Universidad de Buenos Aires
Facultades de Ciencias Económicas, Ciencias Exactas y Naturales e Ingeniería

Carrera de Especialización en Seguridad Informática

Trabajo Final de Especialización

Seguridad en Contratos Inteligentes

Autora: Lic. Andrea Francisca Metetiero
Tutor: Dr. Hugo Scolnik
Año 2020
Cohorte 2019

Declaración jurada de origen de los contenidos

Por medio de la presente, el autor manifiesta conocer y aceptar el Reglamento de Trabajos Finales vigente y se hace responsable que la totalidad de los contenidos del presente documento son originales y de su creación exclusiva, o bien pertenecen a terceros u otras fuentes, que han sido adecuadamente referenciados y cuya inclusión no infringe la legislación Nacional e Internacional de Propiedad Intelectual.

Andrea Francisca Metetiero

DNI 23515279

Resumen

El presente trabajo se concentra en el análisis de los principales ataques perpetrados contra contratos inteligentes hasta el momento, haciendo énfasis en las vulnerabilidades de seguridad que fueron explotadas y los motivos. Con esto se intenta crear conciencia de los factores que se debe tener en cuenta a la hora de desarrollar, testear o auditar dichos contratos. El trabajo comienza con una introducción a la cadena de bloques Ethereum, su estructura y sus principales componentes. Luego se analizan los contratos inteligentes, incluyendo su estructura, casos de uso y vulnerabilidades. Más adelante se describen algunos ataques y sus consecuencias derivadas. Para concluir el trabajo, se hace un breve análisis sobre la implementación concreta de estos contratos y los desafíos que se pueden presentar tanto desde el punto de vista tecnológico como de cumplimiento legal.

Palabras claves: cadena de bloques, contratos inteligentes, criptomonedas, criptografía, ether, Ethereum, hash, seguridad, Solidity, vulnerabilidades.

Tabla de Contenido

Tabla de Ilustraciones

Introducción

Ethereum es una plataforma descentralizada de código abierto basada en tecnología de cadena de bloques, sobre la cual se puede programar aplicaciones conocidas como contratos inteligentes. Los contratos inteligentes se compilan y ejecutan sobre su propia máquina virtual, la máquina virtual de Ethereum (MVE) [1]. Cuenta con una criptomoneda, denominada Ether (ETH), que sirve de incentivo a los validadores, de aquí en más forjadores y como mecanismo para ahorrar costos computacionales. Aunque la red Ethereum posibilita realizar transacciones de criptomonedas, el principal objetivo de sus creadores era diseñar una computadora de uso general que permita el desarrollo de programas autoejecutables, es decir, que se ejecuten sin la intervención de terceros.

Vitalik Buterin, programador y cofundador de Ethereum, se planteó el dilema de construir un proyecto sobre las cadenas de bloques existentes, como Bitcoin o desarrollar una nueva plataforma especialmente orientada hacia este fin. Su idea se basó en trabajos publicados anteriormente por el informático, criptógrafo y jurista Nick Zabo, referentes a protocolos de confianza entre extraños para el procesamiento de operaciones en internet. Más tarde, Bitcoin aportó las bases para que, a finales de 2013, Buterin, publicara un artículo en donde explicaba su propuesta. El Dr. Gavin Wood se interesó en el proyecto y aportó sus conocimientos de programación, así es como a mediados del 2015 logran minar el primer bloque de manera exitosa [2].

A diferencia de Bitcoin, Ethereum se caracteriza por utilizar un lenguaje de programación Turing completo, es decir, que cuenta con la capacidad de realizar cualquier tipo de cálculo computacional gracias a la posibilidad de leer y escribir datos en memoria y de ejecutar programas almacenados en la MVE. Los contratos inteligentes son aplicaciones autoejecutables que intentan eliminar la participación de terceros confiables, a través del uso de aplicaciones descentralizadas y criptografía [2] y cuya principal característica es la capacidad de transferir Ether entre cuentas y entre contratos.

Las transacciones se validan mediante consenso, conocido como *proof of stake*, esto significa que los forjadores aportan sus propias criptomonedas en garantía para poder realizar estas verificaciones [3]. Otro atributo importante es la inmutabilidad, ya que una vez que un contrato es validado y se incorpora a la cadena de bloques, se va a ejecutar de acuerdo con las reglas con las que fue programado. Si bien esto presenta una ventaja, cualquier error en su diseño o implementación lo hace vulnerable, con consecuencias no deseadas como la pérdida o el robo de criptomonedas o el incumplimiento de acuerdos pactados.

Es por esto, que este trabajo estudiará los principales ataques contra contratos inteligentes conocidos hasta el momento y analizará las vulnerabilidades que permitieron que se produzcan. Para poder lograrlo, se hará una introducción a las características fundamentales de Ethereum y sus componentes. Además, se verán en más detalle las propiedades de los contratos, los posibles casos de uso y algunas peculiaridades de Solidity, el lenguaje de programación más utilizado para su desarrollo. Luego se intentará proponer mejores prácticas y contramedidas para prevenirlas o mitigarlas, de manera tal que sirva de guía a los usuarios, sean ellos programadores, probadores, auditores, criptógrafos o inversores.

Finalmente se evaluará la viabilidad de los contratos inteligentes como suplantación de los tradicionales y el impacto social consecuente de su aplicación. Para finalizar se enumerarán las conclusiones obtenidas.

1. Fundamentos de Ethereum

1.1. Cuentas

La red Ethereum posee dos tipos de cuentas y cada una de ellas cumple su funcionalidad. Las primeras son las EOA o cuentas de propiedad externa, que pueden contener saldo y se gestionan por medio de la clave privada del titular. Luego están las cuentas contrato, que son los contratos inteligentes controlados por los algoritmos de programación dispuestos en su código [4]. Las EOA almacenan y pueden transferir Ether y aunque no poseen código asociado son las encargadas de disparar la ejecución de contratos. Todas las transacciones dentro de la red Ethereum se activan desde estas cuentas. Los contratos por su parte también pueden poseer saldo y su ejecución se dispara a través de transacciones, provenientes de EOAs o llamadas provenientes de otros contratos. La MVE se encarga de ejecutar el código definido en los contratos en todos y cada uno de los nodos de la red [5].

Ya sea para la creación de cuentas como para verificar la autenticidad de las transacciones, Ethereum utiliza el Algoritmo de Firma Digital de Curva Elíptica (ECDSA), que se detallará en el apartado ‘Direcciones’. La creación de una cuenta se realiza a partir de un par de claves pública y privada, dentro de la curva secp256k1 [6]. De este modo, la clave pública se utiliza para identificar la dirección de la cuenta, mientras que la clave privada sirve para mantener el control de acceso a los fondos, como también para firmar transacciones digitalmente.

La clave pública se genera a partir de la clave privada, aplicando un algoritmo de hash. La titularidad de una cuenta se verifica a partir de la clave privada, requerida para firmar digitalmente una transacción. Del mismo modo, la firma digital del titular se utiliza para autenticar un contrato o insertar una nueva transacción a la cadena de bloques [7]. El mecanismo de creación de claves se describe a continuación.

1.1.1. Generación de claves

(a) La clave privada es un número aleatorio de 256 bits, elegido al azar, preferentemente proveniente de una fuente entrópica.

(b) Para obtener la clave pública, se multiplica la clave privada por el punto generador de la curva elíptica. La clave pública resultante es un punto en la curva que posee valores x,y.

(c) Estos valores x,y se concatenan y sobre el resultado se aplica la función hash keccak256.

(d) Luego para generar la dirección, se toman los últimos 20 bytes de la cadena de caracteres y se le agrega el prefijo 0x.

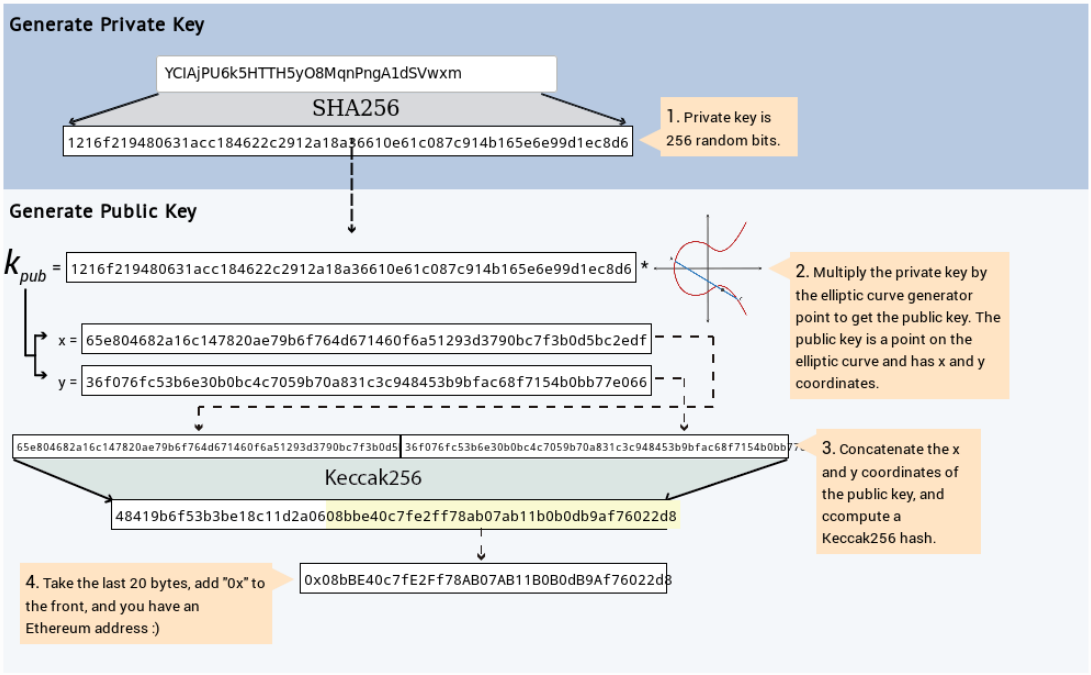


Ilustración 1. Generación de direcciones [8].

