cliente multi-homed se le ”escapa” un anuncio de un proveedor de tránsito ascendente (ISP 1) a otro (ISP 2), y de esta forma el cliente se convierte efectivamente en un proveedor de tránsito. Esto abre la puerta para un ataque de man in the middle (MITM), o un ataque de denegación de servicio (DoS) autoprovocado.

Ninguna de las soluciones de seguridad mencionadas hasta el momento protege contra este tipo de ataque, simplemente porque BGP no tiene la capacidad de señalar relaciones como cliente-proveedor. Esta capacidad se puede obtener extendiendo el protocolo BGP al implementar la RFC 9234, que se titula ”Prevención de fugas de rutas y detección de roles utilizando mensajes UPDATE y OPEN.

Pero mientras tanto, hoy en día se pueden prevenir muchas fugas de ruta manteniendo los filtros actualizados para los anuncios de los clientes. Esta práctica está incorporada en una de las acciones requeridas en las Normas mutuamente acordadas para la seguridad del enrutamiento (MANRS).

### **5.2.1 Clasificación de Fugas de Ruta**

Además, la RFC 7908 enumera diferentes tipos de fugas de ruta en función de sucesos observados en la red Internet.

- Type 1: Hairpin Turn with Full Prefix: Esta es una fuga de ruta que ocurre cuando un prefijo se redirige a través de Internet y vuelve a la red privada de la organización.
- Type 2: Lateral ISP-ISP-ISP Leak: Esta es una fuga de ruta que ocurre cuando el tráfico se redirige a través de múltiples proveedores de servicios de Internet (ISP) en lugar de ir directamente a su destino final.
- Type 3: Leak of Transit-Provider Prefixes to Peer: Esta es una fuga de ruta que ocurre cuando los prefijos de un proveedor de tránsito se filtran a un peer.
- Type 4: Leak of Peer Prefixes to Transit Provider: Esta es una fuga de ruta que ocurre cuando los prefijos de un peer se filtran a un proveedor de tránsito.
- Type 5: Prefix Re-origination with Data Path to Legitimate Origin: Esta es una fuga de ruta que ocurre cuando un prefijo es re-origenado, es decir, se redirige a través de Internet y luego vuelve a su origen legítimo.
- Type 6: Accidental Leak of Internal Prefixes and More-Specific Prefixes: Esta es una fuga de ruta que ocurre cuando los prefijos internos o más específicos de una organización se

filtran accidentalmente a Internet.

Es importante tener en cuenta que, en general, las fugas de ruta pueden ser causadas por una variedad de factores, incluyendo problemas de hardware, errores de configuración, ataques cibernéticos, ingeniería social y fallos en el software.

## **5.2.2 Propuesta para solucionar los problemas de fugas de ruta: RFC 9234.**

### **5.2.2.1 Introducción**

La RFC 9234 ("Route Leak Prevention and Detection Using Roles in UPDATE and OPEN Messages"), extiende el mensaje BGP OPEN para manejar roles según la relación que existe entre los SAs. Esto permite establecer un acuerdo en la relación de peering en cada eBGP sesión entre sistemas autónomos, con el fin de hacer cumplir la configuración en ambos lados (cliente, proveedor, peer, etc). A continuación, automáticamente las rutas propagadas se marcan según la relación acordada, permitiendo tanto la prevención como la detección de fugas de ruta; reemplazando el accionar o configuración del operador por un método in-band.

Este método utiliza un nuevo parámetro de configuración, BGP Role, que es negociado utilizando una "capacidad de rol BGP" en el mensaje OPEN (RFC5492). Un SA en eBGP puede requerir el uso de esta capacidad y confirmar los roles involucrados con el peer vecino para que la sesión BGP se establezca satisfactoriamente.

Un atributo de ruta BGP transitivo opcional, denominado "Only to Customer - OTC" (Solo para el cliente), evita que los AS creen fugas y detecta fugas creadas por los AS que se encuentran en el camino o AS-PATH.

### **5.2.2.2 Relaciones entre peers**

El documento o RFC, define diferentes roles para los peers en una conexión BGP, para establecer restricciones en la propagación de rutas. Estos roles incluyen Proveedor, Cliente, Servidor de ruta (Route Server, RS), Route Server Client (RS-Client) y Peer. Cada rol tiene reglas específicas para la propagación de rutas, que se describen en la siguiente lista:

- Proveedor (Provider): puede propagar cualquier ruta disponible a un Cliente.
- Cliente (Client): puede propagar cualquier ruta aprendida de un Cliente o que se origine localmente a un Proveedor. Todas las demás rutas no deben ser propagadas.
- Servidor de ruta (Route Server, RS): puede propagar cualquier ruta disponible a un RS-Cliente.
- Route Server Client (RS-Client): puede propagar cualquier ruta aprendida de un Cliente o que se origine localmente a un RS. Todas las demás rutas no deben ser propagadas.
- Peer: puede propagar cualquier ruta aprendida de un Cliente o que se origine localmente a un peer. Todas las demás rutas no deben ser propagadas.

Los términos aquí definidos se utilizan para representar restricciones en la propagación de rutas

BGP, y no representan relaciones comerciales basadas sobre acuerdos de pago.

La relación entre dos peers se considera "normal" si el AS local tiene uno de los roles anteriores (en el orden en que se muestran) y la relación de peering correspondiente con el AS remoto es Proveedor a cliente, Cliente a proveedor, RS-a-RS-Cliente, RS-Cliente a RS o Peer-to-Peer (es decir, pares laterales), respectivamente. Si el AS local tiene más de un rol de emparejamiento con el AS remoto, tal relación de emparejamiento se considera "compleja".

La violación de las reglas de propagación de rutas mencionadas anteriormente puede resultar en fugas de ruta. La aplicación automática de estas normas debería reducir significativamente las fugas de ruta que de otro modo podrían ocurrir debido a errores de configuración manual. Además, la RFC 9234 introduce un atributo de ruta BGP opcional llamado "Only to Customer (OTC)", que se utiliza para identificar todas las rutas en el AS que se han recibido de un Peer, un Proveedor o un RS.

### 5.2.2.3 Corrección de roles

Es importante explicar cómo se deben manejar las discrepancias de roles en una conexión BGP. Si se recibe una capacidad de rol BGP del AS remoto y también se envía una, los roles deben corresponder a las relaciones especificadas en la tabla 5.1.

Rol de AS local	Rol de AS remoto
Proveedor	Cliente
Cliente	Proveedor
RS	Cliente RS
Cliente RS	RS
Par	Par

Tabla 5.1: Pares de roles permitidos

Si los roles no corresponden, el peer BGP debe rechazar la conexión mediante la notificación de discrepancia de roles. Para la compatibilidad con versiones anteriores, si se envía la capacidad de rol BGP pero no se recibe una, el peer BGP debe ignorar la ausencia de la capacidad de rol BGP y continuar con el establecimiento de la sesión. Los operadores también pueden optar por aplicar un "modo estricto" en el que se requiere la recepción de una capacidad de rol BGP del AS remoto. En este caso, si se envía la capacidad de rol BGP pero no se recibe una, se rechaza la conexión utilizando la notificación de discrepancia de roles. Si se reciben múltiples capacidades de rol BGP y no todas tienen el mismo valor, el hablante BGP debe rechazar la conexión utilizando la notificación de discrepancia de roles.

### 5.2.2.4 Atributo BGP solo para el cliente (OTC)

El atributo OTC es un atributo de ruta transitivo opcional del mensaje de UPDATE con código de tipo de atributo 35 y una longitud de 4 octetos. El propósito de este atributo es hacer cumplir que

una vez que se envía una ruta a un cliente, un peer o un RS-Client, posteriormente irá únicamente a los Clientes.

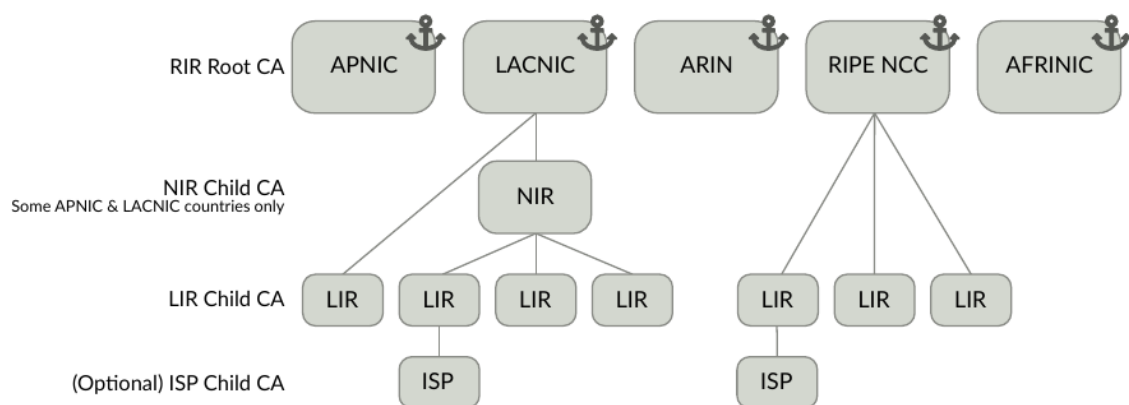
## 5.3 RPKI

### 5.3.1 Introducción a RPKI

Con el crecimiento actual de la red, la cadena de confianza entre operadores se volvió más débil, incrementando la cantidad de ataques y secuestros de ruta. Las medidas de protección para estos ataques eran frágiles e inseguras, cartas de puño y letra, coordinación mediante email y llamados telefónicos entre operadores para acordar los bloques IP a anunciar y poder configurar filtros a medida. Otra manera de verificar que entidades pueden utilizar los recursos de internet son los servicios de IRR, aunque el mismo es de gran ayuda, no provee información firmada que garantice la autenticación del derecho de uso. Esto abrió camino a la implementación de un marco de seguridad, llamado RPKI, para ayudar a los operadores de red a tomar decisiones de enrutamiento más informadas y seguras.

RPKI es un sistema impulsado por la comunidad, en el que participan los cinco Registros Regionales de Internet (RIR), desarrolladores software de código abierto y proveedores de equipos de enrutamiento, que combina el modelo de asignación de recursos a través de los RIRs, con el uso de certificados basados en el estándar X.509 (con una extensión para soportar IPs y ASNs - RFC 3779). Es importante destacar que es un estándar de IETF, según los RFC 6480 – 6493.

El mismo define una infraestructura de clave pública que crea una cadena de certificados de recursos que sigue la misma estructura que la jerarquía de asignación de recursos de números de Internet, con la excepción del IANA. La IANA no opera ninguna autoridad de certificación raíz, en cambio, cada uno de los cinco RIRs ejecutan una CA raíz con un ancla de confianza (trust anchor<sup>1</sup>) del que se deriva una cadena de confianza para los recursos que administra cada uno [45].



**Figura 5.1:** Cadena de confianza de RPKI [45]

<sup>1</sup>En la arquitectura X.509, un certificado raíz sería el ancla de confianza del que toda la cadena de confianza deriva. En el contexto de la arquitectura de RPKI el "TAL" (Trust Anchor Locator) es un archivo que contiene la información necesaria para que una herramienta de validación de RPKI pueda acceder a la localización del repositorio y comenzar el proceso de validación.



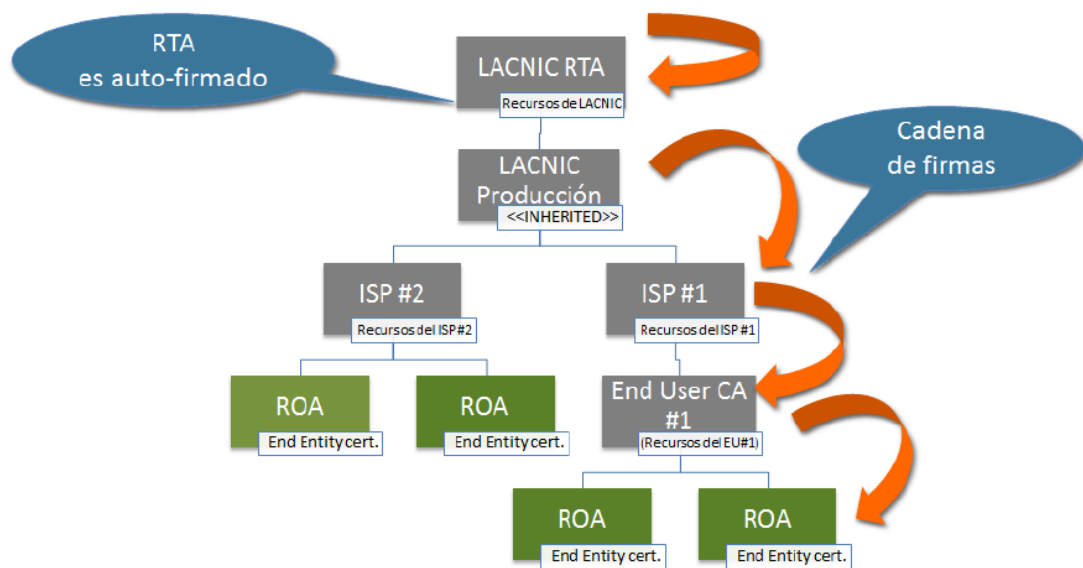


Figura 5.2: Cadena de confianza de LACNIC - RPKI [42]

Los dueños de los recursos, tanto de bloques de direcciones IP como de ASNs, pueden obtener un certificado digital a través del RIR, que valida que los recursos son de la organización. La obtención de estos certificados habilita a los operadores de red a crear declaraciones criptográficamente validables sobre los anuncios de ruta que autorizan a realizarse con los prefijos que tienen. Estas declaraciones se denominan ROA (Autorizaciones de Origen de Ruta) y además de establecer que SA está autorizado para originar un determinado prefijo IP, permite determinar la longitud máxima del prefijo que el AS está autorizado a anunciar. Es importante aclarar que el titular del certificado puede autorizar a cualquier AS a originar su prefijo, no solo sus propios sistemas autónomos.

Los ROAs se almacenan en un repositorio públicamente accesible donde otros operadores de red pueden descargar y validar estas declaraciones y tomar decisiones de enrutamiento basadas en ellas. Este proceso se conoce como validación del origen de la ruta (ROV).

### 5.3.2 Validación del origen de la ruta

Cualquier operador es libre de obtener esa lista de ROA de los RIR, y usarla para que sus enrutadores tomen decisiones basadas en el ROA. Un anuncio en particular generalmente tendrá uno de tres estados posibles:

Válido: El AS de origen y el largo máximo permitido coinciden con la información del ROA.

Invalído: La información del ROA no coincide. O bien difieren en el ASN de origen o es más específico de lo que permite la longitud máxima de prefijo que se establece en el ROA. Si un operador utiliza RPKI de forma estricta, es muy probable que este anuncio no se instale en sus routers.

No encontrado: No hay un ROA para el prefijo dado.

A continuación detallamos los comandos utilizados para chequear un ROA:

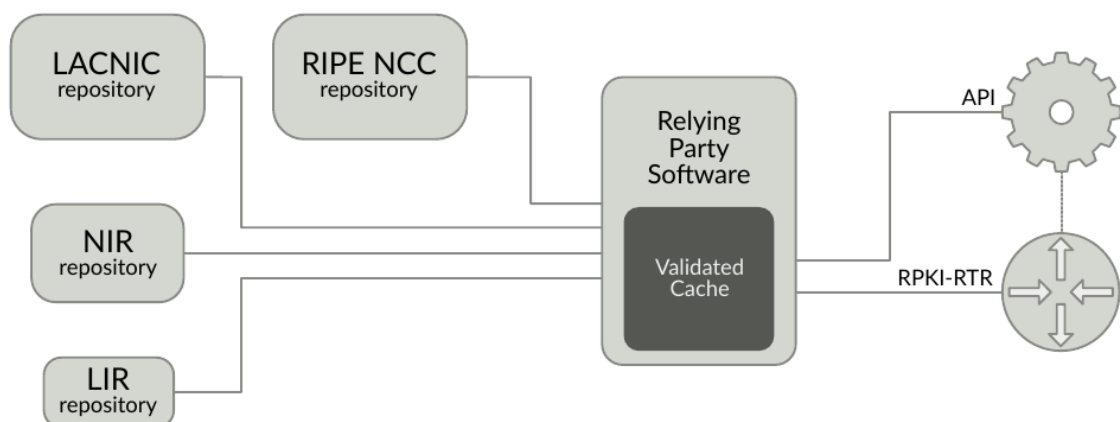
```
$ whois -h whois.bgpmon.net 190.124.224.0/20
```

```
% This is the BGPmon.net whois Service
% You can use this whois gateway to retrieve information
% about an IP address or prefix
% We support both IPv4 and IPv6 address.
%
% For more information visit:
% https://portal.bgpmon.net/bgpmonapi.php

Prefix:                190.124.224.0/20
Prefix description:    UNIVERSIDAD_NACIONAL_DE_SAN_JUAN_Argentina
Country code:         AR
Origin AS:             28107
Origin AS Name:        UNIVERSIDAD NACIONAL DE SAN JUAN, AR
RPKI status:           ROA validation successful
First seen:            2017-08-31
Last seen:             2021-02-23
Seen by #peers:        39
```

```
$ whois -h whois.bgpmon.net --roa 28107 190.124.224.0/20
```

```
0 - Valid
-----
ROA Details
-----
Origin ASN:            AS28107
Not valid Before:     2019-05-03 03:00:00
Not valid After:      2021-05-03 03:00:00 Expires in 64d10h34m22s
Trust Anchor:         repository.lacnic.net
Prefixes:             190.124.224.0/20 (max length /20)
190.124.240.0/22 (max length /22)
2801:0:2f0::/48 (max length /48)
```



**Figura 5.3:** Recuperación y validación de datos RPKI [45]

En el sistema RPKI hay un componente llamado Relying Party (RP), comúnmente llamado vali-

gador RPKI. EL mismo realiza las siguientes tareas:

- Recopila objetos ROA de los repositorios RPKI distribuidos de cada ancla de confianza(TA) utilizando los protocolos rsync o RRDP (RFC 8182 [46]).
- Valida la firma de cada entrada para construir una caché validada.
- Finalmente proporciona a los routers compatibles con RPKI los orígenes de ruta validados(VRP) a través del protocolo RPKI-RTR (RFC 8210 [47]).

Los routers arman un base de datos con la información recibida de la caché y los operadores pueden usar ese atributo para definir filtros. En pocas palabras, un validador RPKI desempeña un papel de interfaz en un sistema RPKI que proporciona un registro de validación de ruta(RV) a los enrutadores conectados a Internet. Los enrutadores actualizan periódicamente la base de datos de registros de RV de los validadores RPKI.

Para que la validación del origen de la ruta tenga éxito los operadores deben eliminar todos los anuncios de BGP que están marcados como no válidos. Antes de tomar este paso, las organizaciones deben aceptar inicialmente anuncios no válidos y darles una preferencia menor, para poder analizar los efectos y así evitar resultados no deseados.

Si bien al implementar RPKI gran parte de los casos mencionados sobre secuestros de ruta podrían prevenirse, este no valida el camino que recorren los anuncios, solo evita los ataques que devienen del origen de la ruta. Por este motivo, la validación de los ROA resulta efectiva para los robos de rutas debido a errores de configuración, y no para los ataques de hijacking maliciosos o fugas de ruta.

A medida que la validación del origen de la ruta se implemente en más y más lugares, se van preparando los cimientos para implementar la validación de ruta. Hay varios esfuerzos para ofrecer validación de ruta fuera de banda. La autorización de proveedor de sistema autónomo (ASPA), es la que actualmente tiene mayor tracción dentro del IETF, y se describe en estos borradores: A Profile for Autonomous System Provider Authorization [48] y BGP AS\_PATH Verification Based on Resource Public Key Infrastructure (RPKI) Autonomous System Provider Authorization (ASPA) Objects[49].

## 5.4 MANRS

Las Normas comúnmente acordadas para la seguridad del enrutamiento (MANRS) son una iniciativa comunitaria organizada por Internet Society, que tiene como objetivo mejorar la seguridad y la resiliencia del sistema de enrutamiento global.

Todos los que operan una red son corresponsables de la estabilidad del ruteo global, es una responsabilidad compartida, y se requiere de la colaboración de todos los participantes para lograr los objetivos de seguridad perseguidos.

MANRS provee un marco de seguridad que describe cuatro acciones simples pero concretas que los operadores de red deben tomar. Las dos primeras acciones (filtrado y anti-spoofing) son me-

mejoras operativas que apuntan a eliminar los problemas de enrutamiento y los ataques comunes, mientras que las dos segundas (coordinación y validación global) son procedimientos que nos acercan a la adopción universal, mejoran la coordinación y disminuyen la probabilidad de incidentes futuros[50].

Las 4 acciones se detallan a continuación:

**Filtrado:** La acción de filtrado evita la propagación de información de enrutamiento incorrecto.

**Anti-spoofing:** Prevención del tráfico con direcciones IP de origen falsificadas.

**Coordinación:** Mantener la información de contacto actualizada y accesible para facilitar la comunicación y coordinación operativa global entre los operadores de red.

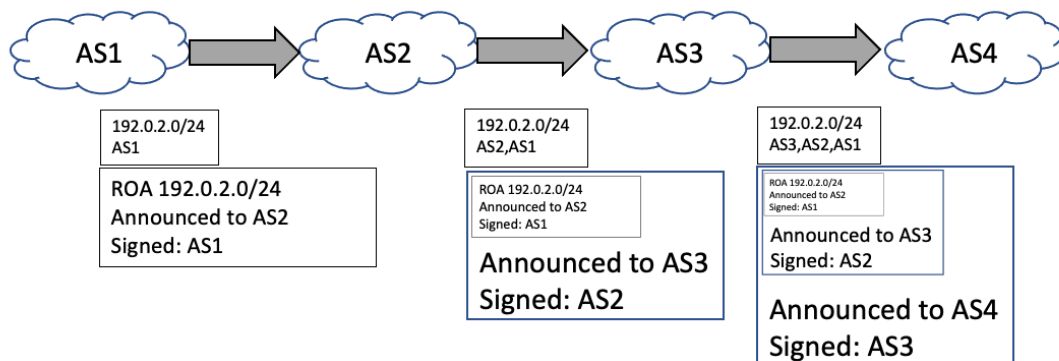
**Validación global:** Facilitar la validación de la información del enrutamiento a nivel mundial.

Con la implementación de estas acciones por parte de todos los participantes se logra una reducción de las amenazas de enrutamiento más comunes. El filtrado correcto basado en la información de los IRR, es lo que mantiene el ruteo global hoy en día, evitando los hijacks y los route leaks ocasionados por configuraciones incorrectas.

## 5.5 BGPSEC

BGPsec es una extensión del protocolo Border Gateway Protocol que se diseñó para resolver el problema de la validación del AS\_PATH, aunque podría considerarse un nuevo protocolo debido a las diferencias que presenta en el formato de los mensajes UPDATE definidos en BGP.

En BGPsec cuando un SA le envía un anuncio al vecino, el SA firma hacia donde se dirige el anuncio (SA destino), e incluye esta firma en el propio anuncio. Si el SA1 envía un prefijo al SA2 el anuncio contiene un mensaje firmado por SA1 que declara que el destino es el SA2, y así sucesivamente. De esta forma se genera una secuencia de firmas que garantiza el camino recorrido por el anuncio, protegiendo criptográficamente el AS\_PATH ocasionando que el atacante no puede modificar el mismo[17].



**Figura 5.4:** Recuperación y validación de datos RPKI [51]

La validación de los ROA en RPKI es realizada en el Relying Party (RP) y la encriptación se realiza fuera de línea. El principal inconveniente de BGPsec es que la encriptación se realiza online y esto produce una sobrecarga computacional muy elevada en los enrutadores BGP[52].

Otro inconveniente de BGPsec es el ataque de degradación de protocolo, donde el atacante decide no utilizar BGPsec y obligar al vecino a hacer un downgrade al antiguo BGP4, anulando los beneficios de la secuencia de firmas. Finalmente, el ataque se produce cuando la víctima descarta una ruta segura prefiriendo una ruta falsa obtenida a través de BGP heredado o no seguro, simplemente porque la ruta falsa es más corta[53].

Desde septiembre del año 2017, BGPsec es finalmente un estándar, definido en la RFC8205. Esta RFC explica como BGPsec reemplaza el clásico atributo de BGP AS\_PATH con un nuevo atributo llamado BGPsec\_Path. Si bien existen investigaciones acerca de como disminuir los costos de implementación de BGPsec y hacerlo asequible a los ISPs más pequeños, detectar o resolver de algún modo los ataques de downgrade es más complejo ocasionando que la implementación en el mundo real parezca imposible en un futuro previsible.

Con una tasa de adopción del 100% BGPsec resolvería de manera exitosa los ataques de hijacking maliciosos, es decir, no permitiendo las sesiones de peer con BGP heredado. Pero en un ecosistema donde las adopciones suelen ser lentas si afectan las finanzas de las entidades adoptantes, BGPsec no es una posibilidad, y se requiere de soluciones más económicas y/o que produzcan impacto con tasas de adopción más bajas.

## 5.6 ASPA - Autonomous System Path Authorization

ASPA propone definir roles al establecer la sesión BGP (provider, customer, peer, internal) que permitan detectar AS\_PATH inválidos o malformados.

El procedimiento en sí utiliza una base de datos compartida y firmada de relaciones cliente-proveedor (C2P), que se construye con un nuevo objeto RPKI: Autorización de proveedor de sistema autónomo (ASPA). Es liviano y rápido de implementar, detectando AS\_PATHs inválidos incluso durante la adopción temprana e incremental. Los ASPA son objetos firmados digitalmente que permiten verificar que un dueño de un AS Cliente(CAS) ha autorizado a un AS Proveedor(PAS) en particular para propagar los anuncios de ruta BGP IPv4 o IPv6 pertenecientes al cliente a los upstreams o pares del AS Proveedor [54].

```
ASPA := {  
    customer_asn – signer  
    provider_asn – authorized to send routes to  
                  upper providers or peers  
    AFI – IPv4 or IPv6  
}
```

Figura 5.5: ASPA - Formato de objetos firmados [55]

### 5.6.1 Procedimiento de verificación cliente-proveedor

El validador RPKI ó Relying Party(RP) debe tener acceso a un caché local del conjunto completo de objetos ASPAs criptográficamente validados al realizar el procedimiento de verificación cliente-proveedor.

El procedimiento de verificación cliente-proveedor toma como argumentos de entrada a SA1, SA2 y AFI, donde SA1 es el SA del cliente, SA2 es el SA candidato a verificar como proveedor y AFI es el indicador de familia de direcciones (IPv4/IPv6).

Este procedimiento devuelve uno de tres resultados: "Válido", "No válido", y "Desconocido". Si el conjunto de proveedores candidatos está vacío, el procedimiento finaliza con el resultado "Desconocido". Si AS2 está incluido en los proveedores candidatos, el procedimiento finaliza con un resultado de "Válido". De lo contrario, el procedimiento finaliza con un resultado de "No válido".

### 5.6.2 Procedimiento de verificación de AS\_PATH

El atributo AS\_PATH identifica los sistemas autónomos a través de los cuales ha pasado un mensaje UPDATE. La manera en que se verifica ese camino al utilizar ASPA difiere si la ruta se recibe de un cliente, o sea upstream, fluye hacia arriba; o se recibe de un proveedor, fluye hacia abajo o downstream[49].

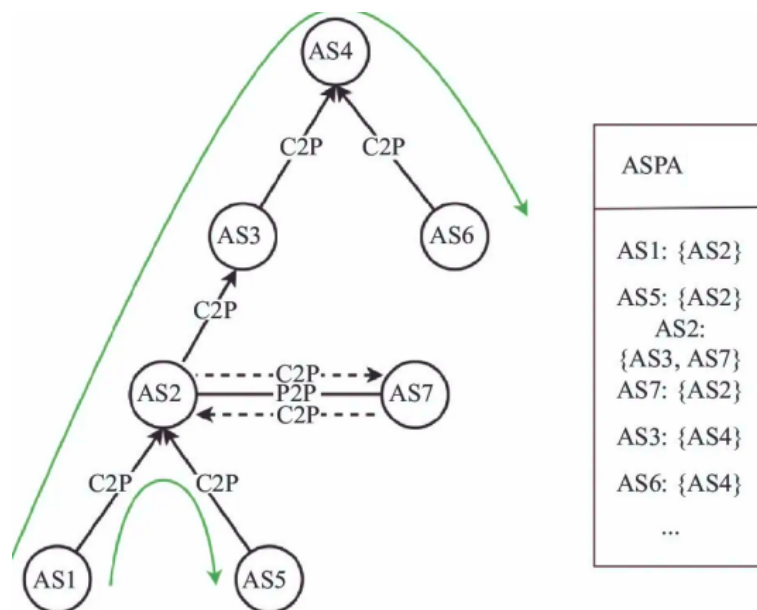


Figura 5.6: ASPA - Procedimiento de verificación [56]

### Camino UPSTREAM

Cuando se recibe una ruta de un cliente, un peer o por un RS<sup>2</sup> en un IXP, cada par de SAs con-

<sup>2</sup>Route Server: Los servidores de ruta se utilizan normalmente en redes de medios de acceso compartido, como los puntos de intercambio de Internet (IXP), para facilitar la interconexión simplificada entre varios enrutadores de Internet. No envía paquetes, únicamente maneja la lógica de ruteo, separando el control del ruteo, del reenvío de paquetes (forwarding)

secutivos debe ser igual (as path prepending<sup>3</sup>) o pertenecer a una relación cliente-proveedor o de tránsito mutuo.

Si hay otros tipos de relaciones, significa que ocurrió un route leak, o el atributo AS\_PATH estaba mal formado.

En la imagen 5.6 se observa el recuadro con los objetos firmados digitalmente que permiten verificar las relaciones cliente-proveedor (c2p) que existen entre AS1, AS2, AS3 y AS4 (AS1:AS2 , AS2:AS3,AS7 y AS3:AS4).

### **Camino DOWNSTREAM**

Cuando la ruta se recibe del proveedor, puede tener fragmentos ascendentes y descendentes, donde un fragmento descendente sigue a un fragmento ascendente. Si la ruta difiere de esta regla, por ejemplo, el fragmento descendente es seguido por el fragmento ascendente, significa que ocurrió un route leak o que el atributo AS\_PATH se formó incorrectamente. Este caso particular de route leak se observaría en la figura 5.6 si AS5 tuviera una relación cliente-proveedor con AS7 ( AS5: AS7 ) y los anuncios originados por AS1 llegaran al AS7 debido a una fuga en el AS5.

El primer par desigual de SAs consecutivos que tiene un resultado "No válido" del procedimiento de verificación cliente-proveedor indica el final del fragmento ascendente. Todos los pares de SAs posteriores invertidos deben ser iguales (as path prepending) o tener un resultado válido de la verificación de cliente-proveedor, o verificar una relación de tránsito mutuo utilizando objetos ASPA.

Una ruta recibida de un RS(Route Server) en un IXP tiene mucho en común con la ruta recibida de un proveedor. Una ruta válida del RS contiene un fragmento de flujo ascendente y puede contener un fragmento de flujo descendente que contiene el SA del IXP. La ambigüedad se crea mediante IXPs transparentes que, de forma predeterminada, no agreguen su SA en el AS\_PATH. Por lo tanto las rutas de un RS pueden procesarse de la misma manera que las rutas de los proveedores.

### **Tránsito mutuo, acuerdos de peering complejos**

Hay relaciones de pares entre un SA1 y un SA2 que son complejas y no pueden describirse estrictamente como pares o como cliente-proveedor; por ejemplo cuando ambas partes están enviando intencionalmente prefijos recibidos entre sí a sus pares y/o en sentido ascendente a sus proveedores. En este caso, dos objetos ASPA deben ser creados, por SA1 y SA2 respectivamente.

---

<sup>3</sup>La técnica de AS path prepending consiste en agregar saltos "falsos" a la trayectoria anteponiendo nuestro AS múltiples veces, con el fin de disminuir la prioridad de esa ruta

# Conceptos de Sistemas Distribuidos

## 6.1 Definición de Sistemas distribuidos

Comprender los sistemas distribuidos es fundamental para entender blockchain, ya que blockchain es un sistema distribuido típico, que en sus orígenes con Bitcoin tenía características descentralizadas, pero no tardaron en aparecer soluciones con un mayor grado de centralización para subsanar el problema de la escalabilidad. Una cadena de bloques está pensada originalmente como un sistema que tiene propiedades de los paradigmas descentralizado y distribuido, es un sistema distribuido-descentralizado.

Los sistemas distribuidos son aquellos en los que varias computadoras trabajan juntas para llevar a cabo una tarea y se conectan entre sí a través de una red. La tecnología blockchain, que fue creada originalmente para su uso en Bitcoin, se basa en la descentralización, lo que significa que no hay una sola entidad que controle la red. Sin embargo, en los últimos años, se han desarrollado soluciones con un mayor grado de centralización en el intento de mejorar la escalabilidad de la red.

Existen varias definiciones para los sistemas distribuidos, nombramos a continuación las dos definiciones clásicas más relevantes:

Según Colouris[57], un sistema distribuido es aquel en el cual los componentes, localizados en equipos en red, se comunican y coordinan sus acciones mediante el envío de mensajes.

Según Tanenbaum[58], un sistema distribuido es una colección de computadoras independientes que aparece a sus usuarios como un único sistema coherente.

Los objetivos de los sistemas distribuidos que plantean estos autores no difieren de la finalidad que buscamos hoy en día en los sistemas de cadena de bloques, estos son:

- **Transparencia:** Es la capacidad de ocultar al usuario o a la aplicación que hace uso del sistema distribuido, la complejidad de la distribución de los procesos y los recursos a lo largo de varios sistemas.
- **Sistemas abiertos:** Es la característica de un sistema que permite añadirle nuevas características y servicios de forma dinámica. Se consigue mediante una buena especificación y la publicación de las interfaces de los componentes.
- **Escalabilidad:** Un sistema es escalable si se mantiene el nivel de confiabilidad y la performance cuando hay un incremento significativo en la cantidad de recursos y usuarios.
- **Heterogeneidad:** Al hablar de heterogeneidad nos referimos a la variedad y diferencia que podemos encontrar en los elementos que componen una red de computadoras sobre la que se ejecuta un sistema distribuido. Dicha heterogeneidad no sólo se aplica a las redes y al hardware de las computadoras, sino también a los sistemas operativos, los lenguajes de programación y las implementaciones en las que trabajan los diferentes desarrolladores.
- **Seguridad:** Es el elemento más importante y más complejo debido a la existencia de equipos



físicamente distribuidos. Las técnicas de encriptación pueden ser utilizadas para proporcionar la adecuada protección a los recursos compartidos cuando se transmite mediante la red. La seguridad, cuando se refiere a los datos digitales, tiene tres componentes:

- Confidencialidad: protección de acceso a usuarios no autorizados
  - Integridad: protección de alteración o corrupción
  - Disponibilidad: protección de interferencias que no permitan acceder al servicio (DoS).
- Manejo de fallos: El rango de fallos en sistemas distribuidos es mayor que en cualquier otro sistema. Además, es fundamental que los sistemas distribuidos toleren los distintos tipos de fallos logrando que los usuarios del servicio distribuido no perciban la degradación.
  - Concurrencia: El control de concurrencia trata con los problemas de aislamiento y consistencia del procesamiento de transacciones, asegurando que las operaciones concurrentes sobre objetos puedan sincronizarse de forma tal de lograr consistencia en los datos.

Las técnicas habituales para asegurar un control de la concurrencia son semáforos, hilos, locks, etc. Herramientas como el uso de cachés, consistencia eventual, replicación, balanceo de carga ayudan a abordar el reto de la concurrencia.

Una definición de sistemas distribuidos más actual y orientada a Blockchain, es la de Imran Bashir[59], que dice que los sistemas distribuidos son un paradigma informático mediante el cual dos o más nodos trabajan entre sí de manera coordinada para lograr un resultado común. Además está modelado de tal manera que los usuarios finales lo ven como una única plataforma lógica. También se realizan las siguientes afirmaciones sobre los nodos:

- Un nodo se puede definir como un jugador individual en un sistema distribuido.
- Todos los nodos son capaces de enviar y recibir mensajes entre sí.
- Los nodos pueden ser honestos, defectuosos o maliciosos, y tienen memoria y procesador.
- Un nodo que exhibe un comportamiento irracional también se conoce como nodo bizantino (problema de los generales bizantinos).

El desafío principal del diseño de un sistema distribuido es la coordinación entre los nodos y la tolerancia a fallas. Incluso si algunos de los nodos se vuelven defectuosos (un cierto umbral dictado por el protocolo de consenso) o los enlaces de la red fallan, el sistema distribuido debería poder tolerar esto y continuar trabajando para lograr el resultado deseado.

## 6.2 Problema de los generales bizantinos

El problema de los generales bizantinos es un experimento propuesto en 1982 por Lamport et al. en el paper “The Byzantine Generals Problem” [60] con la finalidad de ejemplificar el dilema de lograr consenso entre un conjunto de entidades con un objetivo común, cuando entre ellas pueden aparecer entidades que quieren impedir el cumplimiento del objetivo. El experimento consiste en lo siguiente: Un conjunto de generales tienen rodeada una ciudad y tienen que ponerse de

acuerdo para atacar coordinadamente ya que fracasarán sino lo hacen todos al mismo tiempo. Los generales se comunican únicamente a través de tenientes o mensajeros que transmitirán el mensaje tal cual lo han recibido. Los generales traidores darán órdenes incorrectas y los tenientes traidores alterarán el mensaje indicando un horario distinto para el ataque. Si no se logra coordinar que todos los generales leales ataquen coordinadamente perderán, por eso el planteo del dilema es buscar algoritmos que aseguren que los generales leales no sean engañados, y puedan llegar a un consenso para cumplir el objetivo. Esto es posible siempre que exista un mínimo de generales leales que puedan acordar un mismo plan de acción.

La solución a este dilema fue planteada en 1999 por Castro y Liskov, quienes presentaron el algoritmo Practical Byzantine Fault Tolerance (PBFT) [61], que resuelve el problema de consenso en presencia de fallas bizantinas en redes asincrónicas utilizando el protocolo de replicación de la máquina de estado. PBFT pasa por una serie de rondas para finalmente llegar a un acuerdo entre los nodos sobre el valor propuesto. Se entiende como “práctico” el sentido de que la propuesta puede trabajar en entornos asíncronos como internet. Otra característica importante es el rendimiento para procesar un gran número de transacciones.

Para que el modelo de PBFT funcione se parte de la premisa que la cantidad de nodos bizantinos (maliciosos) simultáneos nunca puede ser igual o superior a un tercio del total de nodos. Por tanto, a mayor cantidad de nodos en el sistema, más difícil será llegar a dicho tercio. Pero una de las desventajas de PBFT es su escalabilidad, funciona de una manera eficiente en grupos de nodos pequeños, pero se vuelve ineficiente cuando la red está compuesta por un gran número de estos.

PBFT es un excelente algoritmo de consenso para consorcios empresariales en los que se confía parcialmente en los miembros y se aplica en blockchain permissionadas, las cuales están compuestas de pocos nodos conocidos, y no necesitan llegar a un consenso en una red abierta y pública como Bitcoin. De hecho Bitcoin puede considerarse el primer mecanismo de consenso que resuelve el dilema de los generales bizantinos cuando los miembros son desconocidos.

### 6.3 Teorema CAP

El teorema CAP, también conocido como teorema de Brewer, fue introducido por Eric Brewer en el año 2000 en el Simposio “Principles of Distributed Computing”, como una conjetura. Dos años más tarde, Seth Gilbert y Nancy Lynch, del MIT, demostraron su validez con evidencia formal, estableciéndola como teorema.

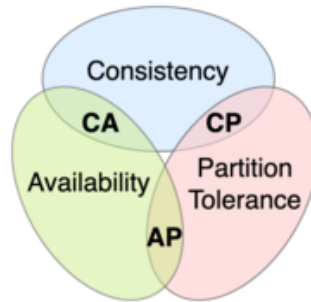
Los sistemas distribuidos son tan complejos de diseñar que se ha probado una teoría conocida como el teorema CAP (Consistency, Availability, Partition tolerance), que establece que es imposible para un sistema de cómputo distribuido garantizar simultáneamente las tres propiedades.

- **Consistencia o Coherencia:** Que todos los nodos vean la misma información al mismo tiempo, o sea tengan una copia única, actual e idéntica de los datos en todo momento.
- **Disponibilidad:** Es la garantía de que los nodos en el sistema están activos, accesibles para su uso y están aceptando solicitudes entrantes y respondiendo con datos sin fallas pero sin

la garantía de que contenga el dato más reciente.

- Tolerancia al particionado: Asegura que el sistema continúa funcionando correctamente como un todo incluso cuando los nodos de la red fallan o no se comunican entre sí.

En el siguiente diagrama se observa que solo se pueden lograr dos propiedades a la vez, a costa de renunciar a la tercera.



**Figura 6.1:** Teorema CAP, ó Conjetura de Brewer[62]

- Podemos optar por CP (consistencia y tolerancia de partición), y sacrificamos la disponibilidad.
- Podemos optar por AP (disponibilidad y tolerancia de partición) y sacrificamos la consistencia.
- Podemos elegir AC (disponibilidad y consistencia) y sacrificamos la tolerancia de partición.

Por lo general, no se puede ignorar el hecho de que una partición o falla de red puede ocurrir, se debe tener en cuenta la latencia en la red para las blockchains cuyos nodos se distribuyen a lo largo de internet. Por lo tanto, la elección se convierte principalmente en priorizar la consistencia o la disponibilidad, renunciando a alguna de ellas.

Bitcoin parece lograr las tres propiedades simultáneamente, pero en realidad la consistencia se sacrifica a favor de la disponibilidad y la tolerancia de partición (AP). En este escenario, la consistencia (C) en la cadena de bloques no se logra simultáneamente con la tolerancia de partición (P) y la disponibilidad (A), pero se logra con el tiempo. Esto se llama consistencia eventual, donde la consistencia se logra como resultado de la validación a lo largo del tiempo de múltiples nodos, a medida que se siguen sumando bloques a la cadena.

La consistencia se logra en las redes blockchain utilizando algoritmos de consenso que aseguran que todos los nodos tienen la misma copia de los datos, puede haber un desacuerdo temporal entre los nodos sobre el estado final, pero finalmente se acuerda. El concepto de minería se introdujo en Bitcoin con este propósito, para mantener la consistencia de la red blockchain.

