



US012388666B2

(12) **United States Patent**
Marosi-Bauer

(10) **Patent No.: US 12,388,666 B2**
(45) **Date of Patent: Aug. 12, 2025**

(54) **PROXY AUTONOMOUS PROTOCOL FOR
BLOCKCHAIN ACCESS CONTROL**

(56) **References Cited**

U.S. PATENT DOCUMENTS

(71) Applicant: **CUBE Security Inc.**, Palo Alto, CA
(US)

2015/0200934 A1 7/2015 Naguib
2017/0364552 A1 12/2017 Pattanaik et al.
2020/0118096 A1 4/2020 Yang et al.
2020/0119904 A1 4/2020 Philyaw et al.
2020/0162448 A1 5/2020 Rao et al.

(Continued)

(72) Inventor: **Attila Marosi-Bauer**, Üröm (HU)

(73) Assignee: **CUBE Security Inc.**, Palo Alto, CA
(US)

FOREIGN PATENT DOCUMENTS

(*) Notice: Subject to any disclaimer, the term of this
patent is extended or adjusted under 35
U.S.C. 154(b) by 0 days.

KR 2022-01 42254 A 10/2022
WO WO-2021085701 A1 * 5/2021 G06F 21/6218

OTHER PUBLICATIONS

(21) Appl. No.: **18/590,041**

“Jason Paul Cruz, RBAC-SC: Role-Based Access Control Using
Smart Contract, 2018, IEEE Access, pp. 12240-12251” (Year:
2018).*

(22) Filed: **Feb. 28, 2024**

(Continued)

(65) **Prior Publication Data**

US 2024/0291679 A1 Aug. 29, 2024

Primary Examiner — Kevin Ayala

(74) *Attorney, Agent, or Firm* — Fenwick & West LLP

(57)

ABSTRACT

An access control server may receive, from an access requester, an access request for accessing a protected autonomous program protocol stored on a blockchain. The access control server may analyze parameters related to the access requester to determine whether the access requester is authorized to access the protected autonomous program protocol. An access control server may generate a cryptographic signature associated with the access request. An access control autonomous protocol recorded on the blockchain may receive the cryptographic signature, verify the cryptographic signature, and store proof of authorization in association with the protected autonomous program protocol. The proof of authorization is accessible by the protected autonomous program protocol to verify that the access requester is authorized to access the protected autonomous program protocol.

Related U.S. Application Data

(60) Provisional application No. 63/487,566, filed on Feb. 28, 2023.

(51) **Int. Cl.**

H04L 9/00 (2022.01)

G06Q 20/38 (2012.01)

H04L 9/32 (2006.01)

(52) **U.S. Cl.**

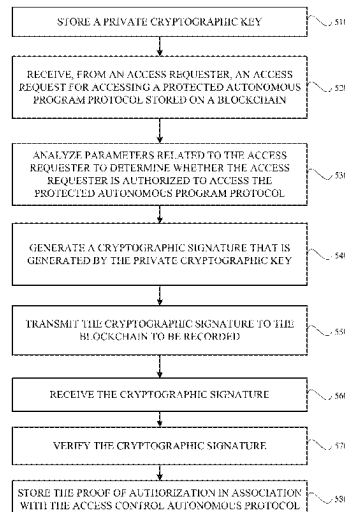
CPC **H04L 9/50** (2022.05); **G06Q 20/3821**
(2013.01); **H04L 9/3218** (2013.01); **H04L**
9/3247 (2013.01)

(58) **Field of Classification Search**

CPC H04L 9/50; H04L 9/3218; H04L 9/3247;
H04L 9/3239; H04L 9/3297

See application file for complete search history.

20 Claims, 10 Drawing Sheets



(56)