1.1.2. Estado de las cuentas

Tanto las cuentas como los contratos poseen un estado, formado por cuatro componentes: el *nonce*, el saldo, la *storageRoot* y el *codeHash* [4]. En las cuentas, el nonce representa el número de transacciones enviadas desde la dirección de esta cuenta, en los contratos representa el número de contratos creados por esa cuenta. El saldo, es el de Ether en esa dirección. StorageRoot

es el hash de 256 bits del nodo raíz de un árbol Merkle[1] Patricia[2] modificado que codifica el contenido de almacenamiento de la cuenta, codificado en el *trie*[3] como un mapeo del hash Keccak 256-bit. Por defecto se encuentra vacío. El *codeHash* es el hash del código que se ejecuta si una dirección recibe una llamada, es inmutable por lo que no se puede modificar después de su creación. Ese código se encuentra en la base de datos de estado bajo su correspondiente hash hasta su posterior recuperación [9].

1.2. Direcciones

Ethereum emplea criptografía de clave pública mediante direcciones y claves privadas para controlar la titularidad de fondos, para ello, utiliza el algoritmo de firma digital de curva elíptica (ECDSA). La criptografía de curva elíptica está basada en el problema del algoritmo discreto, es decir, aprovecha las propiedades de suma y multiplicación sobre los puntos de una curva elíptica, que son fáciles de obtener en un sentido, pero prácticamente imposibles de revertir. La criptografía de curvas elípticas presenta diversas ventajas sobre otros protocolos como RSA. Requiere claves de menor tamaño, consume menos potencia eléctrica, las operaciones se concentran en menos ciclos de CPU y es más segura ya que no se conocen algoritmos de ataque [10].

El protocolo que aplica es el secp256k1, un estándar del NIST (Instituto Nacional de Estándares y Tecnología), que corresponde a la curva elíptica $y^2 = x^3 + 7$ sobre un campo finito de Galois. En el caso de secp256k1, es el campo finito módulo p, donde $p = 2^{256} - 2^{32} - 997$ [11]. Este protocolo es el mismo que utiliza Bitcoin, pero contrariamente a otras curvas de estructura aleatoria, secp256k1, posee una estructura determinista.

La clave privada necesaria para la generación de la clave pública está constituida por un número aleatorio de 256 bits, preferentemente proveniente de una fuente segura de entropía, ya que no debe ser predecible ni fácil de

[1] https://respuestas.me/q/eli5-co-mo-funciona-un-a-rbol-merkle-patricia-trie-25229020477
[2] https://es.wikipedia.org/wiki/%C3%81rbol_de_Merkle
[3] https://es.wikipedia.org/wiki/Trie

obtener mediante fuerza bruta. La clave pública es un punto sobre la curva elíptica, es decir, un par de coordenadas xy obtenidas a partir de la clave privada. Como se explica en [7], a partir de una clave privada k, se selecciona un punto en la curva denominado generador G y se los multiplica entre sí para obtener otro punto en la curva que será la clave pública K resultante, de modo que K = k * G.

Se debe tener en cuenta que el punto generador G será el estándar de secp256k1, lo que implica que todas las claves públicas de todos los usuarios de Ethereum derivan del mismo punto generador G. En otras palabras, la misma clave privada k multiplicada por G siempre dará por resultado la misma clave pública K. Por este motivo se debe prestar especial cuidado a la correcta elección de la clave privada.

La dirección de Ethereum se crea a partir de la clave pública K y puede ser compartida con seguridad ya que la clave privada no podrá ser deducida a partir de ella. A la clave pública K se le aplica el hash conocido como Keccak-256 o SHA-3, luego se toman los últimos 20 bytes y se agrega 0x como prefijo a la cadena, que serán la dirección de Ethereum [7].

Las transacciones que se insertan a la cadena de bloques también utilizan criptografía de curva elíptica para su firma. Los detalles incluidos en los mensajes, más la clave privada crean un código que solo puede producirse si se conoce esa clave. Este código resultante representa la firma digital.

Pero a pesar del uso extensivo de criptografía, ni las transacciones ni los contratos inteligentes se encriptan, todo lo contrario, están disponibles públicamente y pueden ser visualizadas mediante herramientas como las que figuran en el sitio público Etherscan[4].

1.3. Billeteras

Las billeteras son aplicaciones de software o hardware que le permiten al usuario interactuar con la red. A través de una billetera, el usuario puede

[4] https://etherscan.io/

gestionar todas sus cuentas y controlar sus saldos, pero principalmente auspician como administrador y custodio de claves y direcciones y para la creación y firma de transacciones. Algunas billeteras además pueden interactuar con contratos inteligentes y aplicaciones descentralizadas.

Existen dos modelos de billeteras, no deterministas y deterministas. La diferencia está dada por el tipo de relación entre las claves. En las billeteras no deterministas cada clave se genera independientemente de las otras, a partir de un número aleatorio diferente y no mantienen ningún tipo de relación entre sí. En cambio, en las billeteras deterministas todas las claves derivan de una misma clave maestra que se conoce como semilla, están relacionadas entre sí y pueden volver a generarse a partir de la semilla original [12].

El último modelo es el más recomendable por su versatilidad, ya que facilita la realización de copias de resguardo, importación y exportación, facilitando la migración de claves a otras billeteras gracias a que solo se necesita la semilla original para recuperar todas las demás claves. Este modelo puede ser menos seguro, ya que a partir de la semilla se puede acceder a todas las demás claves y tomar el control absoluto de la billetera.

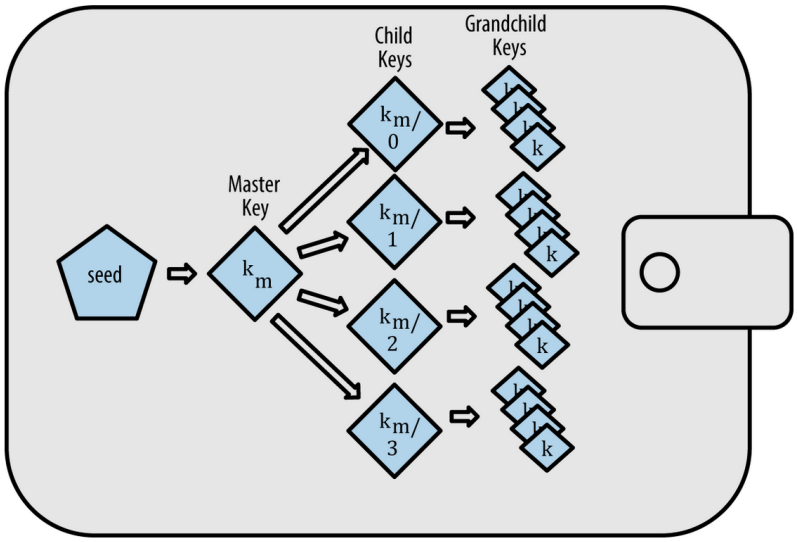


Ilustración 2 Billetera determinista jerárquica [12].

1.4. Gas

Gas es la unidad de medida de esfuerzo computacional que se requiere para ejecutar operaciones dentro de la red, ya sean transacciones o ejecución de contratos [13]. Su valor se establece en ETH que se envían a los forjadores como recompensa por verificar las transacciones. Pero además tiene otros propósitos como evitar ataques de DoS (denegación de servicio) en la red y forzar a los desarrolladores de contratos a evitar gastos computacionales excesivos e innecesarios, ya que cada línea de código requiere cierta cantidad de gas para ejecutarse [14].

Por ejemplo, si se envía 1 ETH de una cuenta a otra se le debe sumar el costo de gas de la transacción. Si ese costo fuese de 0,00057 ETH el usuario generador de la transacción deberá enviar 1,000057 ETH.

El gas tiene asociados dos atributos, el precio y el límite. El precio equivale a la cantidad de Wei (menor unidad de medida de ETH) que se debe pagar por unidad de gas para cubrir los costos computacionales generados de ejecutar una transacción. El límite es la cantidad máxima de gas que se puede consumir para ejecutar la transacción [9].

Value (in wei)	Exponent	Common name	SI name
1	1	wei	Wei
1,000	10^3	Babbage	Kilowei or femtoether
1,000,000	10^6	Lovelace	Megawei or picoether
1,000,000,000	10^9	Shannon	Gigawei or nanoether
1,000,000,000,000	10^{12}	Szabo	Microether or micro
1,000,000,000,000,000	10^{15}	Finney	Milliether or milli
1,000,000,000,000,000,000	*10^{18}*	*Ether*	*Ether*
1,000,000,000,000,000,000,000	10^{21}	Grand	Kiloether
1,000,000,000,000,000,000,000,000	10^{24}		Megaether

Ilustración 3. Denominaciones de la moneda Ether [2].

1.5. Transacciones

Toda actividad en Ethereum comienza con una transacción. Las transacciones son mensajes binarios firmados digitalmente, originados a partir de una cuenta, que se transmiten por la red y se graban en la cadena de bloques. Por medio de transacciones se producen los cambios de estado y se crean y ejecutan contratos.

Existen tres tipos de transacciones: (a) las que transfieren valores entre dos EOAs y modifican los saldos de ambas cuentas, (b) las que realizan llamadas a un contrato para ejecutar sus funciones y (c) las que crean contratos [15]. Una transacción contiene los siguientes datos:

(a) *Nonce*: es el número de secuencia de la transacción emitido por la cuenta emisora para evitar duplicados.
(b) *Precio del Gas*: el valor en Wei dispuesto a pagar por unidad de gas para afrontar los gastos computacionales de ejecutar una transacción. Es necesario para evitar ataques de denegación de servicio o transacciones que produzcan un alto consumo de recursos.
(c) *Límite del Gas:* el valor máximo de gas dispuesto a pagar por una transacción. Cuanto mayor sea el precio pagado por una transacción más rápida será su confirmación en la cadena de bloques.
(d) *Receptor:* es la cuenta receptora de una transacción. Se trata de una dirección de Ethereum de 20 bits y puede ser una cuenta o un contrato.
(e) *Valor:* la cantidad de Ether que se enviará al destinatario.
(f) *Dato:* los datos binarios enviados.
(g) *v,r,s:* los tres componentes de una firma digital ECDSA de la cuenta emisora.
(h) *Init:* Código utilizado para la inicialización de un contrato.