# Conceptos de Criptografía

## 7.1 Introducción

El interés global que despierta la cadena de bloques se debe a que busca mediante la confianza en la tecnología, poder desplazar a intermediarios o terceras partes innecesarias de los procesos administrativos y/o productivos. Si bien la cadena de bloques no nace a partir de una nueva tecnología, se crea con la conjunción de tres tecnologías conocidas existentes: redes P2P, criptografía, y programas (los contratos inteligentes). De estos tres la criptografía es el elemento principal, debido a que es el que nos aporta la confianza.[63]

La criptografía (del griego “ocultar” y “escribir”), literalmente “escritura oculta”, es el arte o ciencia de cifrar y descifrar información utilizando técnicas matemáticas que hagan posible el intercambio de mensajes de manera segura de forma tal que sólo puedan ser leídos por las personas a quienes van dirigidos.

La criptología es el nombre genérico que une dos disciplinas opuestas y a la vez complementarias: criptografía y criptoanálisis. La criptografía se ocupa del estudio de los algoritmos, protocolos y sistemas que se utilizan para permitir que las personas se comuniquen de forma segura a través de un canal inseguro, de forma que se garantice la privacidad de la transmisión y la autenticidad. En forma más clara, se encarga de construir los procedimientos para cifrar, es decir para ocultar información confidencial. El criptoanálisis es el opuesto a la criptografía, se ocupa de romper esos procedimientos para así recuperar la información, su objetivo es buscar el punto débil de los criptosistemas para así reducir o eliminar la seguridad que teóricamente aporta este.

El uso principal de la criptografía es brindar un servicio de confidencialidad, pero generalmente es un componente fundamental dentro de un sistema de seguridad más amplio, para abordar un problema de seguridad complejo (por ejemplo, asegurar una cadena de bloques). El ecosistema blockchain requiere de muchas primitivas criptográficas diferentes, como funciones hash, criptografía de clave simétrica, firmas digitales y criptografía de clave pública.

## 7.2 Servicios de la criptografía

Los servicios que brinda la criptografía u objetivos que persigue son los siguientes [59]:

- **Confidencialidad:** La confidencialidad garantiza que la información está disponible solo para entidades autorizadas.
- **Integridad:** La integridad garantiza que la información solo sea modificada por entidades autorizadas.
- **Autenticación:** La autenticación proporciona seguridad sobre la identidad de una entidad o la validez de un mensaje. Hay dos tipos de mecanismos de autenticación: la autenticación de entidad y la autenticación de origen de los datos.

Autenticación de entidad: Es la que garantiza que la entidad está actualmente involucrada y activa en una sesión de comunicación.

Autenticación de mensaje u origen de datos: Es la que garantiza que la fuente de la información está realmente verificada, y al corroborar la fuente, se garantiza también que no se alteraron los datos. Los métodos más frecuentemente utilizados son las firmas digitales y los códigos de autenticación de mensajes(MAC).

- No repudio: El no repudio es la seguridad de que una entidad no pueda negar un compromiso o acción realizada con anterioridad, proporcionando evidencia irrefutable. Ofrece una prueba o evidencia definitiva de que una entidad ha realizado una actividad en particular, el autor no puede negar haber realizado la misma.
- Transparencia: La transparencia, o rendición de cuentas, es la garantía de que existen registros detallados de las acciones de una entidad.

### 7.3 Primitivas criptográficas

Las primitivas criptográficas son los componentes básicos de un protocolo o sistema de seguridad más complejo. A continuación, en el siguiente diagrama de taxonomía de primitivas criptográficas, se presentan los algoritmos criptográficos que son esenciales para construir protocolos y sistemas seguros.

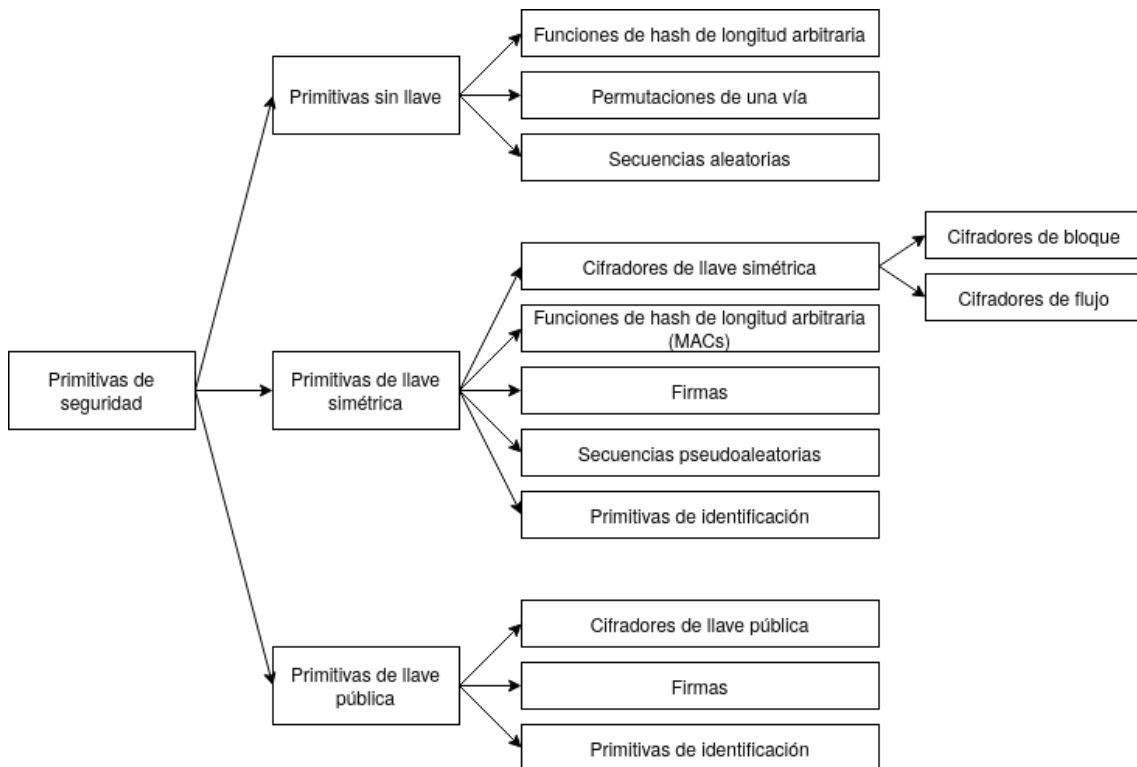


Figura 7.1: Taxonomía de las primitivas criptográficas [64]

En el mismo se puede observar que la criptografía se divide principalmente en tres categorías:

primitivas sin clave, primitivas ó criptografía de clave simétrica y primitivas o criptografía de clave asimétrica.

Los sistemas de clave única o métodos simétricos son aquellos en los que los procesos de cifrado y descifrado son llevados a cabo por una única clave, y los sistemas de clave pública o asimétrica son aquellos en los que los procesos de cifrado y descifrado son llevados a cabo por dos claves distintas y complementarias.

### **7.3.1 Primitivas criptográficas sin clave**

#### **7.3.1.1 Secuencias aleatorias**

La aleatoriedad proporciona un elemento indispensable para la seguridad de los protocolos criptográficos. La mayor parte de los algoritmos y protocolos criptográficos requieren de la generación de grandes cantidades de bits (generados de forma aleatoria o pseudoaleatoria), bien sea para los procesos de cifrado o para la generación de claves. Tales secuencias de bits deben poder ser consideradas aleatorias en el sentido de que su comportamiento no debe ser previsible por un atacante al criptosistema.

No es sencillo generar una aleatoriedad que posea un alto grado de incertidumbre, pero existen métodos que garantizan que se genere un nivel adecuado de aleatoriedad para su uso en algoritmos criptográficos.

Hay dos categorías de fuentes de aleatoriedad, los generadores de números aleatorios "verdaderos" (TRNG) y los generadores de números pseudoaleatorios (PRNG). Un TRNG genera números que, para todos los propósitos prácticos, no se pueden adivinar o predeterminar y, por lo general, utilizan fenómenos físicos externos, generalmente del mundo analógico (variaciones de temperatura, ruido de los componentes electrónicos, etc) para generar una verdadera aleatoriedad. Un PRNG solo aparenta generar números aleatorios pero en realidad genera números deterministas que tienen ciertas propiedades estadísticas que, a partir de un examen superficial, parecen ser aleatorios, pero que pueden probarse de otra manera mediante un análisis[65].

#### **7.3.1.2 Funciones hash**

Las funciones hash son algoritmos que se utilizan para mapear un conjunto de datos de entrada a un valor hash de salida. Estas funciones son unidireccionales, lo que significa que no se puede utilizar el valor hash de salida para volver a obtener los datos de entrada originales. El valor hash de salida es generalmente de un tamaño fijo(entre 128 bits y 512 bits), independientemente del tamaño de los datos de entrada, y se utiliza como una representación compacta o "huella digital" de los datos de entrada. Las funciones hash son utilizadas ampliamente para garantizar la integridad de los datos y para verificar la autenticidad de mensajes.

Las funciones hash tienen algunas propiedades que se deben cumplir de acuerdo al nivel de seguridad requerido, las principales son:

- Determinista: Esto significa que para un mismo conjunto de datos de entrada siempre se ha

de obtener el mismo valor del código hash. Ejemplos de funciones hash no deterministas son aquellas funciones hash que dependen de parámetros externos, tales como generadores de números pseudoaleatorios o la fecha.

- Bajo costo/computacionalmente eficientes: los algoritmos generados por una función hash no requieren de una gran capacidad de cálculo o una gran cantidad de memoria para ejecutarse, deben ser eficientes incluso para mensajes de gran tamaño.
- Tamaño fijo: no importa la cantidad de datos que se quiera convertir, ya que esta siempre será una cadena de tamaño fijo. Si se usa el SHA-256, siempre la cadena tendrá 64 caracteres.
- Resistente a colisiones: Hace referencia a que la probabilidad de que dos conjuntos de datos diferentes generen el mismo código hash ha de ser altamente improbable.
- Efecto avalancha: Los algoritmos han de ser sensibles a cualquier mínimo cambio en el conjunto de datos de entrada. Al realizar un pequeño cambio en la entrada, por ejemplo, un solo bit, los cambios en el código hash resultante han de ser enormes.
- Son irreversibles: una de las cosas que le da más seguridad a las funciones hash, es que no es posible tomar un hash y obtener los datos que le dieron origen al mismo.

Las funciones hash se utilizan comúnmente para firmas digitales, para garantizar la integridad de los datos y en códigos de autenticación de mensajes, como los HMAC.

Existen varias familias de funciones hash, como MD, SHA1, SHA-2, SHA-3, RIPEMD y Whirlpool. La siguiente lista describe los algoritmos hash seguros (SHA) más comunes:

SHA-0: esta es una función de 160 bits introducida por el Instituto Nacional de Estándares y Normas de EE. UU. Tecnología (NIST) en 1993.

SHA-1: SHA-1 fue presentado en 1995 por NIST como el reemplazo de SHA-0, también es de 160 bits. Actualmente también se considera inseguro y está declarado obsoleto por las autoridades de certificación. Se desaconseja su uso en cualquier implementación nueva.

SHA-2: esta categoría incluye cuatro funciones definidas por la cantidad de bits del hash: SHA-224, SHA-256, SHA-384 y SHA-512.

SHA-3: Esta es la última familia de funciones SHA. SHA3-224, SHA3-256, SHA3-384 y SHA3-512 son miembros de esta familia. SHA3 es una versión estandarizada por el NIST del algoritmo Keccak

RIPEMD: RIPEMD es el acrónimo de RACE Integrity Primitives Evaluation Message Digest. Se basa en las ideas de diseño utilizadas en la compilación MD4. Existen múltiples versiones de RIPEMD, incluyendo 128 bits, 160 bits, 256 bits y 320 bits.

Whirlpool: se basa en una versión modificada del cifrado Rijndael conocido como W. Utiliza la función de compresión de Miyaguchi-Preneel, que es un tipo de función unidireccional utilizada para la compresión de dos entradas de longitud fija en una única salida de longitud fija. Se utiliza a menudo en sistemas de criptomonedas y se considera segura.

Ethereum utiliza la función hash criptográfica Keccak-256 con frecuencia. Keccak-256 fue diseñado como candidato para la competencia de función hash criptográfica SHA-3 realizada en 2007 por el NIST. Keccak fue el algoritmo ganador, que se estandarizó como Estándar Federal de Procesamiento de Información (FIPS) 202 en 2015. Las diferencias de implementación son leves y tienen que ver con los parámetros de relleno, pero son significativas porque Keccak-256 produce salidas hash diferentes de FIPS-202 SHA-3 para la misma entrada.

Dentro del ecosistema blockchain la Función Hash tiene múltiples usos[66], los mas destacados son:

- Minería de las Criptomonedas: las cadenas de bloques, son el área donde trabajan los mineros y ellos almacenan toda la información en Hash.
- Seguridad en las Transacciones: todas las transacciones que se realizan con criptomonedas se encriptan a través de los Hash.
- Generación de Claves: el Hash es quien se encarga de generar la clave pública y privada de cada monedero de criptomonedas.

### **7.3.2 Criptografía simétrica**

En el cifrado simétrico, dos entidades comparten una sola clave de cifrado/descifrado y se utiliza la misma clave tanto para el cifrado como para el descifrado de un mensaje. Entre los algoritmos de cifrado simétrico más conocidos se encuentran el DES (Data Encryption Standard), el 3DES (Triple Data Encryption Standard) y el AES (Advanced Encryption Standard). Este último algoritmo fue el que desplazó al DES y es el empleado actualmente debido a que su método de cifrado se adapta mejor a las necesidades actuales. El cifrado de AES puede ser empleado tanto en software como en hardware y el tamaño fijo del bloque es de 128 bits. Mientras que las claves pueden ser elegida a voluntad entre 128, 192 y 256 bits [67].

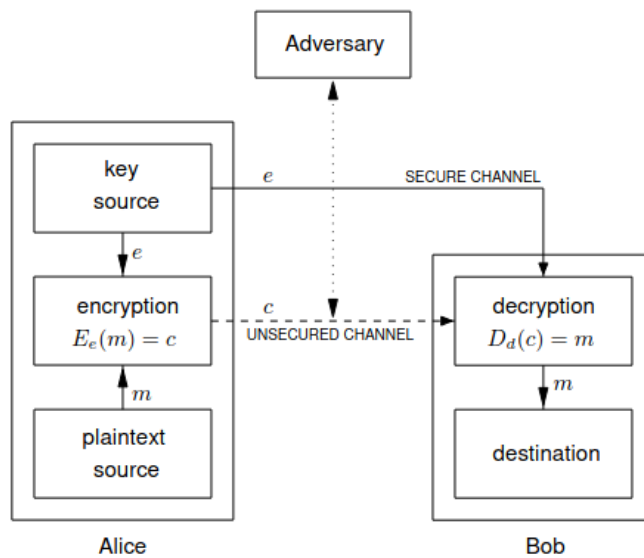
El principal reto del cifrado simétrico consiste en la distribución y protección de las claves, por tal motivo, con frecuencia se combina con el cifrado asimétrico para resolver el problema de la transferencia segura de claves. Entre los ejemplos más destacados de dicho sistema híbrido se encuentra el protocolo criptográfico de seguridad de la capa de transporte (TLS) utilizado para proteger grandes porciones de la Internet moderna.

Hoy en día, los algoritmos de clave simétrica se aplican ampliamente en aplicaciones de mensajería segura como en almacenamiento en la nube, para mejorar la seguridad de los datos y la privacidad del usuario. La tecnología Blockchain no hace uso de la criptografía simétrica en su infraestructura más fundamental, en su lugar utiliza un tipo específico de algoritmo de firmas digitales (DSA) conocido como el algoritmo de firma digital de curva elíptica ECDSA.

### **7.3.3 Criptografía de clave asimétrica**

La infraestructura de cifrado asimétrico, ó cifrado de clave pública, fue inventada por Merkle, Diffie y Hellman (MDH) en 1976, garantizando con este esquema comunicaciones completamente

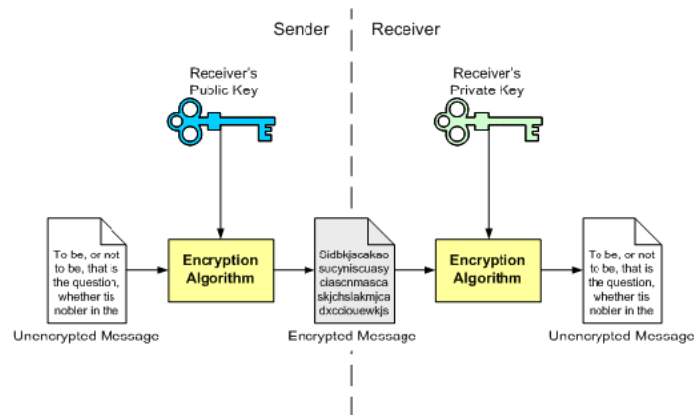




**Figura 7.2:** Criptografía simétrica, donde dos entidades comparten una sola clave de cifrado/descifrado.[64]

seguras sobre canales inseguros; siendo en la actualidad el método de cifrado más usado en Internet. Un esquema de cifrado de clave pública (PKC) supone el uso de dos claves, una clave pública y otra clave privada para realizar diferentes tareas. Las claves públicas se distribuyen y difunden ampliamente, mientras que las claves privadas se mantienen en secreto.

Usando la clave pública de una persona, es posible cifrar un mensaje para que solo la persona con la correspondiente clave privada pueda descifrarlo y leerlo. Usando una clave privada, se puede crear una firma digital para que cualquier persona con la clave pública correspondiente pueda verificar que el mensaje fue creado por el propietario de la clave privada y no se modificó desde entonces. En blockchain se hace un uso extensivo de la criptografía de clave pública, la misma se utiliza en forma de direcciones('addresses') que derivan de la clave pública y claves privadas, para controlar la propiedad de los fondos y la ejecución de los contratos inteligentes.



**Figura 7.3:** Criptografía asimétrica. Usando la clave pública de una persona, es posible cifrar un mensaje para que solo la persona con la correspondiente clave privada pueda descifrarlo.[68]

Estas claves se basan en funciones matemáticas que tienen una propiedad especial: es sencillo calcularlas, pero es muy difícil calcular su inversa. Sobre la base de estas funciones, la criptografía permite la creación de secretos digitales y firmas digitales infalsificables, que están protegidas por las leyes de las matemáticas.

Por ejemplo, uno de los algoritmos más comunes para el cifrado asimétrico en uso hoy en día se conoce como RSA. En el esquema RSA, las claves se generan utilizando un módulo al que se llega al multiplicar dos números primos grandes, que es algo trivial. Pero dado el producto de dos números primos grandes, es muy difícil encontrar los factores primos (un problema llamado descomposición en factores primos). En términos básicos, el módulo genera dos claves (una pública que puede compartirse y una privada que debe mantenerse en secreto). El algoritmo RSA fue descrito por primera vez en 1977 por Rivest, Shamir y Adleman (por lo tanto, RSA) [69] y sigue siendo un componente importante de los sistemas de criptografía de clave pública.

El algoritmo de firma digital de curva elíptica (ECDSA) utiliza una categoría más avanzada de funciones matemáticas, se basa en operaciones aritméticas en una curva elíptica. En la aritmética de curvas elípticas, la multiplicación módulo a primo es simple pero la división (la inversa) es prácticamente imposible. La criptografía de curva elíptica se usa ampliamente en los sistemas informáticos modernos y es la base para verificar las transacciones en la blockchain Ethereum [70].

Debido a que los algoritmos de encriptación asimétrica generan pares de claves que están matemáticamente vinculados, sus longitudes de clave son mucho más largas que las utilizadas en la criptografía simétrica. Esta longitud más larga, generalmente entre 1.024 y 2.048 bits, hace que sea extremadamente difícil calcular una clave privada de su contraparte pública.

## 7.4 Firma digital

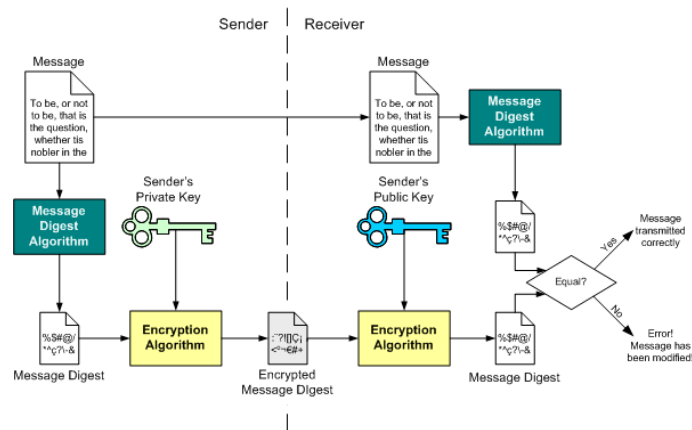
Una primitiva criptográfica que es fundamental en la autenticación, autorización y el no repudio es la firma digital. Una firma digital certifica un documento y lo atribuye fehacientemente a su autor [64]. Usando el modo autenticación, el receptor del mensaje puede estar seguro que nosotros generamos dicho mensaje. Sin embargo, esto puede resultar en un proceso ineficiente, lento y que produce gran volumen de datos (al menos el doble que los datos originales). Para mejorar el proceso, se utilizan funciones de hash.

Desde el punto de vista técnico es una secuencia de bits de longitud fija, resultante de aplicar un conjunto de algoritmos criptográficos que permiten identificar al autor y verificar la integridad del contenido firmado. El proceso de firmar implica el cifrado, con la clave privada del firmante, del hash del mensaje a firmar.

Entre los propósitos de la firma digital se encuentran los siguientes:

- Atribuir el documento a su autor de manera fehaciente (autenticidad del autor)
- Verificar que el contenido del documento no fue alterado (integridad del documento).
- Garantizar que el autor no pueda negar haber firmado el documento o mensaje (no repudio)

de la acción).

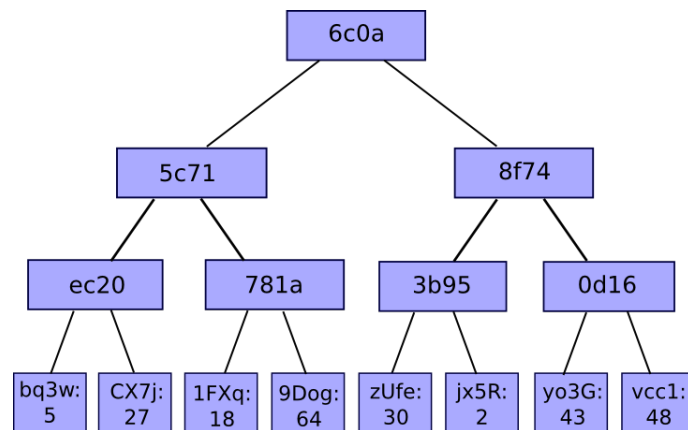


**Figura 7.4:** Firma digital. Usando una clave privada, se puede crear una firma digital para que cualquier persona con la clave pública correspondiente pueda verificar que el mensaje fue creado por el propietario de la clave privada y no se modificó desde entonces.[68]

## 7.5 Árbol de Merkle

Fue patentado en 1979 por Ralph Merkle y es uno de los métodos más utilizados para la verificación de datos en redes distribuidas p2p y blockchains, software de base de datos, sistemas de archivos, estructuras de llaves públicas, sistemas de versionamiento, redes distribuidas (P2P), entre otros.

Un árbol hash de Merkle (en inglés, Merkle hash tree) o árbol de Merkle es una estructura ramificada (Fig 7.5), como un árbol invertido, en la que se parte desde los nodos base (hojas) y se escala progresivamente a través de nodos padre (ramas) hasta llegar al nodo raíz o Merkle root. Este último es el identificador principal que permitirá verificar el conjunto de datos como un todo. A cada nodo base se le asigna un hash (identificador) único, luego se procede a concatenarlos en pares y a esta suma se le asigna un nuevo identificador. De esta manera se prosigue en cada nivel, concatenando por pares y la repetición se lleva a cabo hasta que ya no quedan pares que sumar, obteniendo un único hash principal (o Merkle root)[71].



**Figura 7.5:** Merkle Tree[71]

Permite que gran número de datos separados puedan ser ligados a un único valor de hash, el hash del nodo raíz del árbol. De esta forma proporciona un método de verificación segura y eficiente de los contenidos de grandes estructuras de datos. En sus aplicaciones prácticas, normalmente el hash del nodo raíz va firmado para asegurar su integridad y que la verificación sea totalmente fiable. La demostración de que un nodo hoja es parte de un árbol hash dado requiere una cantidad de datos proporcional al logaritmo del número de nodos del árbol.

La razón por lo que esto funciona es porque los hashes se propagan hacia arriba: si un usuario malintencionado intenta hacer un cambio en una transacción falsa en la parte inferior del árbol de Merkle, este cambio provocará un cambio en el nodo superior y seguidamente otro cambio en el nodo por encima de este, hasta que finalmente, se produzca un cambio en la raíz del árbol y por tanto en el hash del bloque, haciendo que el protocolo tenga que registrarlo como un bloque completamente diferente (y casi con toda seguridad con una prueba de trabajo inválida).

Actualmente el mayor uso de los árboles de Merkle es hacer seguros los bloques de datos recibidos de otros pares en las redes peer-to-peer, asegurar que estos son recibidos sin daños y sin ser alterados de un modo computacionalmente eficiente. Además permiten que los datos de un bloque puedan ser entregados por partes: un nodo puede descargar solamente la cabecera de un bloque (árbol) desde una fuente, y otra pequeña parte del árbol relevante para él, desde otra fuente, y todavía asegurar que los datos son correctos. Esto posibilita un ahorro en recursos de almacenamiento y un mejor rendimiento en la transmisión de datos en redes distribuidas.

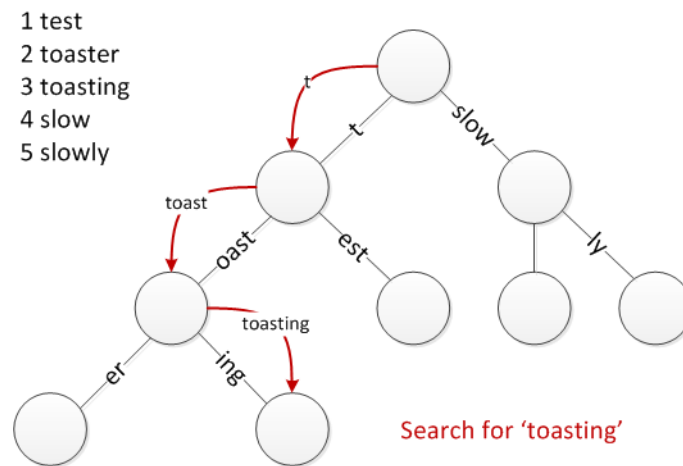
Otra ventaja notable que aporta a las redes blockchain es la verificación de pagos simples o SVP, que consiste en solicitar una prueba de Merkle para corroborar que una transacción se encuentra en un bloque particular. Esto suele hacerse cuando se está ejecutando un nodo en un dispositivo con recursos limitados y no se desea descargar y hashear todas las transacciones de un bloque.

## 7.6 Árbol Patricia Trie

Patricia trie es una estructura de datos que también se denomina árbol de prefijos, árbol radix o trie que permite la recuperación de información (de ahí su nombre del inglés reTRIEval). Trie usa una clave como ruta para que los nodos que comparten el mismo prefijo también puedan compartir la misma ruta. Esta estructura es más rápida para encontrar prefijos comunes, fácil de implementar y requiere poca memoria[72].

Las claves son almacenadas en las hojas del árbol y los nodos internos son pasarelas para guiar la búsqueda. El árbol se estructura de forma que cada letra de la clave se sitúa en un nodo de forma que los hijos de un nodo representan las distintas posibilidades de símbolos diferentes que pueden continuar al símbolo representado por el nodo padre.

Por tanto la búsqueda en un trie se hace de forma similar a como se hacen las búsquedas en un diccionario: Se empieza en la raíz del árbol. Si el símbolo que estamos buscando es A entonces la búsqueda continúa en el subárbol asociado al símbolo A que cuelga de la raíz. Se sigue de forma análoga hasta llegar al nodo hoja. Entonces se compara la cadena asociada al nodo hoja y si coincide con la cadena de búsqueda entonces la búsqueda ha terminado en éxito, si no entonces el elemento



**Figura 7.6:** Merkle Tree[71]

no se encuentra en el árbol.

Por eficiencia se suelen eliminar los nodos intermedios que sólo tienen un hijo, es decir, si un nodo intermedio tiene sólo un hijo con cierto carácter entonces el nodo hijo será el nodo hoja que contiene directamente la clave completa.

Es muy útil para conseguir búsquedas eficientes en repositorios de datos muy voluminosos. La forma en la que se almacena la información permite hacer búsquedas eficientes de cadenas que comparten prefijos, por esto se utiliza a menudo en sistemas distribuidos y tecnologías blockchain, ya que permite actualizaciones rápidas del árbol sin tener que recalcular todo el árbol. En la blockchain Ethereum, se utiliza para almacenar la información de estado de la red en la cadena de bloques.

# Conceptos de Blockchain

## 8.1 Definición de blockchain

Las cadenas de bloques se han considerado una tecnología disruptiva y el comienzo de lo que se ha llamado Web 3.0, o web3. La Web 3.0 no solo se refiere a descentralización, sino que es el próximo paradigma tecnológico de la Web donde muchos dispositivos están interconectados (llamado Internet de las cosas) y se utilizan con tecnologías como la inteligencia artificial.

Para comprender de dónde provienen las cadenas de bloques y las criptomonedas, es importante reflexionar sobre el nacimiento de Bitcoin. Bitcoin fue el verdadero comienzo de la tecnología blockchain porque proporcionó un caso de uso a la sociedad. Hoy en día, después de años de investigación en el área sigue creciendo la cantidad de cadenas de bloques existentes que se adaptan a los diferentes casos de uso; algunas cadenas de bloques son solo para criptomonedas, mientras que otras no admiten criptomonedas.

En el libro “Architecting Enterprise Blockchain Solutions”[63] los autores proponen tres definiciones de blockchain según la audiencia o el espectador, a continuación las citamos a las tres:

- Definición técnica: una estructura de datos segura y compartida globalmente que mantiene una base de datos back-end transaccional que es inmutable.
- Definición de negocio: una red de negocios que se utiliza entre pares para intercambiar valor. El valor puede ser monedas, información de seguimiento o cualquier cosa que las partes interesadas requieran que se mantenga en el libro mayor de la cadena de bloques.
- Definición legal: una cadena resistente a la corrupción de asientos contables compartidos a través de una red por múltiples partes que no requieren un intermediario centralizado para presentar y validar transacciones.

## 8.2 Componentes de una Blockchain

Los componentes de una cadena de bloques abierta y pública usualmente son:

Los nodos son los equipos o dispositivos que conforman la red y que usualmente almacenan una copia completa del historial de transacciones de la cadena de bloques. Estos nodos forman una red peer-to-peer (P2P) que conecta a los participantes que actúan como iguales, actuando tanto como cliente y servidor para el resto de los nodos. En las cadenas de bloques los nodos se encargan de propagar las transacciones y los bloques de transacciones verificadas, mediante un protocolo Gossip estandarizado para difundir la información en la red. La arquitectura peer-to-peer de blockchain ofrece muchos beneficios, entre los más importantes está el hecho de que las redes P2P ofrecen mayor seguridad que los arreglos tradicionales de cliente-servidor.

Los mensajes o transacciones son las operaciones que se registran en la cadena de bloques, y producen las transiciones de estado. Las transacciones pueden ser de cualquier tipo, pero en el

contexto de las criptomonedas, una transacción suele ser el intercambio de monedas entre dos partes.

Los bloques son los elementos más importantes de una cadena de bloques. Cada bloque contiene una serie de transacciones, así como un hash (un valor único generado a partir de la información del bloque) y un hash del bloque anterior en la cadena. Esto hace que la cadena de bloques sea una estructura de datos inmutable, ya que cualquier cambio en un bloque implicaría cambiar su hash y, por lo tanto, todos los bloques posteriores en la cadena. Así la cadena de bloques asegurada criptográficamente actúa como un registro contable de todas las transiciones de estado verificadas y aceptadas.

Un algoritmo de consenso que descentraliza el control sobre la cadena de bloques, al obligar a los participantes a cooperar en la aplicación de las reglas de consenso. Un conjunto de reglas de consenso que rigen lo que constituye una transacción válida, y lo que constituye una transición de estado válida posibilitando llegar a un acuerdo sobre el estado actual de la cadena de bloques.

Una máquina de estado que procesa transacciones de acuerdo con las reglas de consenso.

Un esquema de incentivos diseñado para asegurar económicamente la máquina de estado en un entorno abierto. En entornos cerrados, o cadenas de bloques permissionadas, no es necesario que exista una criptomoneda subyacente para asegurar la red.

Todos o la mayoría de estos componentes suelen combinarse en un único cliente de software. En Ethereum existe una especificación de referencia, una descripción matemática del sistema en el "Yellow paper"[73] que sirve para que estos se construyan de acuerdo con la especificación de referencia.

Las cadenas de bloques privadas a menudo tienen una autoridad certificadora (CA, por sus siglas en inglés) que se encarga de emitir y revocar los certificados digitales utilizados para autenticar a los nodos de la red. En estas cadenas de bloques el acceso a la red está restringido a un conjunto específico de nodos autorizados debido a que se requiere de un mayor control y privacidad sobre la red.

### **8.3 Arquitectura de una blockchain**

La arquitectura de una cadena de bloques es un elemento clave para determinar cómo funciona y se utiliza la red. Algunas de las características y consideraciones más importantes a la hora de diseñar la arquitectura de una cadena de bloques incluyen el consenso, la escalabilidad, el desempeño, la seguridad, la privacidad, la transparencia y la facilidad de programación.

- El algoritmo de consenso es un elemento fundamental de la arquitectura de una cadena de bloques, ya que determina cómo los nodos llegan a un acuerdo sobre el estado actual de la cadena y cómo se validan las nuevas transacciones. Algunos algoritmos de consenso comunes incluyen Prueba de Trabajo (PoW, por sus siglas en inglés) y Prueba de Participación (PoS, por sus siglas en inglés). Cada algoritmo tiene sus propias ventajas y desventajas en términos de costos, escalabilidad, desempeño y seguridad.

- La escalabilidad es otra consideración importante a la hora de diseñar la arquitectura de una cadena de bloques. Una cadena de bloques debe ser capaz de manejar un alto volumen de transacciones sin comprometer el rendimiento o la seguridad. Algunas cadenas de bloques, como la cadena de bloques Ethereum, utilizan técnicas como la fragmentación o el uso de contratos inteligentes para mejorar la escalabilidad.
- La seguridad también es un aspecto clave de la arquitectura de una cadena de bloques. Las cadenas de bloques deben ser resistentes a ataques cibernéticos y a cualquier otro tipo de manipulación no autorizada. Esto se logra a través de la criptografía y del uso de algoritmos de consenso robustos.
- La privacidad es otra consideración importante en algunos entornos, como las cadenas de bloques privadas. En estos casos, se deben tomar medidas adicionales para proteger la privacidad de las transacciones y evitar que sean visibles para cualquier nodo no autorizado.
- La transparencia es una característica clave de muchas cadenas de bloques públicas, que permite a cualquier persona ver y verificar las transacciones registradas en la cadena. Sin embargo, en algunos casos, es posible que sea necesario limitar la transparencia, como en el caso de las cadenas de bloques privadas.

El compromiso entre privacidad y transparencia es un tema complejo y depende de las necesidades y preferencias de cada usuario o arquitectura. En general, es importante tener en cuenta que la privacidad y la transparencia no son mutuamente excluyentes y que es posible encontrar un equilibrio adecuado entre ambas. Por ejemplo, se pueden utilizar técnicas de privacidad para proteger la privacidad de las transacciones individuales mientras se mantiene la transparencia general de la red.

La facilidad de programación también es un factor importante a considerar al elegir una cadena de bloques para un proyecto o aplicación específica. Una cadena de bloques que ofrece una amplia variedad de herramientas y lenguajes de programación, y que es fácil de utilizar puede ser más atractiva para los desarrolladores y contribuir al éxito de la red.

En resumen, la arquitectura de blockchain debe estar inteligentemente diseñada y es un elemento clave para determinar cómo funciona una red. Los algoritmos de consenso son el núcleo de esta arquitectura, y son la principal característica para diferenciar unas blockchain de otras. Se analizarán las distintas blockchain existentes para determinar cuál se ajusta mejor a este caso de uso (Ethereum, Solana, Cardano, Quorum, Hyperledger Fabric, Corda, o frameworks como LACChain, BFA, etc).

#### **8.4 Clasificación de una blockchain**

Es importante tener claro que no todas blockchain existentes se ajustan a las necesidades de cualquier proyecto, por esto es esencial conocer los tipos de blockchain y las diferencias que poseen, para elegir la más adecuada para cada caso de uso. Antes de profundizar en las diferencias que tienen, se enumerarán las características en común que deben cumplir estos sistemas para consi-



derarse una cadena de bloques. Estas son:

- Son una base de datos o libro contable mayor que solo permite añadir datos, donde la estructura de la cadena de bloques es tal que cada bloque de transacciones se vincula al bloque anterior.
- Son una red de nodos o pares, donde cada uno posee una copia de la red. Son considerados iguales e interactúan de manera p2p.
- Existe un mecanismo de consenso necesario para que los nodos se pongan de acuerdo en cuáles son las transacciones propagadas en la red que pasarán a formar parte de la cadena de bloques.

La norma ISO/TC 307[74], titulada "Blockchain y tecnologías de registro distribuido", proporciona una visión general de las blockchains y las tecnologías de registro distribuido (DLT, por sus siglas en inglés). La norma cubre una amplia gama de temas relacionados con las blockchains y las DLT, incluyendo la clasificación según los tipos de redes blockchain.

La siguiente tabla resume las diferencias en las blockchains según si son públicas, privadas o público permisionadas:

Blockchain Pública		Blockchain Privada		(Público-Permisionada)	
Pública (abierta para todos)	✓	Pública (abierta para todos)	✗	Pública (abierta para todos)	✓
Decentralizada	✓	Decentralizada	✗	Decentralizada	✓
Transparente	✓	Transparente	✗	Transparente	✓
Bajo costo de transacción	✗	Bajo costo de transacción	✓	Bajo costo de transacción	✓
No basada en criptomoneda	✗	No basada en criptomoneda	✓	No basada en criptomoneda	✓
No anónimo (por lo tanto puede ser regulado)	✗	No anónimo (por lo tanto puede ser regulado)	✓	No anónimo (por lo tanto puede ser regulado)	✓
Privacidad habilitada	✗	Privacidad habilitada	✓	Privacidad habilitada	✓

Figura 8.1: Tipos de redes blockchain.[74]

### 8.4.1 Blockchain pública

Las blockchain públicas o despermisionadas permiten unirse a la red a cualquier usuario que lo desee de forma pseudoanónima. Es decir, en las blockchain públicas nadie puede restringir u obstaculizar el derecho a ser parte de la red (convertirse en nodo de la red), participar en el mecanismo de consenso (PoW, PoS, etc) y ser recompensado por esa tarea.

Se considera que las blockchain públicas son mayormente descentralizadas, más resistentes a la censura, y brindan mayor seguridad, debido a que existe un mayor número de nodos validando las transacciones. Las mismas tienen mecanismos de seguridad para desalentar a los actores maliciosos, y recompensar a los honestos para que tengan un buen desempeño y lograr el objetivo de mantener la seguridad de la red.

Uno de los desafíos actuales y de mayor relevancia en las blockchain públicas, es poder escalar a

un mayor número de transacciones, que permita que más usuarios participen de la red. Ejemplo de estas redes son Bitcoin, Ethereum y demás redes asociadas a criptomonedas.

#### **8.4.2 Blockchain privadas o permissionadas**

Las cadenas privadas se adaptan mejor al mundo empresarial, donde una organización quiere disfrutar de las propiedades de blockchain sin hacer que su red sea accesible externamente. En un marcado contraste con las blockchains públicas, las blockchains privadas establecen reglas que dictan quién puede ver y escribir en la cadena de bloques (son entornos autorizados o permissionados). Estos sistemas poseen un alto grado de centralización, ya que existe una jerarquía clara con respecto al control o gobernanza de la red. Sin embargo, se obtiene la confianza a partir de que varios nodos aún mantienen una copia de la cadena en cada nodo y pueden verificar la misma.

Si bien el mecanismo de consenso Proof of work ha demostrado ser muy efectivo en mantener la seguridad en entornos públicos, en una blockchain privada carece de sentido, ya que se conoce la identidad de cada participante y la gobernanza es más simple. Un algoritmo más eficiente, en este caso, es uno con validadores designados, que son nodos seleccionados para asumir ciertas funciones como la validación de transacciones. En caso de que los nodos comiencen a actuar de forma maliciosa, pueden ser rápidamente detenidos y/o eliminados de la red.

Al ser privadas, estas blockchains suelen ser más rápidas y escalables que las blockchains públicas, ya que tienen menos nodos y menos transacciones que validar. Sin embargo, al no ser descentralizadas, pueden presentar vulnerabilidades relacionadas con la seguridad y la gobernanza. Cuantos menos nodos forman parte de la validación de transacciones, más fácil será para los actores maliciosos coludirse, por lo que los administradores de las blockchain privadas deben asegurarse de que los nodos que agregan y verifican bloques son altamente confiables. Dado que el control de la blockchain es más centralizado, será más sencillos coordinar tareas como hacer una reversión o "apagar" la blockchain. El ejemplo típico de este tipo de redes son las redes privadas con tecnología Hyperledger Fabric, también existen otras como Corda o derivadas de Ethereum.

#### **8.4.3 Blockchain público-permisionadas**

Las blockchains público/permisionadas son una combinación de las dos anteriores. Son públicas en el sentido de que cualquiera puede acceder a ellas y ver los datos almacenados en la cadena de bloques, pero solo ciertas personas o entidades tienen permiso para participar en la red y validar transacciones. Las blockchains público/permisionadas a menudo se utilizan en aplicaciones empresariales y en aplicaciones gubernamentales y se consideran una opción más segura y privada que las blockchains públicas, a costa de una menor descentralización. Además, a menudo tienen un rendimiento más alto y una mayor escalabilidad que las blockchains públicas, lo que las hace más adecuadas para aplicaciones de alta demanda. Ejemplo de este tipo de redes son frameworks como LACChain, Alastria, etc.