References Cited

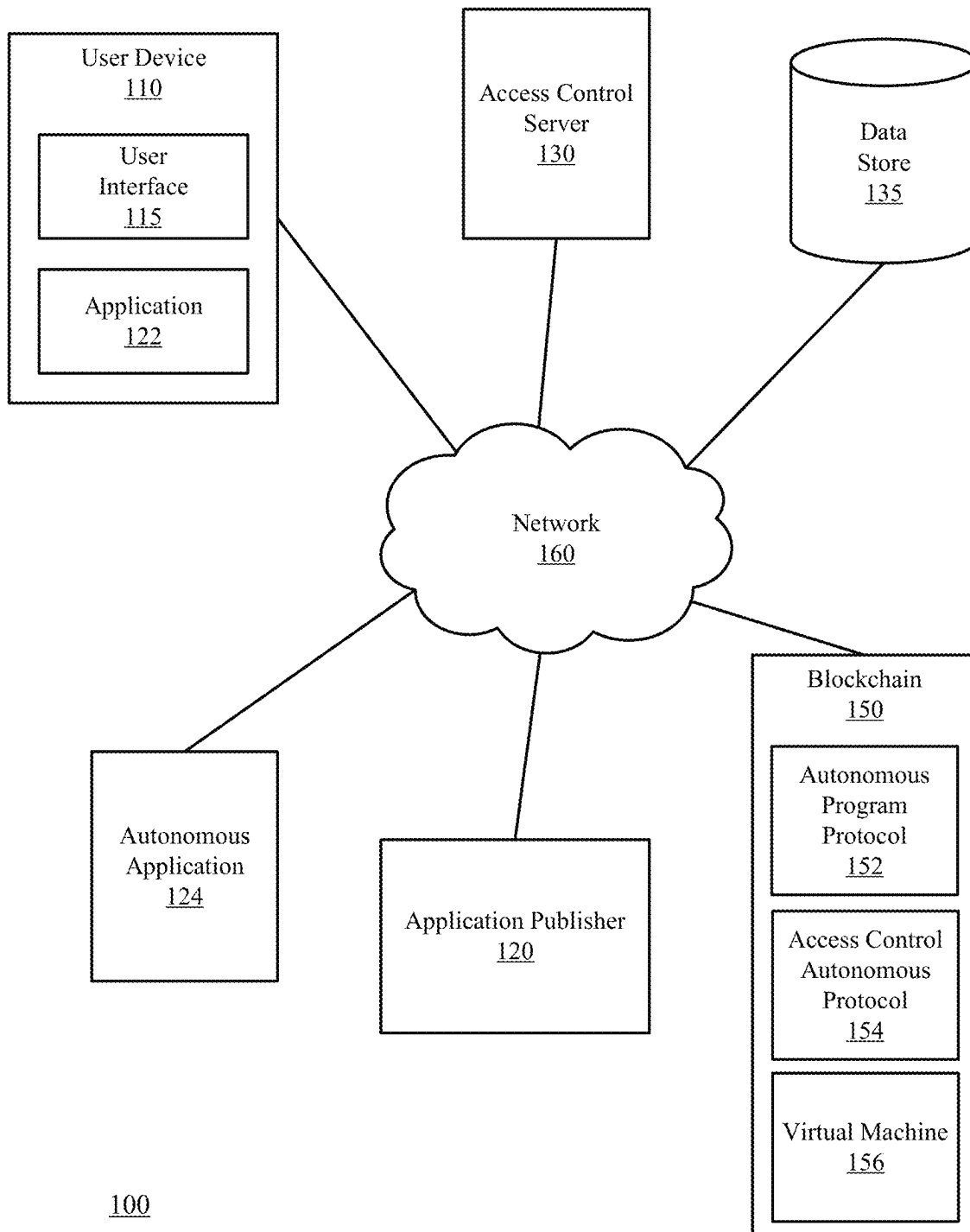
U.S. PATENT DOCUMENTS

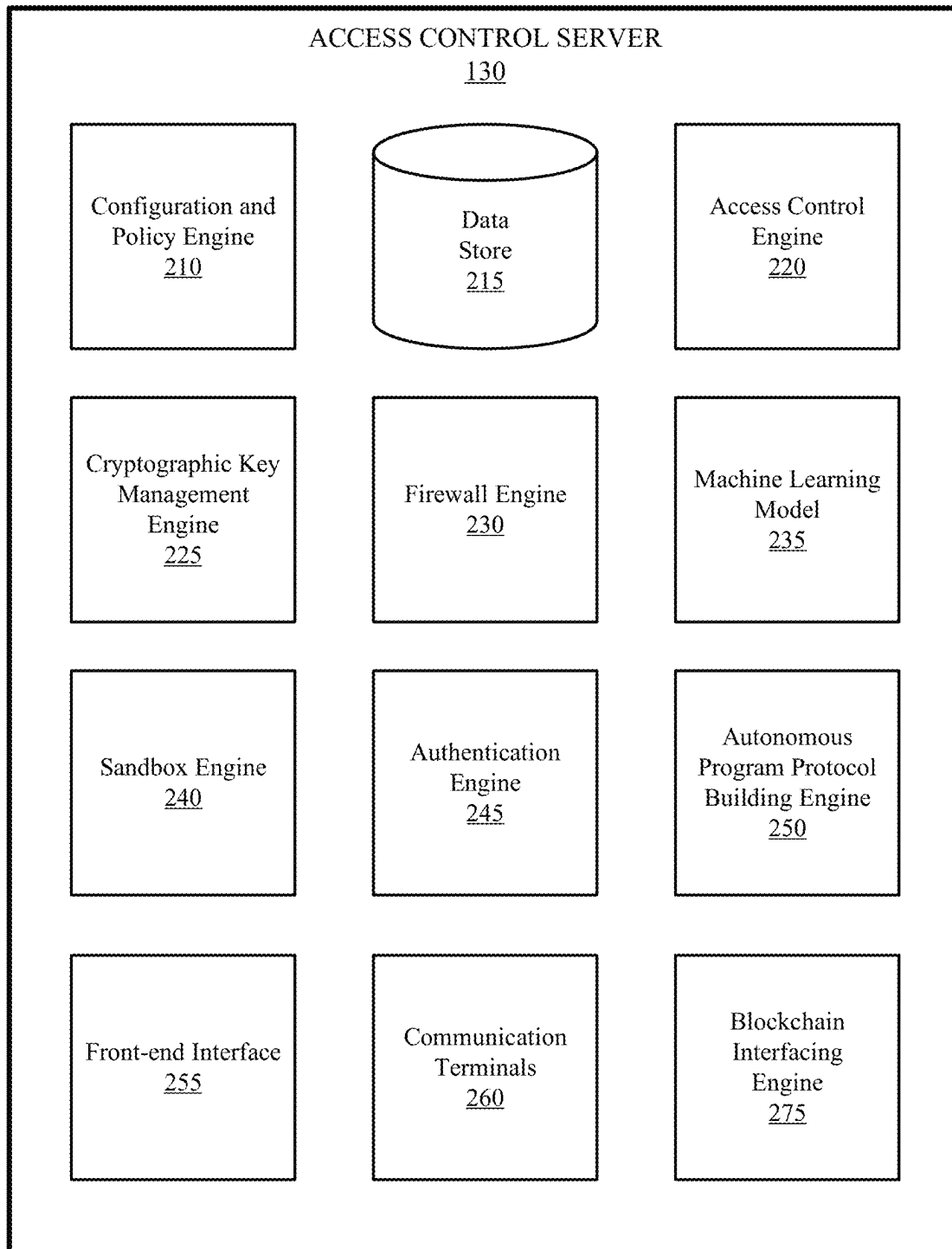
2020/0213117	A1	7/2020	Resch et al.
2020/0334668	A1	10/2020	Nicli et al.
2020/0344070	A1	10/2020	Li
2021/0374731	A1 *	12/2021	Kim G06Q 20/3827
2021/0398017	A1 *	12/2021	Garg H04L 9/50
2022/0045868	A1 *	2/2022	Falk H04L 9/3247
2022/0147961	A1	5/2022	Yoon et al.
2022/0309494	A1	9/2022	Crintea
2022/0321364	A1	10/2022	Sholtis et al.
2022/0353092	A1	11/2022	Rivera et al.
2023/0325822	A1	10/2023	Nagahora et al.
2024/0031146	A1	1/2024	Marosi-Bauer et al.
2024/0135370	A1	4/2024	Cain et al.
2024/0187259	A1	6/2024	Bergmann