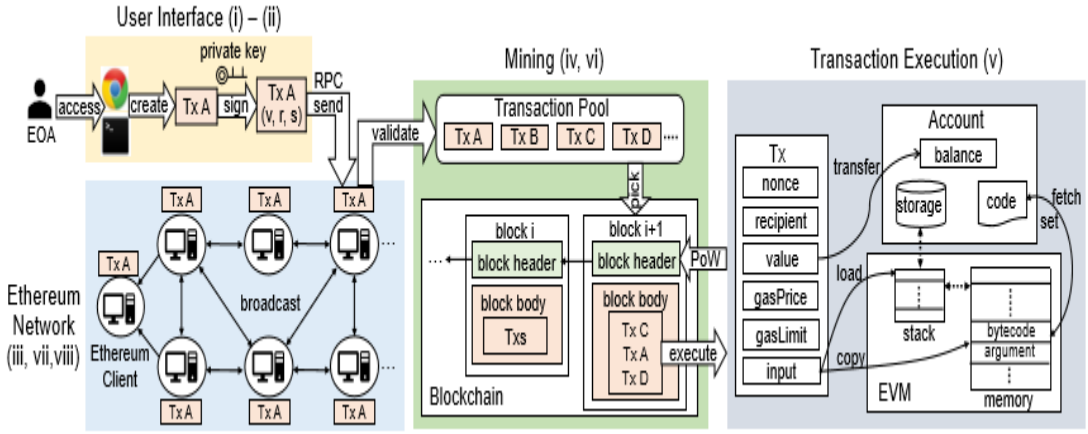


Ilustración 4. Ciclo de vida de una transacción [16].

1.6. Bloques

El tiempo estimado de generación de un nuevo bloque es de aproximadamente 14 segundos. El tamaño del bloque depende de la complejidad de los contratos que están siendo ejecutados y se determina por el límite de gas por bloque, que se estima alrededor de 1500000. Una transacción básica de transferencia de ETH puede requerir 21000 unidades de gas [17].

Un bloque es un registro de hashes de todas las transacciones que entran en ese bloque. Los hashes de dos transacciones se combinan para generar un nuevo hash hasta que todas las transacciones dentro de un bloque queden representadas por un único hash. Este hash se almacena en el encabezado del bloque y se lo conoce como raíz de Merkle.

Esto aporta la seguridad de que cualquier cambio que se realice en una transacción cambiará su hash y consecuentemente el hash raíz. El hash del bloque también se modificará y su bloque hijo cambiará, ya que éste almacena el hash de su padre. Todo este proceso asegura la inmutabilidad de las transacciones [18].

La estructura de un bloque se compone por el encabezado y el cuerpo. El encabezado del bloque es lo que forma parte de la cadena y contiene el

hash de su bloque padre. El cuerpo contiene el listado de transacciones incluidas en ese bloque y una lista de encabezados tíos (ommer[5]) [19].

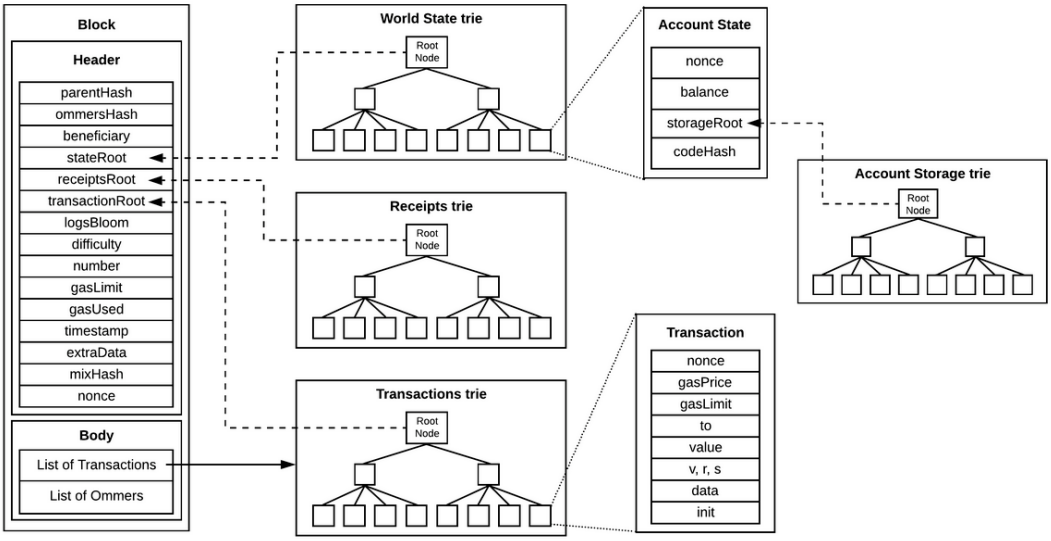


Ilustración 5. Esquema de árboles, cuentas, transacciones y bloques [19].

1.7. Consenso

Según el diccionario, consenso significa un acuerdo sobre algo entre todas las personas que pertenecen a una colectividad. Dentro del contexto de las cadenas de bloques, es la capacidad de arribar a un estado común dentro de un sistema descentralizado [20]. Ethereum genera consenso mediante el algoritmo de prueba de participación (*proof of stake*, PoS) y en este caso, al proceso de validación de nuevos bloques se lo denomina forjado y a los validadores forjadores.

El algoritmo funciona de la siguiente manera. Cualquier usuario que posea ETH puede postularse como forjador enviando una transacción que bloquea parte de sus fondos y los ofrece a modo de garantía. El monto de la garantía determina las posibilidades que tiene un nodo de ser elegido, ya a mayor sea el monto ofrecido en garantía mayores serán las posibilidades. Pero, para que la esta selección sea justa y no favorezca siempre a los nodos

[5] Ommer: bloque que fue minado, pero no se insertó aún en la cadena de bloques.

más acaudalados, se agrega un método adicional al proceso que puede ser la selección aleatoria o la antigüedad de la moneda [21].

En el método de selección aleatorio, se escoge el nodo mediante una combinación entre el valor más bajo de hash y la apuesta más alta. El tamaño de las apuestas es un dato público y posibilita conocer quién será el próximo herrero. El método de selección por antigüedad de la moneda calcula la cantidad de monedas acumuladas por la cantidad de días que se han mantenido en la cuenta. La antigüedad de la moneda vuelve a cero una vez que un bloque es forjado y se debe esperar un tiempo para poder forjar el siguiente bloque. Esto ayuda a evitar que los nodos más poderosos controlen la cadena de bloques.

El algoritmo PoS ofrece varias ventajas. En principio hace que los forjadores actúen con honestidad, ya que el monto de la recompensa es pequeño comparado al riesgo de perder sus depósitos en garantía. Esto aporta mayor seguridad y previene el forjado de transacciones fraudulentas conocido como ataque del 51% ya que, para que sea posible, el herrero debería poseer la mayoría de las criptomonedas en circulación. Otra ventaja de este algoritmo es que no requiere emisión de moneda y es eficiente en cuanto al gasto de energía.

2. Contratos Inteligentes

2.1. Propiedades

Ethereum posee la capacidad de permitir el desarrollo de programas ejecutables sobre su máquina virtual. Un contrato inteligente es un programa que se autoejecuta cuando los acuerdos plasmados en él por dos o más partes se cumplen sin necesidad de ser exigido por una autoridad [22]. A modo más técnico, es un conjunto de funciones convertidas en código legible por la MVE. La particularidad de los contratos inteligentes es poder transferir o recibir ETH entre cuentas o entre otros contratos, situación que los hace vulnerables a recibir ataques contra su seguridad.

Se puede comparar las similitudes entre un contrato inteligente y una clase en un lenguaje orientado a objetos como Java o Python [23]. Pero a diferencia de la última, estos programas son inmutables, lo que significa que una vez insertados a la cadena de bloques sus cláusulas no pueden modificarse y tampoco podrían rescindirse.

Los contratos inteligentes intentan reemplazar las funciones que cumplen los contratos tradicionales, pero a su vez presentan marcadas diferencias. Un contrato tradicional es un acuerdo entre partes con el objetivo de constituir, modificar o extinguir una regulación jurídica relacionada al patrimonio y consta de: el consentimiento de las partes, el objeto del contrato y cláusulas [17]. Los contratos inteligentes, en cambio, son anónimos y las partes se identifican únicamente por medio de sus direcciones de Ethereum. Asimismo, la titularidad de las criptomonedas depositadas en sus cuentas se demuestra por medio de la clave privada del titular, mediante firma digital, como se explica en el capítulo 1.

Para que un contrato inteligente se pueda validar e insertar a la cadena de bloques, las partes deben consensuar los términos de la transacción y realizar un depósito que garantice el cumplimiento de la obligación adquirida. Otra diferencia es el objeto, mientras que en el contrato tradicional pueden ser prestaciones o la transmisión de un derecho, en un contrato inteligente la obligación debe tener un respaldo digital para poder asegurar su

cumplimiento. Es decir, la obligación se debe poder cumplir de manera electrónica.

2.2. Ciclo de vida

El ciclo de vida comienza con la codificación del contrato inteligente en un lenguaje de programación de alto nivel. Existen varios lenguajes compatibles con la MVE, pero el más utilizado es Solidity. Una vez escrito el contrato, se compila transformándose en código legible por la MVE. Luego de ser compilado se inserta a la cadena de bloques mediante una transacción especial diseñada para la creación de contratos y queda a la espera de ser validado. Cada contrato se identifica con una dirección de Ethereum derivada de esta transacción, en función de la cuenta de origen y el nonce de esa cuenta.

Una vez insertados en la cadena de bloques, los contratos inteligentes son públicos y están disponibles para que cualquiera los pueda ver y auditar. No existen claves asociadas a ellos y es por este motivo que el creador del contrato no posee ningún control sobre el mismo a menos que esté especificado con anterioridad dentro del el código fuente. La ejecución de un contrato siempre se activa desde una transacción proveniente de una cuenta y aunque pueden hacer llamadas a otros contratos, su creación siempre se activará de esa manera. La ejecución se realiza de principio a fin de acuerdo con lo indicado en el código fuente, pero si ocurriera una falla todos los cambios de estado que se hubieran generado se revierten.