## 8.5 Descentralización

En blockchain, la descentralización se refiere a dispersar el control y/o toma de decisiones de una entidad centralizada (individuo, organización o grupo de los mismos) a una red más descentralizada. Las redes descentralizadas se esfuerzan por minimizar el nivel de confianza que los participantes deben depositar entre sí y disuadir su capacidad de ejercer autoridad o control sobre los demás de maneras que degraden la funcionalidad de la red. Se observa en la figura 8.2 como la en red descentralizada no hay ningún ente, control o autoridad central, lo cual es la característica que las diferencias de los sistemas distribuidos, similares en la topología [75].

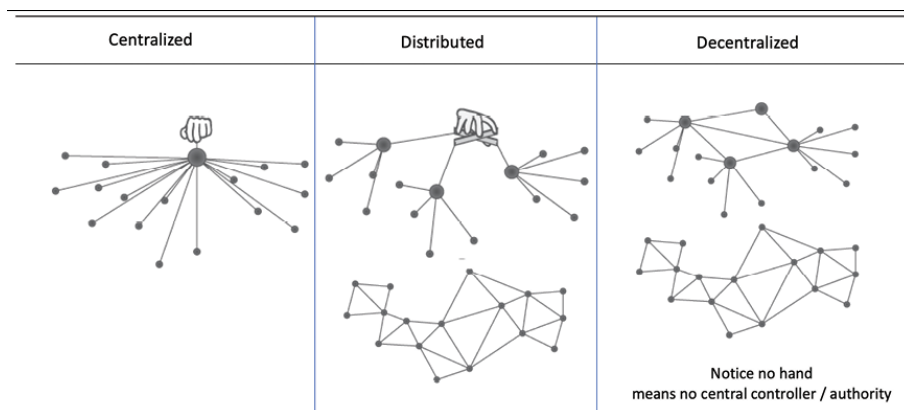


Figura 8.2: Diferentes tipos de red (centralizada, distribuida y descentralizada)[59]

Vitalik Buterin en su artículo "The Meaning of Decentralization"[76] define 3 ejes diferentes al referirnos a la descentralización:

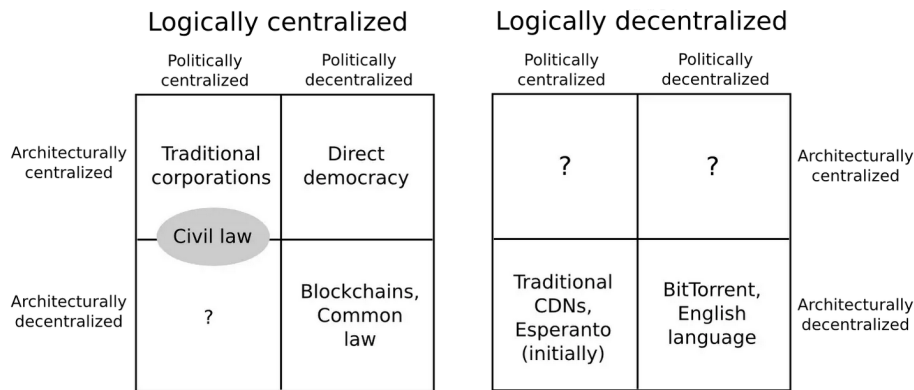
- Descentralización arquitectural: ¿De cuántas computadoras físicas está compuesto un sistema? ¿Cuántas computadoras pueden estar rotas en un momento dado sin que el sistema se resienta?
- Descentralización política: ¿Cuántos individuos u organizaciones controlan las computadoras de las que se compone el sistema?
- Descentralización lógica: ¿Las interfaces y estructuras de datos del sistema se parecen más a un único objeto monolítico o a una colmena amorfa? Una simple regla es: si cortaras el sistema en dos, ¿las dos mitades seguirían operando como unidades independientes?

Podemos poner estas tres dimensiones en un gráfico:

Las cadenas de bloques públicas están políticamente descentralizadas (nadie las controla) y arquitectónicamente descentralizadas (no hay un punto central de falla de infraestructura), pero están lógicamente centralizadas (hay un estado comúnmente acordado y el sistema se comporta como una sola computadora).

El artículo también propone tres argumentos que responden a la pregunta de por qué la descentralización es realmente útil. Estos argumentos son:

- Tolerancia a fallas: Los sistemas descentralizados tienen menor probabilidad de fallar acci-



**Figura 8.3:** Dimensiones de la descentralización[76]

dentalmente porque se forman de varios componentes separados. ¿Qué es menos probable? ¿Que falle una computadora, o que 5 de 10 computadoras fallen al mismo tiempo? Este principio no es controvertido, y se usa en la vida real en muchas situaciones, incluyendo motores de avión, grupos electrógenos de reserva en lugares como hospitales, infraestructura militar y la diversificación del portafolios financiero.

- Resistencia a ataques: Los sistemas descentralizados son más costosos de atacar, destruir o manipular, porque no tienen puntos centrales sensibles que puedan ser atacados.
- Resistencia a la colusión: Es mucho más difícil para los participantes de sistemas descentralizados colusionarse de maneras que los beneficien a expensas de los otros participantes, mientras que los líderes de corporaciones y gobiernos pueden colusionarse de formas que los benefician pero que perjudican a ciudadanos con menor coordinación, clientes y empleados.

Algunas ejemplos de los beneficios que puede proveer blockchain, vinculados a la descentralización, son los siguientes [77]:

El actual panorama de la web 2.0, la monopolización por parte de Silicon Valley de todos los datos de los usuarios, puede comenzar a redistribuirse hacia los usuarios individuales en la Web 3.0

La descentralización social facilitada por blockchain y las DAOs, puede potencialmente redistribuir y volver a democratizar los patrones de participación y cooperación humana.

Las cadenas de bloques no están controladas por una autoridad central, sino por toda la red de participantes, quienes establecen las reglas de participación por sí mismos y pueden elegir hacer evolucionar el sistema de acuerdo con el consenso; esto los hace resistentes a la censura e inherentemente más elásticos que la mayoría de los otros mecanismos de toma de decisiones para grandes grupos de personas.

## 8.6 Mecanismos de consenso

Un algoritmo de consenso es un procedimiento mediante el cual todos los pares de la red blockchain llegan a un acuerdo común sobre el estado actual del registro distribuido e inmutable. De esta manera, los algoritmos de consenso logran confiabilidad en la red blockchain y establecen confian-

za entre pares desconocidos en un entorno informático distribuido. Esencialmente, el algoritmo de consenso asegura que cada nuevo bloque que se agrega a la blockchain sea la única versión de la verdad acordada por todos los nodos [78].

Esencialmente, el protocolo de consenso asegura que cada nuevo bloque que se agrega a la cadena de bloques es la única versión de la verdad que es aceptada por todos los nodos en la cadena de bloques. El protocolo de consenso de la cadena de bloques consta de algunos objetivos específicos como llegar a un acuerdo, colaboración, cooperación, igualdad de derechos para cada nodo y participación obligatoria de cada nodo en el proceso de consenso. Por lo tanto, un algoritmo de consenso tiene como objetivo encontrar un acuerdo común que sea beneficioso para toda la red.

Los algoritmos de consenso se pueden clasificar en dos categorías amplias[59]:

- Consenso tradicional (basado en votación): son algoritmos de consenso tradicionales que han sido investigados en sistemas distribuidos durante muchas décadas. Ejemplos de estos algoritmos son Paxos y PBFT. También se les conoce como consenso distribuido tolerante a fallos. Los mismos suelen modificarse para su uso en blockchains permisionadas.
- Consenso Nakamoto (basado en sorteo) o post Nakamoto: fue introducido por primera vez con Bitcoin y se le conoce también como consenso de blockchain. Están orientados a blockchains públicas como Bitcoin y Ethereum.

Durante la investigación se examinarán los distintos algoritmos de consenso, como funcionan y cuales se ajustan más al caso de uso planteado. Algunos de los algoritmos más conocidos son: Practical Byzantine Fault Tolerance (PBFT), Proof of Work (PoW), Proof of Stake (PoS), Delegated Proof of Stake (DPoS), Proof of Authority (PoA), Proof of Burn (PoB), Proof of Capacity, Proof of Elapsed Time, Proof of Activity, Proof of Weight, Proof of Importance, Leased Proof of Stake, etc [78].

### 8.6.1 PBFT

PBFT es un práctico algoritmo bizantino tolerante a fallos, propuesto por Miguel Castro y Barbara Liskov en 1999[79], que puede garantizar la corrección del sistema (evitar la bifurcación) cuando el número de nodos maliciosos es inferior a un tercio. En comparación con el algoritmo BFT original, la complejidad del algoritmo se reduce de nivel exponencial a nivel polinomial, lo que hace posible la aplicación práctica del algoritmo BFT.

El algoritmo PBFT se enfoca en resolver el problema del ordenamiento de transacciones en nodos distribuidos que pueden tener errores bizantinos. Bajo la condición de que el número de nodos bizantinos con errores bizantinos represente hasta  $1/3$  del total, el estado de los nodos no bizantinos en el sistema puede ser consistente.

En general, un líder elegido (nodo principal) crea una lista ordenada de transacciones que se transmite a otros nodos de validación, quienes luego las ejecutan. Una vez que se han ejecutado las transacciones, los nodos de validación calculan el código hash para el nuevo bloque que luego se transmite a sus pares. Si dos tercios de los códigos hash recibidos son iguales, el bloque se asigna

a la copia local de la cadena de bloques del nodo. PBFT garantiza la tolerancia a fallas de la red y permite miles de operaciones por segundo con un aumento insignificante en el tiempo de espera.

Supongamos que hay  $3f + 1$  nodos en el proceso de trabajo del algoritmo PBFT ( $f$  es el nodo malicioso); entre  $3f + 1$  nodos, hay 1 nodo maestro,  $2f$  nodos honestos y  $f$  nodos maliciosos. La fase de trabajo del algoritmo PBFT se puede dividir en tres fases: fase previa a la preparación, fase de preparación y fase de envío. En la fase de preparación previa, el nodo maestro envía el número de secuencia asignado y la información de preparación previa a otros nodos. Después de que los otros nodos reciban la información enviado por el nodo maestro y confirme, comienza la fase de preparación. En la fase de preparación, cada nodo envía información de preparación a todos los nodos excepto a sí mismo. Cuando cada nodo recibe  $2f + 1$  piezas de información, todos los nodos de consenso se consideran listos. Entonces, está en la fase de confirmación. Todos los demás nodos transmitirán un mensaje de confirmación a toda la red. Cuando cada nodo recibe  $2f + 1$  mensajes de confirmación, la fase de trabajo de confirmación se completa y se devuelve la información de registro correcta; luego, se registrarán en la cadena de bloques. El flujo de trabajo del algoritmo PBFT se muestra en la Figura 8.4.

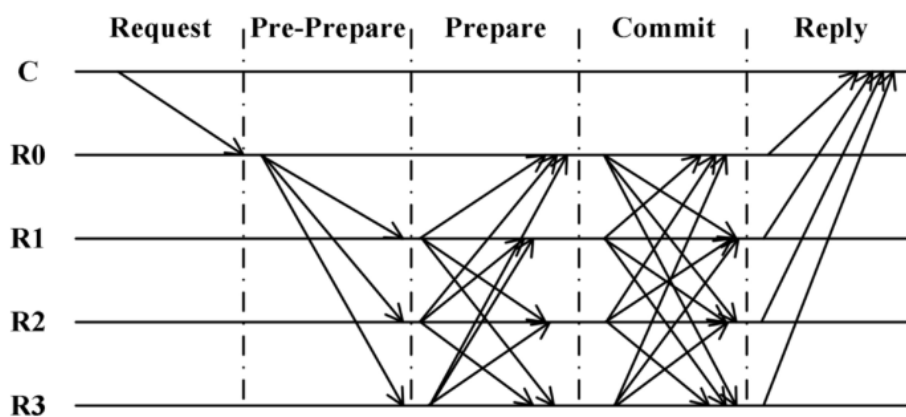


Figura 8.4: Consenso PBFT. Diagrama de flujo[80].

Sin embargo, uno de los principales inconvenientes de PBFT es la capacidad de implementar prácticamente el algoritmo, debido a la enorme cantidad de cálculos necesarios. Sin embargo Tendermint, Diem blockchain, Hashgraph e Hyperledger Fabric logran un consenso basado en PBFT.

### 8.6.2 RAFT

Raft es un algoritmo de consenso distribuido que se utiliza para garantizar que todos los nodos del sistema lleguen a un acuerdo sobre el estado de la red. Este algoritmo se basa en un esquema de liderazgo, en el que un nodo es elegido como líder y los demás nodos son seguidores.

El proceso de consenso en Raft se divide en dos fases: la elección del líder y la replicación de la información. En la fase de elección del líder, los nodos compiten entre sí para ser elegidos como líder. Una vez que un nodo es elegido como líder, comienza la fase de replicación de la información. El líder recibe transacciones de los nodos clientes y las agrega a un bloque, que luego envía a los nodos seguidores para su verificación. Si los nodos seguidores verifican la transacción y el bloque

y envían una confirmación al líder, este agrega el bloque a la cadena de bloques y lo envía a todos los nodos.

Una de las ventajas de Raft es que es más sencillo de implementar y entender que otros algoritmos de consenso, como Paxos o PBFT. Además, es más rápido que estos algoritmos, ya que solo un nodo (el líder) tiene que interactuar con las transacciones en lugar de tener que realizar un proceso de votación o confirmación entre todos los nodos.

En resumen, Raft es un algoritmo de consenso distribuido conocido por su utilización en el servicio de ordenamiento de Hyperledger Fabric, que es responsable de ordenar las transacciones y crear bloques de transacciones que se añaden a la cadena de bloques con el fin de garantizar que todos los nodos del sistema lleguen a un acuerdo sobre el estado de la red. Raft se basa en un esquema de liderazgo donde se asume que el líder es siempre honesto, es más sencillo de implementar y más rápido que otros algoritmos de consenso.

### **8.6.3 Proof of Work (PoW) ó Prueba de trabajo**

El algoritmo de prueba de trabajo (PoW), se diseñó originalmente en 1992 para resistir los ataques de spam, cuando Cynthia Dwork y Moni Naor publican el paper “Pricing via Processing Combating Junk de of Mail” [81]. Este se considera el primer uso del algoritmo de prueba de trabajo, en donde el remitente del correo debe gastar una cierta cantidad de poder de cómputo para resolver un problema matemático antes de enviar un correo electrónico, lo que aumenta en gran medida el costo del envío de spam, lo que garantiza la seguridad del sistema. Sin embargo, es en el whitepaper de Bitcoin[82], donde Satoshi Nakamoto<sup>1</sup> usó por primera vez un algoritmo de consenso bizantino tolerante a fallas en una red blockchain abierta y pública[83].

La idea central del algoritmo PoW es garantizar la consistencia de los datos y la seguridad del consenso a través de la competencia de poder de cómputo. Los actores o participantes de la red usan este poder computacional para encontrar la solución de una compleja operación matemática, ganar el derecho de proponer el próximo bloque de transacciones a la cadena y obtener así la recompensa predefinida y las tarifas de transacción.

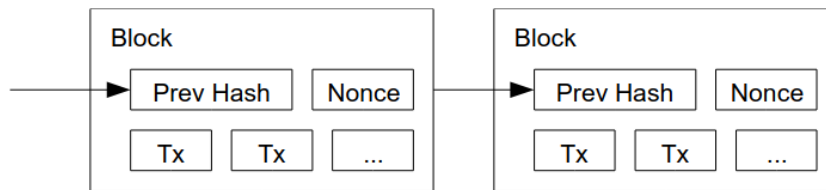
En detalle cada nodo de la blockchain utiliza su poder de cómputo para calcular un valor de nonce aleatorio<sup>2</sup>, que produzca un hash del bloque que cumpla con el valor objetivo del sistema. Si este resultado es menor que el objetivo, el nodo obtiene el derecho de agregar ese bloque y lo transmite empaquetado a toda la red. Los demás nodos después de recibir el bloque empaquetado verificarán rápidamente que el mismo cumplió el objetivo, y dejan de resolver el problema matemático para agregar el bloque recibido a la blockchain, y así continuar en busca del bloque siguiente y garantizar

---

<sup>1</sup>Bitcoin fue creado por una persona o grupo de personas bajo el seudónimo de Satoshi Nakamoto. La verdadera identidad de Satoshi Nakamoto sigue siendo desconocida, y ha sido objeto de mucha especulación. Lo que sí se sabe es que Satoshi Nakamoto publicó el whitepaper de Bitcoin en 2008, y luego lanzó la primera versión del software de Bitcoin en 2009.

<sup>2</sup>Nonce: Se utiliza en la minería de Bitcoin para crear nuevos bloques y validar transacciones. Los mineros de Bitcoin utilizan sus computadoras para resolver un problema matemático complejo, y el nonce es uno de los elementos que se utilizan para resolver este problema. Cada vez que se intenta resolver el problema, se cambia el nonce y se vuelve a intentar. Si se encuentra una solución, el bloque se considera válido y se añade a la cadena de bloques de Bitcoin.

la consistencia y corrección de los datos en el registro de transacciones.



**Figura 8.5:** Diagrama de la cadena Bitcoin. Se observa el valor de nonce que se utiliza para minar un bloque y como cada bloque posee en su interior el hash del bloque anterior[82].

El proceso de resolver el rompecabezas es computacionalmente intensivo y de naturaleza estocástica. Encontrar una solución se asemeja a un minero que descubre oro; por lo tanto, el proceso de hashing se conoce como minería y el nodo que lo realiza se conoce como minero. Para tener más probabilidades de resolver el acertijo y obtener el premio, los mineros buscarán permanentemente realizar una mayor cantidad de trabajo, por lo que se requiere tener el mayor poder computacional, lo que ha llevado a una intensificación en la competencia por mejorar los recursos, dando origen a granjas o pool de servidores, ya que la minería de datos requiere hardware informático altamente especializado, con los considerables costos asociados de energía en el uso de los procesadores y la refrigeración.

PoW es un mecanismo de consenso ideal para blockchains públicas, ya que en el proceso de participación no es necesario introducir un mecanismo de verificación de identidad. Cualquier entidad dispuesta e interesada puede participar en la minería utilizando su propio poder de cómputo, lo que lleva a que PoW consuma una inmensa cantidad de energía, que si bien garantizan la seguridad de la red, no es aplicable en ningún otro lado y por lo tanto se desperdicia. Los mineros prueban y descartan millones de nonces cada segundo y la energía requerida por la red Bitcoin para funcionar es equivalente a la de un país como Bangladesh[84]. Otro de los inconvenientes de PoW es que el período de acuerdo de consenso es demasiado largo, actualmente se tarda unos 10 minutos en resolver el puzzle matemático y el rendimiento de las transacciones es demasiado bajo.

#### 8.6.4 Proof of Stake (PoS) ó Prueba de Participación

Tras el éxito de Bitcoin, muchas cadenas de bloques han utilizado PoW, sin embargo, la explosión de la investigación en algoritmos de consenso también ha resucitado la prueba de participación, también llamado PoS (del inglés proof of stake) una alternativa más amigable con el medioambiente. Históricamente la prueba de trabajo no fue el primer algoritmo de consenso propuesto, la PoW se inventó como una alternativa a la prueba de participación.

PoS es un protocolo de consenso para redes distribuidas que asegura una red mediante la participación financiera. A diferencia de PoW, que recompensa a los participantes que resuelven encriptaciones criptográficas para validar transacciones y poder crear nuevos bloques (es decir, minería); en PoS un grupo de validadores se turnan para proponer y votar el siguiente bloque, y el peso del



voto de cada validador depende del tamaño de su depósito. Es decir, los validadores apostarán con la criptomoneda base de la cadena de bloques y la seguridad de la red depende de esos depósitos y del interés económico de los validadores de red[85].

Es importante destacar que un validador corre el riesgo de perder parte de su depósito si el bloque propuesto es rechazado por la mayoría de los validadores. Por el contrario, los validadores obtienen una pequeña recompensa, proporcional a su participación depositada, por cada bloque aceptado por la mayoría. Por lo tanto, PoS obliga a los validadores a actuar con honestidad y seguir las reglas de consenso, mediante un sistema de recompensa y castigo.

Las ventajas significativas de PoS incluyen seguridad, mejorar la escalabilidad de la red y reducir el consumo de electricidad. Algunas de las cadenas de bloques que utilizan el protocolo PoS o variantes del mismo son : Ethereum, Tezos, Cardano, EOS, Cosmos y TRON. Además Ethereum, una de las redes blockchain más relevantes, migró su red desde PoW a PoS durante septiembre del 2022.

Algunas variantes de Proof of Stake son Chain-based PoS, Committee-based PoS y Delegated PoS. A continuación se describen algunas de estas variantes.

### **8.6.5 Committee-based PoS**

El Protocolo de PoS basado en comités es un algoritmo de consenso utilizado en algunas criptomonedas para validar las transacciones y agregarlas a la cadena de bloques. En lugar de requerir la resolución de problemas computacionales costosos para validar las transacciones, como lo hace el Protocolo PoW, PoS utiliza un sistema de "participación" o apuesta para determinar quién puede validar las transacciones.

En un PoS basado en comités, un grupo de nodos validadores (conocidos como un "comité") son seleccionados para validar un bloque de transacciones. Estos nodos son seleccionados en base a la cantidad de "participación" (o "stake") que han hecho en la red. Cuanto mayor sea la cantidad de participación que un nodo tiene, mayor será su probabilidad de ser seleccionado para formar parte del comité. Algunos algoritmos de PoS basado en comités utilizan una función verificable aleatoria (VRF) para seleccionar de manera aleatoria a los nodos del comité.

Una vez que un comité ha sido seleccionado, ellos son responsables de validar el próximo bloque de transacciones y agregarlo a la cadena de bloques. Si un nodo del comité intenta agregar un bloque inválido a la cadena, su participación puede ser confiscadas como castigo.

Ouroboros es un algoritmo de consenso PoS basado en comités que se utiliza en la criptomoneda Cardano.

### **8.6.6 DPoS - Delegated PoS**

En este tipo de PoS, los usuarios de la red votan por nodos que se convierten en validadores (también conocidos como "delegados") que son responsables de validar transacciones y crear nuevos bloques. Los votos son proporcionales a la cantidad de participación (también conocida como "sta-

ke”) que tienen los usuarios en la red.

DPoS se parece en algunos aspectos a PoS committee-based, pero con una diferencia crucial: en lugar de utilizar una función aleatoria para determinar el grupo de participantes, el grupo es elegido mediante la delegación de participación. Los delegados son seleccionados por votación y forman un comité fijo de validadores que crean bloques de manera rotativa.

Una de las principales ventajas de DPoS es que es más rápido y eficiente que otros algoritmos de consenso, ya que utiliza un pequeño grupo de validadores conocidos para proponer y crear bloques. Sin embargo, DPoS también se ha criticado por no ser tan descentralizado como otros algoritmos de consenso, ya que un pequeño grupo de validadores conocidos es responsable de la mayor parte del procesamiento de transacciones.

Este tipo de PoS se utiliza en algunas cadenas de bloques, como EOS y Tezos.

### **8.6.7 Proof of Authority ó Prueba de Autoridad**

En el mecanismo de consenso de Prueba de Autoridad (PoA), existen una cantidad predeterminada de nodos que tienen permisos para agregar nuevos bloques; los que se seleccionan en base a la certeza que se posee sobre su identidad, lo que significa que un validador de bloques no está apostando criptomonedas, sino su propia reputación[86].

PoA utiliza una cantidad limitada de validadores de bloques, ya que los mismos deben ser verificados, confiables y seleccionados por la red, lo que hace que el sistema sea altamente escalable. Estos nodos validadores o selladores, son los únicos habilitados para crear nuevos bloques, y deben ser aceptados previamente por votación de la mayoría de los nodos ya establecidos, lo que le brinda el control total sobre qué nodos serán selladores en la red. Además, en caso de que un nodo no esté disponible, al estar preautenticados, la red puede eliminarlo del proceso de validación.

A raíz del número limitado de validadores que se requieren, la red tiene un muy alto rendimiento y no necesitan tener una gran capacidad de cómputo (para resolver algoritmos complejos), en consecuencia son muy eficientes energéticamente. Debido a que no hay recompensa económica por la participación en el proceso de agregar nuevos bloques a la cadena, usualmente estos modelos no tienen criptomonedas asociadas, por lo tanto las tarifas de transacción suelen ser nulas.

Si bien PoA es un método más ecológico que PoW y es más rápido, la red siempre estará limitada por el hecho de que, por diseño, no se puede usar de manera descentralizada. PoA tiende hacia una fuerte centralización ya que toda la red está en manos de un pequeño número de actores, creando una solución práctica y eficiente para las redes blockchain privadas.

Algunos de los algoritmos de Prueba de Autoridad (PoA) ampliamente utilizados son Clique y Aura. Clique se utilizaba previamente en la red de prueba Rinkeby y en la red de validación de Goerli, aunque cabe destacar que Rinkeby ha sido marcada como obsoleta.

### 8.6.8 Tower Byzantine Fault Tolerance - Tower BFT

El mecanismo Proof-of-History (PoH) es la innovación principal presentada por la blockchain Solana. Este pretende agilizar el proceso de consenso de los nodos de la red en el procesamiento de los bloques, proporcionando un medio para codificar el tiempo en sí mismo dentro de la misma cadena de bloques. En una cadena de bloques PoW, llegar a un consenso sobre el momento en que se minó un bloque específico es tan necesario como llegar a un consenso sobre la existencia de las transacciones en ese bloque. Con PoH tenemos un método de validación que demuestra el momento en que las transacciones tuvieron lugar y la secuencia particular de las mismas[87].

Mientras que otras Blockchains requieren que los validadores se comuniquen entre sí para acordar que ha pasado en la red en el tiempo, cada nodo de Solana mantiene su propio reloj al codificar el paso del tiempo en una simple función de VDF (sequential hashing verifiable delay function). Se puede pensar a PoH como la solución descentralizada para lograr el consenso sobre el tiempo en los sistemas distribuidos, dada la inexistencia de un reloj central.

La clave de la función VDF es que sólo puede ser resuelta por un único núcleo de CPU aplicando un conjunto particular de pasos secuenciales. No se permite el procesamiento en paralelo, por lo que es fácil definir exactamente el tiempo que se tarda en aplicar esos pasos.

Solana combina PoH con un protocolo de seguridad llamado Tower Byzantine Fault Tolerance (Tower BFT) con el fin de evitar a los actores maliciosos. Permite a los participantes stakear e inmovilizar tokens, para poder votar sobre la validez de un hash de PoH. Este protocolo penaliza a los malos actores, con la reducción de su participación, si se descubre que votan a favor de una bifurcación que no coincide con los registros del PoH [88]. Además, Solana despliega el Proof-of-Stake (PoS) como medio para determinar quién puede participar como validador de bloques.

## 8.7 Escalabilidad de una blockchain

La escalabilidad en blockchain se refiere a la capacidad de una red de blockchain para manejar un gran volumen de transacciones y usuarios sin disminuir significativamente su velocidad o rendimiento. Es uno de los desafíos más importantes en la adopción generalizada de blockchain, ya que las redes blockchain actuales, como Bitcoin y Ethereum, tienen limitaciones en cuanto al número de transacciones que pueden procesar por segundo.

Existen varios enfoques para escalar una blockchain, entre ellos:

- **Off-chain Transactions:** Esta técnica implica llevar transacciones fuera de la cadena principal, lo que permite aumentar la capacidad de procesamiento de transacciones, ya que una transacción off-chain no tiene que ser registrada en la cadena principal, lo que reduce la carga en la red principal y aumenta la velocidad de las transacciones.
- **Sharding:** Es una técnica que consiste en dividir una base de datos en varias partes, conocidas como shards, que se almacenan en diferentes servidores. En el contexto de blockchain, el sharding se refiere a la división de una red de nodos en varios grupos, cada uno de los cuales procesa un subconjunto de transacciones. Esto permite aumentar significativamente

la capacidad de procesamiento de transacciones.

- Capas adicionales: También conocido como "Layer 2" se refiere a soluciones que operan "encima" de una blockchain existente para aumentar su capacidad. Un ejemplo de una solución de capa 2 es los "canales de pago" ( Payment Channels), que permiten a dos partes realizar un número ilimitado de transacciones entre ellas sin tener que registrarlas en la blockchain principal.
- Consenso Alternativos: Existe una variedad de algoritmos de consenso, cada uno con diferentes ventajas y desventajas en términos de escalabilidad, seguridad y descentralización.

Es importante mencionar que ninguna solución ofrece una escalabilidad ilimitada, y cada una tiene sus propios desafíos y limitaciones.

# Plataformas blockchain consideradas

## 9.1 Bitcoin

La criptomoneda Bitcoin nace en 2008 como obra de un individuo o grupo anónimo conocido popularmente como Satoshi Nakamoto. En un intento de crear una moneda virtual descentralizada, Bitcoin soluciona el problema del doble gasto empleando una red que permitía que los pagos en línea fueran enviados directamente de una entidad a otra, sin tener que pasar por medio de una institución financiera.

Las tecnologías subyacentes a Bitcoin no eran necesariamente nuevas, pero la unión de todas ellas permitió crear las cadenas de bloques tal cual las conocemos hoy. La red Bitcoin está compuesta por una red de nodos P2P (Peer to Peer) donde cada nodo actúa enviando una serie de transacciones a los demás nodos y escuchando las transacciones generadas por los demás. Los nodos Bitcoin son iguales entre sí, no hay un líder ni diferencia alguna en cuanto a capacidades ni funcionalidades, esto hace que la red pueda operar libremente sin el control de ningún ente central.

Para interactuar con la red, los nodos y los usuarios mediante sus billeteras de Bitcoin, gestionan un número arbitrario de pares de claves criptográficas (pública y privada) que pueden ser generadas sin ningún tipo de restricción. A partir de las claves públicas, se generan hashes que se conocen como direcciones, y se usan como entidades remitentes y receptoras de los pagos, mientras que, a partir de las claves privadas correspondientes, se autorizan los pagos mediante firmas criptográficas[82].

### 9.1.1 Transacciones en la red Bitcoin

Definimos la moneda electrónica bitcoin como una cadena de firmas digitales. Cada dueño transfiere la moneda al próximo al firmar digitalmente un hash de la transacción previa y la clave pública del próximo dueño y agregando estos al final de la moneda. Un beneficiario puede verificar las firmas para verificar la cadena de propiedad.

Bitcoin es un sistema basado en UTXO (siglas en inglés de «Unspent Transaction Output», comúnmente traducido al español como "monedas no gastadas"). Las cantidades de los UTXO están vinculadas a las direcciones que las pueden gastar por medio del registro de la cadena de bloques. Las UTXO son, por así decirlo, las monedas que conforman el "saldo de una determinada cuenta de Bitcoin".

Cabe destacar que, en Bitcoin, a diferencia de Ethereum, no existe el concepto de "saldo de cuenta", sino que son las aplicaciones de cartera las encargadas de crear esta ilusión de balance de una cuenta tradicional. En realidad, los fondos asociados a una dirección Bitcoin, no son más que el total de las transacciones UTXO que son reclamables por una determinada dirección a lo largo de la blockchain.

Una transacción Bitcoin es una estructura de datos con el siguiente formato:

Versión: versión del nodo que crea la transacción. Número de entradas: número de salidas de otras

transacciones que se toman como entradas en la nueva transacción. Entradas: salidas de transacciones previas que toma como valores de entrada la transacción. Número de salidas: número total de las salidas. Salidas: salidas de los fondos Bitcoin mediante las cuales se transfieren fondos a otras direcciones. Locktime: Un sello de tiempo en formato timestamp (Unix Epoch), o una altura de bloque. Este tipo de parámetro se usa para especificar el momento en el que una transacción debe ser validada y transmitida.

### 9.1.2 Red Bitcoin

La red coloca estampas de tiempo a las transacciones al agruparlas en bloques y crear un hash de las mismas en una cadena continua de pruebas de trabajo basadas en hashes, formando un registro que no puede ser cambiado sin volver a recrear la prueba de trabajo.

La cadena más larga no solo sirve como la prueba de la secuencia de los eventos testificados, sino como prueba de que vino del grupo de poder de procesamiento de CPU/GPU más grande. Siempre que la mayoría del poder de procesamiento esté bajo el control de los nodos que no cooperan para atacar la red, estos generarán la cadena más larga y le llevarán la ventaja a los atacantes. La red en sí misma requiere una estructura mínima, los mensajes son enviados bajo la base de mejor esfuerzo, y los nodos pueden desconectarse y volver a unirse a la red como les parezca, aceptando la cadena de prueba de trabajo de lo que sucedió durante su ausencia.

### 9.1.3 RSK: Contratos inteligentes sobre la red Bitcoin

RSK es una plataforma de contratos inteligentes que funciona sobre la red de Bitcoin. RSK utiliza un protocolo de sidechain llamado "two-way peg" para conectar su red a la red de Bitcoin, lo que permite que los usuarios transfieran bitcoins a la red RSK y los utilicen para interactuar con contratos inteligentes.

El lenguaje de programación utilizado para escribir contratos inteligentes en RSK es similar al de Ethereum, ya que se basa en Solidity. Actualmente, RSK es compatible con todos los códigos de operación y los contratos de precompilación de Ethereum y, por lo tanto, puede admitir cualquier lenguaje que compile en EVM. Esto incluye Solidity, Julia y lenguajes de programación nuevos o experimentales como Vyper.

Algunos de los proyectos más populares sobre RSK incluyen:

- RIF OS, que es un conjunto de protocolos y herramientas para facilitar el desarrollo de dApps sobre RSK.
- Rootstock, una plataforma de contratos inteligentes para la creación y ejecución de contratos en el mercado financiero.
- TEMCO, una plataforma de cadena de suministro que utiliza RSK para rastrear la cadena de suministro de los productos a través de la tecnología de IoT.

En resumen, RSK es una plataforma de contratos inteligentes conectada a Bitcoin que es utilizada para construir aplicaciones descentralizadas, sus lenguajes de programación son similares a Ethe-

reum, pero con algunas diferencias. En cuanto a su uso, RSK es menos conocido que Ethereum, y su uso es menos amplio. Sin embargo, todavía existen varios proyectos y dApps construidos sobre RSK, y algunos consideran que la conexión con la red de Bitcoin le da a RSK una ventaja adicional en términos de seguridad y escalabilidad.

## **9.2 Ethereum**

A continuación se describirá de forma más extensa la blockchain de Ethereum, por ser la elegida para el desarrollo de este proyecto.

### **9.2.1 ¿Qué es Ethereum?**

La blockchain Ethereum es considerada una blockchain de segunda generación, esto se debe a que además de tener una moneda intrínseca y permitir realizar transferencias de valor en forma descentralizada, proporciona una plataforma para la creación y ejecución de aplicaciones descentralizadas.

Desde una perspectiva más práctica, Ethereum es una infraestructura informática descentralizada y de código abierto que mantiene un estado único accesible globalmente. Esta máquina de estado determinista es prácticamente ilimitada y ejecuta los contratos inteligentes sobre una máquina virtual (Ethereum Virtual Machine) que aplica cambios a ese estado. Utiliza las llamadas transacciones para producir los cambios de estado y la cadena de bloques para sincronizar el estado del sistema en todos los nodos. Posee una criptomoneda llamada Ether que sirve para pagar el uso de los recursos de ejecución (de la plataforma Ethereum) y limitar el uso de los mismos, brindando seguridad a la red evitando ataques de denegación de servicio.

### **9.2.2 Características de la red Ethereum**

La plataforma Ethereum permite a los desarrolladores crear potentes aplicaciones descentralizadas con funciones de pago integradas. Los pagos nativos son en la moneda ETH y los desarrolladores no necesitan pasar tiempo integrando sistemas con proveedores de pagos de terceras partes.

La red es de acceso abierto (ó “permissionless”), esto quiere decir que las aplicaciones o servicios que corren sobre Ethereum no requieren de registro alguno. Si el usuario posee una dirección de ethereum con fondos y una conexión de internet puede desplegar nuevas aplicaciones u operar con las aplicaciones existentes en la blockchain.

Ethereum es 100% descentralizada, sin propietarios. Una vez desplegado el contrato sobre la red el mismo es inmutable y no se puede modificar, incluso aunque el equipo que desarrollo la aplicación se disolviera, se podría seguir utilizando el mismo contrato con una interfaz diferente. La red además de proporcionar una alta disponibilidad, auditabilidad, transparencia y neutralidad, gracias a su descentralización, es resistente a la censura ó intentos de bloqueo por parte de gobiernos o agentes externos.

### 9.2.3 Historia de la red Ethereum

Ethereum fue creado por Vitalik Buterin, un desarrollador de blockchain canadiense-ruso, en 2013. Buterin había estado trabajando en Bitcoin desde 2011 y se dio cuenta de que la plataforma tenía limitaciones en cuanto a la creación y ejecución de aplicaciones descentralizadas. Así, él y un equipo de desarrolladores comenzaron a trabajar en una nueva plataforma llamada Ethereum, que se enfocaría en proporcionar una plataforma genérica donde cualquiera pudiera crear y ejecutar aplicaciones descentralizadas.

En 2014, Gavin Wood publicó el "Yellow Paper", que es un documento técnico que describe la arquitectura y funcionamiento de Ethereum. El "Yellow Paper" proporciona una descripción detallada de la arquitectura subyacente de Ethereum, incluyendo los contratos inteligentes, el protocolo de consenso, y el mecanismo de tarificación de gas.

La publicación del "Yellow Paper" por parte de Gavin Wood fue un hito importante en el desarrollo de Ethereum ya que ayudó a establecer las bases técnicas para la plataforma y atrajo a un gran número de desarrolladores y entusiastas interesados en trabajar en la misma. Esto llevó a la realización de una oferta inicial de monedas (ICO) para recaudar fondos para financiar el desarrollo de la red, la cual recaudó más de 18 millones de dólares en Ether.

En julio de 2015, Ethereum lanzó su red principal y comenzó a operar como una plataforma independiente. Sin embargo, en junio de 2016, un atacante explotó una vulnerabilidad en el código de The DAO (una organización autónoma descentralizada) que le permitió retirar aproximadamente \$50 millones en Ether. Esto generó un gran revuelo en la comunidad Ethereum y hubo un intenso debate sobre cómo manejar el incidente.

La solución propuesta por la mayoría de la comunidad Ethereum fue una "bifurcación dura" o hard fork, que implicaba crear una nueva cadena de bloques basada en el estado anterior al ataque, y devolver los fondos robados a sus legítimos propietarios. Esta solución fue implementada en julio de 2016, y se conoce como Ethereum (ETH) y se distingue de la cadena de bloques original, que se mantuvo bajo el nombre Ethereum Classic (ETC).

La bifurcación dura fue controvertida ya que algunos argumentaron que violaba la inmutabilidad de la cadena de bloques y que debía permitir que el mercado decidiera el resultado. A pesar de esto, Ethereum ha continuado evolucionando y ha sido ampliamente adoptado en la industria, mientras que Ethereum Classic ha mantenido una menor capitalización de mercado y adopción[89].

Desde que Ethereum experimentó un gran aumento en el precio de su criptomoneda nativa, Ether, en 2017, ha habido un aumento en el interés y el desarrollo en la plataforma. Entre los cambios importantes que han sucedido desde entonces hasta la actualidad, se incluyen:

- Mejora de la seguridad: Ethereum ha mejorado su resistencia contra ataques de denegación de servicio (DoS) mediante la implementación de medidas de seguridad adicionales y actualizaciones en el protocolo.
- Optimización del costo del gas: A través de diversas actualizaciones, se ha trabajado para optimizar el costo del gas de ciertas acciones en la EVM (Máquina Virtual de Ethereum) con



el objetivo de hacer más predecibles los costos para los desarrolladores y usuarios.

- **Contrato de depósito de Stake:** El contrato de depósito de apuesta introdujo el staking en el ecosistema de Ethereum, lo que permite a los usuarios obtener recompensas por el apoyo a la red mediante la validación de bloques.
- **Cadena de baliza:** La cadena de baliza o beaconchain es una cadena paralela a la principal de Ethereum, diseñada para mejorar la velocidad y escalabilidad de la red mediante un sistema basado en "épocas" en lugar de bloques. Su objetivo es proporcionar una base segura para una red de contrato inteligentes escalable y con bajas tarifas de transacción.
- **Y finalmente El Merge** es el cambio importante que se llevo a cabo el 15 de septiembre de 2022, donde se migró de un algoritmo de consenso de prueba de trabajo (PoW) a un algoritmo de consenso de prueba de participación (PoS) y se fusionó la red Ethereum Mainnet original con la cadena de baliza con el objetivo de mejorar la sostenibilidad, escalabilidad, seguridad de la red. Para cambiar de PoW a PoS en Ethereum, fue necesario indicar a la Beacon Chain que aceptara las transacciones de la cadena original de Ethereum, las agrupara en bloques y las organizara en una blockchain utilizando un mecanismo de consenso basado en la prueba de participación. Al mismo tiempo, los clientes originales de Ethereum apagaron su minería, propagación de bloques y lógica de consenso, dejándolo todo en manos de la Beacon Chain.

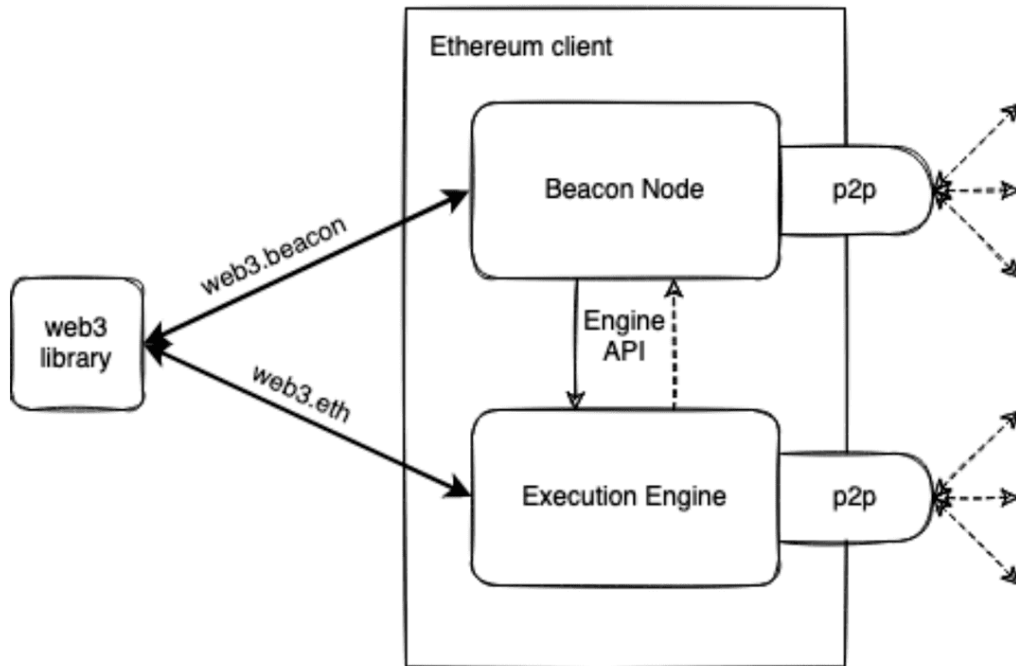
#### **9.2.4 Ecosistema de la red Ethereum**

El ecosistema de Ethereum está compuesto por varios componentes clave. En el centro se encuentra la cadena de bloques Ethereum, que está formada por una red peer-to-peer descentralizada donde los nodos o instancias de software cliente de Ethereum se conectan entre sí y forman la red.