OTHER PUBLICATIONS

PCT International Search Report and Written Opinion, PCT Application No. PCT/US2024/017543, Jun. 19, 2024, eight pages.

* cited by examiner

**FIG. 1**

**FIG. 2**

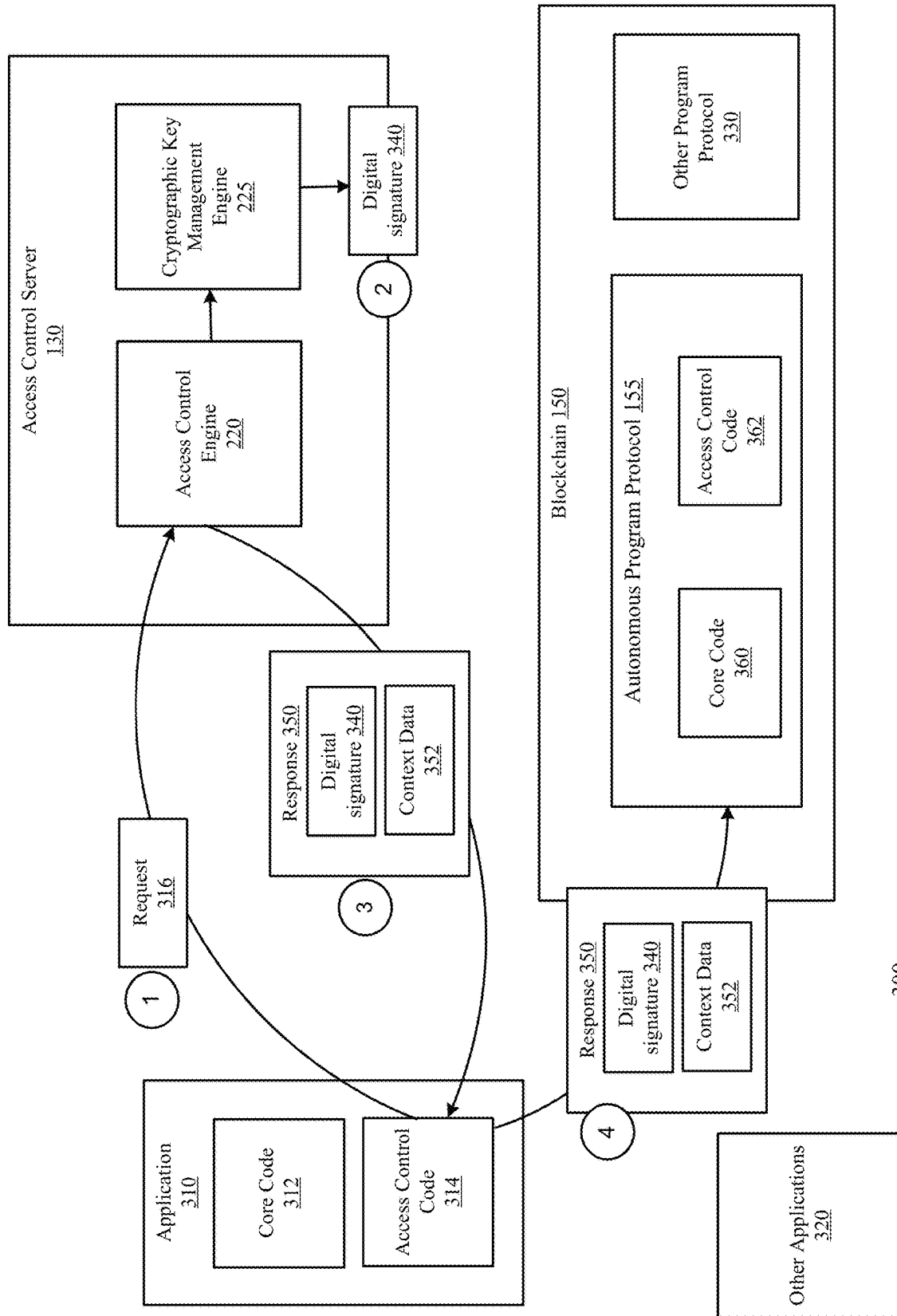
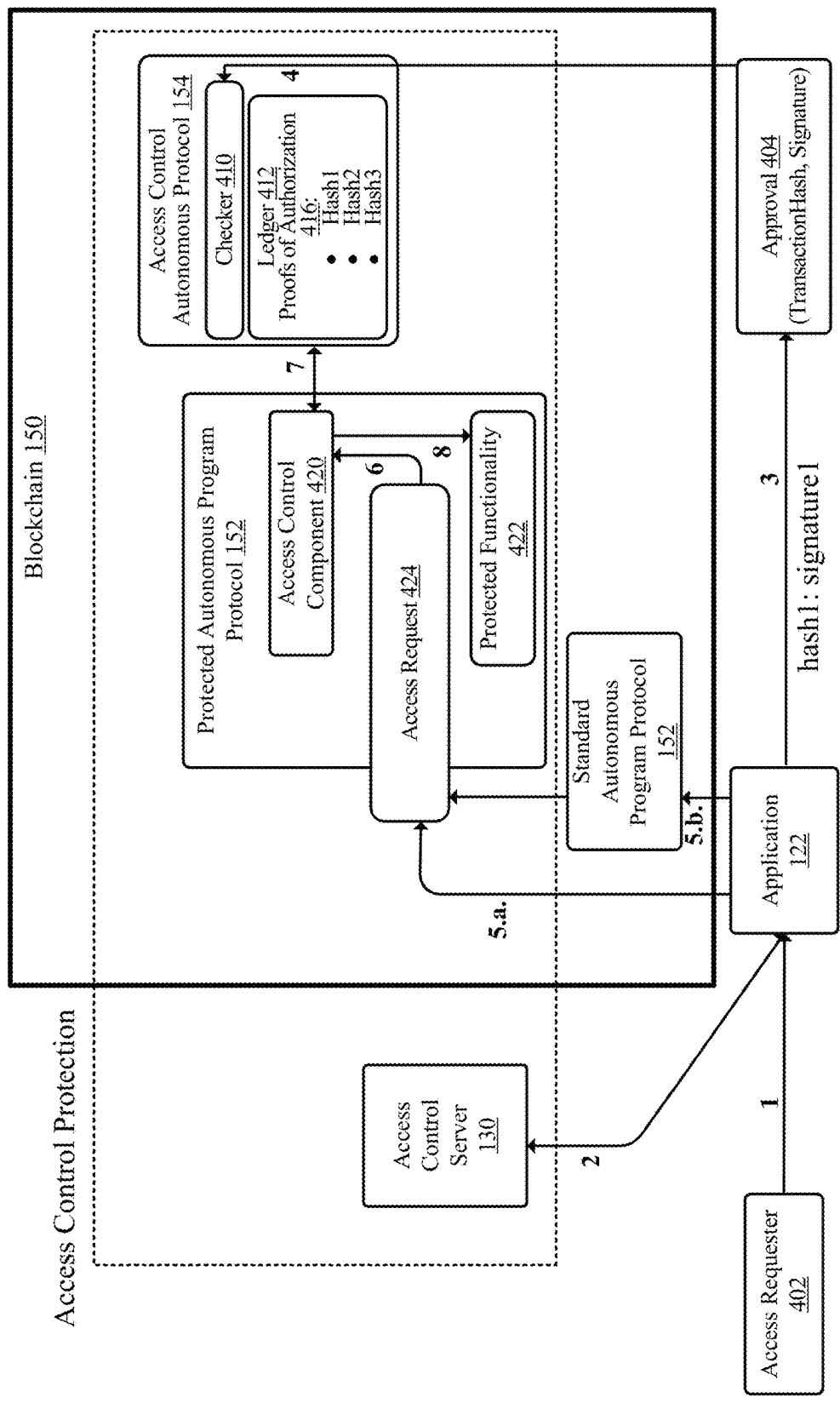