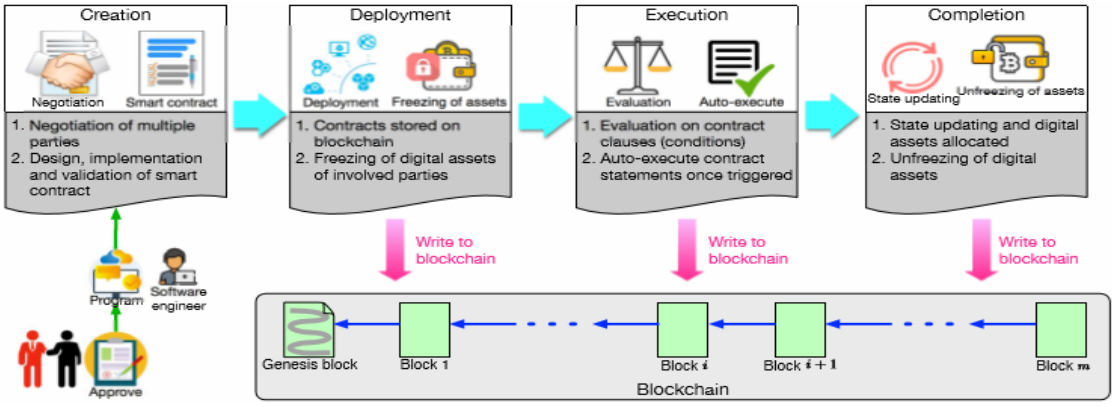


Ilustración 6. Ciclo de vida de un contrato inteligente [24]

2.3. Oráculos

Para que se active la ejecución de una transacción, los contratos deben alimentarse de información que indique el cumplimiento de sus condiciones. Por ejemplo, fechas, eventos deportivos, datos del tiempo, cotizaciones de los mercados, datos estadísticos, números aleatorios, etc. Existen para ello, aplicaciones denominadas oráculos que cumplen con esta función. Los oráculos son un tipo de contrato que obtiene datos desde algún recurso externo a la cadena de bloques, los transfieren hacia su interior y los almacena en un contrato para hacerlos disponibles. De este modo, podrán ser accedidos y utilizados por otros contratos mediante mensajes que lo invoquen.

Si bien los oráculos son necesarios y amplían las posibilidades de uso, a su vez pueden introducir vulnerabilidades. Por ejemplo, se podría falsificar el resultado de un evento deportivo introducido por un oráculo para cobrar una apuesta ganadora. Por otra parte, existen también, oráculos que suministran información de sus propias organizaciones, públicas o privadas, que resultarán confiables o no de acuerdo con la entidad que lo administre.

2.4. Casos de uso

Existen amplias posibilidades respecto a los casos de uso posibles. Los más utilizados hasta este momento son los envíos de criptomonedas como medio de pago, las aplicaciones IoT, certificados y registros públicos, seguros, apuestas, derechos de autor, contratos de alquiler y títulos de propiedad.

A continuación, se describen ejemplos de bancos y compañías de seguros que utilizan contratos inteligentes.

(a) Bancos: el uso de criptomonedas como medio de pago logró romper barreras regulatorias posibilitando transacciones internacionales más rápidas y económicas. Para no quedar fuera de juego, bancos como Barclays y Deustche Bank implementaron sus propias redes de cadenas de bloques que ofrecen servicios a sus clientes. Otros bancos optan por invertir en pequeñas empresas dedicadas a esta tecnología.
(b) Seguros: algunas compañías de seguros aprovechan esta tecnología para verificar la identidad del asegurado y se basan en oráculos para

obtener información en tiempo real acerca de las pólizas y los montos de los contratos. De esta manera se pueden calcular y emitir pagos sobre incidentes sin necesidad de intervención humana. Un ejemplo es la aseguradora francesa AXA que utiliza contratos inteligentes para emitir pagos automatizados a sus clientes en el caso de producirse demoras en los vuelos de más de dos horas.

(c) Dapps: son aplicaciones descentralizadas o semi descentralizadas que se operan desde un navegador con tecnología P2P y guardan su lógica en contratos inteligentes. Las Dapps tienen sus propios casos de uso como la creación de juegos, siendo Criptokitties una de las más populares, finanzas, servicios y otros. La siguiente figura muestra el ejemplo de una Dapp utilizada para realizar subastas.

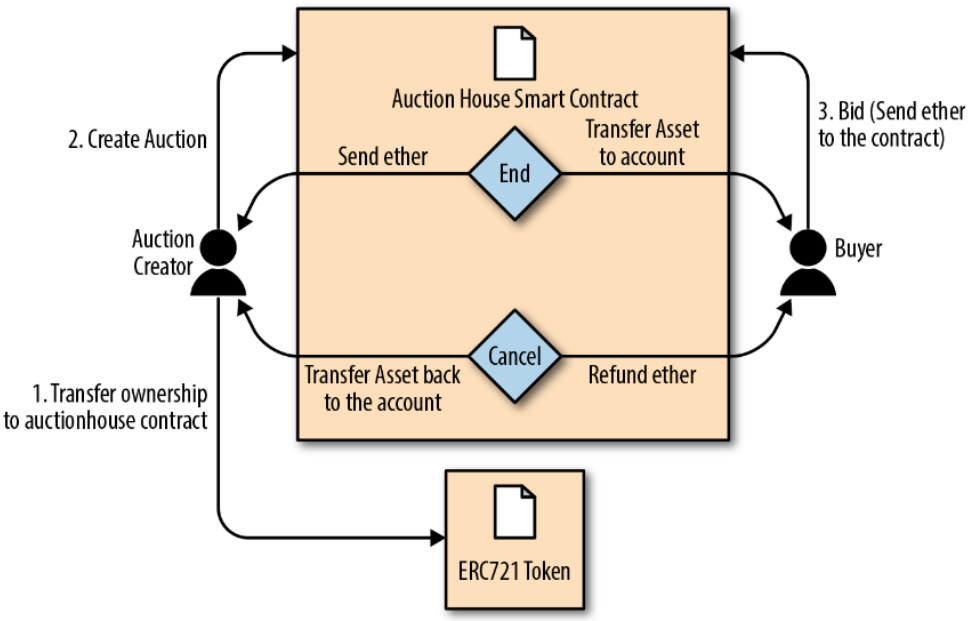


Ilustración 7. Dapp subastas descentralizada [25].

(d) Tokens criptográficos: se puede definir un token como un activo digital de usos múltiples aceptado por una comunidad, que, por estar basado en la tecnología de cadena de bloques, hereda sus características de trazabilidad, seguridad e imposibilidad de falsificación. Existen contratos que permiten la creación rápida de tokens, estandarizando una interfaz para su creación y emisión. Ellos son el ERC-20 y ERC-721 [26].

2.5. Programación de contratos inteligentes

Solidity fue el primer lenguaje de programación utilizado para la creación de contratos inteligentes y hoy en día es el más utilizado. Está inspirado en C++, Python y JavaScript y se compila en instrucciones de bajo nivel para poder ser comprendidas por la MVE [27]. Posee semejanzas con estos lenguajes, lo que puede confundir a los programadores si no tienen en cuenta que, si bien se asemejan, su comportamiento no es el mismo [28].

Esto representa un riesgo importante para la seguridad, ya que muchas veces los resultados obtenidos luego de la ejecución de un contrato no son los esperados, debido a que, una vez registrado el contrato en la cadena de bloques, sus instrucciones pueden sufrir alguna alteración, ya sea de reordenamiento o demoradas. Esta es una diferencia importante respecto a los lenguajes de programación antes mencionados, donde el programador puede tener control casi absoluto sobre el orden en el que se ejecutará cada una de las instrucciones.

Asimismo, todo software es susceptible de presentar vulnerabilidades y los contratos inteligentes no representan una excepción. Si se tiene en cuenta que, además, el código fuente suele ser público y que los contratos gestionan valor económico, estas vulnerabilidades pueden ser aprovechadas por un atacante para su propio beneficio. Por lo que se recomienda que, antes de enviar un contrato para su inserción en la cadena de bloques, realizar un análisis de vulnerabilidades y se apliquen buenas prácticas de desarrollo de software como revisiones de código, pruebas de calidad, auditoría, etc. También se debe considerar a las cadenas de bloques, la compilación sobre la MVE y los canales laterales respecto a la criptografía como vectores de ataque, por formar parte de una tecnología novedosa.

El siguiente contrato [28], muestra a modo de ejemplo, la implementación de una billetera. El contrato puede enviar y recibir ETH desde y hacia otras cuentas mediante la función *pay*, que emplea como parámetros la cantidad de criptomonedas (uint amount) y la dirección del receptor (address recipient). La tabla hash de transacciones registrará todas las

direcciones involucradas en el envío y recepción de ETH y la cantidad de criptomoneda transferida.

Los ETH recibidos se almacenan en el contrato en una variable especial llamada *'balance'*, que representa el saldo y no puede ser alterada por el programador. Junto con la transacción se debe incluir el costo de su ejecución medido en gas, que será utilizado para compensar a los forjadores y no es reembolsable en caso de que la ejecución falle. La función AWallet representa al constructor y es llamada una sola vez al momento de la creación. El contrato además verifica que la cuenta emisora pertenezca al dueño y que posea una cantidad de ETH mayor a los que intenta enviar, ya que, a la transacción se le debe sumar el gas necesario para su ejecución.

```
contract AWallet{
    address owner;
    mapping (address => uint) public outflow;

    function AWallet(){ owner = msg.sender; }

    function pay(uint amount, address recipient) returns (bool){
        if (msg.sender != owner ||  msg.value != 0) throw;
        if (amount > this.balance) return false;
        outflow[recipient] += amount;
        if (!recipient.send(amount)) throw;
        return true;
    }
}
```

Ilustración 8. Contrato representando una billetera [28].

Cada contrato puede contener los siguientes tipos de datos [29]:

(a) Variables de estado: valores almacenados en el contrato de forma permanente.
(b) Funciones: unidades ejecutables de código.
(c) Modificadores de función: incrementan la funcionalidad de las funciones.
(d) Eventos: comunicaciones con la MVE, logs o alertas.
(e) Estructuras: estructuras de datos que almacenan diferentes variables.
(f) Enumeraciones: son utilizadas para crear tipos de datos a medida.

2.6. Descripción general de vulnerabilidades

2.6.1. Función de reentrada

Como se mencionó anteriormente, toda acción en Ethereum se genera mediante transacciones o mensajes (llamadas) disparados desde una cuenta. Las transacciones pueden ser de dos tipos, transferencia de ETH o lanzamiento de contratos. Los contratos además pueden hacer llamados al código de otros contratos [30].

La interacción de un contrato A con otro contrato B y cualquier transferencia de Ether transfiere el control al contrato B. Esto hace posible que B vuelva a llamar a A antes de que se complete la interacción. De esta manera el contrato B puede obtener múltiples reembolsos hasta vaciar el saldo del contrato A. Es importante considerar que un contrato puede modificar el estado de otro contrato.