Actualmente existen dos tipos de clientes en la red Ethereum: el cliente de ejecución (Execution Client) y el cliente de consenso (Consensus Client). El cliente de ejecución, también conocido como el motor de ejecución (Execution Engine), escucha las nuevas transacciones que se envían en la red, las ejecuta en la Máquina Virtual de Ethereum (EVM) y mantiene actualizado el registro de transacciones y los datos (estado global de la red Ethereum). El cliente de consenso, también conocido como el Nodo de Cadena de Baliza (Beacon Node), implementa el algoritmo de consenso de proof-of-stake (PoS), que permite a la red llegar a un acuerdo basado en datos validados del cliente de ejecución.

Antes de la implementación del Merge o fusión, la capa de ejecución y la capa de consenso eran redes separadas. Toda la actividad de transacciones y usuarios en Ethereum ocurría en la capa de ejecución. Un solo software de cliente proporcionaba tanto el entorno de ejecución como la verificación de consenso de los bloques producidos por los mineros. La capa de consenso o beaconchain, se ejecutaba de manera separada desde diciembre de 2020, está introdujo el proof-of-stake y coordinó la red de validadores basándose en los datos de la red Ethereum.

Con la implementación del Merge, Ethereum realiza la transición a proof-of-stake conectando estas redes. Los clientes de ejecución y consenso trabajan juntos para verificar el estado de Ethereum.



**Figura 9.1:** Diagrama simplificado de un cliente de consenso y ejecución acoplados.[90]

Además, Ethereum cuenta con bibliotecas (como ether.js o web3.js) que permiten la interacción con el cliente a través de la interfaz de llamada a procedimiento remoto (RPC). Esta interfaz proporciona una forma estándar de comunicarse con el cliente Ethereum y realizar operaciones como enviar transacciones y leer datos de la cadena de bloques.

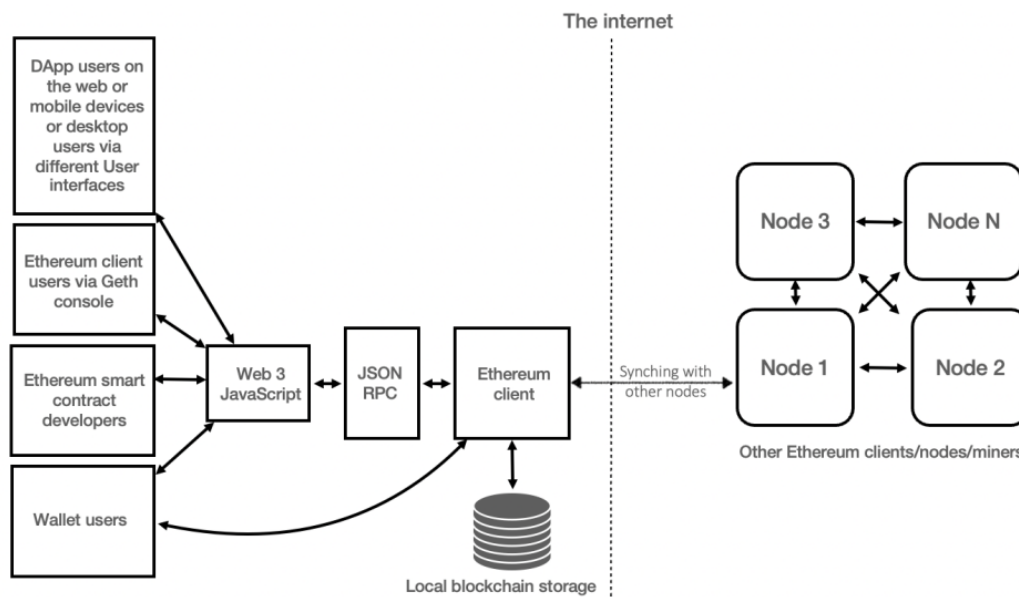
En resumen, el ecosistema está formado por varios componentes interconectados, incluyendo la cadena de bloques de Ethereum, los clientes Ethereum, las bibliotecas y otros protocolos y herramientas. Juntos, estos componentes permiten la creación y ejecución de aplicaciones descentralizadas en la red, que van desde aplicaciones financieras hasta juegos.

A continuación se presenta una breve explicación de los diferentes elementos en la red Ethereum.

#### 9.2.4.1 Claves y direcciones

Las claves y direcciones son la forma en que se identifican y autentican los usuarios en la red Ethereum. Cada usuario tiene una clave privada que se utiliza para firmar transacciones y una dirección pública que se utiliza para recibir pagos e identificar al usuario poseedor de la clave privada correspondiente.

Primero, se elige aleatoriamente una clave privada (un entero positivo de 256 bits) bajo las reglas definidas por la especificación de curva elíptica secp256k1 (en el rango  $[1, \text{secp256k1n} - 1]$ ). Luego, la clave pública se deriva de la clave privada mediante el algoritmo de curva elíptica ECDSA y las direcciones se derivan de la clave pública, específicamente, de los 160 bits más a la



**Figura 9.2:** Diagrama simplificado de un cliente de consenso y ejecución acoplados.[91]

derecha del hash Keccak del llave pública.

A continuación se muestra un ejemplo de cómo se ven las claves y las direcciones en Ethereum generadas con el comando `helpeth`:

Address: 0xe0defb92145fef3c3a945637705fafd3aa74a241

Address (checksum): 0xe0DefB92145FeF3c3a945637705fAfd3AA74a241

Public key: 0x93e39cde5cdb3932e204cdd43b89578ad58d7489c31cbc30e61d167f67e3c8e76b9b2249377fa84f73b11

Private key: 0xba1488fd638adc2e9f62fc70d41ff0ffc0e8d32ef6744d801987bc3ecb6a0953

#### 9.2.4.2 Cuentas (Addresses)

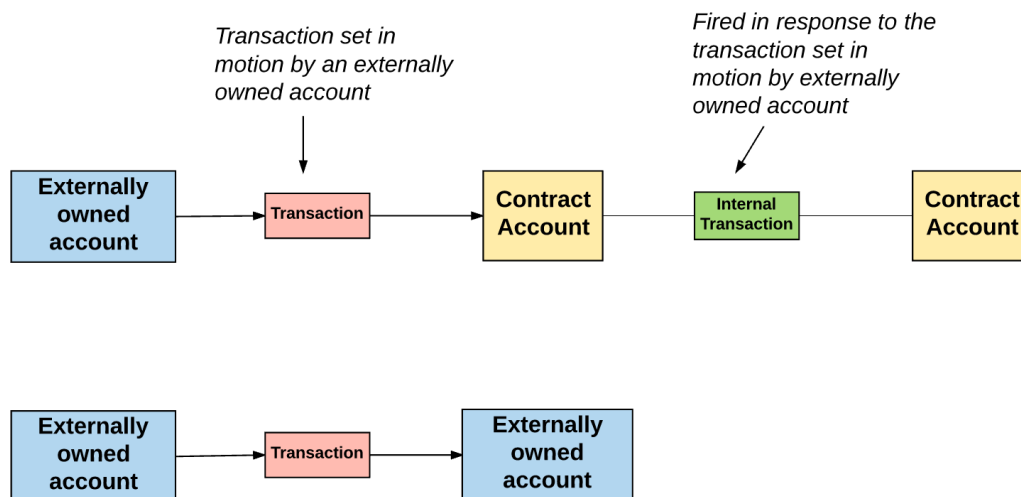
En Ethereum, la “estado compartido” global está compuesto por objetos llamados “cuentas” que pueden interactuar entre sí mediante un marco de mensajes. Cada cuenta tiene un estado asociado y una dirección en Ethereum, que es un identificador de 20 bytes (160 bits) que se utiliza para identificar una cuenta.

Existen dos tipos de cuentas:

- Cuentas controladas externamente (EOA), que son controladas por claves privadas y no tienen código asociado con ellas.
- Cuentas de contrato, que tienen código asociado con ellas y son controladas por su código de contrato.

Es importante entender la diferencia fundamental entre las cuentas controladas externamente y las cuentas de contrato. Una cuenta controlada externamente puede enviar mensajes a otras cuentas controladas externamente o a cuentas de contrato mediante la creación y firma de una transacción

utilizando su clave privada. Un mensaje entre dos cuentas controladas externamente es simplemente una transferencia de valor. Pero un mensaje desde una cuenta controlada externamente a una cuenta de contrato activa el código de la cuenta de contrato, permitiéndole realizar varias acciones (por ejemplo, transferir tokens, escribir en el almacenamiento interno, crear nuevos contratos, etc.).



**Figura 9.3:** Diagrama simplificado de un cliente de consenso y ejecución acoplados.[92]

A diferencia de las cuentas controladas externamente, las cuentas de contrato no pueden iniciar nuevas transacciones por sí solas. En su lugar, las cuentas de contrato solo pueden enviar transacciones en respuesta a otras transacciones que han recibido (de una cuenta controlada externamente o de otra cuenta de contrato). Por lo tanto, cualquier acción que ocurre en la cadena de bloques de Ethereum siempre es iniciada por transacciones enviadas desde cuentas controladas externamente[92].

#### 9.2.4.3 Abstracción de cuentas ERC-4337

Hoy en día para que algo suceda en la cadena de bloques, una transacción debe ser iniciada y pagada por una EOA. El control de una EOA se realiza a través de su clave privada, lo que hace que sea vulnerable, ya que si la pierdes, pierdes tu cuenta y todos tus fondos.

La abstracción de cuentas unifica las cuentas de contrato y las EOAs, lo que hace que las cuentas de usuario sean más programables. Se elimina la lógica de firmar transacciones de la cuenta, lo que desbloquea mucha más flexibilidad, como la posibilidad de implementar billeteras de múltiples firmas, autenticación de dos factores, límites de retiro y caducidad de claves. Esto permite que la experiencia de usuario en blockchain sin custodia sea escalable.

#### 9.2.4.4 Contratos inteligentes

Los contratos inteligentes son programas informáticos que se ejecutan dentro de la red Ethereum. Estos contratos son autónomos y se ejecutan sin la necesidad de una entidad central. Pueden ser

utilizados para crear aplicaciones descentralizadas, para automatizar procesos de negocios y para crear nuevos tipos de organizaciones.

En Ethereum, los contratos inteligentes son un tipo de cuenta. Esto significa que tienen una dirección, un saldo y pueden enviar transacciones por la red. Sin embargo, no están controlados por un usuario, sino que están implementados en la red y se ejecutan de acuerdo a como se hayan programado. Las cuentas de usuario pueden interactuar con un contrato inteligente enviando transacciones que ejecuten una función definida en el contrato inteligente. Los contratos inteligentes pueden definir reglas, como un contrato normal, y automáticamente se ejecutan a través del código.

#### **9.2.4.5 Éter criptomoneda/tokens:**

El Ether es la criptomoneda nativa de Ethereum y se utiliza como una forma de pago para las transacciones y los contratos inteligentes en la red. También hay otros tokens que se pueden crear y negociar en la red, como los tokens ERC-20.

Los tokens ERC-20 son tokens que se crean y operan en la cadena de bloques de Ethereum e incluyen una interfaz estándar que especifica cómo se deben implementar ciertas funciones, lo que permite que los tokens sean compatibles con múltiples aplicaciones y billeteras. La interfaz ERC-20 incluye funciones como el `balanceOf` (que devuelve la cantidad de tokens que una dirección de Ethereum en particular tiene), `transfer` (que permite transferir tokens de una dirección a otra), `approve` (que permite a una dirección aprobar que otra dirección retire tokens en su nombre) y `transferFrom` (que permite que una dirección aprobada retire tokens en nombre de otra dirección). La interfaz estandarizada hace que los tokens ERC-20 sean más fáciles de integrar en aplicaciones y permite que los desarrolladores creen aplicaciones que funcionen con múltiples tokens ERC-20 sin tener que escribir código específico para cada uno.

#### **9.2.4.6 Transacciones y mensajes**

Ethereum es una máquina de estado basada en transacciones, y son las transacciones entre diferentes cuentas lo que modifican el estado global de Ethereum de un estado a otro. Una transacción es una pieza de instrucción criptográficamente firmada que es generada por una cuenta controlada externamente (EOA), se serializa y luego se envía a la blockchain[92].

Hay dos tipos de transacciones: llamadas de mensaje y creaciones de contratos (es decir, transacciones que crean nuevos contratos Ethereum).

Todas las transacciones contienen los siguientes componentes, independientemente de su tipo:

- **Nonce:** un contador del número de transacciones enviadas por el remitente.
- **GasPrice:** la cantidad de Wei que el remitente está dispuesto a pagar por unidad de gas necesario para ejecutar la transacción.
- **GasLimit:** la cantidad máxima de gas que el remitente está dispuesto a pagar por la ejecución de esta transacción. Esta cantidad se establece y se paga por adelantado, antes de que se realice cualquier cálculo.

- To: la dirección del destinatario. En una transacción de creación de contrato, la dirección de la cuenta del contrato aún no existe, por lo que se utiliza un valor vacío.
- Value: la cantidad de Wei que se transferirá del remitente al destinatario. En una transacción de creación de contrato, este valor sirve como el saldo inicial dentro de la cuenta del contrato recién creado.
- v, r, s: se utilizan para generar la firma que identifica al remitente de la transacción.
- Init (sólo existe para transacciones de creación de contratos): un fragmento de código EVM que se utiliza para inicializar la nueva cuenta del contrato. Init se ejecuta solo una vez, y luego se descarta. Cuando se ejecuta init por primera vez, devuelve el cuerpo del código de cuenta, que es la pieza de código que se asocia permanentemente con la cuenta del contrato.
- Data (campo opcional que solo existe para llamadas de mensaje): los datos de entrada (es decir, los parámetros) de la llamada de mensaje.

La "carga útil" principal de una transacción está contenida en dos campos: valor(value) y datos(data). Las transacciones pueden tener valor y datos, solo valor, solo datos o ni valor ni datos. Las cuatro combinaciones son válidas. Una transacción con solo valor es un pago. Una transacción con solo datos es una invocación de una función. Una transacción con valor y datos es tanto un pago como una invocación[70].

En la sección "Cuentas" aprendimos que las transacciones, tanto las llamadas de mensaje como las transacciones de creación de contrato, siempre son iniciadas por cuentas controladas externamente y enviadas a la blockchain. Otra forma de pensar en ello es que las transacciones son lo que conectan el mundo externo con el estado interno de Ethereum.

Pero esto no significa que los contratos no puedan hablar con otros contratos. Los contratos que existen dentro del ámbito global del estado de Ethereum pueden comunicarse con otros contratos dentro de ese mismo ámbito. La forma en que hacen esto es a través de "mensajes" o "transacciones internas" a otros contratos. Podemos pensar en los mensajes o transacciones internas como similares a las transacciones, pero con la diferencia principal de que no son generados por cuentas controladas externamente. En cambio, son generados por contratos.

#### 9.2.4.7 Tarifas y gas

Un concepto muy importante en Ethereum es el concepto de tarifas. Cada cálculo que se produce como resultado de una transacción en la red Ethereum incurre en una tarifa. Esta tarifa se paga en una denominación llamada "gas". El gas es la unidad utilizada para medir las tarifas requeridas para un cómputo particular.

El precio del gas es la cantidad de Ether que se está dispuesto a gastar en cada unidad de gas y se mide en "gwei". "Wei" es la unidad más pequeña de Ether, donde  $1 * 10^{18}$  Wei representa 1 Ether.

Con cada transacción, un remitente establece un límite de gas y un precio de gas. El producto del precio del gas y el límite del gas representa la cantidad máxima de Wei que el remitente está

dispuesto a pagar por ejecutar una transacción.

La EIP-1159 introduce un nuevo tipo de transacción (0x02 || rlp) que incluye dos nuevos campos: `max_priority_fee_per_gas` y `max_fee_per_gas`. La `max_priority_fee_per_gas` es la tarifa máxima que un usuario está dispuesto a pagar para que su transacción sea procesada con prioridad. La `max_fee_per_gas`, por otro lado, es la tarifa máxima que un usuario está dispuesto a pagar en total por su transacción, incluyendo la tarifa de prioridad.

En EIP-1159 se incluye una tarifa base por gas que se quema y que se ajusta según una fórmula que depende del gas utilizado en el bloque anterior y el objetivo de gas del bloque anterior (es decir, el límite de gas del bloque dividido por un multiplicador de elasticidad). Este algoritmo hace que la tarifa base por gas aumente cuando los bloques superan el objetivo de gas y disminuya cuando los bloques están por debajo del objetivo de gas.

#### 9.2.4.8 EVM

La Máquina Virtual de Ethereum (EVM) es la plataforma de ejecución de contratos inteligentes en la red Ethereum. Es una máquina virtual que interpreta y ejecuta el código de los contratos inteligentes.

El estado de Ethereum es una gran estructura de datos que contiene no solo todas las cuentas y saldos, sino también un estado de la máquina que puede cambiar de bloque a bloque según un conjunto predefinido de reglas y que puede ejecutar código de máquina arbitrario. Las reglas específicas de cambio de estado de bloque a bloque son definidas por la EVM[93].

La EVM se comporta como lo haría una función matemática: dado un input, produce una salida determinística. Por lo tanto, es muy útil describir formalmente a Ethereum como teniendo una función de transición de estado:

$$Y(S, T) = S'$$

Dado un estado antiguo válido (S) y un nuevo conjunto de transacciones válidas (T), la función de transición de estado de Ethereum  $Y(S, T)$  produce un nuevo estado de salida válido  $S'$ .

En el contexto de Ethereum, el estado es una enorme estructura de datos llamada Modified Merkle Patricia Trie, que mantiene todas las cuentas vinculadas por hashes y reducibles a un solo hash raíz almacenado en la cadena de bloques.

La creación de contratos resulta en la creación de una nueva cuenta de contrato que contiene el bytecode del contrato inteligente compilado. Cada vez que otra cuenta hace una llamada de mensaje a ese contrato, ejecuta su bytecode.

Durante la ejecución, la EVM mantiene una memoria transitoria (como una matriz de bytes direccionada por palabras) que no persiste entre transacciones. Los contratos, sin embargo, contienen un trie de almacenamiento de Merkle Patricia (como una matriz de palabras direccionada por palabras), asociado con la cuenta en cuestión y parte del estado global.

El bytecode del contrato inteligente compilado se ejecuta como una serie de opcodes de EVM, que

realizan operaciones estándar de pila como XOR, AND, ADD, SUB, etc. La EVM también implementa una serie de operaciones de pila específicas de blockchain, como ADDRESS, BALANCE, BLOCKHASH, etc.

### **9.3 Escalabilidad de la Blockchain Ethereum**

Ethereum ha alcanzado la capacidad actual de la red con más de 1.9 millones de transacciones al día en diciembre del 2022. El éxito de Ethereum y el aumento en la demanda, ocasiona la congestión de la red, el aumento de las tarifas de gas y se excluye a los usuarios que no pueden pagar las elevadas tarifas. Por lo tanto, la necesidad de soluciones de escalado también ha ido en aumento. Las soluciones de escalabilidad de ethereum se puede dividir en dos enfoques diferentes, que a futuro se podrán implementar en forma conjunta: las soluciones de escalado en cadena (on-chain) y escalado fuera de la cadena (off-chain).

#### **9.3.1 Soluciones de escalado on-chain**

El método de escalado en cadena ("on-chain") requiere cambios en el protocolo de Ethereum (red principal de capa 1). El enfoque principal para este método de escalado es actualmente la fragmentación.

Fragmentación (o Sharding) es una técnica para mejorar la escalabilidad de una red de blockchain al dividirla en piezas más pequeñas y manejables, llamadas shard("fragmentos"). Cada shard es una cadena separada e independiente que puede procesar transacciones en paralelo con los demás shards. Esto permite un mayor rendimiento general y reduce la carga en los nodos individuales de la red. En el contexto de Ethereum, el sharding es una solución propuesta que todavía se está investigando y desarrollando para abordar las limitaciones de escalabilidad de la actual red Ethereum mainnet.

Originalmente, el plan era trabajar en el sharding antes de "The Merge" (La Fusión) para abordar la escalabilidad. Sin embargo, con el aumento de las soluciones de escalabilidad de capa 2, la prioridad se cambió a trabajar primero en la migración de PoW a PoS.

#### **9.3.2 Soluciones de escalado off-chain**

Este enfoque implica trasladar una parte del procesamiento de las transacciones fuera de la cadena de bloques de Ethereum. Las soluciones de escalado off-chain buscan procesar transacciones de manera más eficiente a través de capas adicionales que no forman parte de la cadena de bloques principal. Por la motivo no requieren cambios en el protocolo de Ethereum (red principal de capa 1). A continuación se presentan algunas de las soluciones de escalado off-chain:

##### **9.3.2.1 Sidechain o cadena lateral**

Una forma de abordar el problema de escalabilidad es mediante el uso de sidechains. Un sidechain es una cadena lateral que se conecta a una blockchain principal mediante un mecanismo de puente,



permitiendo que las transacciones se lleven a cabo en una cadena paralela. Esto permite aumentar la capacidad de procesamiento de transacciones, ya que una transacción en un sidechain no tiene que ser registrada en la cadena principal, lo que reduce la carga en la red principal y aumenta la velocidad de las transacciones.

### 9.3.2.2 Soluciones de escalado de capa 2

Las tres propiedades deseables de una cadena de bloques es que sea descentralizada, segura y escalable, pero el trilema de la cadena de bloques establece que una arquitectura de cadena de bloques simple sólo puede lograr dos de las tres. Por lo tanto si se desea una cadena de bloques descentralizada y segura, es razonable sacrificar la escalabilidad y delegar esa funcionalidad a una capa superior (capa 2), para que provea ese servicio.

Capa 2 (L2) es un término colectivo para describir un conjunto específico de soluciones de escalado de Ethereum. La capa 1 representa la blockchain base (Ethereum, Bitcoin, etc) y sobre estas se construyen varias redes de capa 2 diferentes. Ethereum también funciona como una capa de disponibilidad de datos donde los proyectos de la capa 2 publicarán sus datos de transacciones en Ethereum, confiando en Ethereum para la disponibilidad de los mismos. Estos datos pueden ser utilizados para obtener el estado de la capa 2, o para disputar transacciones en la capa 2.

Estas soluciones de capa 2 son cadenas de bloques independientes que extienden a Ethereum y heredan las garantías de seguridad de Ethereum. Toda la actividad de las transacciones de los usuarios en estos proyectos de capa 2 puede, en última instancia, volver a la blockchain de capa 1.

El principal beneficio de las soluciones de capa 2 es que hace que la red Ethereum sea más accesible para todos. Al combinar múltiples transacciones fuera de la cadena en una sola transacción de capa 1, las tasas de transacción se reducen masivamente. Con transacciones más altas por segundo y tarifas más bajas, aparecerán nuevos casos de uso y aplicaciones que no hubieran sido factibles de implementar directamente sobre la capa 1, ya sea por motivos técnicos u económicos.

Además, son soluciones que logran mantener la seguridad en niveles muy altos. Las blockchains de capa 2 asientan sus transacciones en la Mainnet de Ethereum, permitiendo a los usuarios beneficiarse de la seguridad heredada de la red principal.

- Un ejemplo de una solución de capa 2 son los "canales de pago" (Payment Channels), que permiten a dos partes realizar un número ilimitado de transacciones entre ellas sin tener que registrarlas en la blockchain principal.
- Otra forma de escalabilidad son los optimistic roll ups y zk roll ups, que se utilizan para agrupar varias transacciones en un solo registro en la cadena de bloques, lo que reduce la cantidad de información que se debe almacenar en la cadena de bloques y, por lo tanto, aumenta la capacidad de procesamiento.

Los Optimistic Rollups se basan en el protocolo rollup optimista, que permite a las transacciones ser verificadas y validadas en una cadena de bloques secundaria antes de ser agregadas a la cadena de bloques principal. Es llamado optimista debido a que confía en la veracidad

de los datos que se registran en la cadena secundaria, y solo verifica los datos en la cadena principal si hay algún problema.

Por otro lado los Zk-Rollups utilizan criptografía zk-SNARKs para agrupar transacciones en un único registro en la cadena de bloques, que permite verificar la validez de estas transacciones sin tener que escanear cada transacción individual. Esto permite aumentar significativamente la capacidad de procesamiento de transacciones, ya que solo se requiere verificar un único registro en lugar de cada transacción individual.

## 9.4 Arbitrum

Arbitrum es una red de escalabilidad de segunda capa para Ethereum que utiliza una tecnología llamada "Optimistic Rollup", la misma permite comprimir grandes cantidades de transacciones en una sola transacción en la red principal de Ethereum. Esto se logra mediante la creación de una capa de abstracción sobre la red principal, en la cual se procesan y verifican las transacciones de manera eficiente, logrando procesar un gran número de transacciones, a una velocidad mucho mayor y con un costo mucho menor que en la red L1 Ethereum[94].

En lugar de procesar cada transacción individualmente en la red Ethereum, el Optimistic Rollup agrupa varias transacciones en una sola transacción que se envía a la red principal. Esto reduce significativamente la cantidad de espacio necesario para almacenar y verificar cada transacción.

Para garantizar que las transacciones sean seguras y precisas, el Optimistic Rollup utiliza una técnica conocida como "verificación optimista". En lugar de verificar cada transacción en tiempo real, la red supone que todas las transacciones son válidas hasta que se demuestre lo contrario. Si se detecta una transacción falsa o inválida, la red puede revertir rápidamente la transacción y restaurar la cadena de bloques a su estado anterior.

En Arbitrum, un secuenciador centralizado recibe las transacciones de los usuarios y envía regularmente el lote de transacciones a la red principal de Ethereum. Los validadores independientes (actualmente en lista blanca) leen los lotes de transacciones de L1, las ejecutan y envían una raíz de estado resultante de L2 a L1. Cualquier otro validador puede desafiar la raíz de estado dentro de la ventana de desafío (7 días). El desafío resultará en un juego interactivo de prueba de fraude que eventualmente será resuelto por L1. Mientras haya al menos un validador honesto, se garantiza a los usuarios que eventualmente se publicará la raíz de estado L2 correcta en L1. Si el Secuenciador está censurando las transacciones de los usuarios, es posible forzar la transacción a través de la cola de L1. Si ningún validador publica una raíz de estado L2 dentro de 7 días, se elimina la lista blanca de validadores y cualquier persona puede convertirse en un nuevo validador [95]. En conclusión, aunque el Secuenciador Centralizado en la red Arbitrum puede ser un problema para la descentralización y la seguridad de la red, la red ha implementado medidas de seguridad para abordar este problema y garantizar que la red siga siendo segura y confiable para los usuarios.

En resumen, Arbitrum es completamente compatible con la red principal de Ethereum, lo que significa que los desarrolladores pueden construir dApps en Arbitrum sin tener que preocuparse por la compatibilidad con la red principal. Además, la tecnología Optimistic Rollup permite a la

red procesar un gran número de transacciones a una velocidad mucho mayor que la red principal, lo que la convierte en una solución escalable y eficiente para las dApps en Ethereum.

## **Parte III**

# **DESARROLLO DEL TRABAJO**

# Desarrollo de la solución propuesta

## 10.1 Introducción

Los objetivos perseguidos con esta propuesta son analizar las propiedades de blockchain y su aplicación específica en la asignación y delegación de recursos de la infraestructura de Internet, así como también en la seguridad de ruteo externo BGP contra ataques de Route Hijacking o similares.

## 10.2 Arquitectura de la solución propuesta

La arquitectura de la solución propuesta se basa en una revisión exhaustiva del estado del arte de la problemática que se desea solucionar. Se tomó como punto de partida el código `bgp_ethereum` del repositorio de Github de Marcelo de Abranches [96], al cual se le sumaron los aportes realizados durante la elaboración del paper *Routing Security Using Blockchain Technology* [97].

El contrato inteligente implementado debe tener funciones que permitan la creación de SAs, la asignación de prefijos a un SA, la consulta sobre si se ha delegado un prefijo determinado a un SA, la revocación de prefijos, entre otras. Es crucial tener en cuenta que las partes implicadas tendrán diferentes permisos para interactuar con la blockchain.

Tras el despliegue de la solución, se realizaron pruebas y análisis que permitieron proponer mejoras y variantes al código original. El repositorio del contrato se encuentra disponible en el siguiente enlace:

[https://github.com/margmz/bgp\\_ethereum](https://github.com/margmz/bgp_ethereum)

Los requerimientos que se tuvieron en cuenta en el diseño, categorizados de acuerdo al objetivo a cumplir son los siguientes:

- Requerimientos de la asignación y delegación de recursos de Internet mediante blockchain:

Se debe simplificar la gestión de los recursos (ASN , direcciones IPv4 e IPv6).

Debe ser más descentralizado que el modelo actual, pero manteniendo cierto nivel de control de los recursos en el IANA y los RIRs.

Debe proporcionar información actualizada sobre las asignaciones de prefijos para poder verificar la propiedad de los mismos.

Debe ser 100% auditable.

Debe permitir la transferencia libre de recursos entre participantes

- Requerimientos para brindar seguridad de ruteo externo BGP contra ataques de Route Hijacking mediante blockchain:

El router de borde que recibe los anuncios BGP UPDATE debe poder verificar el origen de la ruta, es decir, determinar en la blockchain que el primer SA en la ruta recibida está

autorizado para anunciar el prefijo.

El router de borde que recibe los anuncios BGP UPDATE debe poder validar la ruta o camino de los SAs que recorre el anuncio, es decir, el atributo AS-PATH.

El router de borde debe poder modificar su política de ruteo de acuerdo al resultado de las validaciones realizadas. Por ejemplo, seleccionar rutas validadas sobre rutas no validadas, a pesar de poseer un AS-PATH más largo.

En cuanto a la puesta en marcha, debe poder implementarse en forma incremental y no debe consumir una gran cantidad de recursos en los routers (memoria, almacenamiento y CPU).

La topología de red IPv6 con la que se trabajó se describe en el diagrama de la Figura 10.1, donde queda claro las relaciones entre los SAs y los prefijos IPs con los que se ejemplificará el desarrollo. El prefijo IPv6 reservado para documentación es el siguiente: 2001:DB8::/32

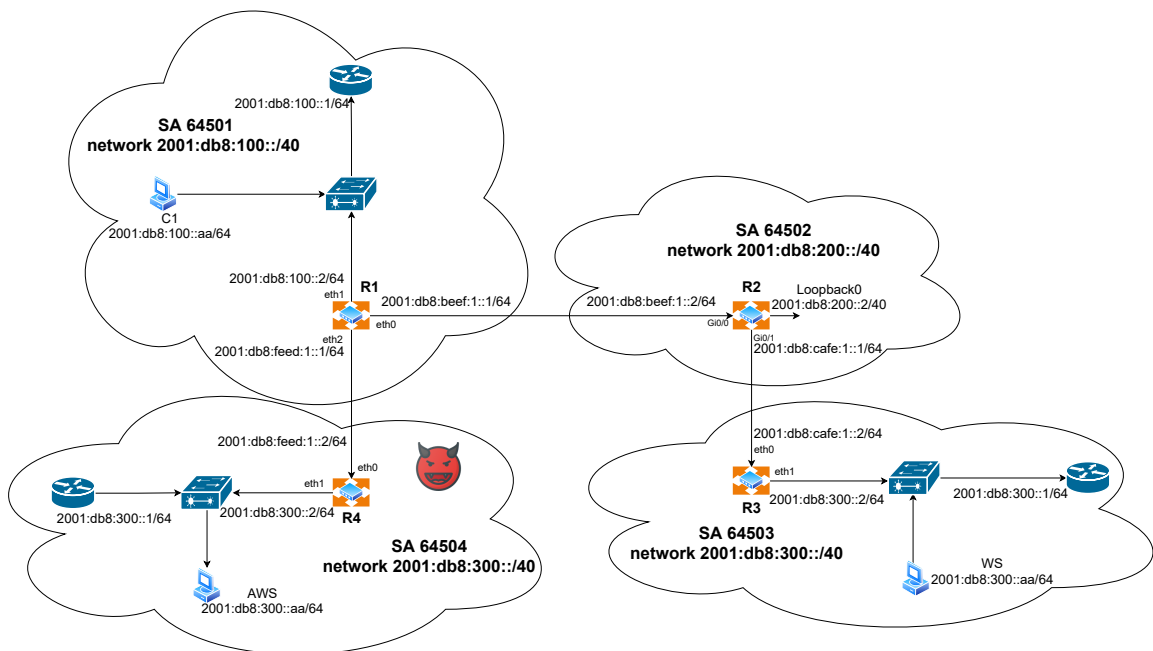


Figura 10.1: Topología de red IPv6 - Emulación de secuestro de rutas

## 10.2.1 Aportes realizados al contrato inteligente

### 10.2.1.1 Soporte IPv6

La implementación realizada se realizó sobre el stack IPv6, tanto para la asignación de los prefijos como para la posterior implementación de seguridad en ruteo externo.

### 10.2.1.2 Utilización de funciones para verificación de firmas realizadas offchain

Se implementó el código para utilizar la librería ethers para firma offchain, y la verificación de firmas on chain, la cual es segura para crear código de producción. También se utilizaron las fun-

ciones hash keccak256 debido a que consume menos gas que sha256. [98]

### **10.2.1.3 Ataques de repetición de firmas (Signature replay attacks)**

Existen casos donde se requiere que un usuario firme datos fuera de la cadena(off-chain) y esos datos firmados se entregan a otro usuario autorizado que los enviará al contrato para invocar alguna función, con el aval del usuario que realizó la firma. Este proceso permite a los usuarios expresar fuera de la cadena el deseo de realizar un transacción y, más tarde, la confirmación se actualiza dentro de la cadena bloques a través del usuario autorizado[99]. En nuestro caso el usuario que firma los datos es el SA o ISP, y el usuario autorizado es el IANA.

El código utilizado inicialmente era propenso al “signature replay attack” porque la entidad autorizada, el IANA, podía volver a enviar los mismos datos firmados y realizar operaciones no deseadas que se espera que se realicen solo una vez.

Para evitar los ataques de repetición se realizaron modificaciones para que los SAs firmen los datos junto a un valor nonce, único cada vez. Además incluimos el nombre de la función(o msg.sig) en los datos firmados, evitando que la entidad autorizada utilice los datos firmados en otra función diferente.

### **10.2.1.4 Heartbeat function**

En un modelo centralizado basado en RPKI, los RIRs tienen la capacidad para impedir el uso de un prefijo IP asignado a un SA mediante la revocación del certificado. El modelo propuesto si bien mantiene al IANA en el rol de gestión de los recursos, no le permite bloquear el uso de los prefijos, a menos que el SA no realice la renovación anual y pago por el uso prefijo. Una vez que se agote el tiempo de espera, el contrato inteligente permite al IANA/RIR recuperar los recursos. Los motivos por los cuales se puede producir el vencimiento son:

- Olvido o decisión propia del responsable del SA. En caso de olvido deberá ponerse en contacto con el IANA/RIR para evitar la baja y realizar el trámite de renovación a la brevedad.
- Extravío de la clave privada por descuido, fallecimiento, etc. En estos casos el IANA puede recuperar los prefijos luego del vencimiento. Esto se debe implementar obligatoriamente sino se perderían para siempre espacios de direcciones y SAs, sin posibilidad de recuperar los mismos o reclamar el pago por el uso del recurso.

### **10.2.1.5 Migración a Solidity v0.8.18**

Al implementar contratos, se recomienda utilizar la ante última versión publicada de Solidity. Esto se debe a que se introducen regularmente cambios importantes, así como nuevas funciones y correcciones de errores.

### 10.2.1.6 Verificación del AS\_PATH

Se evaluaron 2 alternativas diferentes para verificar el AS-PATH, y se seleccionó el método de la adyacencia doble lateral. A continuación se describen los métodos analizados:

- El método de la adyacencia doble lateral es el que plantea que cada SA registre en la blockchain a sus propios peers, sin diferenciar los uplink de los downlink. La validez del AS\_PATH dependerá de que en cada par de SAs en el camino se verifique la adyacencia con el otro extremo. Es el método que utiliza el repositorio original y la desventaja es que sin una alta tasa de adopción no provee una manera confiable de automatizar la detección de route leaks, debido a que el atacante podría crear una adyacencia de un solo sentido. También puede presentar problemas con las adyacencias indirectas en algunos IXPs.
- El método “ASPA” ó Autorización de proveedor de sistema autónomo (ASPA) se puede implementar almacenando en la blockchain las distintas asociaciones del SA con sus peers. Esta información permite que se pueda verificar que un dueño de un AS Cliente(CAS) ha autorizado a un AS Proveedor(PAS) en particular para propagar los anuncios de ruta BGP IPv4 o IPv6 pertenecientes al cliente, a los upstreams o pares del AS Proveedor. Permitiría detectar AS\_PATHs inválidos incluso durante la adopción temprana e incremental.

## 10.3 Despliegue del contrato inteligente

El contrato inteligente llamado IANA.sol se desplegó en la red Ethereum de pruebas Sepolia, con la dirección de contrato 0xEeAaaCdD3d1F77AB33F5fB5448f5843d04E9A496.

<https://sepolia.etherscan.io/address/0xEeAaaCdD3d1F77AB33F5fB5448f5843d04E9A496>

Este contrato inteligente se utilizó tanto para la asignación y delegación de recursos de Internet, como para que las diferentes entidades puedan obtener seguridad en el ruteo externo al validar los mensajes UPDATE de BGP.

Si analizamos la estructura del contrato se observa que la información que se almacenará de forma permanentemente en la blockchain (storage del contrato) se modela como un arreglo dinámico de datos tipo estructura llamado “prefixes”, en el cual se guarda una lista de los prefijos IP asignados, teniendo en cuenta a que SA corresponden.

Para las pruebas iniciales en el ambiente de test se utilizó una plataforma web para desarrollar contratos inteligentes en Solidity llamada Remix[100], que permite escribir, compilar, deployar y depurar código de forma sencilla e interactiva.

```
struct Prefix {
    uint128 ip;
    uint8 mask;
    uint32 owningAS;
    uint256 expiryOf;
    uint[] subPrefixes;
}
```



```
// Arreglo de prefijos
Prefix[] public prefixes;
```

## 10.4 Asignación y delegación de recursos de Internet mediante blockchain

La o las autoridades de registro asignan los prefijos IPv6 a los sistemas SAs, y estos pueden a su vez delegarlos a otros SAs.