FIG. 3



400

FIG. 4A

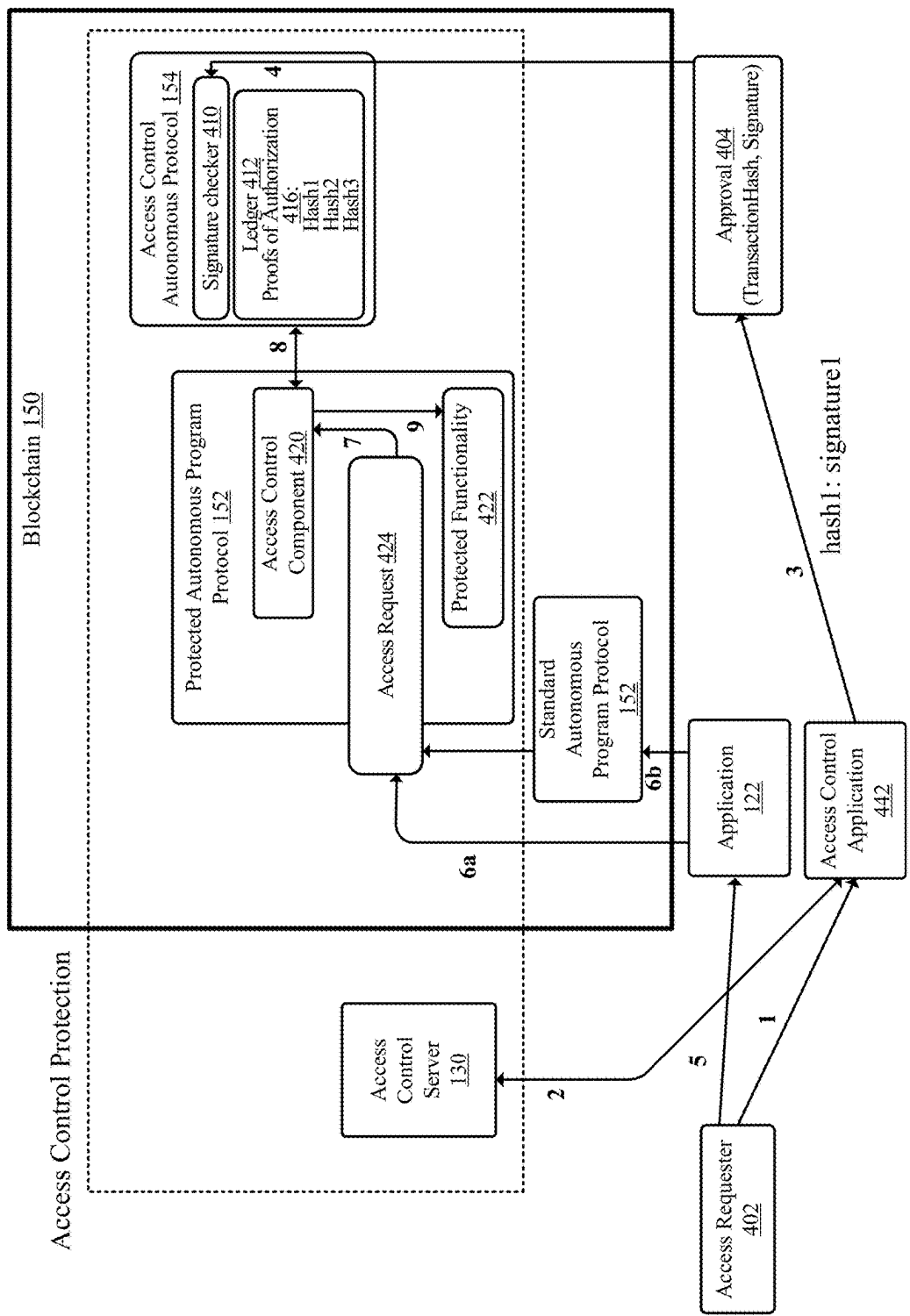


FIG. 4B

440