2.6.2. Desbordamiento de enteros

La MVE fija el tamaño del tipo de dato números enteros en 256 bits (2^{256} - 1). Un desbordamiento ocurre cuando una variable del tipo entero se incrementa por encima de su valor máximo. Si se lo incrementa por 1 bit vuelve a cero.

```
  0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
+ 0x0000000000000000000000000000000000001
----------------------------------------
= 0x0000000000000000000000000000000000000
```

Ilustración 9. Desbordamiento de enteros, límite superior [31].

En el caso contrario, cuando el número no tiene signo, si se resta 1 bit se vuelve a obtener el valor máximo para el tipo de dato, resultando en un desbordamiento. El último tiene más posibilidades de ocurrir ya que le permite al atacante gastar más activos de los que posee.

```
  0x0000000000000000000000000000000000
- 0x0000000000000000000000000000000001
--------------------------------------
= 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
```

Ilustración 10. Desbordamiento de enteros, límite inferior [31].

2.6.3. Nombre del constructor

Los constructores son funciones especiales que realizan tareas como inicializar contratos. En Solidity, hasta antes de la versión v0.4.22, los constructores se definían como una función que poseía el mismo nombre que el contrato donde se encontraba. Si se modificaba el nombre del contrato sin actualizar el nombre de la función, el comportamiento pasaba a ser el de una función común que podía llamarse desde otro contrato [32].

2.6.4. Cantidad de gas insuficiente

Cuando se utiliza la función send para transferir ETH, es posible que falle si no se envía la cantidad suficiente de gas. Si se produce una falla de este estilo se recomienda arrojar una excepción. También es importante crear funciones que no demanden grandes cantidades de gas, ya que además de producir fallas se pueden volver costosas [23].

2.6.5. Falsa entropía

Las transacciones sobre la cadena de bloques de Ethereum son completamente deterministas, lo que significa que el código ejecutado con los mismos datos de entrada debe producir la misma salida en todos los nodos que lo ejecutan. Por este motivo, se puede decir que no existe ninguna fuente de aleatoriedad o entropía. Esto representa un problema para ciertas aplicaciones que requieren aleatoriedad como en el caso de juegos de azar [32].

Para generar aleatoriedad, algunos contratos utilizan un generador inicializado con la misma semilla por todos los forjadores. Generalmente, se suele utilizar como semilla el hash de algún bloque futuro. Esta podría ser una buena opción, pero dado que los forjadores pueden elegir las transacciones que van a insertar en el siguiente bloque, podrían ponerse de acuerdo y alterar el siguiente generador [29].

2.6.6. Tx.origin

Tx.origin posiciona la dirección del atacante como propietario del contrato. De esta manera puede apropiarse de todo el saldo.

2.6.7. Dependencia de timestamp

Si el contrato utiliza funciones como now, StartTime, EndTime, basadas en el timestamp del bloque, un atacante puede modificar el timestamp por unos segundos y obtener un resultado que lo favorezca.

2.6.8. Race conditions

Ocurre cuando algunos eventos en el Sistema no se generan en el orden esperado. En el caso de los contratos inteligentes, puede ocurrir cuando contratos externos toman el control de la ejecución [33].

2.6.9. Ataque del 51% o doble gasto

El ataque del 51% es una vulnerabilidad de las cadenas de bloques que utilizan como algoritmo de consenso prueba de trabajo (proof of work, PoW). Si bien este no es el caso de Ethereum, pero sí de Ethereum Classic.

Si un grupo de mineros obtuvieran el control de más del 50% del poder computacional de la red, podrían hacer que las nuevas transacciones no se confirmen o frenar algunos pagos. También podrían revertir las transacciones realizadas mientras tuvieron el control de la red y de esta manera producir el doble gasto, es decir volver a utilizar la misma moneda [34].

3. Ataques a Contratos Inteligentes

3.1. Ataque al proyecto DAO

Una Organización Autónoma Descentralizada (del inglés *Decentralized Autonomous Organization*) es un modelo empresarial basado en cadena de bloques cuyas reglas son impuestas por la lógica de un contrato inteligente. Se podría trazar un paralelo y compararlo con un fondo de inversión donde los inversores tienen derecho a voto y a decidir sobre futuros contratos donde realizar inversiones.

Para mayo de 2016, el contrato 'El DAO', creado para realizar alquileres temporarios inteligentes de propiedades, había recaudado USD $150.000.000 antes de sufrir un ataque ante la vista de sus 11.000 socios. El ataque ocurrió el 17 de junio de 2016 y se sustrajeron ETH 3.641.694, el equivalente en ese momento a USD $50.000.000, mediante diferentes transacciones anónimas. La estafa pudo concretarse debido a un error en la codificación que posibilitaba la desviación de fondos a una DAO hija, que copiaba la estructura de la DAO original y en cada una de esas duplicaciones realizaba una transferencia de criptomonedas [35]. Tal es así que el atacante publicó una carta declarando no haber cometido ningún delito y que lo que había hecho estaba permitido por las reglas del contrato [36].

En medio de una gran controversia para revertir los efectos del ataque, Ethereum se dividió en dos ramas: la rama principal de Ethereum (ETH) y la rama Ethereum Classic (ETC). A este hecho se lo conoce como '*The Hard Fork',* lo que sería una división permanente de la cadena. De esa manera se logró deshacer las transacciones fraudulentas y devolver los ETH robados a sus respectivos dueños [37]. El hecho no escapó de las críticas sobre la filosofía de Ethereum, la inmutabilidad de una cadena de bloques y las dudas generadas acerca de su seguridad. Ya que la decisión de bifurcar la red fue tomada por los líderes de Ethereum junto a algunos inversores, lo que implica que hubo una intervención humana en la toma de decisiones sobre una tecnología que se jacta justamente de lo contrario, ser autónoma y no estar controlada por ninguna autoridad [38].

3.1.1. Detalles técnicos del ataque

El sistema de inversión DAO funciona de la siguiente manera. El inversor que quiera participar debe adquirir tokens que le dan derecho a realizar propuestas sobre futuras inversiones. Cuando una propuesta es votada por la mayoría, la inversión de los accionistas se transfiere a la cuenta del ideólogo para poder concretar su proyecto. Al resto de los participantes que no apoyaron la inversión, se les devuelve su dinero mediante la creación de nuevos contratos.

La transferencia de criptomonedas se implementa por medio de la función splitDAO(), que fue explotada a causa de una vulnerabilidad de reentrada producida por un error de programación [16]. La siguiente figura, explica paso a paso cómo se produjo el ataque: un inversor minoritario reclama su reembolso.

(a) El contrato DAO crea un nuevo contrato DAO hijo y transfiere la devolución de fondos a la cuenta del nuevo contrato. Este inversor minoritario, además, recibe una recompensa por su participación.
(b) La vulnerabilidad reside en el número de tokens solicitados por este accionista que depende de las variables de estado balances[msg.sender] y totalSupply, que se actualizan al final de la función splitDAO(), después de la llamada externa msg.sender.call.value().
(c) Esto le permite al atacante llamar a la función splitDAO() de forma recursiva antes que la variable de estado que controla la cantidad de tokens se actualice.

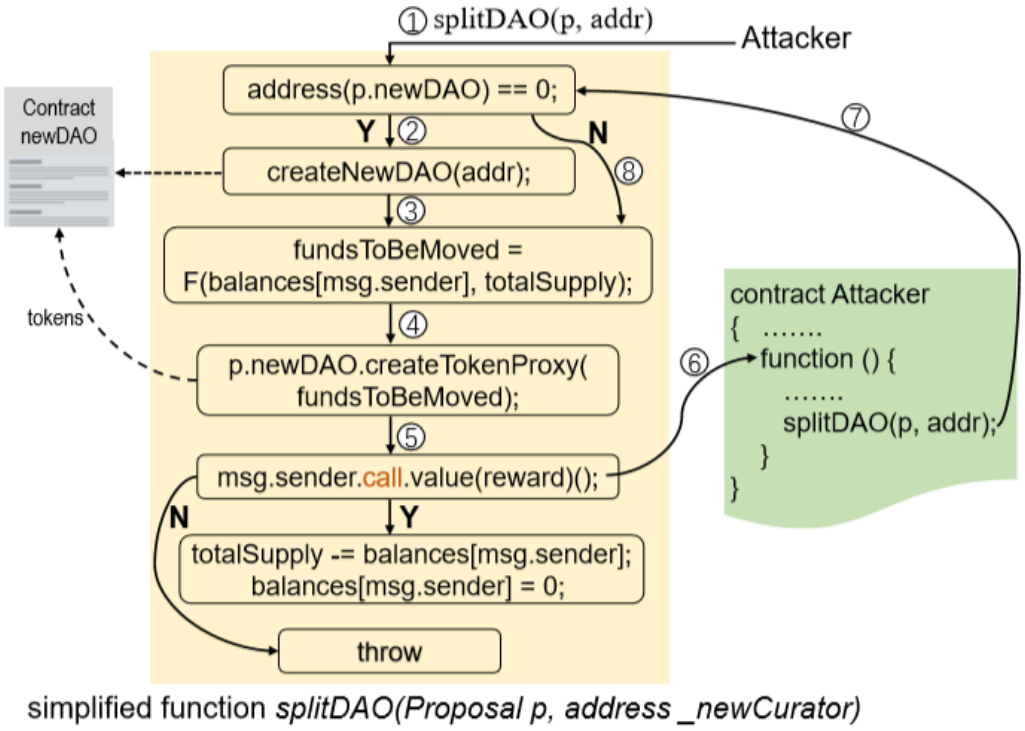


Ilustración 11. Ataque al DAO que explota la vulnerabilidad de reentrada [16].

3.1.2. Métodos de prevención

Se observan dos vulnerabilidades en el contrato. La primera es que el estado del contrato se actualiza después de haber enviado los fondos y no antes. La segunda es la utilización del método `sender.call.value()` para transferir fondos. En su lugar debería usarse `address.transfer()` o `address.send()`, que tienen un gasto de gas limitado y pueden evitar la recursividad [39].

3.2. Beauty Chain (BEC)

Beauty Chain es una plataforma que utiliza tecnología de cadena de bloques para generar contenido sobre temas de belleza y cuidado personal, donde los participantes asumen diferentes roles en la creación, difusión, desarrollo y operaciones. De acuerdo con cada rol el participante obtiene recompensas en forma de BEC [40].