### 10.4.1 Secuencia de pasos para dar de alta un SA

1. Se utiliza la firma de mensajes off-chain para expresar la voluntad del responsable del SA de vincular su dirección de Ethereum al ASN que se le asignará. Mediante la función `IANA_getSignatureMessage` se genera el mensaje de texto que se debe firmar para autenticarse como la entidad solicitante del SA. Los argumentos de la función son la dirección, el ASN, el nonce y un hash que representa el nombre de la función(`msg.sig`), estos dos últimos se incluyen para evitar los ataques de repetición mencionados anteriormente.

```
function IANA_getSignatureMessage(
    uint32 ASN,
    address ASNOwner,
    uint _nonce,
    bytes4 _functionName)
    pure public returns(bytes32) {
    // The user should sign the data along with a unique nonce value
    // each time
    return keccak256(abi.encodePacked(ASN, ASNOwner, _nonce,
        _functionName));
}
```

2. Posteriormente se realiza la firma del mensaje con la clave privada correspondiente a la dirección de la entidad solicitante(10.2).
3. A continuación la autoridad recibe el mensaje firmado y lo utiliza para invocar a la función `IANA_addASN` mediante la cual queda establecida en la blockchain el vínculo entre la dirección solicitante y el ASN.

```
/// Adds an additional ASN to the ASN list. The operation has to include
/// a signature from the ASN owner signing keccak256(abi.encodePacked(ASN
/// ,ASNOwner,_nonce,_functionName))
/// which can be generated by calling IANA_getSignatureMessage()
/// @param ASN The ASN to be added
/// @param ASNOwner The public key of the new owner.
/// @param nonce A security nonce.
/// @param _signature The signature.
```

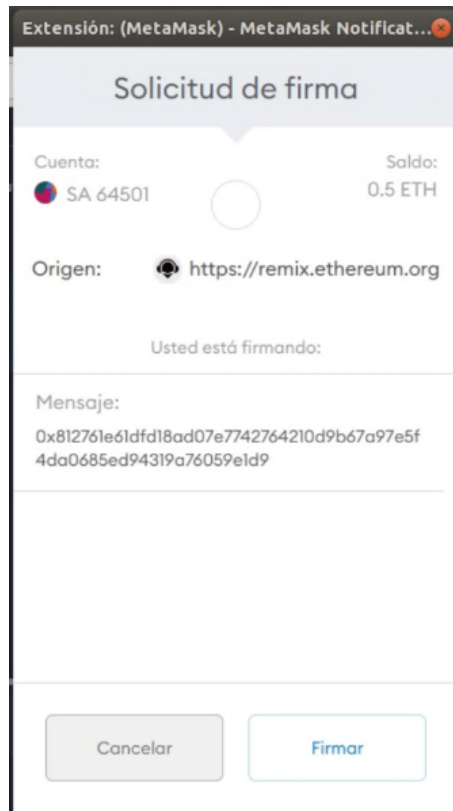


Figura 10.2: Solicitud de firma del SA 64501

```
function IANA_addASN(uint32 ASN, address ASNOwner, uint nonce, bytes
    memory _signature) public onlyOwners {
    // check if the user has previously sent the signature with that nonce
    require(nonceUsedMap[ASNOwner][nonce] == false);
    // It must be signed by the new ASNOwner. We don't have to check for the
    IANA owner // because the onlyOwners routine does that for us.

    require(recoverSigner(getEthSignedMessageHash(IANA_getSignatureMessage(
        ASN, ASNOwner, nonce, msg.sig)), _signature) == ASNOwner);
    require(ASN != 0);

    // At this point, we have two party agreement on ASN ownership. Add it to
    the ANSList.
    ANSList[ASN] = ASNOwner;
    // the nonce should be stored on-chain
    nonceUsedMap[ASNOwner][nonce] = true;
}
```

Para remover un ASN(vínculo entre la dirección y el ASN) se utiliza la función IANA\_removeASN

```
/// Removes an ASN to the ASN list. The operation has to include a
signature
/// from the ASN owner signing keccak256(abi.encodePacked(ASN,ASNOwner,
_nonce,_functionName)) which can be
```

```

/// generated by calling IANA_getSignatureMessage()
/// @param ASN The ASN to be added
/// @param ASNOwner The public key of the new owner.
/// @nonce A security nonce.
/// @param _signature The signature.
function IANA_removeASN(uint32 ASN, address ASNOwner, uint nonce, bytes
memory _signature) public onlyOwners {
// check if the user has previously sent the signature with that nonce
require(nonceUsedMap[ASNOwner][nonce] == false);
// It must be signed by the new ASNOwner. We don't have to check for the
IANA owner because the onlyOwners routine does that for us.
require(recoverSigner(getEthSignedMessageHash(IANA_getSignatureMessage(
ASN, ASNOwner, nonce, msg.sig)), _signature) == ASNOwner);
require(ASN != 0);

for (uint i=0; i<prefixes.length; i++) {
require(prefixes[i].owningAS != ASN, "The ASN cannot be removed
because it has prefixes associated with it.");
}

// At this point, we have two party agreement on ASN ownership. Mark the
ASN as unowned
ASNList[ASN] = address(0);
// the nonce should be stored on-chain
nonceUsedMap[ASNOwner][nonce] = true;
}

```

#### 10.4.2 Secuencia de pasos para asignar el prefijo IPv6 al SA

1. Se utiliza la firma de mensajes off-chain para expresar la voluntad del responsable del SA de vincular el prefijo IPv6 que es de su interés, al ASN que le ha sido asignado. Mediante la función `IANA_getPrefixSignatureMessage` genera el mensaje de texto que se firmará para autenticar al propietario del ASN. Los argumentos de la función son el prefijo(ip y máscara de red), el ASN, la dirección y el nonce.

```

/// Generates the message text to be signed for add authentication.
/// @param ip The ip to be added
/// @param mask The mask
/// @param ASN The ASN
/// @param ASNOwner The public key of the owner.
/// @return bytes32 The keccak256 hash of abi.encodePacked(ip,mask,ASN,
ASNOwner,nonce).
function IANA_getPrefixSignatureMessage(uint128 ip, uint8 mask, uint32
ASN, address ASNOwner, uint nonce) pure public returns(bytes32) {
// The user should sign the data along with a unique nonce value each
time
return keccak256(abi.encodePacked(ip, mask, ASN, ASNOwner,nonce));
}

```

2. Se realiza la firma del mensaje con la clave privada correspondiente a la dirección ASNOwner.
3. Posteriormente la autoridad recibe el mensaje firmado y lo utiliza para invocar a la función `prefix_addPrefix` mediante la cual queda establecida en la blockchain el vínculo entre el prefijo IPv6 y el ASN.

```

/// Adds the specified prefix to the prefix table. Must be done by the
/// owner of the prefixes containing AS and must include the signature
/// of the message returned by IANA_getPrefixSignatureMessage for the
/// new AS.
/// @param ip The IP address of the prefix to add
/// @param mask The number of bits in the netmask of the prefix to add
/// @param newOwnerAS The AS number to associate with the new prefix to.
/// @param nonce
/// @param _signature The signature.
function prefix_addPrefix(uint128 ip, uint8 mask, uint32 newOwnerAS, uint
    nonce, bytes memory _signature) public {
    // Only valid subnet masks
    require (mask <= 48);
    // Get the ASN's owner
    address newOwnerAddress = ASNList[newOwnerAS];
    // The owning ASN must exist
    require (newOwnerAddress != address(0));
    // check if the user has previously sent the signature with that nonce
    require(nonceUsedMap[newOwnerAddress][nonce] == false);
    // The owning ASN must have signed the message.
    require(recoverSigner(getEthSignedMessageHash(
        IANA_getPrefixSignatureMessage(ip, mask, newOwnerAS, newOwnerAddress
        , nonce)), _signature) == newOwnerAddress);

    // Find who owns the space this mask is in.
    uint parentIndex = prefix_getContainingPrefix(0,ip, mask);
    Prefix storage parent = prefixes[parentIndex];

    // Require that the public address calling us owns the AS that owns the
    // parent prefix.
    // (i.e. you can't claim addresses on someone else's prefix)
    // If the parent's owningAS is 0, that means that we're making changes
    // to the root. Check against our internal owner list.
    if (parent.owningAS == 0) {
        require (ownerList[msg.sender] == true);
    } else {
        // Otherwise, check that our sender is the AS's owner.
        require (msg.sender == ASNList[parent.owningAS]);
    }

    // Require that the parent contains us.
    require(!(parent.ip == ip && parent.mask == mask));

```

```

// For every child,
for (uint i=0; i<parent.subPrefixes.length; i++) {
    uint testIndex = parent.subPrefixes[i];
    Prefix memory child = prefixes[testIndex];
    PrefixCompareResult result = prefix_comparePrefix(child.ip, child.
        mask, ip, mask);
    // Require that we're not trying to take their prefix.
    require(result == PrefixCompareResult.NoIntersection);
}
// the nonce should be stored on-chain
nonceUsedMap[newOwnerAddress][nonce] = true;

Prefix memory newPrefix;
newPrefix.ip = ip;
newPrefix.mask = mask;
newPrefix.owningAS = newOwnerAS;
newPrefix.expiryOf = block.timestamp + leaseTime;
prefixes.push(newPrefix);
// Add it to the list
uint index = prefixes.length - 1;
parent.subPrefixes.push(index);
}

```

Para remover el vínculo entre el ASN y el prefijo se utiliza la función `prefix_removePrefix`.

```

/// Removes the specified prefix to the prefix table. Must be done by the
    owner of the prefixes containing
/// @param ip The IP address of the prefix to remove
/// @param mask The number of bits in the netmask of the prefix to remove
function prefix_removePrefix(uint128 ip, uint8 mask) public {
    // Only valid subnet masks
    require (mask <= 48);

    // Find who owns the space this mask is in.
    (uint index, uint parentIndex) = prefix_getContainingPrefixAndParent(0,
        ip, mask);
    Prefix storage target = prefixes[index];
    Prefix storage parent = prefixes[parentIndex];

    // Require that the public address that calls us has the prefix
    // (i.e. you can't claim addresses in someone else's prefix)
    // and the expiryOf period is not expired.
    // If expiryOf is expired it requires that the public address belongs
    to ownerList
    if (target.expiryOf > block.timestamp) {
        require (msg.sender == ASNList[target.owningAS], "The public address
            that calls us doesn't own the prefix");
    } else {
        require (ownerList[msg.sender] == true, "The public address doesn't
            belong to ownerList");
    }
}

```

```

}

// The prefix must exactly reference the listed prefix.
require(prefix_comparePrefix(target.ip, target.mask, ip, mask) ==
    PrefixCompareResult.Equal);

// We only delete prefixes that have no children. They have to be
    deleted seperately.
require(target.subPrefixes.length == 0);

//Blank the prefix (since deleting it is impossible in the contract and
    doesn't help anything anyway.)
Prefix memory blankPrefix;
blankPrefix.ip = 0;
blankPrefix.mask = 0;
blankPrefix.owningAS = 0;
blankPrefix.expiryOf = 0;
prefixes[index] = blankPrefix;

// This operation is EXPENSIVE. However, without it, the arrays could
    get long. Basically we're looking
// for our specific child node in the parent, and once we find it, we'
    re deleting it from the array and
// shrinking the array down.
for(uint i=0;i<parent.subPrefixes.length;i++) {
    Prefix storage child = prefixes[parent.subPrefixes[i]];
    if (child.ip == target.ip && child.mask == target.mask) {
        // Skip ahead one and start shifting the array left
        for(i=i+1;i<parent.subPrefixes.length;i++) {
            parent.subPrefixes[i-1] = parent.subPrefixes[i];
        }
        // Shrink the array.
        parent.subPrefixes.pop();
    }
}
}
}
}

```

### 10.4.3 Los SAs indican mediante transacciones cuales son sus SAs vecinos

Cada SA, para indicar la vecindad, llama a la función link\_addLink una vez por cada vecino. Los argumentos de la función son: el SA que llama a la función y el SA con el cual quiere establecer la relación de peering. Para poder posteriormente validar la relación de peering, es necesario que ambos SAs invoquen a la función estableciendo el vínculo con el otro.

En las transacciones realizadas para agregar los vecinos solo intervienen las cuentas de los SAs, no interviene la dirección del IANA o autoridad.

```

/// Marks that the caller believes it has a link to a particular destination
ASN.

```

```

/// @param myASN The ASN the caller owns
/// @param destinationASN The ASN that the caller links to.
function link_addLink(uint32 myASN, uint32 destinationASN) public {
    require(msg.sender == ASNList[myASN]);
    bytes32 linkhash = keccak256(abi.encodePacked(myASN, destinationASN));
    links[linkhash] = true;
}

```

Para remover el vínculo de peering entre los ASN basta con que alguno de los dos SAs realice una transacción invocando a la función `link_removeLink`.

```

/// Marks that the caller believes it no longer has a link to a particular
    destination ASN.
/// @param myASN The ASN the caller owns
/// @param destinationASN The ASN that the caller links to.
function link_removeLink(uint32 myASN, uint32 destinationASN) public {
    require(msg.sender == ASNList[myASN]);
    bytes32 linkhash = keccak256(abi.encodePacked(myASN, destinationASN));
    links[linkhash] = false;
}

```

#### 10.4.4 Conclusiones del trabajo experimental con Remix

En la sección anterior se describieron las funciones principales del contrato y se realizaron las interacciones con el mismo mediante la plataforma web Remix. En la siguiente fase de pruebas se utilizará Hardhat[101], una herramienta de desarrollo de contratos inteligentes que facilita el proceso de compilación, despliegue y pruebas de los contratos. Una de las principales ventajas de usar Hardhat es que permite simular entornos de prueba de manera rápida y eficiente, lo que permite a los desarrolladores probar casos límites y escenarios complejos sin incurrir en costos de gas en la red Ethereum. Esto es especialmente útil para trabajar con casos extremos, bucles y arreglos dinámicos, ya que los desarrolladores pueden probar y optimizar su código sin tener que pagar por cada transacción.

Además, Hardhat incluye herramientas de pruebas unitarias integradas que permiten a los desarrolladores automatizar pruebas para validar el comportamiento de los contratos. Esto ayuda a detectar y corregir errores temprano en el proceso de desarrollo, lo que puede ahorrar tiempo y recursos en el futuro.

En resumen, el uso de Hardhat puede mejorar significativamente la eficiencia, escalabilidad y calidad del proceso de desarrollo de contratos inteligentes en las plataformas EVM compatibles, especialmente para proyectos complejos y en constante evolución.

#### 10.5 Despliegue del contrato inteligente con Hardhat

Inicialmente se crea la carpeta del proyecto y se crea un nuevo proyecto de Hardhat, posteriormente se instalan algunos plugin necesarios como `@nomicfoundation/hardhat-toolbox` que tiene las

herramientas necesaria para el desarrollo de contratos inteligentes. Finalmente se ejecuta Visual Studio Code para trabajar en el proyecto.

```
$ mkdir bgp_blockchain
$ cd bgp_blockchain
$ npm init
$ npm install --save-dev hardhat
$ npm install --save-dev @nomicfoundation/hardhat-toolbox
$ npm install dotenv --save
$
$ npx hardhat
$
$ 888      888                888 888                888
$ 888      888                888 888                888
$ 888      888                888 888                888
$ 888888888888 8888b. 888d888 .d88888 88888b. 8888b. 888888
$ 888      888      "88b 888P"  d88" 888 888 "88b      "88b 888
$ 888      888 .d888888 888      888 888 888 888 .d888888 888
$ 888      888 888 888 888      Y88b 888 888 888 888 888 Y88b.
$ 888      888 "Y888888 888      "Y88888 888 888 "Y888888 "Y888
$
$ Welcome to Hardhat v2.13.0
$
$ ? What do you want to do? ...
$ Create a JavaScript project
$ Create a TypeScript project
$ > Create an empty hardhat.config.js
$ Quit
$
$ code .
```

Se crea el archivo .env, donde almacenamos variables de entorno con la información de la clave privada y la url RPC para conectarnos a la red Sepolia.

### Código 10.1 .env

```
1 GOERLI_RPC_URL=https://goerli.infura.io/v3/31a6da5e*****
2 SEPOLIA_RPC_URL=https://sepolia.infura.io/v3/31a6da5*****
3
4 PRIVATE_KEY=90c5391f5dc0177cc*****
5 PRIVATE_KEY_1=3458c9c62296973a1*****
6 PRIVATE_KEY_2=d5b2a530a0e777c38*****
7 PRIVATE_KEY_3=07ebc556d26b0f142*****
8 PRIVATE_KEY_4=0c5f9a73bdaa0a897*****
9 PRIVATE_KEY_5=87c17d8b8e71c27e7*****
```



```

10
11 COINMARKETCAP_API_KEY=6c489421-****-****-****-*****
12
13 ETHERSCAN_API_KEY=PQ3A4ZGRRJKEI*****

```

Las variables del archivo .env son leídas desde la configuración de Hardhat almacenada en el archivo hardhat.config.js dentro de la carpeta principal del proyecto.

```

require("@nomicfoundation/hardhat-toolbox");
require('dotenv').config();
require("hardhat-gas-reporter");
require('hardhat-storage-layout');

const GOERLI_RPC_URL = process.env.GOERLI_RPC_URL
const SEPOLIA_RPC_URL = process.env.SEPOLIA_RPC_URL

const PRIVATE_KEY = process.env.PRIVATE_KEY
const PRIVATE_KEY_1 = process.env.PRIVATE_KEY_1
const PRIVATE_KEY_2 = process.env.PRIVATE_KEY_2
const PRIVATE_KEY_3 = process.env.PRIVATE_KEY_3
const PRIVATE_KEY_4 = process.env.PRIVATE_KEY_4
const PRIVATE_KEY_5 = process.env.PRIVATE_KEY_5

const COINMARKETCAP_API_KEY = process.env.COINMARKETCAP_API_KEY
const ETHERSCAN_API_KEY = process.env.ETHERSCAN_API_KEY

task("accounts", "Prints the list of accounts", async (taskArgs, hre) => {
  const accounts = await hre.ethers.getSigners();

  for (const account of accounts) {
    console.log(await account.address);
  }
});

/** @type import('hardhat/config').HardhatUserConfig */
module.exports = {
  defaultNetwork: "hardhat",
  networks: {
    hardhat: {
      accounts: [
        { privateKey: "0
          x503f38a9c967ed597e47fe25643985f032b072db8075426a92110f82df48dfcb",
            balance: "10000000000000000000" },
        { privateKey: "0
          x7e5bfb82febc4c2c8529167104271ceec190eafdca277314912eaabdb67c6e5f",
            balance: "10000000000000000000" },
        { privateKey: "0
          xcc6d63f85de8fef05446ebdd3c537c72152d0fc437fd7aa62b3019b79bd1fdd4",
            balance: "10000000000000000000" },
      ],
    },
  },
};

```

```

    { privateKey: "0
      x638b5c6c8c5903b15f0d3bf5d3f175c64e6e98a10bdb9768a2003bf773dcb86a",
        balance: "10000000000000000000" },
    { privateKey: "0
      xf49bf239b6e554fdd08694fde6c67dac4d01c04e0dda5ee11abee478983f3bc0",
        balance: "10000000000000000000" }
  ],
},
sepolia: {
  url: SEPOLIA_RPC_URL,
  accounts: [`${PRIVATE_KEY_1}`, `${PRIVATE_KEY_2}`, `${PRIVATE_KEY_3}`,
    `${PRIVATE_KEY_4}`, `${PRIVATE_KEY_5}`],
  chainId: 11155111,
},
},
solidity: "0.8.18",
gasReporter: {
  enabled: true,
  currency: "USD", // Choose the currency
  // gasPrice: 15, // Set the gas price in Gwei
  coinmarketcap: COINMARKETCAP_API_KEY,
  // token: "MATIC",
},
etherscan: {
  apiKey: ETHERSCAN_API_KEY,
},
};

```

El contrato IANA.sol explicado previamente, se debe almacenar en la carpeta contracts. El mismo se encuentra en la sección de Anexos.

En la carpeta scripts se almacena los scripts para desplegar el contrato y automatización de algunas tareas auxiliares. Precisamente el archivo deploy.js, se encarga del despliegue del contrato y los archivos start\_r1.js, start\_r2.js, start\_r3.js, y start\_r4.js se encargan de registrar los SAs, los prefijos y las adyacencias en la blockchain. La cual después consultaremos para validar los mensajes UPDATES de BGP. Todos los archivos mencionados se encuentran en el Anexo.

A continuación se ejecutan los siguientes comandos para desplegar e interactuar con el contrato inteligente dentro de una red blockchain local que provee HardHat.

```
$ npx hardhat node
```

El comando "npx hardhat node" se utiliza para iniciar una instancia de nodo de Ethereum local que se ejecuta en segundo plano y se conecta al HRE. El HRE (Hardhat Runtime Environment) es una instancia de tiempo de ejecución de Hardhat que proporciona una interfaz de programación de aplicaciones (API) para interactuar con la red, como enviar transacciones, interactuar con contratos inteligentes y escuchar eventos de la red.

La instancia de nodo local proporciona una red de Ethereum simulada que se puede utilizar para probar y desarrollar aplicaciones de Ethereum de manera segura y eficiente.

```
$ npx hardhat node
```

```
Started HTTP and WebSocket JSON-RPC server at http://127.0.0.1:8545/
```

```
Accounts
```

```
=====
```

```
Account #0: 0x5B38Da6a701c568545dCfcB03Fcb875f56beddC4 (1000 ETH)
```

```
Account #1: 0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2 (1000 ETH)
```

```
Account #2: 0x4B20993Bc481177ec7E8f571ceCaE8A9e22C02db (1000 ETH)
```

```
Account #3: 0x78731D3Ca6b7E34aC0F824c42a7cC18A495cabaB (1000 ETH)
```

```
Account #4: 0x617F2E2fD72FD9D5503197092aC168c91465E7f2 (1000 ETH)
```

Posteriormente se inicia otra terminal donde se ejecutan los scripts con las tareas de compilación, despliegue del contrato y asignación de recursos de internet en la blockchain.

```
$ npx hardhat compile
```

```
$
```

```
$ npx hardhat run scripts/deploy.js --network localhost
```

```
Iana deployed to 0xd9145CCE52D386f254917e481eB44e9943F39138
```

```
$
```

```
$ npx hardhat run scripts/start_r1.js --network localhost
```

```
$
```

```
El ASN solicitado por R1 es 64501
```

```
El address que pretende ser dueño del ASN1 es \\
```

```
0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2
```

```
La firma de la solicitud del ASN 64501 firmado por \\
```

```
0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2 es: \\
```

```
0x8ee2c1bcaecd20076efe62cdac34b52b4231baea502a274fcd583e7aa5610680 \\
```

```
0db82fc64f5bb5cb8af3701eba2912f577d68ba52aea153312337062b3d6efb51b
```

```
El IANA utilizando los datos del ASN, el address solicitante,
```

```
el nonce y la firma, procesa la solicitud...
```

```
IANA vinculo correctamente la address \\
```

```
0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2 con el ASN 64501
```

```
Se vincularon los datos del prefijo IP 0x20010db8010000000000000000000000
```

```
/40 al ASN 64501 : true
```

```
Enlazamos el ASN 64501 a sus vecinos, 64502 y 64504
```

Posteriormente se repite la ejecución para los demás SAs, teniendo en cuenta que el SA 64504 no realiza reserva de prefijos de red.

```
$ npx hardhat run scripts/start_r2.js --network localhost
$
$ npx hardhat run scripts/start_r3.js --network localhost
$
$ npx hardhat run scripts/start_r4.js --network localhost
$
```

## 10.6 Prueba del contrato inteligente

A continuación creamos casos de prueba utilizando Mocha y Chai. Las dependencias de Mocha y Chai se instalaron junto con el plugin de Hardhat Toolbox.

Por este motivo solo tenemos que crear un directorio llamado test en la raíz del proyecto, y aquí dentro crear un archivo de test llamado iana\_tests.js

Ejecutamos los test del archivo iana\_tests.js, donde se encuentran los siguientes casos de prueba:

```
$ npx hardhat test
```

```
IANA contract test cases
```

```
  Should assign an autonomous system to an address,
  and then remove the autonomous system from it. (239ms)
  Should add a prefix to an ASN (282ms)
  Should remove a prefix by the AS owner (347ms)
  Should remove a prefix by the IANA after the leaseTime (336ms)
  Should not allow adding a subprefix to another ASN (528ms)
```

```
5 passing (3s)
```

El contenido del archivo iana\_tests.js se encuentra en el Anexo.

## 10.7 Reporte de gas utilizado

hardhat-gas-reporter es un plugin para Hardhat que genera un informe del consumo de gas, esto nos permite analizar el rendimiento y la eficiencia de las funciones del contrato.

Para instalar hardhat-gas-reporter, ejecuta el siguiente comando en tu proyecto Hardhat:

```
$ npm install hardhat-gas-reporter --save-dev
```

Después agregamos lo siguiente al archivo hardhat.config.js:

```
require("hardhat-gas-reporter");
```

```

module.exports = {
  // Rest of your Hardhat configuration

  gasReporter: {
    enabled: true,
    currency: "USD", // Choose the currency
    // gasPrice: 15, // Set the gas price in Gwei
    coinmarketcap: COINMARKETCAP_API_KEY,
  },
};

```

Con el plugin instalado y configurado, se ejecutan los test del contrato con el informe de gas incluido, usando el siguiente comando:

```
$ npx hardhat test --network localhost
```

Ahora después de ejecutar los tests, se obtiene un informe de gas para cada función y transacción en la salida de la consola.

```
$ npx hardhat test --network localhost
```

```
IANA contract test cases
```

```

  Should assign an autonomous system to an address,
  and then remove the autonomous system from it. (3669451 gas)
  Should add a prefix to an ASN (301516 gas)
  Should remove a prefix by the AS owner (451454 gas)
  Should remove a prefix by the IANA after the leaseTime (321431 gas)
  Should not allow adding a subprefix to another ASN (346375 gas)

```

Estos informes nos brindan información valiosa para optimizar el consumo de gas de nuestros contratos y funciones. Los mismos serán de utilidad para poder realizar análisis de factibilidad económica sobre los costos de despliegue y ejecución de la solución en diferentes redes blockchain.

```

IANA contract test cases
✓ Should assign an autonomous system to an address, and then remove the autonomous system from it. (3669451 gas)
✓ Should add a prefix to an ASN (301516 gas)
✓ Should remove a prefix by the AS owner (451454 gas)
✓ Should remove a prefix by the IANA after the leaseTime (321431 gas)
✓ Should not allow adding a subprefix to another ASN (346375 gas)

-----|-----|-----|-----|-----|-----|-----|
Solc version: 0.8.18 | Optimizer enabled: false | Runs: 200 | Block limit: 6718946 gas |
-----|-----|-----|-----|-----|-----|-----|
Methods | | | 19 gwei/gas | | 1775.66 usd/eth |
-----|-----|-----|-----|-----|-----|-----|
Contract | Method | Min | Max | Avg | # calls | usd (avg) |
-----|-----|-----|-----|-----|-----|-----|
IANA | IANA_addASN | 57949 | 77849 | 71214 | 6 | 2.40 |
IANA | IANA_removeASN | - | - | 60812 | 2 | 2.05 |
IANA | prefix_addPrefix | 157809 | 162855 | 159837 | 5 | 5.39 |
IANA | prefix_removePrefix | 52756 | 52917 | 52837 | 4 | 1.78 |
-----|-----|-----|-----|-----|-----|-----|
Deployments | | | | | % of limit |
-----|-----|-----|-----|-----|-----|-----|
IANA | | | | | 3530790 | 52.5 % | 119.12 |
-----|-----|-----|-----|-----|-----|-----|

5 passing (7s)

```

Figura 10.3: Ejecución de test utilizando el plugin hardhat-gas-reporter

# Validación de la solución

## 11.1 Introducción

En este capítulo se describen las especificaciones del contrato inteligente y scripts utilizados para brindar seguridad al intercambio de mensajes BGP entre pares, a través de la tecnología blockchain.

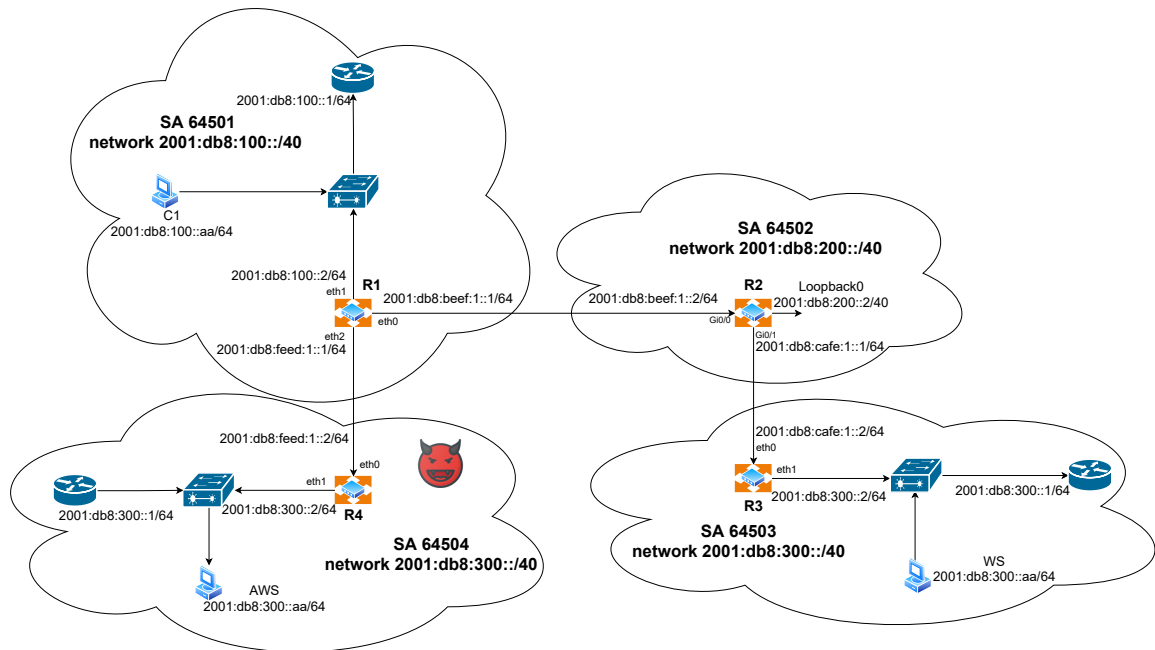


Figura 11.1: Red de prueba IPv6 - Emulación de secuestro de rutas

En el router R1 del SA 64501 de la red de pruebas ilustrada en la figura 11.1, se ejecuta un cliente implementado en Python (Anexo 14.5). Este script utiliza funciones de lectura de la blockchain para verificar dos requisitos después de recibir un paquete UPDATE:

1. Verificar que el primer SA en la ruta recibida está autorizado para anunciar el prefijo.
2. Verificar que el camino de los SAs que recorre el anuncio, el atributo AS-PATH, sea válido.

Finalmente para las rutas que fueron validadas el script cliente configura una preferencia local más alta. Esta medida permite influir en las decisiones de enrutamiento y priorizar rutas que cumplen con los requisitos mencionados anteriormente, permitiendo garantizar una red más segura y confiable.

## 11.2 Entorno de software utilizado para la realización de pruebas

El entorno de software utilizado para la realización de pruebas en la topología de red IPv6 fue el simulador GNS3. GNS3 es un software gratuito que permite emular diferentes tipos de hardware de una red y ejecutar imágenes reales en dispositivos virtuales. Con GNS3 se pueden crear topologías

avanzadas sin limitación en la cantidad de dispositivos, lo que lo convierte en una herramienta ideal para realizar pruebas de red complejas.

En este caso, se emuló la topología de red IPv6 utilizada como banco de pruebas en GNS3. Este software permite crear redes virtuales que se asemejan a redes reales, lo que facilita la realización de pruebas y experimentos. GNS3 incluye imágenes de dispositivos como Cisco VIRL (IOSvL2, IOSvL3, ASAv), así como otras imágenes que requieren Qemu o imágenes de Docker como hosts en topologías de red emuladas. Esto permite una gran flexibilidad y adaptabilidad al entorno de prueba.

El banco de pruebas utilizado para la realización de las pruebas se presenta en el diagrama de la figura 11.2, y se utilizan diferentes imágenes para los routers involucrados en la topología de red.

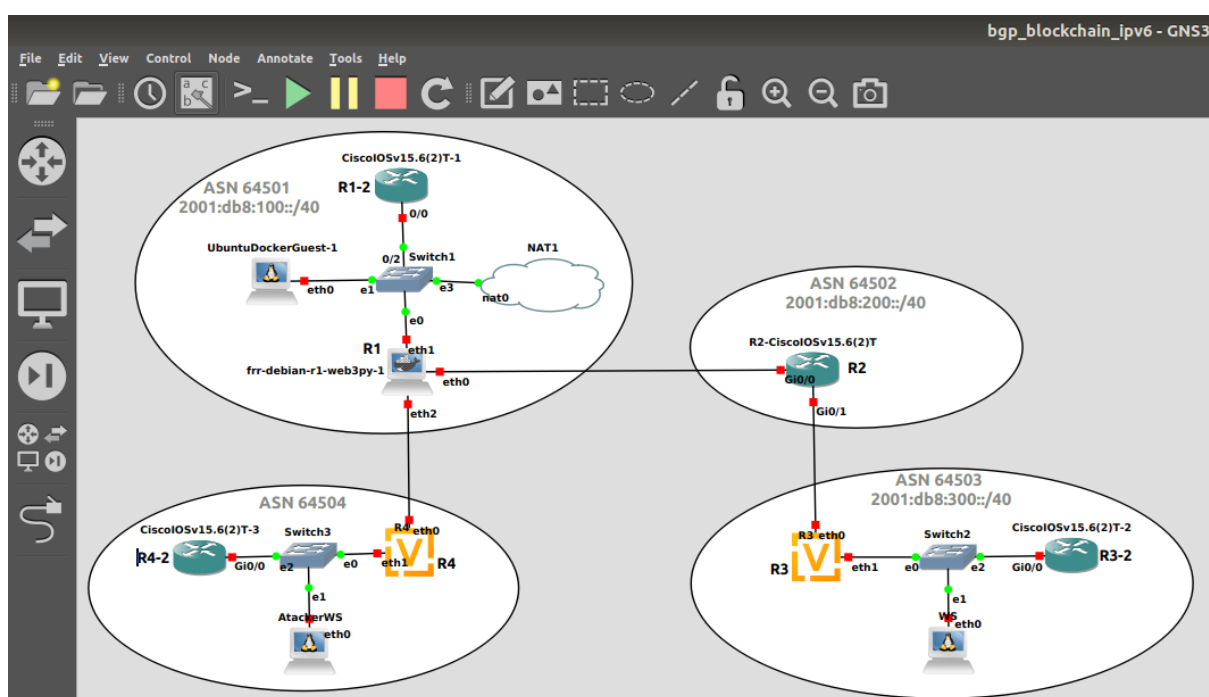


Figura 11.2: Topología de red IPv6 - Emulación de un secuestro de rutas en GNS3

Para el router R1, se utiliza un contenedor docker Debian 10(buster) con FRRouting[102] instalado. FRRouting es una suite de protocolos de ruteo de código abierto para plataformas Linux y Unix. Sobre este router se instaló la librería de python web3.py y otros paquetes necesarios para interactuar con blockchain EVM compatibles. En el Anexo 14.8 se desarrollan los pasos de la instalación.

El router R2, pertenece al ASN 64502, tiene asignado el prefijo 2001:db8:300::/40 y utiliza la imagen Cisco IOSv 15.6(2)T[103], que permite ejecutar IOS en una computadora estándar.

El router R3, perteneciente al ASN víctima (ASN64503), tiene asignado el prefijo 2001:db8:300::/40, y utiliza una imagen del router VyOS[104], un sistema operativo de red de código abierto que nació como una bifurcación comunitaria de Vyatta.

El router R4, perteneciente al ASN64504, es el que realiza el ataque de BGP Hijacking, y también



utiliza una imagen del router VyOS. Las relaciones de peering que se establecen entre los diferentes routers son las siguientes: el router R1 establece una relación de peering con los routers R2 y R4. El router R2 establece una relación de peering con los routers R1 y R3. El router R3 establece una relación de peering con el router R2. Y finalmente, el R4, el atacante, establece una relación de peering con el R1. El detalle de los comandos de configuración de los routers mencionados se muestra en el Anexo 14.8.

En la red GNS3 utilizada para las pruebas de secuestro de rutas (Figura 11.2), también se observa la presencia de un servidor web denominado WS, con dirección IPv6 2001:db8:300:aa. Este servidor web es consultado desde el cliente C1, que se encuentra en el contenedor UbuntuDockerGuest1.

El servidor web WS se implementa en NodeJS (Anexo 14.6) y pertenece al AS64503. Cuando se realiza en forma exitosa el ataque de BGP Hijacking desde el AS64504, el cliente C1 es redirigido al servidor web del atacante, denominado AttackerWS. Esto ocurre debido a que el atacante ha logrado manipular las rutas BGP que se utilizan para dirigir el tráfico en la red. En consecuencia, el tráfico que originalmente debería ser dirigido al servidor web WS es redirigido al servidor web del atacante.

Este comportamiento demuestra la gravedad del ataque de BGP Hijacking y la importancia de tomar medidas de seguridad para evitar este tipo de incidentes en la red. La manipulación de las rutas BGP puede tener un impacto significativo en el tráfico de red y la seguridad de los datos que se transmiten a través de la red. Por lo tanto, es fundamental contar con medidas de seguridad adecuadas para prevenir este tipo de ataques y minimizar su impacto en caso de que se produzcan.

Después de haber establecido el entorno de pruebas utilizando GNS3 y las imágenes de routers correspondientes, se llevan a cabo dos emulaciones del ataque de Route Hijacking. La primera donde el ataque se realiza de forma exitosa, y la segunda donde el mismo se evita utilizando la red blockchain.

Para lograrlo, se utiliza la librería de Python web3.py instalada en el router R1 para interactuar con la blockchain EVM compatible. Con esta tecnología, se consigue implementar una solución de confianza descentralizada en la que las rutas BGP se verifican y validan mediante consultas a la red blockchain.

Cuando se ejecuta un ataque de Route Hijacking desde el ASN64504, se activa en R1 el mecanismo de seguridad basado en la biblioteca web3.py (Anexo 14.5). Este mecanismo tiene como objetivo validar las rutas BGP. Al hacerlo, incrementa la preferencia local de las rutas que han sido validadas, lo que previene que el tráfico del cliente C1 sea redirigido hacia el servidor web del atacante. Como consecuencia de esta medida de seguridad, el tráfico se dirige al servidor web WS original, que está alojado en el servidor web del SA 64503.

Este resultado demuestra el potencial de la tecnología blockchain para mejorar la seguridad en las redes que forman Internet y prevenir ataques como el de Route Hijacking. Al implementar soluciones de confianza descentralizadas, se puede garantizar que las rutas BGP sean seguras y confiables, lo que minimiza el riesgo de ataques malintencionados y protege la integridad del tráfico de red.

En la siguiente sección se describe a grandes rasgos el script en python `ids.py` y se muestran en detalle las emulaciones realizadas.

### 11.3 Emulación de ataque de Route Hijacking exitoso

Una vez iniciada la emulación, se establecen las relaciones de peering entre los routers.

En las siguientes figuras (11.3 y 11.4) se observan las respuestas de los comandos **show bgp ipv6 summary** y **show bgp ipv6 unicast 2001:db8:300::/40** en el router R1, previo al ataque de BGP hijacking, cuando todavía el router R4 no realiza el falso anuncio.

```
frr-debian-r1-web3py-1# show bgp ipv6 summary
IPv6 Unicast Summary:
BGP router identifier 192.168.122.220, local AS number 64501 vrf-id 0
BGP table version 5
RIB entries 5, using 960 bytes of memory
Peers 2, using 43 KiB of memory

Neighbor      V      AS    MsgRcvd  MsgSent  TblVer  InQ  OutQ  Up/Down  State/PfxRcd  PfxSnt
2001:db8:beef:1::2 4      64502    223      201       0     0     0 03:14:43      2          3
2001:db8:feed:1::2 4      64504     33       33       0     0     0 00:26:26      0          3

Total number of neighbors 2
```

Figura 11.3: Ejecución del comando **show bgp ipv6 summary** antes del ataque de BGP hijacking

```
frr-debian-r1-web3py-1# show bgp ipv6 unicast 2001:db8:300::/40
BGP routing table entry for 2001:db8:300::/40
Paths: (1 available, best #1, table default)
  Advertised to non-peer-group peers:
    2001:db8:beef:1::2 2001:db8:feed:1::2
    64502 64503
    2001:db8:beef:1::2 from 2001:db8:beef:1::2 (9.0.0.2)
      Origin IGP, valid, external, best (First path received)
  Last update: Wed May 19 16:39:19 2021
```

Figura 11.4: Ejecución del comando **show bgp ipv6 unicast 2001:db8:300::/40** antes del ataque de BGP hijacking

Mediante la ejecución del comando que se observa en la figura 11.4 se obtiene que el número IP `2001:db8:beef:1::2` es el siguiente salto para el prefijo `2001:db8:300::/40`, y que el tráfico se dirige a través de los SA `64502` y `64503`.

Cuando se consulta el servidor web `WS(2001:db8:300:aa)` desde el cliente `C1 (UbuntuDockerGuest-1)` se observa como se accede al sitio principal del servidor web llamado `WS`, perteneciente al SA `64503` Figura (11.5).

```
root@UbuntuDockerGuest-1:~# curl [2001:db8:300::aa]
<h1>Hola mundo! Soy el servidor web del ASN 64503</h1>
```

Figura 11.5: Ejecución del comando **curl [2001:db8:300::aa]** desde el cliente docker C1 antes del ataque de BGP hijacking

Posteriormente se inicia el robo de rutas al anunciar el prefijo `2001:db8:300::/40` desde el router R4:

```
vyos@vyos# set protocols bgp 64504 address-family ipv6-unicast network 2001:db8:300::/40
```

Una vez que el SA deshonesto(SA 64504) anuncia falsamente una ruta más corta para alcanzar el prefijo 2001:db8:300::/40, el SA 64501 dirige el tráfico destinado a ese prefijo a través del camino más corto, hacia el router R4.

Observamos ahora en las figuras (11.6 y 11.7) como son las salidas de los comandos **show bgp ipv6 summary** y **show bgp ipv6 unicast 2001:db8:300::/40** en el router R1, después del ataque desde el SA 64504.

```
frr-debian-r1-web3py-1# show bgp ipv6 summary

IPv6 Unicast Summary:
BGP router identifier 192.168.122.220, local AS number 64501 vrf-id 0
BGP table version 6
RIB entries 5, using 960 bytes of memory
Peers 2, using 43 KiB of memory

Neighbor      V      AS  MsgRcvd  MsgSent  TblVer  InQ  OutQ  Up/Down  State/PfxRcd  PfxSnt
2001:db8:beef:1::2 4    64502    229     207      0     0     0 03:19:20      2         3
2001:db8:feed:1::2 4    64504     39      39      0     0     0 00:31:03      1         3

Total number of neighbors 2
```

Figura 11.6: Ejecución del comando **show bgp ipv6 summary** después del ataque de BGP hijacking

```
frr-debian-r1-web3py-1# show bgp ipv6 unicast 2001:db8:300::/40
BGP routing table entry for 2001:db8:300::/40
Paths: (2 available, best #1, table default)
  Advertised to non-peer-group peers:
    2001:db8:beef:1::2 2001:db8:feed:1::2
    64504
    2001:db8:feed:1::2 from 2001:db8:feed:1::2 (9.0.4.2)
      (fe80::e6f:51ff:fe32:f00) (used)
      Origin IGP, metric 0, valid, external, best (AS Path)
      Last update: Wed May 19 19:58:18 2021
    64502 64503
    2001:db8:beef:1::2 from 2001:db8:beef:1::2 (9.0.0.2)
      Origin IGP, valid, external
      Last update: Wed May 19 16:39:19 2021
```

Figura 11.7: Ejecución del comando **show bgp ipv6 unicast 2001:db8:300::/40** después del ataque de BGP hijacking

A partir de la ejecución del último comando, se determina que la ruta seleccionada para el prefijo 2001:db8:300::/40 dirige el siguiente salto hacia la dirección IP 2001:db8:feed:1::2. Este camino conduce al SA 64504, en lugar de dirigirse a los SAs 64502 - 64503 como se esperaba.

Se constata que al no ejecutar el script en python de validación en la blockchain en el router R1(SA 64501), el ataque de BGP hijacking ejecutado por parte del router R4(SA 64504) resulta exitoso. De esta manera, logra apoderarse del prefijo 2001:db8:300::/40, que originalmente pertenece al SA 64503

Finalmente, al realizar una consulta al servidor web WS (2001:db8:300:aa) desde el cliente C1 (UbuntuDockerGuest1), se accede al sitio principal del servidor web del atacante, denominado AWS (Attacker Web Server). Este servidor pertenece al SA 64504, lo que confirma el éxito del ataque (Figura 11.8).

```
root@UbuntuDockerGuest-1:~# curl [2001:db8:300::aa]
<h1>Hola mundo! Soy el servidor web del atacante en el ASN 64504</h1>
```

Figura 11.8: Ejecución del comando `curl [2001:db8:300::aa]` desde el cliente docker C1 después del ataque de BGP hijacking. El SA65504 logra apropiarse del prefijo.