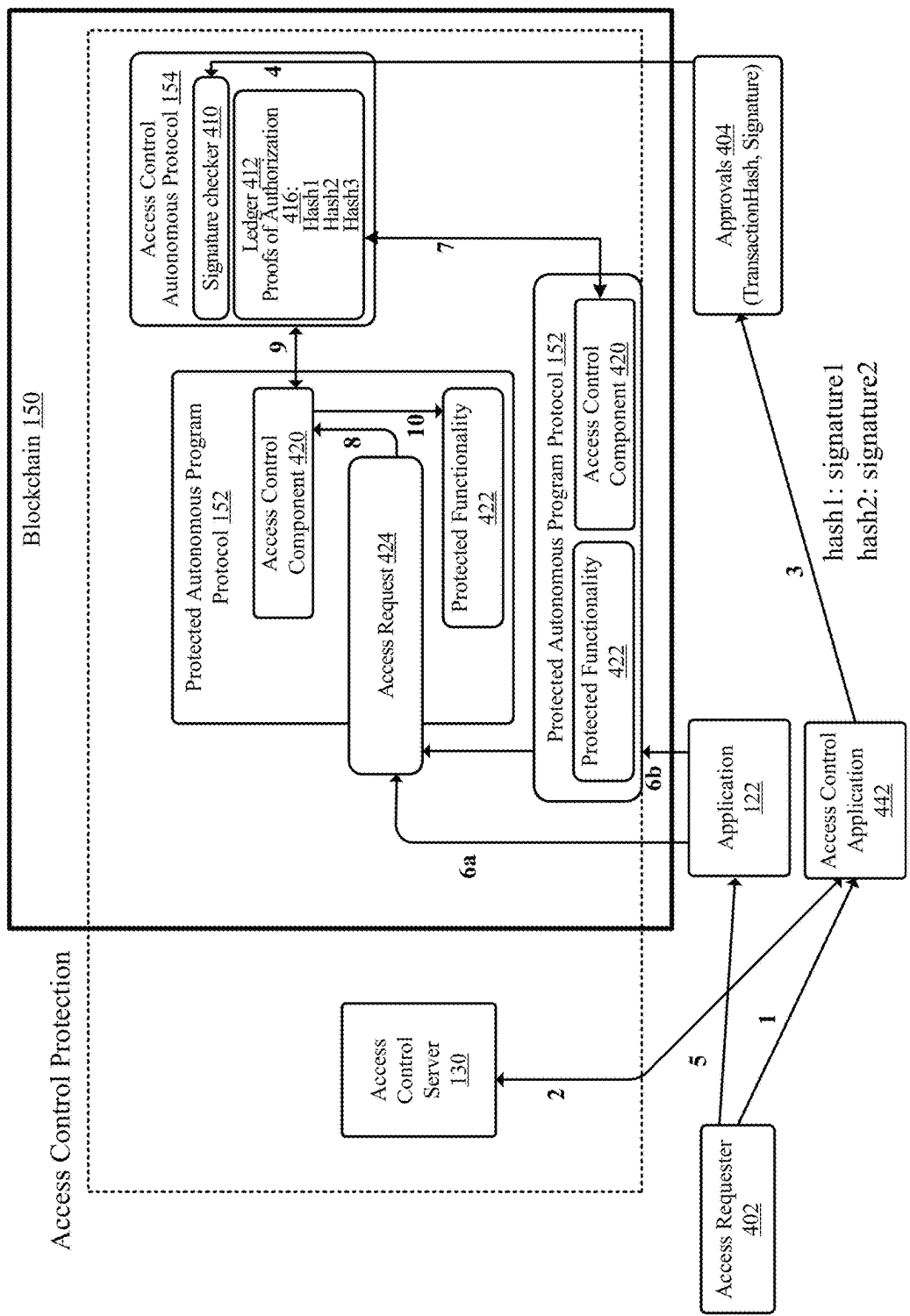
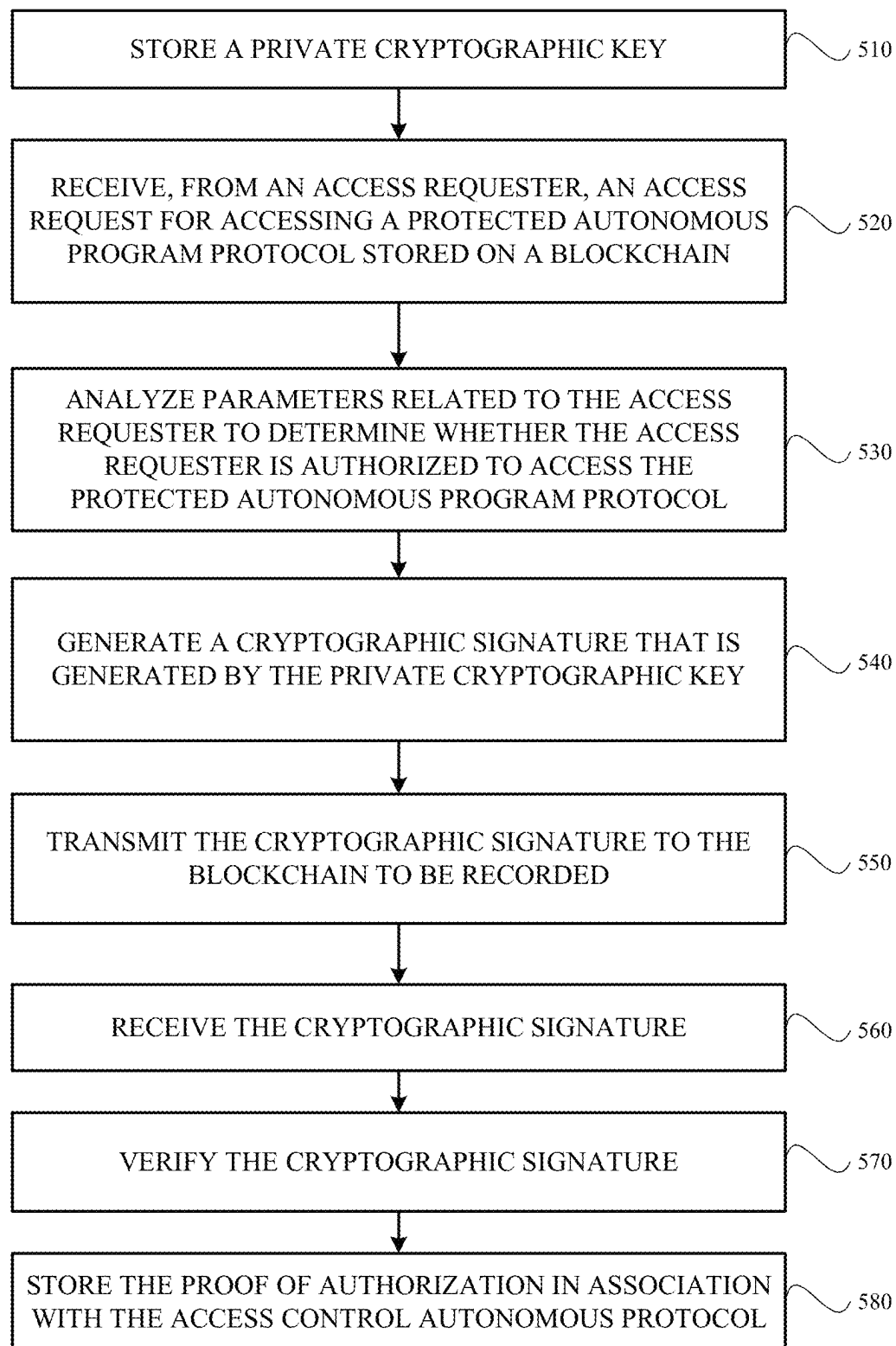