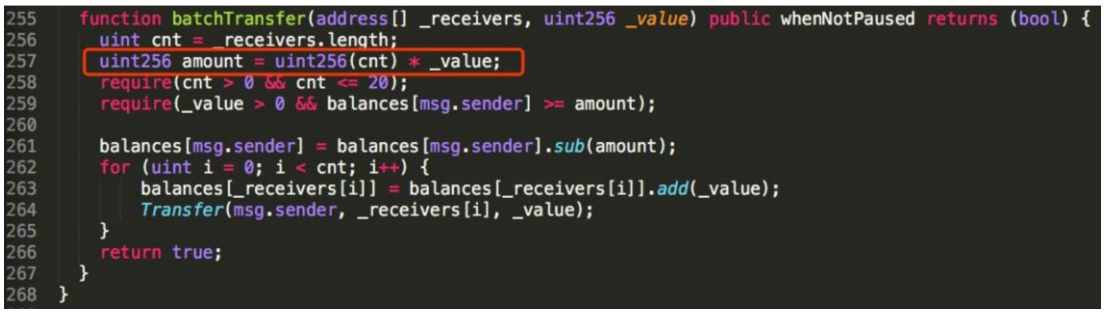
El 22 de abril de 2018, un equipo de expertos de la firma de seguridad PeckShield Inc, detectó a través de su sistema de alertas automatizado, una transacción inusual de tokens BEC. Se trataba de dos transferencias idénticas desde el contrato BeautyChain hacia dos direcciones diferentes. El ataque explotó una vulnerabilidad desconocida hasta ese momento que los expertos llamaron *batch overflow*, basada en desbordamiento de enteros. La vulnerabilidad permite que los atacantes logren un aumento no autorizado de los activos digitales proporcionando dos argumentos `_receivers` junto con un argumento `_value` [41]. Pero BeautyChain no era el único contrato vulnerable, el equipo de PeckShiels Inc, detectó más de una docena de contratos basados en ERC20[6] que presentaban esta vulnerabilidad [42].

3.2.1. Detalles técnicos del ataque

La vulnerabilidad se encuentra en la función *batchTransfer.* Mediante un desbordamiento de enteros el atacante consigue saltear las validaciones de saldo y realizar una transferencia por grandes sumas.

(a) La variable local `amount` es el producto de `cnt` y `_value`. En este caso `value` puede ser un entero cualquiera de 256 bits.
(b) Al pasar un número muy grande a la variable `_val,` cuando se calcula `amount se` puede producir un desbordamiento dando como resultado 0.
(c) Con la variable `amount` igual a 0, el atacante puede saltearse las validaciones de saldo que se requiere para poder realizar la transferencia.
(d) Al finalizar las sentencias de la función el atacante puede enviar transferencias con grandes sumas sin ninguna validación previa.

[6] https://fin.guru/educacion/201706/-que-son-los-tokens-erc20-de-ethereum-y-como-funcionan-_n2207



```
255   function batchTransfer(address[] _receivers, uint256 _value) public whenNotPaused returns (bool) {
256     uint cnt = _receivers.length;
257     uint256 amount = uint256(cnt) * _value;
258     require(cnt > 0 && cnt <= 20);
259     require(_value > 0 && balances[msg.sender] >= amount);
260
261     balances[msg.sender] = balances[msg.sender].sub(amount);
262     for (uint i = 0; i < cnt; i++) {
263         balances[_receivers[i]] = balances[_receivers[i]].add(_value);
264         Transfer(msg.sender, _receivers[i], _value);
265     }
266     return true;
267   }
268 }
```

Ilustración 12. Función vulnerable a desbordamiento de enteros [42].

3.2.2. Métodos de prevención

Esta vulnerabilidad se puede prevenir utilizando bibliotecas matemáticas para remplazar operaciones de suma, resta y multiplicación. Por ejemplo, la biblioteca conocida SafeMath.

3.3. Rubixi

Se trata de un contrato que implementa un esquema Ponzi[7], un programa de inversión fraudulento que utiliza la modalidad piramidal, donde los participantes fundacionales obtienen dinero por medio de las inversiones realizadas por los nuevos participantes. Además, su propietario podía cobrar tarifas relacionadas con esas nuevas inversiones.

El problema de Rubixi fue haber cambiado su nombre original de DynamicPyramid a Rubixi y olvidarse de cambiar el nombre del constructor. La importancia del constructor es que se ejecuta por única vez en el momento de la creación del contrato. Pero el cambio de nombre provocó que el constructor actúe como una función invocable desde cualquier otro contrato. Debido a la característica de inmutabilidad de los contratos inteligentes, el error no pudo solucionarse.

3.3.1. Detalles técnicos del ataque

La función DynamicPyramid establece la dirección del propietario, quien puede retirar en su beneficio a través de la variable collectAllFees.

[7] https://es.wikipedia.org/wiki/Esquema_Ponzi

Después de que el error se hiciera público, los usuarios comenzaron a invocar DynamicPyramid para convertirse en el propietario y retirar los fondos [28].

Esto fue posible debido a que al momento de la creación de Rubixi, Solidity iba por su versión v0.4.24 y no verificaba que el constructor tuviera el mismo nombre que el contrato. A causa de esta inconsistencia, cualquiera desde un contrato externo podía actuar como propietario y realizar transferencias hacia su propia cuenta.

```
5  contract Rubixi {
6
7          //Declare variables for storage critical to contract
8          uint private balance = 0;
9          uint private collectedFees = 0;
10         uint private feePercent = 10;
11         uint private pyramidMultiplier = 300;
12         uint private payoutOrder = 0;
13
14         address private creator;
15
16         //Sets creator
17         function DynamicPyramid() {
18                 creator = msg.sender;
19         }
20
21         modifier onlyowner {
22                 if (msg.sender == creator) _
23         }
24
25         struct Participant {
26                 address etherAddress;
27                 uint payout;
28         }
29
30         Participant[] private participants;
31
32         //Fallback function
33         function() {
34                 init();
35         }
```

Ilustración 13. Contrato Rubixi [43].

3.3.2. Métodos de prevención

A partir de Solidity v0.4.22, el constructor se define directamente por medio de un constructor propiamente declarado y no por una función con el nombre del contrato. Por este motivo se recomienda utilizar siempre la última versión de este lenguaje.

3.4. Rey del ether

Rey del ether era un juego que prometía convertir al jugador en rey o reina, inmortalizar su nombre en una lista de ‘Salón de monarcas’ y otorgarle riquezas. Para eso, los participantes debían cumplir una serie de reglas.

El participante B debía enviar ETH 10 al contrato, de esa manera se lo proclamaba rey y pasaba a integrar el salón de la fama. Siguientemente, el contrato enviaría esos ETH 10 menos una comisión al monarca anterior, participante A, destronado por B. En ese momento, el precio del trono subiría un 33% y un nuevo participante C pagaría ETH 13,3. Así el participante B se quedaría con esos ETH 13,3 menos la comisión.

La rueda continuaba hasta que el trono alcance un valor máximo de ETH 100000 [44]. El método de recaudación parecía sencillo, pero se presentaba una condición. Si al cabo de 14 días no se presentaba un sucesor al trono, el actual rey finalizaba su reinado sin obtener ninguna compensación.

El problema se suscitó entre el 6 y el 8 de febrero de 2016 a causa de una vulnerabilidad en el contrato que causó que algunos pagos no se realicen o lo hagan con errores. La vulnerabilidad se presentaba cuando Rey del ether enviaba pagos a contratos, ya que por un descuido solo se enviaba una pequeña cantidad de gas para su ejecución, que no era suficiente para procesar el pago.

Al no enviarse suficiente gas, la transacción fallaba y se devolvía el pago al contrato Rey del ether. El contrato no advertía esta devolución y entregaba el trono al próximo monarca pensando que la transacción había sido exitosa. De esta manera el rey predecesor no recibía recompensa alguna luego ceder su trono.

3.4.1. Detalles técnicos del ataque

Cuando un jugador envía ETH mediante `msg.value` se dispara también la función de reentrada del contrato KotET. Esta función verifica que el ETH enviado sea suficiente para reclamar el trono, de lo contrario arroja una excepción y revierte la transacción. Si el ETH es suficiente, el jugador se convierte en el nuevo rey y el contrato recibe su comisión a la que accede mediante la función `sweepCommission`.

Este contrato presenta dos vulnerabilidades. La primera es no verificar el código resultante de `send.` La segunda es enviar una cantidad insuficiente

de gas sin arrojar excepciones, lo que permite al contrato KotET apropiarse de la recompensa que le correspondería al rey saliente [28].

```
contract KotET {
  address public king;
  uint public claimPrice = 100;
  address owner;

  function KotET() {
    owner = msg.sender;  king = msg.sender;
  }

  function sweepCommission(uint amount)  {
    owner.send(amount);
  }

  function() {
    if (msg.value < claimPrice) throw;

    uint compensation = calculateCompensation();
    king.send(compensation);
    king = msg.sender;
    claimPrice = calculateNewPrice();
  }
  /* other functions below */
}
```

Ilustración 14. KotET simplificado [28].

3.4.2. Métodos de prevención

Utilizar `transfer` en lugar de `send`, ya que `transfer` tiene la capacidad de revertir una transacción. Si se utiliza `send` asegurarse de verificar su valor de retorno.

4. Desafíos legales y regulatorios

4.1. Problemas de los Algoritmos de Consenso

Los contratos inteligentes se autoejecutan en una plataforma descentralizada sin intermediarios y el único método de verificación es el consenso entre los nodos participantes de la red. Esto presenta diversos desafíos, por empezar, la falta de un marco regulatorio común.

En el caso de Ethereum y el algoritmo de PoS, los forjadores que quieran participar en la creación de un bloque deben depositar una cantidad de criptomonedas en garantía. El ganador de la partida recibe como recompensa el equivalente al valor del costo de las transacciones [45]. Este algoritmo emplea métodos adicionales para que la competencia entre los nodos sea más justa, como aleatoriedad y edad de las monedas. No obstante, si bien el riesgo de un ataque del 51% es mucho menos probable, quienes aporten la mayor cantidad de criptomonedas en garantía podrían tener alguna ventaja sobre el resto.

Los casos mencionados se refieren a las redes públicas de Ethereum y Ethereum Classic, pero también existen redes privadas, gestionadas por un administrador u organización, quienes intentarán aplicar un mecanismo de consenso que se adapte a los objetivos organizacionales. Uno de los interrogantes que se presentan entonces, sería quiénes tendrán el control sobre estas redes. Entre la diversidad de actores que intervienen están los usuarios, inversores, forjadores, dueños de la tecnología, programadores y proveedores de servicios entre otros [46].