## 11.4 Emulación de ataque de Route Hijacking evitado mediante el uso de tecnología blockchain

Al comenzar la emulación en GNS3 se establecen las relaciones de peering entre los routers. Posteriormente se inicia el script Python en el router R1 con el comando: `python3 ids.py` (Anexo 14.5).

Posteriormente se ejecutan los comandos para ver el estado de las sesiones BGP y las rutas posibles hacia el prefijo `2001:db8:300::/40`.

En las siguientes figuras (11.9 y 11.10) se observan las respuestas de los comandos `show bgp ipv6 summary` y `show bgp ipv6 unicast 2001:db8:300::/40` previo al ataque de BGP hijacking, donde todavía el router R4 no realiza el falso anuncio.

```
frr-debian-r1-web3py-1# show bgp ipv6 summary
IPv6 Unicast Summary:
BGP router identifier 192.168.122.220, local AS number 64501 vrf-id 0
BGP table version 7
RIB entries 5, using 960 bytes of memory
Peers 2, using 43 KiB of memory

Neighbor      V      AS  MsgRcvd  MsgSent  TblVer  InQ  OutQ  Up/Down  State/PfxRcd  PfxSnt
2001:db8:beef:1::2 4    64502    25      23       0     0     0 00:02:13    2           3
2001:db8:feed:1::2 4    64504    12      15       0     0     0 00:07:23    0           3

Total number of neighbors 2
```

Figura 11.9: Ejecución del comando `show bgp ipv6 summary` antes del ataque de BGP hijacking

```
frr-debian-r1-web3py-1# show bgp ipv6 unicast 2001:db8:300::/40
BGP routing table entry for 2001:db8:300::/40
Paths: (1 available, best #1, table default)
  Advertised to non-peer-group peers:
    2001:db8:beef:1::2 2001:db8:feed:1::2
    64502 64503
    2001:db8:beef:1::2 from 2001:db8:beef:1::2 (9.0.0.2)
  Origin IGP, localpref 150, valid, external, best (First path received)
  Last update: Thu May 20 10:44:43 2021
```

Figura 11.10: Ejecución del comando `show bgp ipv6 unicast 2001:db8:300::/40` antes del ataque de BGP hijacking

Mediante este último comando se obtiene que el número IP `2001:db8:beef:1::2` es el siguiente salto para el prefijo `2001:db8:300::/40`, y que el tráfico se dirige a través de los SA 64502 y 64503.

En la figura 11.11 vemos los reportes del script de validación en Python. El mismo intercepta los paquetes UPDATE BGP utilizando la librería `NetfilterQueue` del proyecto `Netfilter`, la cual proporciona acceso a paquetes que coinciden con una regla iptables en Linux. Los paquetes así emparejados pueden posteriormente ser analizados con el fin de validar los mismos con la información de la blockchain (verificar que el primer SA en la ruta recibida está autorizado para anunciar el prefijo

y que el camino de los SAs que recorre el anuncio, el atributo AS-PATH, sea válido). Finalmente para las rutas que fueron completamente validadas el script invoca a la función `policy_agent`, la cual configura una preferencia local más alta para las mismas.

```

pkt arrived
{'peer': '2001:db8:beef:1::2', 'AS': '64503', 'next hop': '2001:db8:beef:1::2', 'prefix': '2001:db8:300::', 'length': '40'}
#####
This update message was verified at IANA contract
Checking prefix ...
pkt arrived
Checking path array: ['64502', '64503']
Received valid update, calling policy agent
Installing local preference for checked path

```

Figura 11.11: Ejecución de script de validación en Python

Cuando se consulta el servidor web WS(2001:db8:300:aa) desde el cliente C1 se observa como se accede al sitio principal del servidor web WS, perteneciente al SA 64503 (11.12).

```

root@Ubuntu
root@UbuntuDockerGuest-1:~# curl [2001:db8:300::aa]
<h1>Hola mundo! Soy el servidor web del ASN 64503</h1>

```

Figura 11.12: Ejecución del comando `curl [2001:db8:300::aa]` desde el cliente docker C1 antes del ataque de BGP hijacking

Posteriormente se anuncia el prefijo 2001:db8:300::/40 desde el router R4

**`vynos@vynos# set protocols bgp 64504 address-family ipv6-unicast network '2001:db8:300::/40'`**

Una vez que el SA deshonesto(SA 64504) anuncia falsamente una ruta más corta para alcanzar el prefijo 2001:db8:300::/40, el SA 64501 no dirige el tráfico destinado a ese prefijo hacia el SA64504. Por más que el camino más corto es a través del router R4, se prefiere la ruta validada por el script hacia el SA64503. Esto es a causa de que tiene una preferencia local de 150.

Observamos ahora en la figura 11.13 como es la respuesta del comando **`show bgp ipv6 unicast 2001:db8:300::/40`** después del ataque.

```

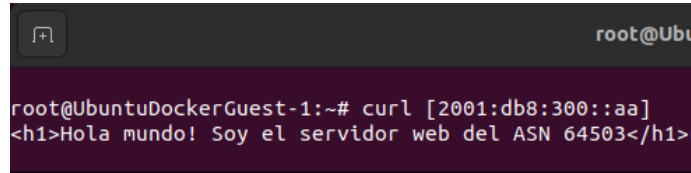
frr-debian-r1-web3py-1# show bgp ipv6 unicast 2001:db8:300::/40
BGP routing table entry for 2001:db8:300::/40
Paths: (2 available, best #2, table default)
  Advertised to non peer-group peers:
    2001:db8:beef:1::2 2001:db8:feed:1::2
  64504
    2001:db8:feed:1::2 from 2001:db8:feed:1::2 (9.0.4.2)
      (fe80::e6f:51ff:fe32:f00) (used)
      Origin IGP, metric 0, valid, external
      Last update: Thu May 20 10:50:01 2021
  64502 64503
    2001:db8:beef:1::2 from 2001:db8:beef:1::2 (9.0.0.2)
      Origin IGP, localpref 150, valid, external, best (Local Pref)
      Last update: Thu May 20 10:44:43 2021

```

Figura 11.13: Ejecución del comando **`show bgp ipv6 unicast 2001:db8:300::/40`** después del ataque de BGP hijacking

Podemos concluir que a causa de que se ejecutó en el router R1(SA 64501) el script Python de validación en la blockchain, el ataque de BGP hijacking por parte del router R4(SA 64504) no tuvo éxito y este no logró apropiarse del prefijo perteneciente al SA 64503.

Si ahora realizamos la consulta al servidor web WS(2001:db8:300:aa) desde el cliente C1 se observa que se accede al sitio principal del servidor web NGINX llamado WS, perteneciente al SA 64503 (11.14).

A terminal window with a dark background. The prompt is 'root@UbuntuDockerGuest-1:~#'. The command entered is 'curl [2001:db8:300::aa]'. The output is '<h1>Hola mundo! Soy el servidor web del ASN 64503</h1>'. The terminal title bar shows 'root@Ubu' on the right and a window icon on the left.

```
root@UbuntuDockerGuest-1:~# curl [2001:db8:300::aa]
<h1>Hola mundo! Soy el servidor web del ASN 64503</h1>
```

**Figura 11.14:** Ejecución del comando `curl [2001:db8:300::aa]` desde el cliente docker C1 después del ataque de BGP hijacking. El mismo no tuvo éxito.

## Conclusiones

En este trabajo se han explorado diversas soluciones para mejorar la seguridad del enrutamiento en Internet, y se ha encontrado que RPKI y ASPA son las soluciones más ampliamente difundidas en la actualidad. Sin embargo, se ha identificado que estas soluciones presentan algunos desafíos y limitaciones en términos de centralización y seguridad.

La hipótesis planteada en esta tesis ha sido respaldada por los resultados obtenidos. Al utilizar blockchain para almacenar las asignaciones y delegaciones de recursos de Internet, se ha demostrado que es posible facilitar un sistema de enrutamiento seguro entre sistemas autónomos. La implementación de contratos inteligentes en una red Ethereum o similar ha demostrado ser viable y efectiva para la gestión de recursos de Internet, brindando un modelo de confianza flexible, equilibrando el poder entre los actores participantes, simplificando la administración y proporcionando un sistema altamente auditable sin depender de certificados digitales ni control centralizado.

En cuanto a las soluciones existentes, se ha identificado que RPKI y ASPA presentan un potencial punto de falla: la autoridad centralizada. Los ataques dirigidos a esta autoridad centralizada pueden comprometer la seguridad del enrutamiento en Internet. Por lo tanto, es necesario buscar alternativas más descentralizadas, equilibradas y seguras para los procesos que gobiernan la infraestructura de Internet.

En este contexto, la solución propuesta en esta tesis, basada en la tecnología blockchain, se presenta como una alternativa prometedora. Al descentralizar la gestión de recursos de Internet y proporcionar un mecanismo de consenso distribuido, se pueden abordar los desafíos de seguridad y confianza asociados con las soluciones centralizadas. Además, se ha demostrado que la implementación de un prototipo funcional utilizando contratos inteligentes en una red Ethereum Virtual Machine (EVM) compatible es factible, y brinda resultados positivos en términos de seguridad y descentralización.

Sin embargo, es importante tener en cuenta los desafíos económicos asociados con el uso de la red Ethereum mainnet. Los costos de transacción pueden hacer que la solución propuesta no sea económicamente viable en su forma actual. Por lo tanto, se necesitará explorar soluciones alternativas, como segundas capas o redes blockchain público-permisionadas, que puedan proporcionar una mayor eficiencia y escalabilidad económica.

En conclusión, esta tesis ha logrado analizar las propiedades de blockchain y su aplicación específica en la asignación y delegación de recursos de la infraestructura de Internet, así como la validación de la solución, en la seguridad del ruteo externo BGP, mediante la emulación de una red en GNS3. Se ha comprobado la hipótesis planteada, validando los paquetes UPDATE de BGP mediante el script en Python, demostrando que la tecnología blockchain puede facilitar un sistema de enrutamiento seguro, descentralizado y confiable entre sistemas autónomos.

Como trabajo futuro, se recomienda continuar investigando soluciones innovadoras basadas en tecnologías emergentes y colaborar en enfoques multidisciplinarios para abordar los desafíos críticos

en la seguridad del enrutamiento en Internet y promover una mayor descentralización, equilibrio y democracia en los procesos que nos gobiernan.



# Futuras líneas de investigación

## 13.1 Información adicional on-chain

Adicionalmente se puede extender la arquitectura para mantener en la blockchain información útil adicional. Por ejemplo: garantizar y validar los contactos de abuso del WHOIS (abuse-c y abuse-mailbox) de las organizaciones (referencias), u objetos AS-Cone1.

## 13.2 Soporte dual stack IPv6/IPv4

La mayoría de las implementaciones de doble pila permiten que las aplicaciones solo IPv6 interoperen con nodos IPv4 e IPv6. Los paquetes IPv4 dirigidos a aplicaciones IPv6 en un nodo de doble pila llegan a destino porque sus direcciones se mapean utilizando direcciones IPv6 tipo IPv4-mapped (RFC 4038).

Las direcciones IPv4-mapped, 0:0:0:0:FFFF:w.x.y.z o ::FFFF:w.x.y.z, se emplean para representar las direcciones de un nodo IPv4 como dirección tipo IPv6 (por ejemplo, la dirección ::FFFF:192.168.0.1 representa a la dirección IPv4 192.168.0.1. Esta notación se utiliza solo para representación interna, la dirección asignada a IPv4 nunca se utiliza como dirección de origen o destino dentro de un paquete IPv6, IPv6 no admite el uso de direcciones IPv4-mapped.

## 13.3 Verificación del AS-PATH mediante el método ASPA

El método “ASPA” ó Autorización de proveedor de sistema autónomo (ASPA) se puede implementar almacenando en la blockchain las distintas asociaciones del SA con sus peers. Esta información permite que se pueda verificar que un dueño de un AS Cliente(CAS) ha autorizado a un AS Proveedor(PAS) en particular para propagar los anuncios de ruta BGP IPv4 o IPv6 pertenecientes al cliente, a los upstreams o pares del AS Proveedor. Permitiría detectar AS\_PATHs inválidos incluso durante la adopción temprana e incremental.

## 13.4 Pruebas de la solución en redes público-permisionadas

La investigación y exploración de infraestructuras blockchain público-permisionadas ofrece una oportunidad para ampliar el alcance y la aplicabilidad de las soluciones propuestas, al tiempo que se abordan desafíos clave como la eficiencia económica, la escalabilidad y el impacto ambiental.

Un ejemplo de ello es la exploración de infraestructuras como LACNet[105], que ofrecen garantías de escalabilidad y bajos costos de transacción. De esta manera, se estaría impulsando la evolución hacia un entorno de gestión de recursos de Internet más descentralizado y equilibrado, que promueva la seguridad, la confianza y el cumplimiento de los acuerdos comerciales en un marco sostenible y eficiente.

# Anexos

## 14.1 Contrato inteligente IANA.sol

### Código 14.1 IANA.sol

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.18;
3
4 contract IANA {
5
6     struct Prefix {
7         uint128 ip;
8         uint8 mask;
9         uint32 owningAS;
10        uint256 expiryOf;
11        uint[] subPrefixes; // Pointer to prefix index in the larger prefixes array.
12    }
13
14    enum PrefixCompareResult {
15        OneContainsTwo,
16        TwoContainsOne,
17        NoIntersection,
18        Equal
19    }
20
21    // All the people who can change the function pointers
22    mapping (address => bool) public ownerList;
23    // The associative mapping that maps ASNs to their owner's public key.
24    mapping (uint32 => address) public ASNList;
25    // The user should sign the data along with a unique nonce value each time
26    mapping (address => mapping(uint => bool)) public nonceUsedMap;
27    // List of prefixes.
28    Prefix[] public prefixes;
29    // Holds the table of links keyed by sha256(encodePacked(ASN1,ASN2))
30    // A link is valid if both ASN1->ASN2 and ASN2->ASN1 exist.
31    // This particular structure has the potential to be astoundingly large.
32    mapping (bytes32 => bool) links;
33
34    uint private leaseTime = 365 days;
35    uint private renewPrefixPrice = 0.1 ether;
36
37    /// Simple modifier to ensure that only owners can make changes
38    modifier onlyOwners {
39        require(ownerList[msg.sender] == true);
40        _;
41    }
42
43    constructor() {
44        // Automatically add the contract creator as an owner
45        ownerList[msg.sender] = true;
46
47        // Build up the prefix for the root prefix
48        Prefix memory rootPrefix;
49        rootPrefix.ip = 0;
```

```

50     rootPrefix.mask = 0;
51     rootPrefix.owningAS = 0;
52     rootPrefix.expiryOf = type(uint).max;
53     prefixes.push(rootPrefix);
54
55     // Mark that the root is owned by a dummy address.
56     ASNList[0] = address(0);
57 }
58
59 function getEthSignedMessageHash(bytes32 _messageHash) public pure returns (bytes32)
60 {
61     /*
62     Signature is produced by signing a keccak256 hash with the following format:
63     "\x19Ethereum Signed Message\n" + len(msg) + msg
64     */
65     return
66         keccak256(
67             abi.encodePacked("\x19Ethereum Signed Message:\n32", _messageHash)
68         );
69
70     function recoverSigner(
71         bytes32 _ethSignedMessageHash,
72         bytes memory _signature
73     ) public pure returns (address) {
74         (bytes32 r, bytes32 s, uint8 v) = splitSignature(_signature);
75
76         return ecrecover(_ethSignedMessageHash, v, r, s);
77     }
78
79     function splitSignature( bytes memory sig ) public pure returns (bytes32 r, bytes32
80     s, uint8 v) {
81         require(sig.length == 65, "invalid signature length");
82
83         assembly {
84             /*
85             First 32 bytes stores the length of the signature
86
87             add(sig, 32) = pointer of sig + 32
88             effectively, skips first 32 bytes of signature
89
90             mload(p) loads next 32 bytes starting at the memory address p into memory
91             */
92
93             // first 32 bytes, after the length prefix
94             r := mload(add(sig, 32))
95             // second 32 bytes
96             s := mload(add(sig, 64))
97             // final byte (first byte of the next 32 bytes)
98             v := byte(0, mload(add(sig, 96)))
99         }
100
101         // implicitly return (r, s, v)
102     }
103
104     function set_renewPrefixPrice(uint _price ) public onlyOwners {
105         renewPrefixPrice = _price ;
106     }

```

```

107
108     /// Compares the address spaces of two prefixes and determines if they are the same,
109     /// do not intersect, or if one contains another.
110     /// @param ip1 IP Address of the first prefix
111     /// @param mask1 Number of bits in the subnet mask for the first prefix
112     /// @param ip2 IP Address of the second prefix
113     /// @param mask2 Number of bits in the subnet mask for the second prefix
114     /// @return PrefixCompareResult The relationship between the two sets of prefix data
115     .
116     function prefix_comparePrefix(uint128 ip1, uint8 mask1, uint128 ip2, uint8 mask2)
117     public pure returns (PrefixCompareResult) {
118         // Only valid subnet masks
119         require (mask1 <= 48 && mask2 <= 48);
120         // If their masks are the same
121         if (mask1 == mask2) {
122             // As are their IPs
123             if (ip1 == ip2) {
124                 // Then they're the same.
125                 return PrefixCompareResult.Equal;
126             } else {
127                 // Otherwise they can't intersect ever.
128                 return PrefixCompareResult.NoIntersection;
129             }
130         }
131
132         // Get the number of bits we have to shift to remove the host from the address.
133         uint8 shift;
134         // We need to pick the one with the largest address space (smallest mask) since
135         // we're testing for containment.
136         if (mask1 < mask2) {
137             shift = (128 - mask1);
138         } else {
139             shift = (128 - mask2);
140         }
141
142         // Shift the two addresses over so we can look at their network address alone.
143         uint128 network1 = ip1 >> shift;
144         uint128 network2 = ip2 >> shift;
145
146         // If they reference different networks, then there isn't an intersection.
147         if (network1 != network2) {
148             return PrefixCompareResult.NoIntersection;
149         }
150
151         // If they reference the same network, it means the bigger one contains the
152         // smaller one.
153         if (mask1 < mask2) {
154             return PrefixCompareResult.OneConatinsTwo;
155         } else {
156             return PrefixCompareResult.TwoContainsOne;
157         }
158     }
159
160     /// Alias function to get just the containingPrefix.
161     /// @param startingPrefixIndex The index of the entry to use to start the search.
162     /// Use 0 to select the root prefix (0.0.0.0/0)
163     /// @param ip The IP address of the prefix to locate
164     /// @param mask The number of bits in the netmask for the prefix to locate

```

```

159  /// @return uint Index of the most specific, wholly containing prefix or 0 if it is
    not contained by the prefix at startingPrefixIndex
160  function prefix_getContainingPrefix(uint startingPrefixIndex, uint128 ip, uint8 mask
    ) public view returns (uint) {
161      (uint containingPrefix, ) = prefix_getContainingPrefixAndParent(
    startingPrefixIndex, ip, mask);
162      return containingPrefix;
163  }
164
165  /// Gets the most specific prefix that wholly contains or is exactly equal to the
    specified prefix as found in the prefixes database.
166  /// @param startingPrefixIndex The index of the entry to use to start the search.
    Use 0 to select the root prefix (0.0.0.0/0)
167  /// @param ip The IP address of the prefix to locate
168  /// @param mask The number of bits in the netmask for the prefix to locate
169  /// @return uint,uint Tuple containing (index of the most specific, wholly
    containing prefix or 0 if it is not contained by the prefix at startingPrefixIndex,
170  /// @return the parent index of the containing prefix or 0 if it is not contained or
    is a child of the root.)
171  function prefix_getContainingPrefixAndParent(uint startingPrefixIndex, uint128 ip,
    uint8 mask) public view returns (uint,uint) {
172      // Only valid subnet masks
173      require (mask <= 48);
174
175      // Get a handle to our starting prefix.
176      Prefix storage startingPrefix = prefixes[startingPrefixIndex];
177
178      // If the prefix has been deleted and isn't root...
179      if (startingPrefix.ip == 0 && startingPrefix.mask == 0 && startingPrefixIndex !=
    0) {
180          // No match.
181          return (0,0);
182      }
183
184      // Find out the relationship of ourselves to the prefix
185      PrefixCompareResult comparison = prefix_comparePrefix(startingPrefix.ip,
    startingPrefix.mask, ip, mask);
186
187      if (comparison == PrefixCompareResult.Equal) {
188          // This is us. Return ourselves.
189          return (startingPrefixIndex, startingPrefixIndex);
190      } else if (comparison == PrefixCompareResult.NoIntersection) {
191          // We don't have anything to do with this. Say we don't know.
192          return (0,0);
193      } else if (comparison == PrefixCompareResult.TwoContainsOne) {
194          // The prefix is bigger than us, we don't contain it.
195          return (0,0);
196      }
197      else {
198          // At this point we know the unknown is contained in us. But we need more;
    is it owned by a subprefix? The best way to find out is to ask.
199          for (uint i=0; i<startingPrefix.subPrefixes.length; i++) {
200              uint testIndex = startingPrefix.subPrefixes[i];
201              (uint result, uint parent) = prefix_getContainingPrefixAndParent(
    testIndex, ip, mask);
202              // If this child reports that they know who contains it
203              if (result != 0) {
204                  // Check if the parent is the same as the result. If it is, they're
    a leaf node.

```

```

205         if (result == parent) {
206             // And that means we're their first parent, so return ourselves
as the parent.
207             return (result, startingPrefixIndex);
208         } else {
209             // Otherwise, we're further up the tree, and that means we just
pass the result along as-is.
210             return (result, parent);
211         }
212     }
213 }
214
215     // If we know we contain it, but none of our children claimed it, then it's
ours. Return ourselves.
216     return (startingPrefixIndex, startingPrefixIndex);
217 }
218 }
219
220 /// Adds the specified prefix to the prefix table. Must be done by the owner of the
prefixes containing
221 /// AS and must include the signature of the message returned by
IANA_getPrefixSignatureMessage for the new AS.
222 /// @param ip The IP address of the prefix to add
223 /// @param mask The number of bits in the netmask of the prefix to add
224 /// @param newOwnerAS The AS number to associate with the new prefix to.
225 /// @param _signature The signature.
226 function prefix_addPrefix(uint128 ip, uint8 mask, uint32 newOwnerAS, uint nonce,
bytes memory _signature) public {
227     // Only valid subnet masks
228     require (mask <= 48, 'Only valid subnet masks');
229     // Get the ASN's owner
230     address newOwnerAddress = ASNList[newOwnerAS];
231     // The owning ASN must exist
232     require (newOwnerAddress != address(0), 'The owning ASN must exist');
233     // check if the user has previously sent the signature with that nonce
234     require (nonceUsedMap[newOwnerAddress][nonce] == false, 'The user has previously
sent the signature with that nonce');
235     // The owning ASN must have signed the message.
236     require (recoverSigner(getEthSignedMessageHash(IANA_getPrefixSignatureMessage(ip,
mask, newOwnerAS, newOwnerAddress, nonce)), _signature) == newOwnerAddress, 'The
owning ASN must have signed the message');
237     // Find who owns the space this mask is in.
238     uint parentIndex = prefix_getContainingPrefix(0, ip, mask);
239     Prefix storage parent = prefixes[parentIndex];
240
241     // Require that the public address calling us owns the AS that owns the parent
prefix.
242     // (i.e. you can't claim addresses on someone else's prefix)
243     // If the parent's owningAS is 0, that means that we're making changes to the
root. Check against our internal owner list.
244     if (parent.owningAS == 0) {
245         require (ownerList[msg.sender] == true, 'We re making changes to the root,
but we re not IANA');
246     } else {
247         // Otherwise, check that our sender is the AS's owner.
248         require (msg.sender == ASNList[parent.owningAS], 'The address calling us NOT
owns the AS that owns the parent prefix');
249     }
250

```

```

251 // Require that the parent contains us.
252 require(!(parent.ip == ip && parent.mask == mask));
253
254 // For every child,
255 for (uint i=0; i<parent.subPrefixes.length; i++) {
256     uint testIndex = parent.subPrefixes[i];
257     Prefix memory child = prefixes[testIndex];
258     PrefixCompareResult result = prefix_comparePrefix(child.ip, child.mask, ip,
mask);
259     // Require that we're not trying to take their prefix.
260     require(result == PrefixCompareResult.NoIntersection);
261 }
262 // the nonce should be stored on-chain
263 nonceUsedMap[newOwnerAddress][nonce] = true;
264
265 Prefix memory newPrefix;
266 newPrefix.ip = ip;
267 newPrefix.mask = mask;
268 newPrefix.owningAS = newOwnerAS;
269 newPrefix.expiryOf = block.timestamp + leaseTime;
270 prefixes.push(newPrefix);
271 // Add it to the list
272 uint index = prefixes.length - 1;
273 parent.subPrefixes.push(index);
274 }
275
276 /// Removes the specified prefix to the prefix table. Must be done by the owner of
the prefixes containing
277 /// @param ip The IP address of the prefix to add
278 /// @param mask The number of bits in the netmask of the prefix to add
279 function prefix_removePrefix(uint128 ip, uint8 mask) public {
280     // Only valid subnet masks
281     require (mask <= 48);
282
283     // Find who owns the space this mask is in.
284     (uint index, uint parentIndex) = prefix_getContainingPrefixAndParent(0, ip, mask)
;
285     Prefix storage target = prefixes[index];
286     Prefix storage parent = prefixes[parentIndex];
287
288     // Require that the public address that calls us has the prefix
289     // (i.e. you can't claim addresses in someone else's prefix)
290     // and the expiryOf period is not expired.
291     // If expiryOf is expired it requires that the public address belongs to
ownerList
292     if (target.expiryOf > block.timestamp) {
293         require (msg.sender == ASNList[target.owningAS], "The public address that
calls us doesn't own the prefix");
294     } else {
295         require (ownerList[msg.sender] == true, "The public address doesn't belong
to ownerList");
296     }
297
298     // The prefix must exactly reference the listed prefix.
299     require(prefix_comparePrefix(target.ip, target.mask, ip, mask) ==
PrefixCompareResult.Equal);
300
301     // We only delete prefixes that have no children. They have to be deleted
seperately.

```

```

302     require(target.subPrefixes.length == 0);
303
304     //Blank the prefix (since deleting it is impossible in the contract and doesn't
help anything anyway.)
305     Prefix memory blankPrefix;
306     blankPrefix.ip = 0;
307     blankPrefix.mask = 0;
308     blankPrefix.owningAS = 0;
309     blankPrefix.expiryOf = 0;
310     prefixes[index] = blankPrefix;
311
312     // This operation is EXPENSIVE. However, without it, the arrays could get long.
Basically we're looking
313     // for our specific child node in the parent, and once we find it, we're
deleting it from the array and
314     // shrinking the array down.
315     for(uint i=0;i<parent.subPrefixes.length;i++) {
316         Prefix storage child = prefixes[parent.subPrefixes[i]];
317         if (child.ip == target.ip && child.mask == target.mask) {
318             // Skip ahead one and start shifting the array left
319             for(i=i+1;i<parent.subPrefixes.length;i++) {
320                 parent.subPrefixes[i-1] = parent.subPrefixes[i];
321             }
322             // Shrink the array.
323             parent.subPrefixes.pop();
324         }
325     }
326 }
327
328 /// Renewal the specified prefix expiryOf. Must be done by the owner of the prefixes
329 /// @param ip The IP address of the prefix to add
330 /// @param mask The number of bits in the netmask of the prefix to add
331 function prefix_renewalPrefix(uint128 ip, uint8 mask) public payable {
332     // Only valid subnet masks
333     require (mask <= 48);
334
335     // Paying the annual fee
336     require (msg.value == renewPrefixPrice);
337
338     // Find who owns the space this mask is in.
339     uint index = prefix_getContainingPrefix(0,ip, mask);
340     Prefix storage target = prefixes[index];
341
342     // The prefix must exactly reference the listed prefix.
343     require(prefix_comparePrefix(target.ip, target.mask, ip, mask) ==
PrefixCompareResult.Equal);
344
345     require (target.expiryOf > block.timestamp);
346     require (msg.sender == ASNList[target.owningAS]);
347
348     target.expiryOf = block.timestamp + leaseTime;
349 }
350 /// Returns the owner's address for the given ASN, or 0 if no one owns the ASN.
351 /// @param ASN The ASN whose owner is to be returned
352 /// @return address The address of the owner.
353 function IANA_getASNOwner(uint32 ASN) public view returns (address) {
354     return ASNList[ASN];
355 }
356

```



```

357     /// Generates the message text to be signed for add authentication.
358     /// @param ASN The ASN to be added
359     /// @param ASNOwner The public key of the new owner.
360     /// @return bytes32 The keccak256 hash of abi.encodePacked(ASN,ASNOwner).
361     function IANA_getSignatureMessage(uint32 ASN, address ASNOwner, uint _nonce, bytes4
    _functionName) pure public returns(bytes32) {
362         // The user should sign the data along with a unique nonce value each time
363         return keccak256(abi.encodePacked(ASN,ASNOwner,_nonce,_functionName));
364     }
365
366     /// Generates the message text to be signed for add authentication.
367     /// @param ASN The ASN to be added
368     /// @param ASNOwner The public key of the new owner.
369     /// @return bytes32 The keccak256 hash of abi.encodePacked(ASN,ASNOwner).
370     function IANA_getPrefixSignatureMessage(uint128 ip, uint8 mask, uint32 ASN, address
    ASNOwner, uint nonce) pure public returns(bytes32) {
371         // The user should sign the data along with a unique nonce value each time
372         return keccak256(abi.encodePacked(ip, mask, ASN, ASNOwner,nonce));
373     }
374
375     /// Adds an additional ASN to the ASN list. The operation has to include a signature
376     /// from the ASN owner signing keccak256(abi.encodePacked(ASN,ASNOwner,_nonce,
    _functionName))
377     /// which can be generated by calling IANA_getSignatureMessage()
378     /// @param ASN The ASN to be added
379     /// @param ASNOwner The public key of the new owner.
380     /// @param nonce A security nonce.
381     /// @param _signature The signature.
382     function IANA_addASN(uint32 ASN, address ASNOwner, uint nonce, bytes memory
    _signature) public onlyOwners {
383         // check if the user has previously sent the signature with that nonce
384         require(nonceUsedMap[ASNOwner][nonce] == false);
385         // It must be signed by the new ASNOwner. We don't have to check for the IANA
    owner because
386         // the onlyOwners routine does that for us.
387
388         require(recoverSigner(getEthSignedMessageHash(IANA_getSignatureMessage(ASN,
    ASNOwner, nonce, msg.sig)), _signature) == ASNOwner);
389         require(ASN != 0);
390
391         // At this point, we have two party agreement on ASN ownership. Add it to the
    ANSList.
392         ANSList[ASN] = ASNOwner;
393         // the nonce should be stored on-chain
394         nonceUsedMap[ASNOwner][nonce] = true;
395     }
396
397     /// Removes an ASN to the ASN list. The operation has to include a signature
398     /// from the ASN owner signing keccak256(abi.encodePacked(ASN,ASNOwner,_nonce,
    _functionName)) which can be
399     /// generated by calling IANA_getSignatureMessage()
400     /// @param ASN The ASN to be added
401     /// @param ASNOwner The public key of the new owner.
402     /// @param nonce A security nonce.
403     /// @param _signature The signature.
404     function IANA_removeASN(uint32 ASN, address ASNOwner, uint nonce, bytes memory
    _signature) public onlyOwners {
405         // check if the user has previously sent the signature with that nonce
406         require(nonceUsedMap[ASNOwner][nonce] == false);

```

```

407     // It must be signed by the new ASNOwner. We don't have to check for the IANA
owner because
408     // the onlyOwners routine does that for us.
409     require(recoverSigner(getEthSignedMessageHash(IANA_getSignatureMessage(ASN,
ASNOwner, nonce, msg.sig)), _signature) == ASNOwner);
410     require(ASN != 0);
411
412     for (uint i=0; i<prefixes.length; i++) {
413         require(prefixes[i].owningAS != ASN, "The ASN cannot be removed because it
has prefixes associated with it.");
414     }
415
416     // At this point, we have two party agreement on ASN ownership. Mark the ASN as
unowned
417     ASNList[ASN] = address(0);
418     // the nonce should be stored on-chain
419     nonceUsedMap[ASNOwner][nonce] = true;
420 }
421
422 /// Adds an additional user to the owners table, allowing them to modify the
discovery tables.
423 /// @param owner The public key of the new owner.
424 function IANA_addOwner(address owner) public onlyOwners {
425     ownerList[owner] = true;
426 }
427
428 /// Removes a user from the owners table, who will no longer be allowed to edit the
discovery table.
429 /// @param owner The public key of the owner to be removed.
430 function IANA_removeOwner(address owner) public onlyOwners {
431     delete(ownerList[owner]);
432 }
433
434 /// Alias for IANA_prefixCheck to allow for compatibility.
435 function prefixCheck(uint8 A, uint8 B, uint8 C, uint8 D, uint8 M, uint _asNumber )
public view returns (bool){
436     return IANA_prefixCheck(A<<24|B<<16|C<<8|D,M,_asNumber);
437 }
438
439 /// Determines if a given prefix is owned by the specified AS.
440 /// @param ip The IP address of the prefix to be checked.
441 /// @param mask The mask of the prefix to be checked.
442 /// @param asNumber the ASN of the AS we're checking the prefix against
443 /// @return bool true if the prefix's address space is owned by the listed AS and
that address space hasn't been subnet to another AS, false otherwise.
444 function IANA_prefixCheck(uint128 ip, uint8 mask, uint asNumber) public view returns
(bool) {
445     // Check if the address is part of a prefix subset that has been transferred to
another user.
446     // This will involve going to the prefix keyed data structure and enumerating it
's sub owners.
447     // If not return a false.
448
449     // At this point we know we own it, and know we haven't sold it.
450
451     uint index = prefix_getContainingPrefix(0,ip,mask);
452
453     // No hits is bad. This means it's in unallocated space.
454     if (index == 0) {

```

```

455     return false;
456 }
457
458 Prefix storage parent = prefixes[index];
459
460 // If the topmost containing isn't our AS, then it's bad.
461 if (parent.owningAS != asNumber) {
462     return false;
463 }
464
465 // For every child, make sure there isn't a subprefix that intersects this
prefix.
466 for (uint i=0; i<parent.subPrefixes.length; i++) {
467     uint testIndex = parent.subPrefixes[i];
468     Prefix storage child = prefixes[testIndex];
469     PrefixCompareResult result = prefix_comparePrefix(child.ip, child.mask, ip,
mask);
470     // Make sure there isn't
471     if (result != PrefixCompareResult.NoIntersection) {
472         return false;
473     }
474 }
475
476 return true;
477 }
478
479 /// Returns the contract address of the contract that contains the desired
function.
480 /// @param AS1 The ASN of the first end of the link
481 /// @param AS2 The ASN of the first end of the link
482 /// @return bool True if there is a valid, bidirectional link from AS1 to AS2
483 function link_validateLink(uint32 AS1, uint32 AS2) public view returns (bool) {
484     // Make the hash for the link in the forward direction
485     bytes32 linkhash1 = keccak256(abi.encodePacked(AS1, AS2));
486     // Make the hash for the link in the reverse direction
487     bytes32 linkhash2 = keccak256(abi.encodePacked(AS2, AS1));
488     // Return true if both links are in the valid link table.
489     return links[linkhash1] && links[linkhash2];
490 }
491
492 /// Marks that the caller believes it has a link to a particular destination ASN.
493 /// @param myASN The ASN the caller owns
494 /// @param destinationASN The ASN that the caller links to.
495 function link_addLink(uint32 myASN, uint32 destinationASN) public {
496     require(msg.sender == ASNList[myASN]);
497     bytes32 linkhash = keccak256(abi.encodePacked(myASN, destinationASN));
498     links[linkhash] = true;
499 }
500
501 /// Marks that the caller believes it no longer has a link to a particular
destination ASN.
502 /// @param myASN The ASN the caller owns
503 /// @param destinationASN The ASN that the caller links to.
504 function link_removeLink(uint32 myASN, uint32 destinationASN) public {
505     require(msg.sender == ASNList[myASN]);
506     bytes32 linkhash = keccak256(abi.encodePacked(myASN, destinationASN));
507     links[linkhash] = false;
508 }
509

```

## 14.2 Archivo de configuración de Hardhat

**Código 14.2** hardhat.config.js

```

1 require("@nomicfoundation/hardhat-toolbox");
2 require('dotenv').config();
3 require("hardhat-gas-reporter");
4
5 const GOERLI_RPC_URL = process.env.GOERLI_RPC_URL
6 const PRIVATE_KEY = process.env.PRIVATE_KEY
7 const COINMARKETCAP_API_KEY = process.env.COINMARKETCAP_API_KEY
8
9 /** @type import('hardhat/config').HardhatUserConfig */
10 module.exports = {
11   defaultNetwork: "hardhat",
12   networks: {
13     hardhat: {
14       accounts: [
15         { privateKey: "0
16           x503f38a9c967ed597e47fe25643985f032b072db8075426a92110f82df48dfcb", balance: "
17             1000000000000000000000000" },
18         { privateKey: "0
19           x7e5bfb82febc4c2c8529167104271ceec190eafdc277314912eaabdb67c6e5f", balance: "
20             1000000000000000000000000" },
21         { privateKey: "0
22           xcc6d63f85de8fef05446ebdd3c537c72152d0fc437fd7aa62b3019b79bd1fdd4", balance: "
23             1000000000000000000000000" },
24         { privateKey: "0
25           x638b5c6c8c5903b15f0d3bf5d3f175c64e6e98a10bdb9768a2003bf773dcb86a", balance: "
26             1000000000000000000000000" },
27         { privateKey: "0
28           xf49bf239b6e554fdd08694fde6c67dac4d01c04e0dda5ee11abee478983f3bc0", balance: "
29             1000000000000000000000000" }
30       ],
31     },
32     goerli: {
33       url: GOERLI_RPC_URL,
34       accounts: ['0x${PRIVATE_KEY}'],
35       chainId: 5,
36     }
37   },
38   solidity: "0.8.18",
39   gasReporter: {
40     enabled: true,
41     currency: "USD", // Choose the currency
42     // gasPrice: 15, // Set the gas price in Gwei
43     coinmarketcap: COINMARKETCAP_API_KEY,
44     // token: "MATIC",
45   },
46 };

```

## 14.3 Scripts para el deploy del contrato inteligente

Código 14.3 deploy.js

```
1 // We require the Hardhat Runtime Environment explicitly here. This is optional
2 // but useful for running the script in a standalone fashion through 'node <script>'.
3 //
4 // You can also run a script with 'npx hardhat run <script>'. If you do that, Hardhat
5 // will compile your contracts, add the Hardhat Runtime Environment's members to the
6 // global scope, and execute the script.
7 const hre = require("hardhat");
8
9 async function main() {
10
11   const lana = await hre.ethers.getContractFactory("IANA");
12   const iana = await lana.deploy();
13
14   await iana.deployed();
15
16   console.log(
17     'Iana deployed to ${iana.address}'
18   );
19 }
20
21 // We recommend this pattern to be able to use async/await everywhere
22 // and properly handle errors.
23 main().catch((error) => {
24   console.error(error);
25   process.exitCode = 1;
26 });
```

Código 14.4 start\_r1.js

```
1 // We require the Hardhat Runtime Environment explicitly here. This is optional
2 // but useful for running the script in a standalone fashion through 'node <script>'.
3 //
4 // You can also run a script with 'npx hardhat run <script>'. If you do that, Hardhat
5 // will compile your contracts, add the Hardhat Runtime Environment's members to the
6 // global scope, and execute the script.
7 const hre = require("hardhat");
8 const crypto = require('crypto');
9 const { BigNumber } = require("ethers");
10
11
12 async function main() {
13
14   const lana = await hre.ethers.getContractFactory("IANA");
15   const iana = await lana.attach('0xEeAaaCdD3d1F77AB33F5fB5448f5843d04E9A496');
16
17   const asnR1 = 64501;
18   const asnR2 = 64502;
19   const asnR3 = 64503;
```

```

20 const asnR4 = 64504;
21
22 const prefixR1 = BigNumber.from("0x20010db80100000000000000000000"); // 2001:db8
    :100::/40
23 const prefixR1length = 40;
24 const prefixR2 = BigNumber.from("0x20010db80200000000000000000000"); // 2001:db8
    :200::/40
25 const prefixR2length = 40;
26 const prefixR3 = BigNumber.from("0x20010db80300000000000000000000"); // 2001:db8
    :300::/40
27 const prefixR3length = 40;
28
29 const [iana_owner1, asn1_owner, asn2_owner, asn3_owner, asn4_owner] = await ethers.
    getSigners();
30
31 // 1) El ISP firma la solicitud del ASN, firmando un mensaje que contiene el ASN, el
    owner y un nonce aleatorio.
32 // creo el nonce
33 const nonce = ethers.BigNumber.from(crypto.randomBytes(32)).toString();
34 const selector = ethers.utils.keccak256(ethers.utils.toUtf8Bytes("IANA_addASN(uint32,
    address,uint256,bytes)")).slice(0, 10);
35 console.log("El ASN solicitado por R1 es %s", asnR1);
36 console.log("El address que pretende ser dueño del ASN1 es %s", asn1_owner.address);
37 // Obtengo el MessageHash
38 const msgHash = await iana.connect(asn1_owner).IANA_getSignatureMessage(asnR1,
    asn1_owner.address, nonce, selector);
39 // El agregado del prefijo lo hace automáticamente la función signMessage de ethers
40 // const prefixedMessage = ethers.utils.solidityKeccak256(["string", "bytes32"], ["\
    x19Ethereum Signed Message:\n32", msgHash]);
41
42 // Convert string to bytes and sign
43 const sig = await asn1_owner.signMessage(ethers.utils.arrayify(msgHash));
44 console.log("La firma de la solicitud del ASN %s firmado por %s es: %s", asnR1,
    asn1_owner.address, sig);
45
46 // El ISP asn1_owner envía al IANA la firma y el nonce.
47 // 2) El IANA vincula el ASN a la address del ISP utilizando la solicitud firmada y el
    nonce para mayor seguridad.
48 console.log("El IANA utilizando los datos del ASN, el address solicitante, el nonce y
    la firma, procesa la solicitud...");
49 const addASNTransaction = await iana.connect(iana_owner1).IANA_addASN(asnR1, asn1_owner
    .address, nonce, sig);
50 await addASNTransaction.wait();
51
52 const asnR1tmp = await iana.connect(iana_owner1).IANA_getASNOwner(asnR1);
53 console.log("IANA vinculo correctamente la address %s con el ASN %s", asnR1tmp, asnR1)
    ;
54
55 // El SA firman la solicitud de prefijo IP. Firma un mensaje que contiene el prefijo,
    el ASN y el owner, y lo envían al IANA.
56 const nonce2 = ethers.BigNumber.from(crypto.randomBytes(32)).toString();
57 const msgHashpfx = await iana.connect(asn1_owner).IANA_getPrefixSignatureMessage(
    prefixR1, prefixR1length, asnR1, asn1_owner.address, nonce2);
58 const sigpx = await asn1_owner.signMessage(ethers.utils.arrayify(msgHashpfx));
59
60 // El IANA, o el dueño del prefijo, recibe el mensaje firmado y lo utiliza para
    invocar a la función prefix_addPrefix mediante la cual
61 // queda establecida en la blockchain el vínculo entre el prefijo IPv6 y el ASN.
62 const addPrefixTransaction = await iana.connect(iana_owner1).prefix_addPrefix(prefixR1

```