FIG. 4C



500

FIG. 5

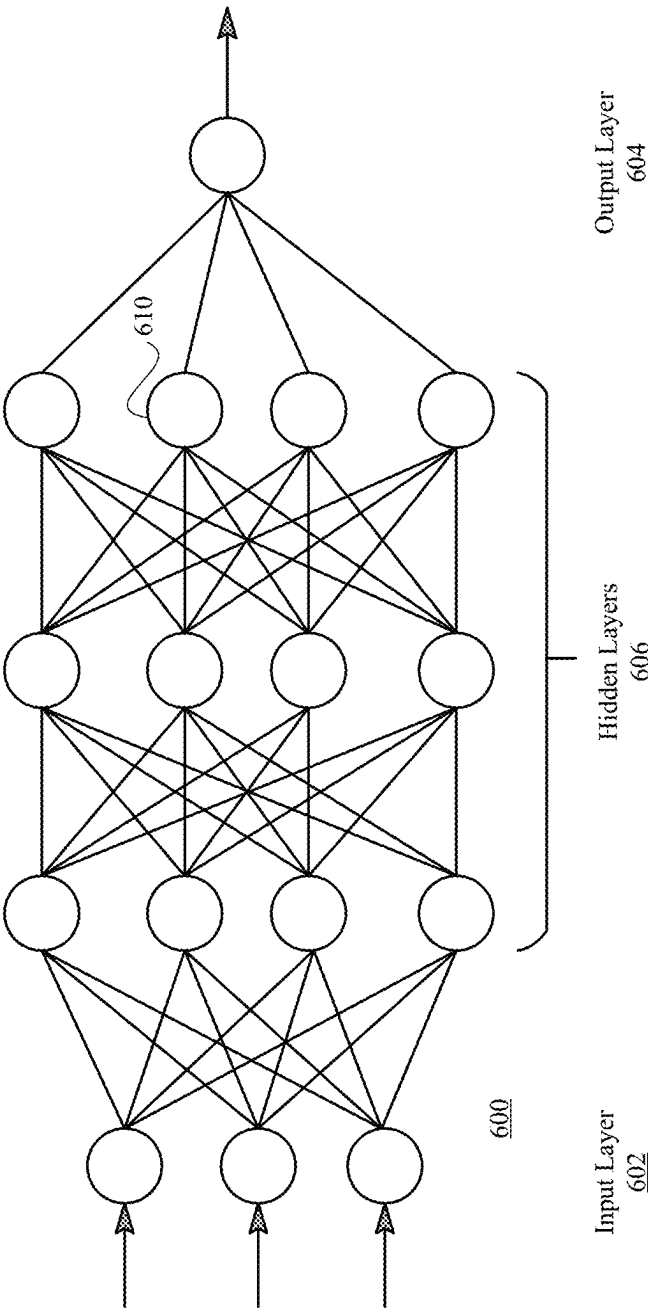