4.2. Gobierno de TI

El siguiente desafío sería analizar qué normas se pueden o deben aplicar, ya que no existen aún normas aceptadas universalmente. Una red de bloques funciona en base a tecnología y protocolos sobre una red de comunicaciones. Estos componentes presentan sus propias características, las que a su vez adicionan riesgos y requieren sus propias reglas de gobierno.

Algunos ejemplos de interrogantes pueden ser, por ejemplo, quiénes deben tener acceso a la red, cuáles serán los procesos de validación de transacciones o de retiro de fondos, cómo se protege la privacidad de los participantes, la gestión de datos personales, la aplicación de políticas de prevención de lavado de dinero, KYC (conoce a tu cliente), etc.

4.3. Riesgo de TI

Los riesgos deben ser gestionados, comprendidos y mitigados para poder sacar provecho de la tecnología [47]. Como se observó, existe el riesgo de que se presenten fallas tecnológicas o de introducir vulnerabilidades por la mala codificación de un contrato o de sufrir ciberataques. Por lo tanto, se requiere aplicar controles eficientes y testeos exhaustivos como salvaguardas.

Puede darse el caso de que la información que llega a la red proveniente del exterior por medio de oráculos contenga software malicioso, diseñado con el fin de perpetrar ataques y causar modificaciones fraudulentas en los datos dentro de la red.

La pérdida de la clave privada es otro factor de riesgo, ya que es la única manera que el usuario pueda demostrar la titularidad de sus cuentas y los activos depositados. El usuario que pierda su clave privada no podrá demostrar la titularidad de una cuenta y perderá todos sus depósitos. Cabe aclarar que la situación descripta no es privativa de Ethereum ya que lo mismo sucede con todos los métodos de firma digital.

El protocolo de consenso también puede presentar vulnerabilidades. Por ejemplo, el ataque del 51% o doble gasto podría presentarse en redes cuyo consenso esté basado en PoW como Ethereum Classic. Así, si un minero posee el 51% de los nodos, ganaría el control de toda la red interfiriendo en la selección de inserción de nuevos bloques. Los mineros maliciosos podrían producir una reorganización de la cadena y originar doble gasto, mecanismo por el cual una moneda puede gastarse en más de una ocasión. Este tipo de ataque ocurrió a las redes Krypton y Shift, ambas basadas en Ethereum Classic [34].

Además, existen riesgos operacionales relacionados a la escalabilidad e interoperabilidad con otras tecnologías y riesgos de fallas causadas por mal desarrollo del código.

4.4. Legislación

Los contratos inteligentes se encuentran afectados por todos los interrogantes planteados más algunos propios. El proceso de creación y ejecución puede ser ágil y económico, pero de difícil comprensión para el usuario común [48], quien deberá depender de un experto informático, ya sea para diseñarlo o para certificar contratos desarrollados por terceros.

Otro inconveniente puede ser el grado de responsabilidad que le corresponderá al programador si comete un error por imprudencia, negligencia o impericia y que esa acción perjudicase a alguna de las partes [49]. Asimismo, es necesario considerar si las leyes existentes son aplicables o no en el caso que falle la ejecución automática y la obligación se deba resolver por fuera de la red.

La falta de regulaciones podría llevar a desarrollar contratos con el objetivo de cometer hechos delictivos. Pero la inexistencia de controles y la posibilidad técnica de hacerlo lo permiten. Esto que no debe tomarse como una invitación a no cumplir, ya que, de lo contrario, las personas humanas u organizaciones podrían sufrir sanciones por incumplimientos legales o regulatorios y afrontar pérdidas económicas o de imagen y hasta sanciones penales. Por eso, cada organización debe seguir las mismas normas que aplica para el resto de sus operaciones, respetando las leyes y normativas que le son propias a su industria o negocio. En el caso de las personas humanas, deben recordar en todo momento, que cualquier infracción cometida en el mundo virtual puede tener su consecuencia en el mundo real.

Finalmente, podrían sobrevenir cambios en las condiciones en el mundo real que no puedan ser volcadas en el contrato. Como ya se dijo, los contratos son inmutables, lo que implica que las cláusulas no se pueden modificar. Por este motivo, tampoco se podrían rescindir.

Conclusiones

Además de poseer la criptomoneda Ether, una de las principales innovaciones que aporta Ethereum es la posibilidad de crear, compilar y ejecutar programas sobre una máquina virtual. Estos programas, conocidos como contratos inteligentes, se autoejecutan de manera descentralizada, sin ninguna intervención por parte de terceros, según las condiciones insertadas su código fuente.

Toda ejecución de un contrato se genera a través de una transacción disparadora y por medio de esa ejecución se envían o reciben Ethers entre contratos o desde y hacia cuentas. Como resultado de esa ejecución, se produce un cambio de estado en la red que se refleja en todos sus nodos.

En el capítulo 1 se explica cómo Ethereum utiliza criptografía de clave pública, es decir, que utiliza una clave privada y una clave pública para la creación de cuentas y direcciones, así como también emplea firma digital para validar contratos o insertar nuevas transacciones por medio del algoritmo de curva elíptica ECDSA. Mediante la clave pública se identifica la dirección de Ethereum del usuario, mientras que por medio de la clave privada se controlan los fondos en las cuentas. Por este motivo debe mantenerse en secreto y a resguardo ya que si esto no sucede se corre el riesgo de perder todos los valores depositados en esa cuenta.

El usuario interactúa con la red y gestiona sus cuentas por medio de billeteras, estas además proporcionan mecanismos para el almacenamiento y gestión de claves. Las billeteras son simplemente programas de software creados para este fin y según el método de generación de claves que utilice se lo denomina determinista, todas las claves se generan a partir de la misma clave, o no determinista, cada generación de clave se genera de manera independiente.

La transferencia de valores entre cuentas y la creación y ejecución de contratos siempre se ejecutan a través de transacciones. Para que estas transacciones puedan realizarse, el usuario debe incluir un importe denominado gas que se calcula de acuerdo con el gasto computacional que ocasionará la ejecución de esa transacción. De esta manera se evitan gastos

computacionales innecesarios en la red, ataques de denegación de servicio y se recompensa a los forjadores.

El registro de hashes de un grupo de transacciones forma un bloque, que es el resultado de la combinación de los hashes de todas las transacciones tomadas de a pares, hasta que queden representadas por un único hash. Este sistema de hashes es lo que asegura la inmutabilidad de las transacciones.

La selección del próximo bloque a insertar en la cadena se realiza por mecanismos de consenso. Los dos más utilizados son prueba de trabajo, que es aún utilizada por Ethereum Classic y prueba de participación, utilizada por Ethereum. La primera consume grandes cantidades de energía y la recompensa es un valor fijo mientras en la segunda, los usuarios arriesgan sus propios activos para poder minar o forjar bloques y la recompensa es un valor relativo representado por la cantidad de gas pagado por los usuarios que desean insertar esas transacciones.

El capítulo 2 nos introduce en más detalle a los contratos inteligentes y las debilidades que pueden presentar. Allí se explica cómo las similitudes entre el lenguaje de programación Solidity y otros lenguajes más tradicionales suelen confundir a los desarrolladores, principalmente por la dificultad de poder predecir el orden de ejecución de las sentencias. Por este motivo, el desarrollador debe tener especial cuidado y en lo posible utilizar código ya testeado y validado.

Sumado a esto, no se debe pasar por alto que la obtención de datos desde el exterior de la red se realiza por medio de oráculos y estos no deberían ser considerados como único medio de alimentación hacia la red. Sería imperioso, ante la necesidad de contar con fuentes de información externas, que sean variadas y confiables.

En el mismo capítulo se describen algunos posibles casos de uso. Aunque después de realizar un recorrido por algunas vulnerabilidades existentes no se puede generar con certeza que los mismos puedan ser aplicados a sistemas críticos o de gran escala, sino que más bien se observa que la tecnología está en plena etapa de prueba.

El capítulo 3 presenta los principales ataques a la seguridad de contratos inteligentes. Se detalla el momento histórico, la descripción técnica de la vulnerabilidad atacada y las cantidades de criptomonedas robadas. Además, nos muestra algunas contradicciones. Primero el ataque al DAO, que produjo la bifurcación de la red entre Ethereum Classic y Ethereum hizo que se rompiera con la principal filosofía de los creadores de la red que dice "…sin la intervención de terceros…", ya que la decisión de llevar adelante la bifurcación de la red para recuperar criptomonedas robadas fue tomada por un grupo de usuarios influyentes. Segundo, que las transacciones pueden no ser inmutables si un grupo de usuarios con poder deciden lo contrario. Finalmente, como se puede ver en el contrato Rey del Ether, puede haber usuarios malintencionados que crean contratos únicamente con el propósito de estafar a otros.

En el capítulo 4 se enumeran algunos riesgos que podrían presentarse tanto en redes públicas como privadas respecto a la implementación y utilización de contratos inteligentes. Estos riesgos incluyen cuál será el algoritmo de consenso que conviene utilizar y quién lo determina, quiénes tendrán el control sobre la red y quiénes tendrán el derecho o no a ser parte de la misma. Asimismo, no se debe pasar por alto todos los riesgos generales de IT, como riesgos operacionales o de integración con otras tecnologías como los propios ya mencionados.

Respecto a la legalidad de estos contratos, queda mucho camino por recorrer y la realidad es que si bien su nombre nos hace pensar que estamos frente a documentos que contienen cláusulas exigibles por la ley, la realidad es que se trata de programas de computación cuyo cumplimiento debería realizarse de manera digital y automática. No obstante, de acuerdo con la jurisdicción en la que nos encontremos, cualquier incumplimiento de una parte que se presente dentro de la red podría ser exigido por la contraria en el mundo real ente un tribunal de justicia y obtener su reparación de ese modo.

Bibliografía

[1] «Ethereum,» BitcoinWiki, [En línea]. Available: https://es.bitcoinwiki.org/wiki/Ethereum. [Último acceso: 28 05 2020].

[2] Antonopoulos A., Wood G., «Mastering Ethereum,» [En línea]. Available: https://github.com/ethereumbook/ethereumbook/blob/develop/01what-is.asciidoc. [Último acceso: 28 05 2020].

[3] D. Won, «Exodus Bog,» 21 02 2020. [En línea]. Available: https://www.exodus.io/blog/ethereum-proof-of-stake-date/. [Último acceso: 28 05 2020].

[4] H. Kenneth, «Medium,» 07 05 2018. [En línea]. Available: https://medium.com/coinmonks/ethereum-account-212feb9c4154. [Último acceso: 30 05 2020].