```

    , prefixR1length , asnR1 , nonce2 , sigpx);
63 await addPrefixTransaction.wait();
64
65 const pfxcheck = await iana.connect(iana_owner1).IANA_prefixCheck(prefixR1 ,
    prefixR1length , asnR1);
66 console.log("Se vincularon los datos del prefijo IP %s / %s al ASN %s : %s", prefixR1.
    toHexString(), prefixR1length , asnR1, pfxcheck );
67
68 console.log("Enlazamos el ASN 64501 a sus vecinos , 64502 y 64504");
69 await iana.connect(asn1_owner).link_addLink(asnR1,asnR2);
70 await iana.connect(asn1_owner).link_addLink(asnR1,asnR4);
71
72 }
73
74 // We recommend this pattern to be able to use async/await everywhere
75 // and properly handle errors.
76 main().catch((error) => {
77     console.error(error);
78     process.exitCode = 1;
79 });

```

#### Código 14.5 start\_r2.js

```

1 // We require the Hardhat Runtime Environment explicitly here. This is optional
2 // but useful for running the script in a standalone fashion through 'node <script>'.
3 //
4 // You can also run a script with 'npx hardhat run <script>'. If you do that, Hardhat
5 // will compile your contracts, add the Hardhat Runtime Environment's members to the
6 // global scope, and execute the script.
7 const hre = require("hardhat");
8 const crypto = require('crypto');
9 const { BigNumber } = require("ethers");
10
11
12 async function main() {
13
14     const lana = await hre.ethers.getContractFactory("IANA");
15     const iana = await lana.attach('0xEeAaCdD3d1F77AB33F5fB5448f5843d04E9A496');
16
17     const asnR1 = 64501;
18     const asnR2 = 64502;
19     const asnR3 = 64503;
20     const asnR4 = 64504;
21
22     const prefixR1 = BigNumber.from("0x20010db80100000000000000000000"); // 2001:db8
    :100::/40
23     const prefixR1length = 40;
24     const prefixR2 = BigNumber.from("0x20010db80200000000000000000000"); // 2001:db8
    :200::/40
25     const prefixR2length = 40;
26     const prefixR3 = BigNumber.from("0x20010db80300000000000000000000"); // 2001:db8
    :300::/40
27     const prefixR3length = 40;
28

```

```

29 const [iana_owner1, asn1_owner, asn2_owner, asn3_owner, asn4_owner] = await ethers.
    getSigners();
30
31 // 1) El ISP firma la solicitud del ASN, firmando un mensaje que contiene el ASN, el
    owner y un nonce aleatorio.
32 // creo el nonce
33 const nonce = ethers.BigNumber.from(crypto.randomBytes(32)).toString();
34 const selector = ethers.utils.keccak256(ethers.utils.toUtf8Bytes("IANA_addASN(uint32,
    address,uint256,bytes)")).slice(0, 10);
35 console.log("El ASN solicitado por R2 es %s", asnR2);
36 console.log("El address que pretende ser dueño del ASN2 es %s", asn2_owner.address);
37 // Obtengo el MessageHash
38 const msgHash = await iana.connect(asn2_owner).IANA_getSignatureMessage(asnR2,
    asn2_owner.address, nonce, selector);
39 // El agregado del prefijo lo hace automáticamente la función signMessage de ethers
40 // const prefixedMessage = ethers.utils.solidityKeccak256(["string", "bytes32"], ["\
    x19Ethereum Signed Message:\n32", msgHash]);
41
42 // Convert string to bytes and sign
43 const sig = await asn2_owner.signMessage(ethers.utils.arrayify(msgHash));
44 console.log("La firma de la solicitud del ASN %s firmado por %s es: %s", asnR2,
    asn2_owner.address, sig);
45
46 // El ISP asn1_owner envía al IANA la firma y el nonce.
47 // 2) El IANA vincula el ASN a la address del ISP utilizando la solicitud firmada y el
    nonce para mayor seguridad.
48 console.log("El IANA utilizando los datos del ASN, el address solicitante, el nonce y
    la firma, procesa la solicitud...");
49 const addASNTransaction = await iana.connect(iana_owner1).IANA_addASN(asnR2, asn2_owner
    .address, nonce, sig);
50 await addASNTransaction.wait();
51
52 const asnR2tmp = await iana.connect(iana_owner1).IANA_getASNOwner(asnR2);
53 console.log("IANA vinculo correctamente la address %s con el ASN %s", asnR2tmp, asnR2)
    ;
54
55 // El SA firman la solicitud de prefijo IP. Firma un mensaje que contiene el prefijo,
    el ASN y el owner, y lo envían al IANA.
56 const nonce2 = ethers.BigNumber.from(crypto.randomBytes(32)).toString();
57 const msgHashpfx = await iana.connect(asn2_owner).IANA_getPrefixSignatureMessage(
    prefixR2, prefixR2length, asnR2, asn2_owner.address, nonce2);
58 const sigpx = await asn2_owner.signMessage(ethers.utils.arrayify(msgHashpfx));
59
60 // El IANA, o el dueño del prefijo, recibe el mensaje firmado y lo utiliza para
    invocar a la función prefix_addPrefix mediante la cual
61 // queda establecida en la blockchain el vínculo entre el prefijo IPv6 y el ASN.
62 const addPrefixTransaction = await iana.connect(iana_owner1).prefix_addPrefix(prefixR2
    , prefixR2length, asnR2, nonce2, sigpx);
63 await addPrefixTransaction.wait();
64
65 const pfxcheck = await iana.connect(iana_owner1).IANA_prefixCheck(prefixR2,
    prefixR2length, asnR2);
66 console.log("Se vincularon los datos del prefijo IP %s / %s al ASN %s : %s", prefixR2.
    toHexString(), prefixR2length, asnR2, pfxcheck );
67
68 console.log("Enlazamos el ASN 64502 a sus vecinos, 64501 y 64503");
69 await iana.connect(asn2_owner).link_addLink(asnR2, asnR1);
70 await iana.connect(asn2_owner).link_addLink(asnR2, asnR3);
71

```



```

72 }
73
74 // We recommend this pattern to be able to use async/await everywhere
75 // and properly handle errors.
76 main().catch((error) => {
77   console.error(error);
78   process.exitCode = 1;
79 });

```

### Código 14.6 start\_r3.js

```

1 // We require the Hardhat Runtime Environment explicitly here. This is optional
2 // but useful for running the script in a standalone fashion through 'node <script>'.
3 //
4 // You can also run a script with 'npx hardhat run <script>'. If you do that, Hardhat
5 // will compile your contracts, add the Hardhat Runtime Environment's members to the
6 // global scope, and execute the script.
7 const hre = require("hardhat");
8 const crypto = require('crypto');
9 const { BigNumber } = require("ethers");
10
11
12 async function main() {
13
14   const iana = await hre.ethers.getContractFactory("IANA");
15   const iana = await iana.attach('0xEeAaaCdD3d1F77AB33F5fB5448f5843d04E9A496');
16
17   const asnR1 = 64501;
18   const asnR2 = 64502;
19   const asnR3 = 64503;
20   const asnR4 = 64504;
21
22   const prefixR1 = BigNumber.from("0x20010db8010000000000000000000000"); // 2001:db8
23     :100::/40
24   const prefixR1length = 40;
25   const prefixR2 = BigNumber.from("0x20010db8020000000000000000000000"); // 2001:db8
26     :200::/40
27   const prefixR2length = 40;
28   const prefixR3 = BigNumber.from("0x20010db8030000000000000000000000"); // 2001:db8
29     :300::/40
30   const prefixR3length = 40;
31
32   const [iana_owner1, asn1_owner, asn2_owner, asn3_owner, asn4_owner] = await ethers.
33     getSigners();
34
35   // 1) El ISP firma la solicitud del ASN, firmando un mensaje que contiene el ASN, el
36   // owner y un nonce aleatorio.
37   // creo el nonce
38   const nonce = ethers.BigNumber.from(crypto.randomBytes(32)).toString();
39   const selector = ethers.utils.keccak256(ethers.utils.toUtf8Bytes("IANA_addASN(uint32,
40     address,uint256,bytes)")).slice(0, 10);
41   console.log("El ASN solicitado por R3 es %s", asnR3);
42   console.log("El address que pretende ser dueño del ASN3 es %s", asn3_owner.address);
43   // Obtengo el MessageHash

```

```

38 const msgHash = await iana.connect(asn3_owner).IANA_getSignatureMessage(asnR3,
    asn3_owner.address, nonce, selector);
39 // El agregado del prefijo lo hace automáticamente la función signMessage de ethers
40 // const prefixedMessage = ethers.utils.solidityKeccak256(["string", "bytes32"], ["\
    x19Ethereum Signed Message:\n32", msgHash]);
41
42 // Convert string to bytes and sign
43 const sig = await asn3_owner.signMessage(ethers.utils.arrayify(msgHash));
44 console.log("La firma de la solicitud del ASN %s firmado por %s es: %s", asnR3,
    asn3_owner.address, sig);
45
46 // El ISP asn1_owner envía al IANA la firma y el nonce.
47 // 2) El IANA vincula el ASN a la address del ISP utilizando la solicitud firmada y el
    nonce para mayor seguridad.
48 console.log("El IANA utilizando los datos del ASN, el address solicitante, el nonce y
    la firma, procesa la solicitud...");
49 const addASNTransaction = await iana.connect(iana_owner1).IANA_addASN(asnR3, asn3_owner
    .address, nonce, sig);
50 await addASNTransaction.wait();
51
52 const asnR3tmp = await iana.connect(iana_owner1).IANA_getASNOwner(asnR3);
53 console.log("IANA vincula correctamente la address %s con el ASN %s", asnR3tmp, asnR3)
    ;
54
55 // El SA firman la solicitud de prefijo IP. Firma un mensaje que contiene el prefijo,
    el ASN y el owner, y lo envían al IANA.
56 const nonce2 = ethers.BigNumber.from(crypto.randomBytes(32)).toString();
57 const msgHashpfx = await iana.connect(asn3_owner).IANA_getPrefixSignatureMessage(
    prefixR3, prefixR3length, asnR3, asn3_owner.address, nonce2);
58 const sigpx = await asn3_owner.signMessage(ethers.utils.arrayify(msgHashpfx));
59
60 // El IANA, o el dueño del prefijo, recibe el mensaje firmado y lo utiliza para
    invocar a la función prefix_addPrefix mediante la cual
61 // queda establecida en la blockchain el vínculo entre el prefijo IPv6 y el ASN.
62 const addPrefixTransaction = await iana.connect(iana_owner1).prefix_addPrefix(prefixR3
    , prefixR3length, asnR3, nonce2, sigpx);
63 await addPrefixTransaction.wait();
64
65 const pfxcheck = await iana.connect(iana_owner1).IANA_prefixCheck(prefixR3,
    prefixR3length, asnR3);
66 console.log("Se vincularon los datos del prefijo IP %s / %s al ASN %s : %s", prefixR3.
    toHexString(), prefixR3length, asnR3, pfxcheck );
67
68 console.log("Enlazamos el ASN 64503 a su vecino 64502");
69 await iana.connect(asn3_owner).link_addLink(asnR3, asnR2);
70
71 }
72
73 // We recommend this pattern to be able to use async/await everywhere
74 // and properly handle errors.
75 main().catch((error) => {
76     console.error(error);
77     process.exitCode = 1;
78 });

```

## Código 14.7 start\_r4.js

```
1 // We require the Hardhat Runtime Environment explicitly here. This is optional
2 // but useful for running the script in a standalone fashion through 'node <script>'.
3 //
4 // You can also run a script with 'npx hardhat run <script>'. If you do that, Hardhat
5 // will compile your contracts, add the Hardhat Runtime Environment's members to the
6 // global scope, and execute the script.
7 const hre = require("hardhat");
8 const crypto = require('crypto');
9 const { BigNumber } = require("ethers");
10
11
12 async function main() {
13
14   const lana = await hre.ethers.getContractFactory("IANA");
15   const iana = await lana.attach('0xEeAaCdD3d1F77AB33F5fB5448f5843d04E9A496');
16
17   const asnR1 = 64501;
18   const asnR2 = 64502;
19   const asnR3 = 64503;
20   const asnR4 = 64504;
21
22   const prefixR1 = BigNumber.from("0x20010db8010000000000000000000000"); // 2001:db8
23     :100::/40
24   const prefixR1length = 40;
25   const prefixR2 = BigNumber.from("0x20010db8020000000000000000000000"); // 2001:db8
26     :200::/40
27   const prefixR2length = 40;
28   const prefixR3 = BigNumber.from("0x20010db8030000000000000000000000"); // 2001:db8
29     :300::/40
30   const prefixR3length = 40;
31
32   const [iana_owner1, asn1_owner, asn2_owner, asn3_owner, asn4_owner] = await ethers.
33     getSigners();
34
35   // 1) El ISP firma la solicitud del ASN, firmando un mensaje que contiene el ASN, el
36   // owner y un nonce aleatorio.
37   // creo el nonce
38   const nonce = ethers.BigNumber.from(crypto.randomBytes(32)).toString();
39   const selector = ethers.utils.keccak256(ethers.utils.toUtf8Bytes("IANA_addASN(uint32,
40     address,uint256,bytes)")).slice(0, 10);
41   console.log("El ASN solicitado por R4 es %s", asnR4);
42   console.log("El address que pretende ser dueño del ASN4 es %s", asn4_owner.address);
43   // Obtengo el MessageHash
44   const msgHash = await iana.connect(asn4_owner).IANA_getSignatureMessage(asnR4,
45     asn4_owner.address, nonce, selector);
46   // El agregado del prefijo lo hace automáticamente la función signMessage de ethers
47   // const prefixedMessage = ethers.utils.solidityKeccak256(["string", "bytes32"], ["\
48     x19Ethereum Signed Message:\n32", msgHash]);
49
50   // Convert string to bytes and sign
51   const sig = await asn4_owner.signMessage(ethers.utils.arrayify(msgHash));
52   console.log("La firma de la solicitud del ASN %s firmado por %s es: %s", asnR4,
53     asn4_owner.address, sig);
54
55   // El ISP asn1_owner envía al IANA la firma y el nonce.
56   // 2) El IANA vincula el ASN a la address del ISP utilizando la solicitud firmada y el
57   // nonce para mayor seguridad.
```

```

48 console.log("El IANA utilizando los datos del ASN, el address solicitante, el nonce y
    la firma, procesa la solicitud...");
49 const addASNTransaction = await iana.connect(iana_owner1).IANA_addASN(asnR4,asn4_owner
    .address,nonce,sig);
50 await addASNTransaction.wait();
51
52 const asnR4tmp = await iana.connect(iana_owner1).IANA_getASNOwner(asnR4);
53 console.log("IANA vinculo correctamente la address %s con el ASN %s",asnR4tmp,asnR4)
    ;
54
55 // No tiene prefijos propios..
56
57 console.log("Enlazamos el ASN 64504 a su vecino 64501");
58 await iana.connect(asn4_owner).link_addLink(asnR4,asnR1);
59
60 }
61
62 // We recommend this pattern to be able to use async/await everywhere
63 // and properly handle errors.
64 main().catch((error) => {
65   console.error(error);
66   process.exitCode = 1;
67 });

```

## 14.4 Scripts para testing del contrato inteligente

**Código 14.8** iana\_test.js

```

1 const { expect } = require("chai");
2 const { ethers } = require("hardhat");
3 const crypto = require('crypto');
4 const { BigNumber } = require("ethers");
5
6 describe("IANA contract test cases", function () {
7   let IANA, iana, signers, asnR1, asnR2, asnR3, asnR4, iana_owner1, asn1_owner,
    asn2_owner, asn3_owner, asn4_owner;
8
9   before(async function () {
10     IANA = await ethers.getContractFactory("IANA");
11     iana = await IANA.deploy();
12     await iana.deployed();
13     signers = await ethers.getSigners();
14     asnR1 = 64501;
15     asnR2 = 64502;
16     asnR3 = 64503;
17     asnR4 = 64504;
18
19   });
20
21   it("Should assign an autonomous system to an address, and then remove the autonomous
    system from it.", async function () {
22
23     const [iana_owner1, asn1_owner, asn2_owner, asn3_owner, asn4_owner] = signers;

```

```

24
25 const nonce = ethers.BigNumber.from(crypto.randomBytes(32)).toString();
26 const selector = ethers.utils.keccak256(ethers.utils.toUtf8Bytes("IANA_addASN(uint32
, address, uint256, bytes)")).slice(0, 10);
27
28 // Get message hash
29 const msgHash = await iana.connect(asn1_owner).IANA_getSignatureMessage(asnR1,
asn1_owner.address, nonce, selector);
30
31 // Sign message
32 const sig = await asn1_owner.signMessage(ethers.utils.arrayify(msgHash));
33
34 // Process the request with the signature and nonce
35 await iana.connect(iana_owner1).IANA_addASN(asnR1, asn1_owner.address, nonce, sig);
36
37 // Check if the ASN is assigned to the address correctly
38 const asnR1Owner = await iana.connect(iana_owner1).IANA_getASNOwner(asnR1);
39 expect(asnR1Owner).to.equal(asn1_owner.address);
40
41 // Prepare for ASN removal
42 const nonceRemove = ethers.BigNumber.from(crypto.randomBytes(32)).toString();
43 const selectorRemove = ethers.utils.keccak256(ethers.utils.toUtf8Bytes("
IANA_removeASN(uint32, address, uint256, bytes)")).slice(0, 10);
44 const msgHashRemove = await iana.connect(asn1_owner).IANA_getSignatureMessage(asnR1,
asn1_owner.address, nonceRemove, selectorRemove);
45 const sigRemove = await asn1_owner.signMessage(ethers.utils.arrayify(msgHashRemove))
;
46
47 // Remove the ASN
48 await iana.connect(iana_owner1).IANA_removeASN(asnR1, asn1_owner.address,
nonceRemove, sigRemove);
49
50 // Check if the ASN is removed correctly
51 const asnR1Owner2 = await iana.connect(iana_owner1).IANA_getASNOwner(asnR1);
52 expect(asnR1Owner2).to.equal(ethers.constants.AddressZero);
53 });
54
55 it("Should add a prefix to an ASN", async function () {
56
57 const [iana_owner1, asn1_owner, asn2_owner, asn3_owner, asn4_owner] = signers;
58
59 const nonce = ethers.BigNumber.from(crypto.randomBytes(32)).toString();
60 const selectorAdd = ethers.utils.keccak256(ethers.utils.toUtf8Bytes("IANA_addASN(
uint32, address, uint256, bytes)")).slice(0, 10);
61
62 // Get message hash
63 const msgHash = await iana.connect(asn1_owner).IANA_getSignatureMessage(asnR1,
asn1_owner.address, nonce, selectorAdd);
64
65 // Sign message
66 const sig = await asn1_owner.signMessage(ethers.utils.arrayify(msgHash));
67
68 // Process the request with the signature and nonce
69 await iana.connect(iana_owner1).IANA_addASN(asnR1, asn1_owner.address, nonce, sig);
70
71 // Check if the ASN is assigned to the address correctly
72 const asnR1Owner = await iana.connect(iana_owner1).IANA_getASNOwner(asnR1);
73 expect(asnR1Owner).to.equal(asn1_owner.address);
74

```

```

75  const prefixR1 = BigNumber.from("0x20010db80100000000000000000000"); // 2001:db8
76  :100::/40
77  const prefixR1length = 40;
78  const nonce2 = ethers.BigNumber.from(crypto.randomBytes(32)).toString();
79  const msgHashpfx = await iana.connect(asn1_owner).IANA_getPrefixSignatureMessage(
80  prefixR1, prefixR1length, asnR1, asn1_owner.address, nonce2);
81  const sigpx = await asn1_owner.signMessage(ethers.utils.arrayify(msgHashpfx));
82
83  // Add the prefix to the ASN
84  await iana.connect(iana_owner1).prefix_addPrefix(prefixR1, prefixR1length, asnR1,
85  nonce2, sigpx);
86
87  // Check if the prefix is added correctly
88  const pfxcheck = await iana.connect(iana_owner1).IANA_prefixCheck(prefixR1,
89  prefixR1length, asnR1);
90  expect(pfxcheck).to.be.true;
91  });
92
93  it("Should remove a prefix by the AS owner", async function () {
94
95  // Add ASN for asn2_owner
96  const [iana_owner1, asn1_owner, asn2_owner, asn3_owner, asn4_owner] = signers;
97
98  const nonce = ethers.BigNumber.from(crypto.randomBytes(32)).toString();
99  const selectorAdd = ethers.utils.keccak256(ethers.utils.toUtf8Bytes("IANA_addASN(
100  uint32,address,uint256,bytes)")).slice(0, 10);
101
102  // Get message hash
103  const msgHash = await iana.connect(asn2_owner).IANA_getSignatureMessage(asnR2,
104  asn2_owner.address, nonce, selectorAdd);
105
106  // Sign message
107  const sig = await asn2_owner.signMessage(ethers.utils.arrayify(msgHash));
108
109  // Process the request with the signature and nonce
110  await iana.connect(iana_owner1).IANA_addASN(asnR2, asn2_owner.address, nonce, sig);
111
112  // Check if the ASN is assigned to the address correctly
113  const asnR2Owner = await iana.connect(iana_owner1).IANA_getASNOwner(asnR2);
114  expect(asnR2Owner).to.equal(asn2_owner.address);
115
116  // Add prefix to ASN
117  const prefixR2 = BigNumber.from("0x20010db80200000000000000000000"); // 2001:db8
118  :200::/40
119  const prefixR2length = 40;
120  const nonce2 = ethers.BigNumber.from(crypto.randomBytes(32)).toString();
121  const msgHashpfx = await iana.connect(asn2_owner).IANA_getPrefixSignatureMessage(
122  prefixR2, prefixR2length, asnR2, asn2_owner.address, nonce2);
123  const sigpx = await asn2_owner.signMessage(ethers.utils.arrayify(msgHashpfx));
124
125  // Add the prefix to the ASN
126  await iana.connect(iana_owner1).prefix_addPrefix(prefixR2, prefixR2length, asnR2,
127  nonce2, sigpx);
128
129  // Check if the prefix is added correctly
130  const pfxcheck = await iana.connect(iana_owner1).IANA_prefixCheck(prefixR2,
131  prefixR2length, asnR2);
132  expect(pfxcheck).to.be.true;

```

```

124
125 // Remove the prefix
126 await iana.connect(asn2_owner).prefix_removePrefix(prefixR2, prefixR2length);
127
128 // Check that the prefix has been removed
129 const removedPrefix = await iana.connect(iana_owner1).IANA_prefixCheck(prefixR2,
130 prefixR2length, asnR2);
131 expect(removedPrefix).to.be.false;
132 });
133
134
135 it("Should remove a prefix by the IANA after the leaseTime", async function () {
136
137 // Add ASN for asn2_owner
138 const [iana_owner1, asn1_owner, asn2_owner, asn3_owner, asn4_owner] = signers;
139
140 const nonce = ethers.BigNumber.from(crypto.randomBytes(32)).toString();
141 const selectorAdd = ethers.utils.keccak256(ethers.utils.toUtf8Bytes("IANA_addASN(
142 uint32, address, uint256, bytes)")).slice(0, 10);
143
144 // Get message hash
145 const msgHash = await iana.connect(asn2_owner).IANA_getSignatureMessage(asnR2,
146 asn2_owner.address, nonce, selectorAdd);
147
148 // Sign message
149 const sig = await asn2_owner.signMessage(ethers.utils.arrayify(msgHash));
150
151 // Process the request with the signature and nonce
152 await iana.connect(iana_owner1).IANA_addASN(asnR2, asn2_owner.address, nonce, sig);
153
154 // Check if the ASN is assigned to the address correctly
155 const asnR2Owner = await iana.connect(iana_owner1).IANA_getASNOwner(asnR2);
156 expect(asnR2Owner).to.equal(asn2_owner.address);
157
158 // Add prefix to ASN
159 const prefixR2 = BigNumber.from("0x20010db8020000000000000000000000"); // 2001:db8
160 :200::/40
161
162 const prefixR2length = 40;
163 const nonce2 = ethers.BigNumber.from(crypto.randomBytes(32)).toString();
164 const msgHashpfx = await iana.connect(asn2_owner).IANA_getPrefixSignatureMessage(
165 prefixR2, prefixR2length, asnR2, asn2_owner.address, nonce2);
166 const sigpx = await asn2_owner.signMessage(ethers.utils.arrayify(msgHashpfx));
167
168 // Add the prefix to the ASN
169 await iana.connect(iana_owner1).prefix_addPrefix(prefixR2, prefixR2length, asnR2,
170 nonce2, sigpx);
171
172 // Check if the prefix is added correctly
173 const pfxcheck = await iana.connect(iana_owner1).IANA_prefixCheck(prefixR2,
174 prefixR2length, asnR2);
175 expect(pfxcheck).to.be.true;
176
177 // Manipulate the time
178 const secondsToIncrease = 31536000; // Increase time by 1 year
179 await network.provider.send("evm_increaseTime", [secondsToIncrease]);
180 await network.provider.send("evm_mine"); // Mine a new block to update the timestamp
181
182 // Remove the prefix

```

```

176     await iana.connect(iana_owner1).prefix_removePrefix(prefixR2, prefixR2length);
177
178     // Check that the prefix has been removed
179     const removedPrefix = await iana.connect(iana_owner1).IANA_prefixCheck(prefixR2,
180     prefixR2length, asnR2);
181     expect(removedPrefix).to.be.false;
182
183   });
184
185   it("Should not allow adding a subprefix to another ASN", async function () {
186
187     // Add ASN for asn2_owner
188     const [iana_owner1, asn1_owner, asn2_owner, asn3_owner, asn4_owner] = signers;
189
190     const nonce = ethers.BigNumber.from(crypto.randomBytes(32)).toString();
191     const selectorAdd = ethers.utils.keccak256(ethers.utils.toUtf8Bytes("IANA_addASN(
192     uint32, address, uint256, bytes)")).slice(0, 10);
193
194     // Get message hash
195     const msgHash = await iana.connect(asn2_owner).IANA_getSignatureMessage(asnR2,
196     asn2_owner.address, nonce, selectorAdd);
197
198     // Sign message
199     const sig = await asn2_owner.signMessage(ethers.utils.arrayify(msgHash));
200
201     // Process the request with the signature and nonce
202     await iana.connect(iana_owner1).IANA_addASN(asnR2, asn2_owner.address, nonce, sig);
203
204     // Check if the ASN is assigned to the address correctly
205     const asnR2Owner = await iana.connect(iana_owner1).IANA_getASNOwner(asnR2);
206     expect(asnR2Owner).to.equal(asn2_owner.address);
207
208     // Add ASN for asn3_owner
209     const nonce_asn3 = ethers.BigNumber.from(crypto.randomBytes(32)).toString();
210
211     // Get message hash
212     const msgHash_asn3 = await iana.connect(asn3_owner).IANA_getSignatureMessage(asnR3,
213     asn3_owner.address, nonce_asn3, selectorAdd);
214
215     // Sign message
216     const sig_asn3 = await asn3_owner.signMessage(ethers.utils.arrayify(msgHash_asn3));
217
218     // Process the request with the signature and nonce
219     await iana.connect(iana_owner1).IANA_addASN(asnR3, asn3_owner.address, nonce_asn3,
220     sig_asn3);
221
222     // Check if the ASN is assigned to the address correctly
223     const asnR3Owner = await iana.connect(iana_owner1).IANA_getASNOwner(asnR3);
224     expect(asnR3Owner).to.equal(asn3_owner.address);
225
226     // Add prefix to ASN
227     const prefixR2 = BigNumber.from("0x20010db8020000000000000000000000"); // 2001:db8
228     :200::/40
229     const prefixR2length = 40;
230     const nonce2 = ethers.BigNumber.from(crypto.randomBytes(32)).toString();
231     const msgHashpfx = await iana.connect(asn2_owner).IANA_getPrefixSignatureMessage(
232     prefixR2, prefixR2length, asnR2, asn2_owner.address, nonce2);
233     const sigpx = await asn2_owner.signMessage(ethers.utils.arrayify(msgHashpfx));

```



```

228
229 // Add the prefix to the ASN
230 await iana.connect(iana_owner1).prefix_addPrefix(prefixR2, prefixR2length, asnR2,
    nonce2, sigpx);
231
232 // Check if the prefix is added correctly
233 const pfxcheck = await iana.connect(iana_owner1).IANA_prefixCheck(prefixR2,
    prefixR2length, asnR2);
234 expect(pfxcheck).to.be.true;
235
236 // Now, try to add the subprefix to asnR3
237 const subPrefixR2 = BigNumber.from("0x20010db8020100000000000000000000"); // 2001:
    db8:201::/48
238 const subPrefixR2length = 48;
239 const nonceSubPrefix = ethers.BigNumber.from(crypto.randomBytes(32)).toString();
240 const msgHashSubPrefix = await iana.connect(asn3_owner).
    IANA_getPrefixSignatureMessage(subPrefixR2, subPrefixR2length, asnR3, asn3_owner.
    address, nonceSubPrefix);
241 const sigSubPrefix = await asn3_owner.signMessage(ethers.utils.arrayify(
    msgHashSubPrefix));
242
243 // Check if the subprefix is not yet assigned
244 const subPrefixCheck = await iana.connect(asn3_owner).IANA_prefixCheck(subPrefixR2,
    subPrefixR2length, asnR3);
245 expect(subPrefixCheck).to.be.false;
246
247 // Attempt to add the subprefix and expect the transaction to be reverted
248 await expect(iana.connect(iana_owner1).prefix_addPrefix(subPrefixR2,
    subPrefixR2length, asnR3, nonceSubPrefix, sigSubPrefix)).to.be.revertedWith('The
    address calling us NOT owns the AS that owns the parent prefix');
249
250 // Check again that the subprefix is not assigned to asnR3
251 const subPrefixCheckAfter = await iana.connect(asn3_owner).IANA_prefixCheck(
    subPrefixR2, subPrefixR2length, asnR3);
252 expect(subPrefixCheckAfter).to.be.false;
253
254 });
255
256 });

```

## 14.5 Script Python para validación de paquetes UPDATE de BGP

```

from netfilterqueue import NetfilterQueue
import socket
from scapy.all import *
import os
import time
import pyshark
import threading
import queue
import random
import netaddr

```

```

from web3 import Web3, HTTPProvider
from eth_account import Account

#Author Marcelo de Abranches (made0661@colorado.edu)

#Policy agent - currently it configures a higher local preference for routes
that were completely validated
def policy_agent(valid_update_dict):
    print('Installing local preference for checked path')
    myPeer = valid_update_dict['peer']
    prefix = valid_update_dict['prefix']
    length = valid_update_dict['length']
    create_prefix_list, apply_policy1, apply_policy2, reload_bgp =
        generate_local_pref_command(myAS, myPeer, prefix, length)
    os.system(create_prefix_list)
    os.system(apply_policy1)
    os.system(apply_policy2)
    os.system(reload_bgp)
    return 0

#Generates the local pref command to be sent to Quagga
def generate_local_pref_command(myAS, myPeer, prefix, length):
    #probably we will also need to cotrol the sequence number of the policies
    prefix_list_name = myPeer.replace('.', '-')
    local_pref_name = 'LP-' + prefix_list_name + '-CKD-150'
    ip_list_seq = random.randint(1,100)
    create_prefix_list = 'vtysh -c "config terminal" -c "ipv6 prefix-list ' +
        prefix_list_name + ' seq ' + str(ip_list_seq) + ' permit ' + prefix + '
        /' + length + '"
    apply_policy1 = 'vtysh -c "config terminal" -c "router bgp ' + myAS + '" -c
        "address-family ipv6 unicast" -c "neighbor ' + myPeer + ' route-map ' +
        local_pref_name + ' in" -c "route-map ' + local_pref_name + ' permit
        10" -c "match ipv6 address prefix-list ' + prefix_list_name + '" -c "
        set local-preference 150"'
    #policy 2 is needed to avoid updates being rejected if they do not match
    conditions at the route-map
    apply_policy2 = 'vtysh -c "config terminal" -c "access-list 1 permit any" -
        c "router bgp ' + myAS + '" -c "address-family ipv6 unicast" -c "route-
        map ' + local_pref_name + ' permit 20" -c "match ipv6 address 1"'
    reload_bgp = 'vtysh -c "clear bgp ' + myPeer + ' soft"'
    # ip_list_seq += 1
    return create_prefix_list, apply_policy1, apply_policy2, reload_bgp

#Code for what to do after receiving a invalid update should go here
def handle_invalid_update(invalid_update_dict):
    print("Invalid update received")
    #print(invalid_update_dict)

```

```

def handle_unknown_update(update_dict):
    print("Validity of update is Unknown")
    #print(update_dict)

#Checks if an AS is using the IANA smart contracts
#It is used to determine if we consider update messages completely or partially
    validated
def check_ASMembership(AS):
    member=IANA.functions.ASNList(int(AS)).call()
    if member == '0x00000000000000000000000000000000':
        return False
    else:
        return True

#Checks AS announced prefixes in the IANA smart contract
def check_as_prefix(_prefix, _length, _AS):
    print("Checking prefix ...")
    prefix = netaddr.IPAddress(_prefix)
    length = _length
    AS = _AS
    ownPrefix = IANA.functions.IANA_prefixCheck(int(prefix), int(length), int(
        AS)).call()
    return ownPrefix

#Checks AS announced paths in the IANA smart contract
def check_path(path_array):
    validPath = True
    print('Cheking path array: ' + str(path_array))
    index = 0
    while(index < len(path_array) - 1):
        hasLink = IANA.functions.link_validateLink(int(path_array[index]), int(
            path_array[index + 1])).call()
        validPath = validPath and hasLink
        index += 1
    return validPath

#Creates a dict with structured data from BGP messages
def create_pkt_dict(pkt):
    fname='file.pcap'
    wrpcap(fname, pkt, append=False)
    cap = pyshark.FileCapture(fname, decode_as={'tcp.port==179':'bgp'})
    bgp_pkt_dict = {}
    bgp_pkt_dict['pkt'] = 0
    for cur_pkt in cap:
        for layer in cur_pkt.layers:
            if layer.layer_name == "bgp":
                #save_packet
                if bgp_pkt_dict['pkt'] == 0:
                    bgp_pkt_dict['pkt'] = cur_pkt
                    bgp_pkt_dict['bgp_layers']=[]

```

```

        bgp_pkt_dict['bgp_layers'].append(layer)
    else:
        bgp_pkt_dict['bgp_layers'].append(layer)
cap.close()
if bgp_pkt_dict['pkt'] == 0:
    return 0
else:
    return bgp_pkt_dict

#This thread is awoken whenever a packet arrives at the BGP port of the host
    captured
#by NF_QUEUE
#We are only saving and processing BGP UPDATE messages
def process_bgp_pkt():
    while True:
        process_packet_event.wait()
        pkt_dict=bgp_pkt_q.get()
        bgp_layers=pkt_dict['bgp_layers']
        bgp_update = False
        bgp_update_list=[] #save all bgp update info from an update msg
        bgp_layer_dict={} #save all info from each update
        bgp_update_messages={} #save info in a dict with peer as key
        pkt=pkt_dict['pkt']
        for bgp_layer in bgp_layers:
            bgp_layer_dict={} #save all info from each update

            if bgp_layer.get('type') == '2': #only save updates
                bgp_update = True

                for field in bgp_layer.field_names:
                    bgp_layer_dict[field] = bgp_layer.get(field)

                bgp_update_list.append(bgp_layer_dict)

            if bgp_update == True:
                bgp_update_messages[pkt.ipv6.addr] = bgp_update_list
                validate_update_message(bgp_update_messages)

#Invokes the functions to validate the BGP UPDATE messages
#It also applies some logic to determine if a the update was valid or invalid
#and also if it was possible to do a complete validation (All ASs were members
    of IANA contrac)
#or a partial validation (not All ASs was members of IANA contract)
def validate_update_message(bgp_update_messages):
    is_as_prefix_valid = False
    is_path_array_valid = False
    state = "Invalid"
    peer = list(bgp_update_messages.keys())[0]
    print('Received update message from peer ' + peer)
    for bgp_update_msg in bgp_update_messages[peer]:
        try:

```

```

# next_hop = bgp_update_msg['update_path_attribute_next_hop']
next_hop = bgp_update_msg['
    update_path_attribute_mp_reach_nlri_next_hop']
# prefix = bgp_update_msg['nlri_prefix']
prefix = bgp_update_msg['mp_reach_nlri_ipv6_prefix']
length = bgp_update_msg['prefix_length']
path_array = bgp_update_msg['update_path_attribute_as_path_segment'
    ].split(':')[1:][0].split(' ')[1:]
AS = path_array[len(path_array) - 1]
update_dict = {}
update_dict['peer'] = peer
update_dict['AS'] = AS
update_dict['next_hop'] = next_hop
update_dict['prefix'] = prefix
update_dict['length'] = length
print(update_dict)
member=check_ASMembership(AS)
print("#####")
if member == False:
    print("This update message was partially verified")
    verified = False
    state = "Unknown"
    handle_unknown_update(update_dict)
else:
    print("This update message was verified at IANA contract")
    verified = True
    is_as_prefix_valid=check_as_prefix(prefix, length, AS)
    if is_as_prefix_valid == False:
        state = "Invalid"
        handle_invalid_update(update_dict)
    else:
        if (len(path_array) > 1):
            is_path_array_valid=check_path(path_array)
            valid = (is_as_prefix_valid and is_path_array_valid)
        if (valid == False):
            if is_as_prefix_valid == False:
                print("Prefix was not verified")
                handle_invalid_update(update_dict)
                state = "Invalid"
            else:
                print("Prefix was verified")
            if is_path_array_valid == False:
                print("Path was not verified")
                handle_unknown_update(update_dict)
            else:
                print("Path was verified")
        else:
            state = "Valid"
            cur_hash = hash(update_dict['peer'] + update_dict['AS']
                + update_dict['prefix'] + update_dict['length'])
            #This will avoid unnecessary policy_agent calls

```

```

        if cur_hash not in policy_hash_list:
            print("Received valid update, calling policy agent"
                  )
            policy_hash_list.append(cur_hash)
            policy_agent(update_dict)
        else:
            print("Policy exists for this update message. Will
                  not call policy agent")

    except:
        pass

#return state

#Gets the packet and sends it to bgpd. Also wakes the thread to process the
packet
def get_packet(pkt):
    #don't wait to forward the pkt
    print('pkt arrived')
    pkt.accept()
    #pkt.drop()
    spkt=IPv6(pkt.get_payload())
    pkt_dict=create_pkt_dict(spkt)
    if pkt_dict != 0:
        bgp_pkt_q.put(pkt_dict)
        process_packet_event.set()
        process_packet_event.clear()

#Policy agent configs (put as env variable)
myAS='64501'
policy_hash_list=[] #This is used to know if a policy has already been applied
ip_list_seq = 1
#Ethereum stuff
infura_provider = HTTPProvider('https://sepolia.infura.io/v3/31
a6da5e9d80423b8c425057cd23b473')
web3=Web3(infura_provider)
#IANA contract with prefix/AS ownership and valid AS links
IANA_addr = Web3.to_checksum_address('0
xEeAaaCdD3d1F77AB33F5fB5448f5843d04E9A496')
IANA_abi= [{"inputs": [], "stateMutability": "nonpayable", "type": "constructor"}, {"
inputs": [{"internalType": "uint32", "name": "", "type": "uint32"}], "name": "
ASNList", "outputs": [{"internalType": "address", "name": "", "type": "address
"}], "stateMutability": "view", "type": "function"}, {"inputs": [{"internalType
": "uint32", "name": "ASN", "type": "uint32"}, {"internalType": "address", "name": "
ASNOwner", "type": "address"}, {"internalType": "uint256", "name": "nonce", "type
": "uint256"}, {"internalType": "bytes", "name": "_signature", "type": "bytes"}], "
name": "IANA_addASN", "outputs": [], "stateMutability": "nonpayable", "type": "
function"}, {"inputs": [{"internalType": "address", "name": "owner", "type": "
address"}], "name": "IANA_addOwner", "outputs": [], "stateMutability": "
nonpayable", "type": "function"}, {"inputs": [{"internalType": "uint32", "name": "
ASN", "type": "uint32"}], "name": "IANA_getASNOwner", "outputs": [{"internalType
": "address", "name": "", "type": "address"}], "stateMutability": "view", "type": "
function"}, {"inputs": [{"internalType": "uint128", "name": "ip", "type": "uint128

```