[5] Ethereum community, «Ethdocs,» 2016. [En línea]. Available: https://ethdocs.org/en/latest/contracts-and-transactions/account-types-gas-and-transactions.html. [Último acceso: 30 05 2020].

[6] Galal, H. S., & Youssef, A. M., «Trustee: full privacy preserving vickrey auction on top of Ethereum. In International Conference on Financial Cryptography and Data Security (pp. 190-207). Springer, Cham.,» 02 2019. [En línea]. [Último acceso: 23 11 2020].

[7] Antonopoulos A., Wood G.,, «Mastering Ethereum,» [En línea]. Available: https://github.com/ethereumbook/ethereumbook/blob/develop/04keys-addresses.asciidoc. [Último acceso: 29 05 2020].

[8] M. Musumeci, «Massmux,» 26 12 2018. [En línea]. Available: https://www.massmux.com/private-and-public-keys-on-ethereum/. [Último acceso: 07 06 2020].

[9] G. Wood, «Ethereum: A secure decentralized generalised transaction ledger,» [En línea]. Available: https://ethereum.github.io/yellowpaper/paper.pdf. [Último acceso: 30 05 2020].

[10] P. Hecht, «Curvas Elípticas. Criptografía 2. Maestría en Seguridad Informática.,» Universidad de Buenos Aires, 2020.

[11] J. D. Cook, «John D Cook Consulting,» 14 08 2018. [En línea]. Available: https://www.johndcook.com/blog/2018/08/14/bitcoin-elliptic-curves/. [Último acceso: 07 06 2020].

[12] Antonopoulos,A., Wood, G., «Mastering Ethereum,» [En línea]. Available: https://github.com/ethereumbook/ethereumbook/blob/develop/05wallets.asciidoc. [Último acceso: 07 06 2020].

[13] A. Rosic, «What is Ethereum Gas? The Most Comprehensive Step-By-Step Guide Ever!,» [En línea]. Available: https://blockgeeks.com/guides/ethereum-gas/#:~:text=What%20is%20Ethereum%20Gas%3F%20TL%3BDR%20Ethereum%20Gas%20is,it%20took%20them%20to%20execute%20a%20complete%20. [Último acceso: 12 10 2020].

[14] Antonopoulos, A., Wood, G., «Mastering Ethereum,» [En línea]. Available: https://github.com/ethereumbook/ethereumbook/blob/develop/02intro.asciidoc. [Último acceso: 12 10 2020].

[15] «PegaaSys,» 15 01 2019. [En línea]. Available: https://kauri.io/ethereum-explained-merkle-trees-world-state-transa/1f4196c3db7f41e5845f063dc1581a4e/a. [Último acceso: 07 06 2020].

[16] Huashan Chen, Marcus Pendleton, Laurent Njilla, Shouhuai Xu, «A Survey on Ethereum Systems Security: Vulnerabilities, Attacks and Defenses,» 13 08 2019. [En línea]. Available: https://arxiv.org/abs/1908.04507. [Último acceso: 27 06 2020].

[17] A. Lewis, «Bits on Blocks,» 2015. [En línea]. Available: https://bitsonblocks.net/2016/10/02/gentle-introduction-ethereum/. [Último acceso: 09 08 2020].

[18] R. Modi, Solidity Programming Essentials, Birmingham: Packt Publishing, 2018.

[19] L. Saldanha, «Merkle Tree and Ethereum Objects,» 11 12 2018. [En línea]. Available: https://www.lucassaldanha.com/ethereum-yellow-paper-walkthrough-2/. [Último acceso: 09 08 2020].

[20] Antonopoulos, A., Wood, G., «Mastering Ethereum,» [En línea]. Available: https://github.com/ethereumbook/ethereumbook/blob/develop/14consensus.asciidoc. [Último acceso: 27 06 2020].

[21] Binance Academy, «¿Qué es la Prueba de Estaca (Proof of Stake)?,» [En línea]. Available: https://academy.binance.com/es/articles/proof-of-stake-explained. [Último acceso: 12 10 2020].

[22] E. Rojas, «¿Qué son los smart contracts o contratos inteligentes?,» 20 01 2020. [En línea]. Available: https://es.cointelegraph.com/explained/what-is-a-smart-contract. [Último acceso: 27 06 2020].

[23] A. Dika, «Ethereum Smart Contracts: Security Vulnerabilities and Security Tools,» 12 2017. [En línea]. Available: https://pdfs.semanticscholar.org/dd6b/1c81c3d8343faf651741ebf778b0eb7cf

d53.pdf?_ga=2.180855463.1668582251.1593270460-675627698.1590796904. [Último acceso: 27 06 2020].

[24] Z. Zheng, Shaoan Xie, Muhammad Imran, «An Overview on Smart Contracts: Challenges, Advances and Platforms,» 22 12 2019. [En línea]. Available: https://arxiv.org/pdf/1912.10370.pdf. [Último acceso: 06 07 2020].

[25] Antonopoulos A., Wood G.,, «Mastering Ethereum,» [En línea]. Available: https://github.com/ethereumbook/ethereumbook/blob/develop/12dapps.asciidoc. [Último acceso: 09 07 2020].

[26] «Qué es un token ERC-20,» [En línea]. Available: https://academy.bit2me.com/que-es-erc-20-token/. [Último acceso: 25 07 2020].

[27] Ethereum , «Solidity,» [En línea]. Available: https://solidity.readthedocs.io/en/v0.4.24/. [Último acceso: 28 06 2020].

[28] Nicola Atzei, Massimo Bartoletti, Tiziana Cimoli, «A survey of attacks on Ethereum smart contracts,» 18 03 2017. [En línea]. Available: https://eprint.iacr.org/2016/1007.pdf. [Último acceso: 28 06 2020].

[29] López Vivar, A., Turégano Castedo, A., et al., «An Analysis of Smart Contracts Security Threats Alongside Existing Solutions,» 11 02 2020. [En línea]. Available: https://www.mdpi.com/1099-4300/22/2/203/htm. [Último acceso: 01 08 2020].

[30] M. P. G. Gelvez, «Explaining the DAO exploit for beginners in Solidity,» 16 10 2016. [En línea]. Available: https://medium.com/@MyPaoG/explaining-the-dao-exploit-for-beginners-in-solidity-80ee84f0d470. [Último acceso: 10 07 2020].

[31] G. Konstantopoulos, «How to Secure Your Smart Contracts: 6 Solidity Vulnerabilities and how to avoid them,» 08 01 2018. [En línea]. Available: https://medium.com/loom-network/how-to-secure-your-smart-contracts-6-solidity-vulnerabilities-and-how-to-avoid-them-part-1-c33048d4d17d. [Último acceso: 18 07 2020].

[32] Antonopoulos, A., Wood, G., «Mastering Ethereum,» [En línea]. Available: https://github.com/ethereumbook/ethereumbook/blob/develop/09smart-contracts-security.asciidoc. [Último acceso: 27 07 2020].

[33] Blockgeeks, «What Kinds of Smart Contract Attacks Are There?,» [En línea]. Available: https://blockgeeks.com/guides/audit-smart-contract/. [Último acceso: 09 08 2020].

[34] J. Frankenfield, «51% Attack,» 06 05 2019. [En línea]. Available: https://www.investopedia.com/terms/1/51-attack.asp. [Último acceso: 25 11 2020].

[35] «¿Qué es DAO?,» [En línea]. Available: https://es.cointelegraph.com/ethereum-for-beginners/what-is-dao. [Último acceso: 09 07 2020].

[36] ". Attacker", «An open letter. Carta pública del atacante del DAO,» [En línea]. Available: https://pastebin.com/CcGUBgDG. [Último acceso: 09 07 2020].

[37] A. Lucian, «La comunidad de Ethereum no está de acuerdo con los 'hard forks' para revertir los ataques,» 31 10 2019. [En línea]. Available: https://es.beincrypto.com/la-comunidad-de-ethereum-no-esta-de-acuerdo-con-los-hard-forks-para-revertir-los-ataques/. [Último acceso: 09 07 2020].

[38] T. Bossomaier, S. D'Alessandro, R. Bradbury, «The Hard Fork,» de *Human Dimensions of Cybersecurity*, Boca Raton FL., CRC Press, pp. 432-434.

[39] P. Humiston, «Smart Contract Attacks,» 13 05 2018. [En línea]. Available: https://hackernoon.com/smart-contract-attacks-part-1-3-attacks-we-should-all-learn-from-the-dao-909ae4483f0a. [Último acceso: 18 07 2020].

[40] «BEC,» [En línea]. Available: http://www.beauty.io/. [Último acceso: 20 07 2020].

[41] «Vulnerabilidad en la función batchTransfer en Beauty Ecosystem Coin,» 29 08 2018. [En línea]. Available: https://www.incibe-cert.es/alerta-temprana/vulnerabilidades/cve-2018-10299. [Último acceso: 25 07 2020].

[42] PeckShield, «New batchOverflow Bug in Multiple ERC20 Smart Contracts (CVE-2018–10299),» 22 04 2018. [En línea]. Available: https://medium.com/@peckshield/alert-new-batchoverflow-bug-in-multiple-erc20-smart-contracts-cve-2018-10299-511067db6536. [Último acceso: 20 07 2020].

[43] «Etherscan,» [En línea]. Available: https://etherscan.io/address/0xe82719202e5965Cf5D9B6673B7503a3b92DE20be#code. [Último acceso: 26 07 2020].

[44] [En línea]. Available: https://www.kingoftheether.com/thrones/kingoftheether/index.html. [Último acceso: 01 08 2020].

[45] R. Mitra, «Prueba de trabajo vs Prueba de participación: Guía básica de minado,» 20 06 2019. [En línea]. Available: https://blockgeeks.com/guides/es/prueba-de-trabajo-vs-prueba-de-participacion/. [Último acceso: 19 06 2020].

[46] R. Karjalainen, «Governance in decentralized networks,» 21 05 2020. [En línea]. Available: https://streamr-public.s3.amazonaws.com/governance-whitepaper-2020-05-21-v1_1.pdf. [Último acceso: 20 06 2020].

[47] Deloitte, «Blockchain risk management – Risk functions need to play an active role in shaping blockchain strategy.,» [En línea]. Available: https://www2.deloitte.com/content/dam/Deloitte/us/Documents/financial-services/us-fsi-blockchain-risk-management.pdf. [Último acceso: 20 06 2020].