```

    }, {"internalType": "uint8", "name": "mask", "type": "uint8"}, {"internalType": "uint32", "name": "ASN", "type": "uint32"}, {"internalType": "address", "name": "ASNOwner", "type": "address"}, {"internalType": "uint256", "name": "nonce", "type": "uint256"}], "name": "IANA_getPrefixSignatureMessage", "outputs": [{"internalType": "bytes32", "name": "", "type": "bytes32"}], "stateMutability": "pure", "type": "function"}, {"inputs": [{"internalType": "uint32", "name": "ASN", "type": "uint32"}, {"internalType": "address", "name": "ASNOwner", "type": "address"}, {"internalType": "uint256", "name": "_nonce", "type": "uint256"}, {"internalType": "bytes4", "name": "_functionName", "type": "bytes4"}], "name": "IANA_getSignatureMessage", "outputs": [{"internalType": "bytes32", "name": "", "type": "bytes32"}], "stateMutability": "pure", "type": "function"}, {"inputs": [{"internalType": "uint128", "name": "ip", "type": "uint128"}, {"internalType": "uint8", "name": "mask", "type": "uint8"}, {"internalType": "uint256", "name": "asNumber", "type": "uint256"}], "name": "IANA_prefixCheck", "outputs": [{"internalType": "bool", "name": "", "type": "bool"}], "stateMutability": "view", "type": "function"}, {"inputs": [{"internalType": "uint32", "name": "ASN", "type": "uint32"}, {"internalType": "address", "name": "ASNOwner", "type": "address"}, {"internalType": "uint256", "name": "nonce", "type": "uint256"}, {"internalType": "bytes", "name": "_signature", "type": "bytes"}], "name": "IANA_removeASN", "outputs": [], "stateMutability": "nonpayable", "type": "function"}, {"inputs": [{"internalType": "address", "name": "owner", "type": "address"}], "name": "IANA_removeOwner", "outputs": [], "stateMutability": "nonpayable", "type": "function"}, {"inputs": [{"internalType": "uint32", "name": "myASN", "type": "uint32"}, {"internalType": "uint32", "name": "destinationASN", "type": "uint32"}], "name": "link_addLink", "outputs": [], "stateMutability": "nonpayable", "type": "function"}, {"inputs": [{"internalType": "uint32", "name": "myASN", "type": "uint32"}, {"internalType": "uint32", "name": "destinationASN", "type": "uint32"}], "name": "link_removeLink", "outputs": [], "stateMutability": "nonpayable", "type": "function"}, {"inputs": [{"internalType": "uint32", "name": "AS1", "type": "uint32"}, {"internalType": "uint32", "name": "AS2", "type": "uint32"}], "name": "link_validateLink", "outputs": [{"internalType": "bool", "name": "", "type": "bool"}], "stateMutability": "view", "type": "function"}, {"inputs": [{"internalType": "address", "name": "", "type": "address"}, {"internalType": "uint256", "name": "", "type": "uint256"}], "name": "nonceUsedMap", "outputs": [{"internalType": "bool", "name": "", "type": "bool"}], "stateMutability": "view", "type": "function"}, {"inputs": [{"internalType": "address", "name": "", "type": "address"}], "name": "ownerList", "outputs": [{"internalType": "bool", "name": "", "type": "bool"}], "stateMutability": "view", "type": "function"}, {"inputs": [{"internalType": "uint8", "name": "A", "type": "uint8"}, {"internalType": "uint8", "name": "B", "type": "uint8"}, {"internalType": "uint8", "name": "C", "type": "uint8"}, {"internalType": "uint8", "name": "D", "type": "uint8"}, {"internalType": "uint8", "name": "M", "type": "uint8"}, {"internalType": "uint256", "name": "_asNumber", "type": "uint256"}], "name": "prefixCheck", "outputs": [{"internalType": "bool", "name": "", "type": "bool"}], "stateMutability": "view", "type": "function"}, {"inputs": [{"internalType": "uint128", "name": "ip", "type": "uint128"}, {"internalType": "uint8", "name": "mask", "type": "uint8"}, {"internalType": "uint32", "name": "newOwnerAS", "type": "uint32"}, {"internalType": "uint256", "name": "nonce", "type": "uint256"}, {"internalType": "bytes", "name": "_signature", "type": "bytes"}], "name": "prefix_addPrefix", "outputs": [], "stateMutability": "nonpayable", "type": "function"}, {"inputs": [{"internalType": "uint128", "name": "ip1", "type": "uint128"}, {"internalType": "uint8", "name": "mask1", "type": "uint8"}]

```

```

uint8"}, {"internalType": "uint128", "name": "ip2", "type": "uint128"}, {"
internalType": "uint8", "name": "mask2", "type": "uint8"}], "name": "
prefix_comparePrefix", "outputs": [{"internalType": "enum IANA.
PrefixCompareResult", "name": "", "type": "uint8"}], "stateMutability": "pure", "
type": "function"}, {"inputs": [{"internalType": "uint256", "name": "
startingPrefixIndex", "type": "uint256"}, {"internalType": "uint128", "name": "ip
", "type": "uint128"}, {"internalType": "uint8", "name": "mask", "type": "uint8
"}], "name": "prefix_getContainingPrefix", "outputs": [{"internalType": "uint256
", "name": "", "type": "uint256"}], "stateMutability": "view", "type": "function
"}, {"inputs": [{"internalType": "uint256", "name": "startingPrefixIndex", "type
": "uint256"}, {"internalType": "uint128", "name": "ip", "type": "uint128"}, {"
internalType": "uint8", "name": "mask", "type": "uint8"}], "name": "
prefix_getContainingPrefixAndParent", "outputs": [{"internalType": "uint256", "
name": "", "type": "uint256"}, {"internalType": "uint256", "name": "", "type": "
uint256"}], "stateMutability": "view", "type": "function"}, {"inputs": [{"
internalType": "uint128", "name": "ip", "type": "uint128"}, {"internalType": "
uint8", "name": "mask", "type": "uint8"}], "name": "prefix_removePrefix", "outputs
": [], "stateMutability": "nonpayable", "type": "function"}, {"inputs": [{"
internalType": "uint128", "name": "ip", "type": "uint128"}, {"internalType": "
uint8", "name": "mask", "type": "uint8"}], "name": "prefix_renewalPrefix", "
outputs": [], "stateMutability": "payable", "type": "function"}, {"inputs": [{"
internalType": "uint256", "name": "", "type": "uint256"}], "name": "prefixes", "
outputs": [{"internalType": "uint128", "name": "ip", "type": "uint128"}, {"
internalType": "uint8", "name": "mask", "type": "uint8"}, {"internalType": "uint32
", "name": "owningAS", "type": "uint32"}, {"internalType": "uint256", "name": "
expiryOf", "type": "uint256"}], "stateMutability": "view", "type": "function"}, {"
inputs": [{"internalType": "uint256", "name": "_price", "type": "uint256"}], "name
": "set_renewPrefixPrice", "outputs": [], "stateMutability": "nonpayable", "type
": "function"}] '

```

```

IANA = web3.eth.contract(address=IANA_addr, abi=IANA_abi)
#IDS stuff
bgp_pkt_q = queue.Queue()
process_packet_event = threading.Event()
process_bgp_pkt_t = threading.Thread(target=process_bgp_pkt)
process_bgp_pkt_t.start()
os.system('ip6tables -I INPUT -p tcp --dport 179 -j NFQUEUE --queue-num 1')
os.system('ip6tables -I INPUT -p tcp --sport 179 -j NFQUEUE --queue-num 1')
# os.system('ip6tables -I INPUT -p tcp -j NFQUEUE --queue-num 1')
nfqueue = NetfilterQueue()
nfqueue.bind(1, get_packet)
s = socket.fromfd(nfqueue.get_fd(), socket.AF_UNIX, socket.SOCK_STREAM)
try:
    print('BGP IDS Started ...')
    nfqueue.run_socket(s)
except KeyboardInterrupt:
    print('Removing iptables rules ...')
    os.system('ip6tables -F INPUT')

```



## 14.6 Servidores WEB en NodeJS en SA64503

Dockerfile y código del servidor en Node.js para que escuche en IPv6.

Dockerfile:

```
FROM node:14

WORKDIR /app

COPY server.js .

CMD ["node", "server.js"]
```

server.js:

```
const http = require('http');

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/html');
  res.end('<h1>Hola mundo! Soy el servidor web del ASN 64503</h1>');
});

const port = process.env.PORT || 80;
server.listen(port, () => {
  console.log(`Server running on port ${port}`);
});
```

## 14.7 Servidores WEB en NodeJS en sistema autónomo atacante SA64504

Dockerfile y código del servidor en Node.js para que escuche en IPv6.

Dockerfile:

```
FROM node:14

WORKDIR /app

COPY server.js .

CMD ["node", "server.js"]
```

server.js:

```
const http = require('http');

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/html');
  res.end('<h1>Hola mundo! Soy el servidor web del atacante en el ASN 64504</h1>');
});
```

```
});

const port = process.env.PORT || 80;
server.listen(port, () => {
  console.log(`Server running on port ${port}`);
});
```

## 14.8 Configuración del R1

Primero, se creó un nodo Docker en GNS3 utilizando la imagen frouting/fr-debian. A continuación, se activó el servicio BGP modificando el archivo `/etc/frr/daemons`. Después, se configuró el servicio DNS en el archivo `resolv.conf` y se estableció un nodo NAT de GNS3 para proporcionar conectividad a Internet al enrutador. Finalmente, se aplicaron los siguientes comandos para configurar el enrutador:

```
fr# conf t
fr(config)# hostname r1
R1(config)# ipv6 forwarding
R1(config)# interface eth0
R1(config-if)# ipv6 address 2001:db8:beef:1::1/64
R1(config)# interface eth1
R1(config-if)# ip address 192.168.122.220/24
R1(config-if)# ipv6 address 2001:db8:100::2/64
R1(config)# interface eth2
R1(config-if)# ip address 9.0.4.1/24
R1(config-if)# ipv6 address 2001:db8:feed:1::1/64
R1(config)# ip route 0.0.0.0/0 192.168.122.1
R1(config)# ipv6 route 2001:db8:100::/40 2001:db8:100::1
R1(config)# router bgp 64501
R1(config-router)# no bgp ebgp-requires-policy
R1(config-router)# neighbor 2001:db8:beef:1::2 remote-as 64502
R1(config-router)# neighbor 2001:db8:beef:1::2 ebgp-multihop 2
R1(config-router)# neighbor 2001:db8:beef:1::2 update-source 2001:db8:beef:1::1
R1(config-router)# neighbor 2001:db8:feed:1::2 remote-as 64504
R1(config-router)# neighbor 2001:db8:feed:1::2 ebgp-multihop 2
```

```
R1(config-router)# neighbor 2001:db8:feed:1::2 update-source 2001:db8:feed:1::1
```

```
R1(config-router)# address-family ipv6 unicast
```

```
R1(config-router-af)# network 2001:db8:100::/40
```

```
R1(config-router-af)# neighbor 2001:db8:beef:1::2 activate
```

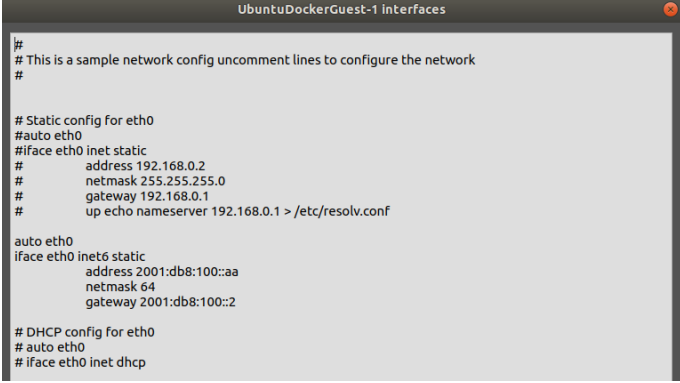
```
R1(config-router-af)# neighbor 2001:db8:feed:1::2 activate
```

Por último, se instalaron los paquetes necesarios para trabajar con Web3:

```
apt-get update --fix-missing
apt upgrade
apt install -y less
apt install -y net-tools
apt-get install -y libtool
apt-get install -y libssl-dev
apt-get install -y python3-pip
apt-get install -y pkg-config
apt-get install -y libsecp256k1-dev
apt-get install libffi-dev
pip3 install wheel
pip3 install eth-testrpc
pip3 install web3
pip3 install ipython
pip3 install netaddr
apt-get install -y build-essential python-dev libffi-dev libxml2-dev libxslt1-
    dev zlib1g-dev libnetfilter-queue-dev iptables
apt-get install libcurl4-openssl-dev
DEBIAN_FRONTEND=noninteractive apt-get -y install tshark
apt install libnetfilter-queue1 libnetfilter-queue-dev
apt install python3-netfilter
pip3 install scapy
pip3 install libbgp
pip3 install pyshark
# pip3 install NetfilterQueue
apt install git
pip3 install cython
pip3 install -U git+https://github.com/kti/python-netfilterqueue
# https://sauravjalui.com/how-to-solve-error-command-x8664-linux-gnu-gcc-failed
    -with-exit-status-1
sysctl -w net.ipv4.ip_forward=1
sed -i 's/#net.ipv4.ip_forward=1/net.ipv4.ip_forward=1/g' /etc/sysctl.conf
sed -i 's/#net.ipv6.conf.all.forwarding=1/net.ipv6.conf.all.forwarding=1/g' /
    etc/sysctl.conf
sysctl -p
```

Copiamos el script python dentro del contenedor: `docker cp ids.py mycontainer:/root/`

## 14.9 Configuración del cliente docker C1



```
#
# This is a sample network config uncomment lines to configure the network
#

# Static config for eth0
#auto eth0
#iface eth0 inet static
#    address 192.168.0.2
#    netmask 255.255.255.0
#    gateway 192.168.0.1
#    up echo nameserver 192.168.0.1 > /etc/resolv.conf

auto eth0
iface eth0 inet6 static
    address 2001:db8:100::aa
    netmask 64
    gateway 2001:db8:100::2

# DHCP config for eth0
# auto eth0
# iface eth0 inet dhcp
```

Figura 14.1: GNS3 - Configuración de las interfaces de red del contenedor Docker

## 14.10 Configuración del router R1-2

```
Router#configure terminal
Router(config)#hostname r1-2
Router(config)# int Gi0/0
Router(config-if)# ipv6 address 2001:db8:100::1/64
Router(config-if)# no shutdown
Router# copy running-config startup-config
Router# show ipv6 interface Gi0/0
```

## 14.11 Configuración del router R2

```
Router#configure terminal
Router(config)# ipv6 unicast-routing
Router(config)# hostname r2
r2(config)#int Gi0/0
r2(config-if)# ipv6 address 2001:db8:beef:1::2/64
r2(config-if)# no shutdown
r2(config)# int Loopback0
r2(config-if)# ipv6 address 2001:db8:200::1/40
r2(config)#int Gi0/1
r2(config-if)# ipv6 address 2001:db8:cafe:1::1/64
```

```

r2(config-if)# no shutdown
r2(config)# router bgp 64502
r2(config-router)# bgp routerid 9.0.0.2
r2(config-router)# neighbor 2001:db8:beef:1::1 remote-as 64501
r2(config-router)# neighbor 2001:db8:beef:1::1 ebgp-multihop 2
r2(config-router)# neighbor 2001:db8:cafe:1::2 remote-as 64503
r2(config-router)# no address-family ipv4
r2(config-router)# address-family ipv6 unicast
r2(config-router-af)# network 2001:db8:200::/40
r2(config-router-af)# neighbor 2001:db8:beef:1::1 activate
r2(config-router-af)# neighbor 2001:db8:cafe:1::2 activate
r2(config-router-af)# exit-address-family
r2# copy running-config startup-config

```

### 14.12 Configuración del router R3

```

set interfaces ethernet eth0 address 2001:db8:cafe:1::2/64
set interfaces ethernet eth0 address 9.0.1.2/24
set interfaces ethernet eth1 address 2001:db8:300::2/64
set protocols bgp 64503 neighbor 2001:db8:cafe:1::1 ebgp-multihop '2'
set protocols bgp 64503 neighbor 2001:db8:cafe:1::1 remote-as '64502'
set protocols bgp 64503 neighbor 2001:db8:cafe:1::1 update-source '2001:db8:cafe:1::2'
set protocols bgp 64503 neighbor 2001:db8:cafe:1::1 address-family ipv6-unicast
set protocols bgp 64503 address-family ipv6-unicast network '2001:db8:300::/40'
set protocols static route6 2001:db8:300::/40 next-hop '2001:db8:300::1'

```

### 14.13 Configuración del router R3-2

```

Router# configure terminal
Router(config)# int Gi0/0
Router(config-if)# ipv6 address 2001:db8:300::1/64
Router(config-if)# no shutdown
Router# copy running-config startup-config

```

```
Router# show ipv6 interface Gi0/0
```

#### **14.14 Configuración del router R4**

```
set interfaces ethernet eth0 address 2001:db8:feed:1::2/64
```

```
set interfaces ethernet eth0 address 9.0.4.2/24
```

```
set interfaces ethernet eth1 address 2001:db8:300::2/64
```

```
set protocols bgp 64504 neighbor 2001:db8:feed:1::1 ebgp-multihop '2'
```

```
set protocols bgp 64504 neighbor 2001:db8:feed:1::1 remote-as '64501'
```

```
set protocols bgp 64504 neighbor 2001:db8:feed:1::1 update-source '2001:db8:feed:1::2'
```

```
set protocols bgp 64504 neighbor 2001:db8:feed:1::1 address-family ipv6-unicast
```

```
set protocols bgp 64504 address-family ipv6-unicast network '2001:db8:300::/40'
```

```
set protocols static route6 2001:db8:300::/40 next-hop '2001:db8:300::1'
```

#### **14.15 Configuración del router R4-2**

```
Router# configure terminal
```

```
Router(config)# int Gi0/0
```

```
Router(config-if)# ipv6 address 2001:db8:300::1/64
```

```
Router(config-if)# no shutdown
```

```
Router# copy running-config startup-config
```

```
Router# show ipv6 interface Gi0/0
```

## Bibliografía

- [1] J. Paillisse, A. Rodriguez-Natal, V. Ermagan, F. Maino, L. Vegoda y A. Cabellos-Aparicio, «An analysis of the applicability of blockchain to secure IP addresses allocation, delegation and bindings.», Internet Engineering Task Force, Internet-Draft draft-paillisse-sidrops-blockchain-02, jun. de 2018, Work in Progress, 30 págs. dirección: <https://datatracker.ietf.org/doc/draft-paillisse-sidrops-blockchain/02/>.
- [2] APNIC, *History of the RIRs*, 10 de jun. de 2014. dirección: <https://www.youtube.com/watch?v=7I0fb2N4wHU> (visitado 01-12-2020).
- [3] «MANUAL DE POLÍTICAS DE LACNIC (v2.16)». (), dirección: <https://www.lacnic.net/innovaportal/file/543/1/manual-politicas-sp-2-16.pdf> (visitado 03-08-2022).
- [4] C. Aguerre. «Libro : El Desarrollo de la Comunidad de LACNIC». (2019), dirección: <https://www.lacnic.net/libro-comunidad> (visitado 15-12-2020).
- [5] B. Kuerbis y M. Mueller, «Internet routing registries, data governance, and security», *Journal of Cyber Policy*, vol. 2, n° 1, págs. 64-81, 2 de ene. de 2017, ISSN: 2373-8871, 2373-8898. DOI: 10.1080/23738871.2017.1295092. dirección: <https://www.tandfonline.com/doi/full/10.1080/23738871.2017.1295092> (visitado 01-10-2019).
- [6] G. Cicileo y G. Rada, «RPKI: Validación de Origen en BGP», pág. 32, 2019.
- [7] «Q4 2022 DDoS Attacks and BGP Incidents». (), dirección: <https://habr.com/en/companies/qrator/articles/713734/> (visitado 14-04-2023).
- [8] R. Singel, «Pakistan’s Accidental YouTube Re-Routing Exposes Trust Flaw in Net», *Wired*, 25 de feb. de 2008, ISSN: 1059-1028. dirección: <https://www.wired.com/2008/02/pakistans-accid/> (visitado 19-09-2019).
- [9] A. Greenberg, «Hacker Redirects Traffic From 19 Internet Providers to Steal Bitcoins», *Wired*, 7 de ago. de 2014, ISSN: 1059-1028. dirección: <https://www.wired.com/2014/08/isp-bitcoin-theft/> (visitado 19-09-2019).
- [10] A. Siddiqui. «Not just another BGP hijack», MANRS. Library Catalog: [www.manrs.org](http://www.manrs.org). (6 de abr. de 2020), dirección: <https://www.manrs.org/2020/04/not-just-another-bgp-hijack/> (visitado 12-04-2020).
- [11] M. Lepinski y S. Kent. «An infrastructure to support secure internet routing». Library Catalog: [tools.ietf.org](http://tools.ietf.org). (24 de feb. de 2012), dirección: <https://tools.ietf.org/html/rfc6480> (visitado 30-04-2020).
- [12] «MANRS – Mutually Agreed Norms for Routing Security». (), dirección: <https://www.manrs.org/> (visitado 27-09-2019).
- [13] «RPKI Deployment Monitor». (), dirección: <https://rpki-monitor.antd.nist.gov/?p=0&s=0> (visitado 29-09-2019).
- [14] D. Cooper, E. Heilman, K. Brogle, L. Reyzin y S. Goldberg, «On the risk of misbehaving RPKI authorities», en *Proceedings of the Twelfth ACM Workshop on Hot Topics in Networks - HotNets-XII*, College Park, Maryland: ACM Press, 2013, págs. 1-7, ISBN: 978-1-4503-2596-7. DOI: 10.1145/2535771.2535787. dirección: <http://dl.acm.org/citation.cfm?doid=2535771.2535787> (visitado 21-09-2019).

- [15] J. Paillisse. «IPchain: Securing IP prefix allocation and delegation with blockchain | jordi paillisse | request PDF», ResearchGate. (14 de mayo de 2018), dirección: [https://www.researchgate.net/publication/333585120\\_IPchain\\_Securing\\_IP\\_Prefix\\_Allocation\\_and\\_Delegation\\_with\\_Blockchain](https://www.researchgate.net/publication/333585120_IPchain_Securing_IP_Prefix_Allocation_and_Delegation_with_Blockchain) (visitado 18-07-2019).
- [16] Sharon Goldberg: *The Transition to BGP Security: Is the Juice Worth the Squeeze?*, 9 de abr. de 2014. dirección: <https://www.youtube.com/watch?v=68W7biIsJ14> (visitado 21-09-2019).
- [17] M. Lepinski y K. Sriram, *BGPsec Protocol Specification*, RFC 8205, sep. de 2017. DOI: 10.17487/RFC8205. dirección: <https://www.rfc-editor.org/info/rfc8205>.
- [18] S. Kent, C. Lynn y K. Seo, «Secure Border Gateway Protocol (S-BGP)», *IEEE Journal on Selected Areas in Communications*, vol. 18, nº 4, págs. 582-592, 2000. DOI: 10.1109/49.839934.
- [19] J. Ng, «Extensions to BGP to Support Secure Origin BGP (soBGP)», Internet Engineering Task Force, Internet-Draft draft-ng-sobgp-bgp-extensions-02, abr. de 2004, Work in Progress, 16 págs. dirección: <https://datatracker.ietf.org/doc/draft-ng-sobgp-bgp-extensions/02/>.
- [20] W. Haag, D. Montgomery, W. C. Barker y A. Tan, «Protecting the integrity of internet routing:: Border gateway protocol (BGP) route origin validation», National Institute of Standards y Technology, Gaithersburg, MD, NIST SP 1800-14, jun. de 2019, NIST SP 1800-14. DOI: 10.6028/NIST.SP.1800-14. dirección: <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.1800-14.pdf> (visitado 30-09-2019).
- [21] R. Lychev, S. Goldberg y M. Schapira, «BGP security in partial deployment: Is the juice worth the squeeze?», *arXiv:1307.2690 [cs]*, 10 de jul. de 2013. arXiv: 1307.2690. dirección: <http://arxiv.org/abs/1307.2690> (visitado 21-09-2019).
- [22] G. Urbini e I. Martín, «Certificados Digitales: De Una Arquitectura Jerárquica Y Centralizada a Una Distribuida Y Descentralizada», Tesis, Universidad Nacional de La Plata, 2018. DOI: <https://doi.org/10.35537/10915/72076>. dirección: <http://sedici.unlp.edu.ar/handle/10915/72076> (visitado 15-10-2019).
- [23] S. Angieri, A. García-Martínez, B. Liu, Z. Yan, C. Wang y M. Bagnulo, «A Distributed Autonomous Organization for Internet Address Management», *IEEE Transactions on Engineering Management*, págs. 1-17, 2019, Conference Name: IEEE Transactions on Engineering Management, ISSN: 1558-0040. DOI: 10.1109/TEM.2019.2924737.
- [24] «DAOs, Multi-Sigs, and Which is Right For You». (), dirección: <https://daohaus.substack.com/p/daos-multi-sigs-and-which-is-right> (visitado 25-09-2020).
- [25] «Proxy patterns», OpenZepplin blog. (20 de abr. de 2018), dirección: <https://blog.openzeppelin.com/proxy-patterns/> (visitado 05-12-2021).
- [26] «Internet Number Resource Status Report. NRO-Statistics-2020-Q3-FINAL.pdf». (30 de sep. de 2020), dirección: <https://www.nro.net/wp-content/uploads/NRO-Statistics-2020-Q3-FINAL.pdf> (visitado 15-12-2020).
- [27] Guangliang Pan. «Keep calm and carry on: The status of IPv4 address allocation», APNIC Blog. Section: Tech matters. (13 de dic. de 2019), dirección: <https://blog.apnic.net>.



net/2019/12/13/keep-calm-and-carry-on-the-status-of-ipv4-address-allocation/ (visitado 01-10-2020).

- [28] Guillermo Cicileo. «LACNIC. Dr IPv6». (2019), dirección: <https://www.lacnic.net/innovaportal/file/2944/1/dripv6-46.mp3> (visitado 02-10-2020).
- [29] D. Tsumak, «Securing BGP using blockchain technology», 2018.
- [30] M. Saad, A. Anwar, A. Ahmad, H. Alasmay, M. Yuksel y A. Mohaisen, «RouteChain: Towards blockchain-based secure and efficient BGP routing», en *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, Seoul, Korea (South): IEEE, mayo de 2019, págs. 210-218, ISBN: 978-1-72811-328-9. DOI: 10.1109/BL0C.2019.8751229. dirección: <https://ieeexplore.ieee.org/document/8751229/> (visitado 04-07-2019).
- [31] Q. Xing, B. Wang y X. Wang, «BGPcoin: Blockchain-based internet number resource authority and BGP security solution», *Symmetry*, vol. 10, n° 9, pág. 408, sep. de 2018. DOI: 10.3390/sym10090408. dirección: <https://www.mdpi.com/2073-8994/10/9/408> (visitado 01-06-2019).
- [32] V. Buterin, D. Hernandez, T. Kamphefner et al., «Combining GHOST and casper», *arXiv:2003.03052 [cs]*, 11 de mayo de 2020. arXiv: 2003.03052. dirección: <http://arxiv.org/abs/2003.03052> (visitado 02-07-2020).
- [33] R. Nakamura, T. Jimba y D. Harz, «Refinement and verification of CBC casper», en *2019 Crypto Valley Conference on Blockchain Technology (CVCBT)*, Rotkreuz, Switzerland: IEEE, jun. de 2019, págs. 26-38, ISBN: 978-1-72813-669-1. DOI: 10.1109/CVCBT.2019.00008. dirección: <https://ieeexplore.ieee.org/document/8787558/> (visitado 12-10-2020).
- [34] C. Schwarz. «Ethereum 2.0: A Complete Guide. Scaling Ethereum — Part Two: Sharding.» (10 de jul. de 2019), dirección: <https://medium.com/chainsafe-systems/ethereum-2-0-a-complete-guide-scaling-ethereum-part-two-sharding-902370ac3be> (visitado 21-11-2020).
- [35] «Sharding on Ethereum - EthHub». (16 de nov. de 2019), dirección: <https://docs.ethhub.io/ethereum-roadmap/ethereum-2.0/sharding/> (visitado 12-10-2020).
- [36] J. Hawkinson y T. Bates. «Guidelines for creation, selection, and registration of an autonomous system (AS)». (mar. de 1996), dirección: <https://tools.ietf.org/html/rfc1930> (visitado 05-03-2021).
- [37] «Special-Purpose Autonomous System (AS) Numbers». (7 de ago. de 2015), dirección: <https://www.iana.org/assignments/iana-as-numbers-special-registry/iana-as-numbers-special-registry.xhtml#special-purpose-as-numbers> (visitado 05-03-2021).
- [38] «Fascículos para entender la infraestructura de Internet - Direcciones IP y ASN». (oct. de 2019), dirección: <https://www.lacnic.net/innovaportal/file/978/3/lacnic-fasciculo-infraestructura-internet-es.pdf> (visitado 19-04-2020).
- [39] «Solicitar IP y ASN». (), dirección: <https://www.lacnic.net/solicitar-ip> (visitado 14-04-2023).

- [40] «Border Gateway Protocol (BGP)», Internet Engineering Task Force, Request for Comments RFC 1105, jun. de 1989, Num Pages: 17. DOI: 10.17487/RFC1105. dirección: <https://datatracker.ietf.org/doc/rfc1105> (visitado 31-12-2022).
- [41] N. d. Rio, «Diseño e implementación de una solución de administración de tráfico de red basada en DNS y chequeos de disponibilidad», Magister en Redes de Datos, Universidad Nacional de La Plata, 12 de feb. de 2016. DOI: 10.35537/10915/51098. dirección: <http://sedici.unlp.edu.ar/handle/10915/51098> (visitado 08-02-2021).
- [42] «Fundamentos de BGP e Introducción a RPKI Info: Fundamentos de BGP e Introducción a RPKI». (), dirección: <https://campus.lacnic.net/mod/page/view.php?id=4707> (visitado 26-02-2021).
- [43] P. d. 1. M. 2. news, ris e I. Governance. «YouTube Hijacking: A RIPE NCC RIS case study», RIPE Network Coordination Centre. Library Catalog: [www.ripe.net](http://www.ripe.net). (), dirección: <https://www.ripe.net/publications/news/industry-developments/youtube-hijacking-a-ripe-ncc-ris-case-study> (visitado 12-04-2020).
- [44] K. Sriram, D. Montgomery, D. McPherson, E. Osterweil y B. Dickson, «Problem definition and classification of BGP route leaks», RFC Editor, RFC7908, jun. de 2016, RFC7908. DOI: 10.17487/RFC7908. dirección: <https://www.rfc-editor.org/info/rfc7908> (visitado 22-03-2021).
- [45] A. Band, «The RPKI Documentation», pág. 173,
- [46] B. Weber, T. Bruijnzeels, O. Muravskiy y R. Austein. «The RPKI repository delta protocol (RRDP)». (), dirección: <https://tools.ietf.org/html/rfc8182> (visitado 05-03-2021).
- [47] R. Bush y R. Austein. «The resource public key infrastructure (RPKI) to router protocol, version 1». (), dirección: <https://tools.ietf.org/html/rfc8210> (visitado 05-03-2021).
- [48] A. Azimov, E. Uskov, R. Bush, J. Snijders, R. Housley y B. Maddison, «A Profile for Autonomous System Provider Authorization», Internet Engineering Task Force, Internet-Draft draft-ietf-sidrops-asma-profile-11, oct. de 2022, Work in Progress, 11 págs. dirección: <https://datatracker.ietf.org/doc/draft-ietf-sidrops-asma-profile/11/>.
- [49] «draft-ietf-sidrops-asma-verification-07 - Verification of AS\_PATH Using the Resource Certificate Public Key Infrastructure and Autonomous System Provider Authorization». (), dirección: <https://datatracker.ietf.org/doc/draft-ietf-sidrops-asma-verification/> (visitado 05-03-2021).
- [50] *MANRS – Mutually Agreed Norms for Routing Security*. dirección: <https://www.manrs.org/> (visitado 27-09-2019).
- [51] «A Survey on Securing Inter-Domain Routing». (), dirección: <https://www.potaroo.net/ispcol/2021-07/sidr2.html> (visitado 07-2021).
- [52] V. K. Sriram y D. Montgomery, «Design and analysis of optimization algorithms to minimize cryptographic processing in BGP security protocols», *Computer Communications*, vol. 106, págs. 75-85, 2017, ISSN: 0140-3664. DOI: <https://doi.org/10.1016/j>.

- comcom. 2017. 03. 007. dirección: <https://www.sciencedirect.com/science/article/pii/S0140366417303365>.
- [53] S. Goldberg, «Why is it taking so long to secure internet routing?», pág. 14,
- [54] Q. Labs. «Eliminating opportunities for traffic hijacking», Medium. (1 de mar. de 2019), dirección: <https://qratorlabs.medium.com/eliminating-opportunities-for-traffic-hijacking-153a39395778> (visitado 02-03-2021).
- [55] «BGP Route Security Cycling to the Future!» (), dirección: [https://pc.nanog.org/static/published/meetings/NANOG76/1978/20190611\\_Azimov\\_Bgp\\_Route\\_Security\\_v1.pdf](https://pc.nanog.org/static/published/meetings/NANOG76/1978/20190611_Azimov_Bgp_Route_Security_v1.pdf) (visitado 06-2019).
- [56] «Mecanismo de protección de ruta BGP mejorado basado en RPKI-ASPA». (), dirección: <http://c-s-a.org.cn/html/2022/2/8321.htm> (visitado 02-2022).
- [57] G. Coulouris, J. Dollimore, T. Kindberg y J. Junquera, *Sistemas distribuidos: conceptos y diseño* (Fuera de colección Out of series). Pearson Educación, 2001, ISBN: 9788478290499. dirección: <https://books.google.com.ar/books?id=aA5BAAAACAAJ>.
- [58] A. Tanenbaum y M. van Steen, *Distributed Systems: Principles and Paradigms*. Pearson Prentice Hall, 2007, ISBN: 9780132392273. dirección: <https://books.google.com.ar/books?id=DL8ZAQAATAAJ>.
- [59] I. Bashir, *Mastering Blockchain 3rd Edition*. 31 de ago. de 2020, ISBN: 978-1-83921-319-9.
- [60] L. Lamport, R. Shostak y M. Pease, «The Byzantine Generals Problem», *ACM Trans. Program. Lang. Syst.*, vol. 4, n° 3, págs. 382-401, jul. de 1982, ISSN: 0164-0925. DOI: 10.1145/357172.357176. dirección: <https://doi.org/10.1145/357172.357176>.
- [61] M. Castro y B. Liskov, «Practical Byzantine Fault Tolerance», *OSDI*, mar. de 1999.
- [62] *CAP theorem*, en, Page Version ID: 1128018230, dic. de 2022. dirección: [https://en.wikipedia.org/w/index.php?title=CAP\\_theorem&oldid=1128018230](https://en.wikipedia.org/w/index.php?title=CAP_theorem&oldid=1128018230) (visitado 04-01-2023).
- [63] J. Holbrook, *Architecting Enterprise Blockchain Solutions*. John Wiley & Sons, 11 de feb. de 2020, 400 págs., Google-Books-ID: ldjIDwAAQBAJ, ISBN: 978-1-119-55769-2.
- [64] A. J. Menezes, P. C. v. Oorschot y S. A. Vanstone, *Handbook of Applied Cryptography*. Boca Raton, 16 de dic. de 1996, 780 págs., ISBN: 978-0-8493-8523-0.
- [65] *Random number generation (TRNG, PRNG, DRBG, RNG) solutions for FPGA (Xilinx, Altera, Microsemi, Lattice) and ASIC - Helion Technology*. dirección: <https://www.heliontech.com/random.htm> (visitado 05-01-2023).
- [66] A. Rodriguez, *Blockchain: funciones hash*, en, ago. de 2019. dirección: <https://adr-rod87.medium.com/blockchain-funciones-hash-a1ff21fe7e7f> (visitado 05-01-2023).
- [67] *¿Qué es la criptografía simétrica?*, es, Section: Fundamentos. dirección: <https://academy.bit2me.com/que-es-criptografia-simetrica/> (visitado 06-01-2023).
- [68] *Public key cryptography*. dirección: <https://docs.huihoo.com/globus/gt3-tutorial/ch10s03.html> (visitado 06-01-2023).

- [69] R. L. Rivest, A. Shamir y L. Adleman, «A method for obtaining digital signatures and public-key cryptosystems», *Communications of the ACM*, vol. 21, n° 2, págs. 120-126, 1978.
- [70] *Mastering Ethereum*, original-date: 2016-08-10T15:07:54Z, ene. de 2023. dirección: <https://github.com/ethereumbook/ethereumbook/blob/c5ddeb3dbec804463c86d0ae2de9f28fbafb8304keys-addresses.asciidoc> (visitado 06-01-2023).
- [71] *Merkling in Ethereum*, en. dirección: <https://blog.ethereum.org/2015/11/15/merkle-in-ethereum> (visitado 06-01-2023).
- [72] *Trie*, es, Page Version ID: 138084519, sep. de 2021. dirección: <https://es.wikipedia.org/w/index.php?title=Trie&oldid=138084519> (visitado 06-01-2023).
- [73] G. Wood, «Ethereum: A secure decentralised generalised transaction ledger», *Ethereum project yellow paper*, vol. 151, págs. 1-32, 2014.
- [74] *ISO/TC 307 - Blockchain and distributed ledger technologies*, en. dirección: <https://www.iso.org/committee/6266604.html> (visitado 07-01-2023).
- [75] *What is Decentralization?*, en-US. dirección: <https://aws.amazon.com/blockchain/decentralization-in-blockchain/> (visitado 09-01-2023).
- [76] V. Buterin. «The meaning of decentralization», Medium. Library Catalog: medium.com. (6 de feb. de 2017), dirección: <https://medium.com/@VitalikButerin/the-meaning-of-decentralization-a0c92b76a274> (visitado 30-06-2020).
- [77] M. Anderson, «Exploring Decentralization: Blockchain Technology and Complex Coordination», en, *Journal of Design and Science*, feb. de 2019. dirección: <https://jods.mitpress.mit.edu/pub/7vxemt3/release/2> (visitado 09-01-2023).
- [78] «Consensus algorithms in blockchain», GeeksforGeeks. (25 de abr. de 2019), dirección: <https://www.geeksforgeeks.org/consensus-algorithms-in-blockchain/> (visitado 03-12-2019).
- [79] M. Castro, B. Liskov et al., «Practical byzantine fault tolerance», en *OsDI*, vol. 99, 1999, págs. 173-186.
- [80] Y. Wang, M. Zhong y T. Cheng, «Research on PBFT consensus algorithm for grouping based on feature trust», *Scientific Reports*, vol. 12, n° 1, págs. 1-12, 2022.
- [81] C. Dwork y M. Naor, «Pricing via processing or combatting junk mail», en *Annual international cryptology conference*, Springer, 1992, págs. 139-147.
- [82] S. Nakamoto, «Bitcoin whitepaper», URL: <https://bitcoin.org/bitcoin.pdf>-(: 17.07. 2019), 2008.
- [83] H. Xiong, M. Chen, C. Wu, Y. Zhao y W. Yi, «Research on Progress of Blockchain Consensus Algorithm: A Review on Recent Progress of Blockchain Consensus Algorithms», *Future Internet*, vol. 14, n° 2, 2022, ISSN: 1999-5903. DOI: 10.3390/fi14020047. dirección: <https://www.mdpi.com/1999-5903/14/2/47>.
- [84] *Bitcoin Energy Consumption Index*, en-US. dirección: <https://digiconomist.net/bitcoin-energy-consumption/> (visitado 09-01-2023).
- [85] *Prueba de participación (PoS)*, es. dirección: <https://ethereum.org> (visitado 09-01-2023).

- [86] D. P. Oyinloye, J. S. Teh, N. Jamil y M. Alawida, «Blockchain Consensus: An Overview of Alternative Protocols», *Symmetry*, vol. 13, nº 8, 2021, ISSN: 2073-8994. DOI: 10.3390/sym13081363. dirección: <https://www.mdpi.com/2073-8994/13/8/1363>.
- [87] A. Yakovenko, *Proof of History: A Clock for Blockchain*, en, mayo de 2020. dirección: <https://medium.com/solana-labs/proof-of-history-a-clock-for-blockchain-cf47a61a9274> (visitado 09-01-2023).
- [88] A. Yakovenko, *Tower BFT: Solana's High Performance Implementation of PBFT*, en, feb. de 2020. dirección: <https://medium.com/solana-labs/tower-bft-solanas-high-performance-implementation-of-pbft-464725911e79> (visitado 09-01-2023).
- [89] C. Russo, *The Infinite Machine: How an Army of Crypto-hackers is Building the Next Internet with Ethereum*. HarperBusiness, 2020, ISBN: 9780062886163. dirección: [https://books.google.com.ar/books?id=Q6W%5C\\_zQEACAAJ](https://books.google.com.ar/books?id=Q6W%5C_zQEACAAJ).
- [90] *Nodes and clients*, ethereum.org. dirección: <https://ethereum.org/en/developers/docs/nodes-and-clients/#consensus-clients> (visitado 24-02-2023).
- [91] I. Bashir, *Mastering Blockchain 4th Edition*. mar. de 2023, ISBN: 978-1-80324-106-7.
- [92] «How does ethereum work, anyway?» (), dirección: <https://www.preethikasireddy.com/post/how-does-ethereum-work-anyway> (visitado 13-09-2017).
- [93] «Ethereum Virtual Machine (EVM)». (), dirección: <https://ethereum.org/en/developers/docs/evm/> (visitado 19-01-2023).
- [94] «A Gentle Introduction to Arbitrum». (), dirección: <https://developer.offchainlabs.com/intro/> (visitado 01-03-2023).
- [95] «l2beats arbitrum». (), dirección: <https://l2beat.com/scaling/projects/arbitrum> (visitado 13-03-2023).
- [96] «bgp<sub>e</sub>thereum». (), dirección: [https://github.com/mcabranches/bgp%5C\\_ethereum](https://github.com/mcabranches/bgp%5C_ethereum) (visitado 25-01-2020).
- [97] M. A. Gomez, P. Bazán, N. Río y M. Morandi, «Routing Security Using Blockchain Technology», en ago. de 2021, págs. 46-59, ISBN: 978-3-030-84824-8. DOI: 10.1007/978-3-030-84825-5\_4.
- [98] S. B. Example, *Signature*, <https://solidity-by-example.org/signature/>, [Accedido el 21 de marzo de 2023], 2023.
- [99] J. Chittoda, *Mastering Blockchain Programming with Solidity: Write production-ready smart contracts for Ethereum blockchain with Solidity*. Packt Publishing, 2019, ISBN: 9781839218637. dirección: <https://books.google.com.ar/books?id=wWOnDwAAQBAJ>.
- [100] *Remix IDE*, <https://remix.ethereum.org/>, Accedido el 22 de marzo de 2023.
- [101] «Hardhat - Development environment». (), dirección: <https://hardhat.org/> (visitado 04-2023).
- [102] «Debian docker image with FRR». (), dirección: <https://hub.docker.com/r/frouting/frr-debian> (visitado 27-07-2021).
- [103] «Cisco IOSv». (), dirección: <https://gns3.com/marketplace/appliances/cisco-iosv> (visitado 08-12-2015).
- [104] «Vyos». (), dirección: <https://vyos.io/> (visitado 30-04-2023).

[105] «LACNet». (), dirección: <https://lacnet.lacchain.net/> (visitado 29-05-2023).