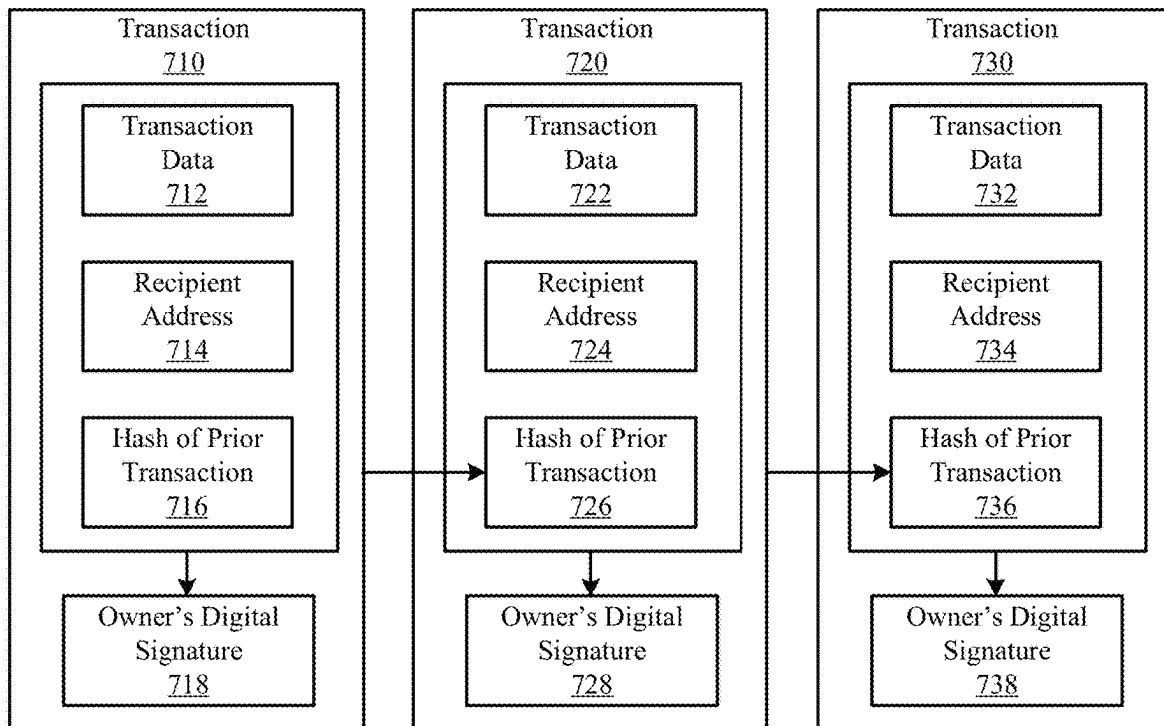
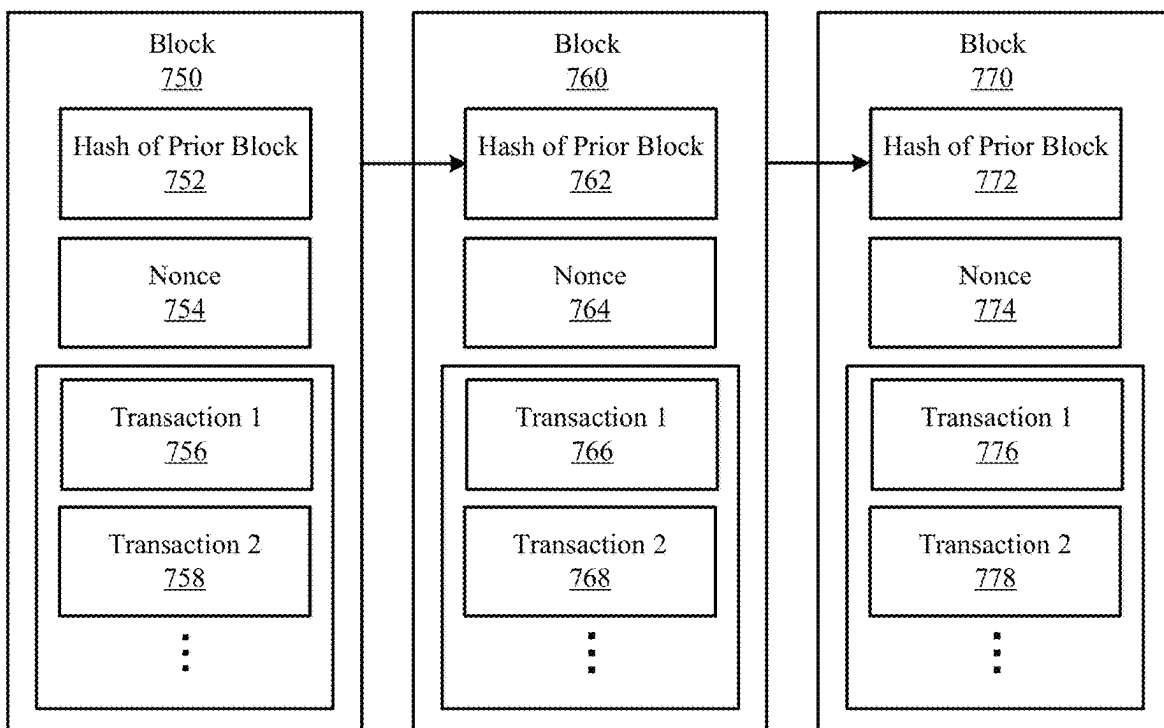
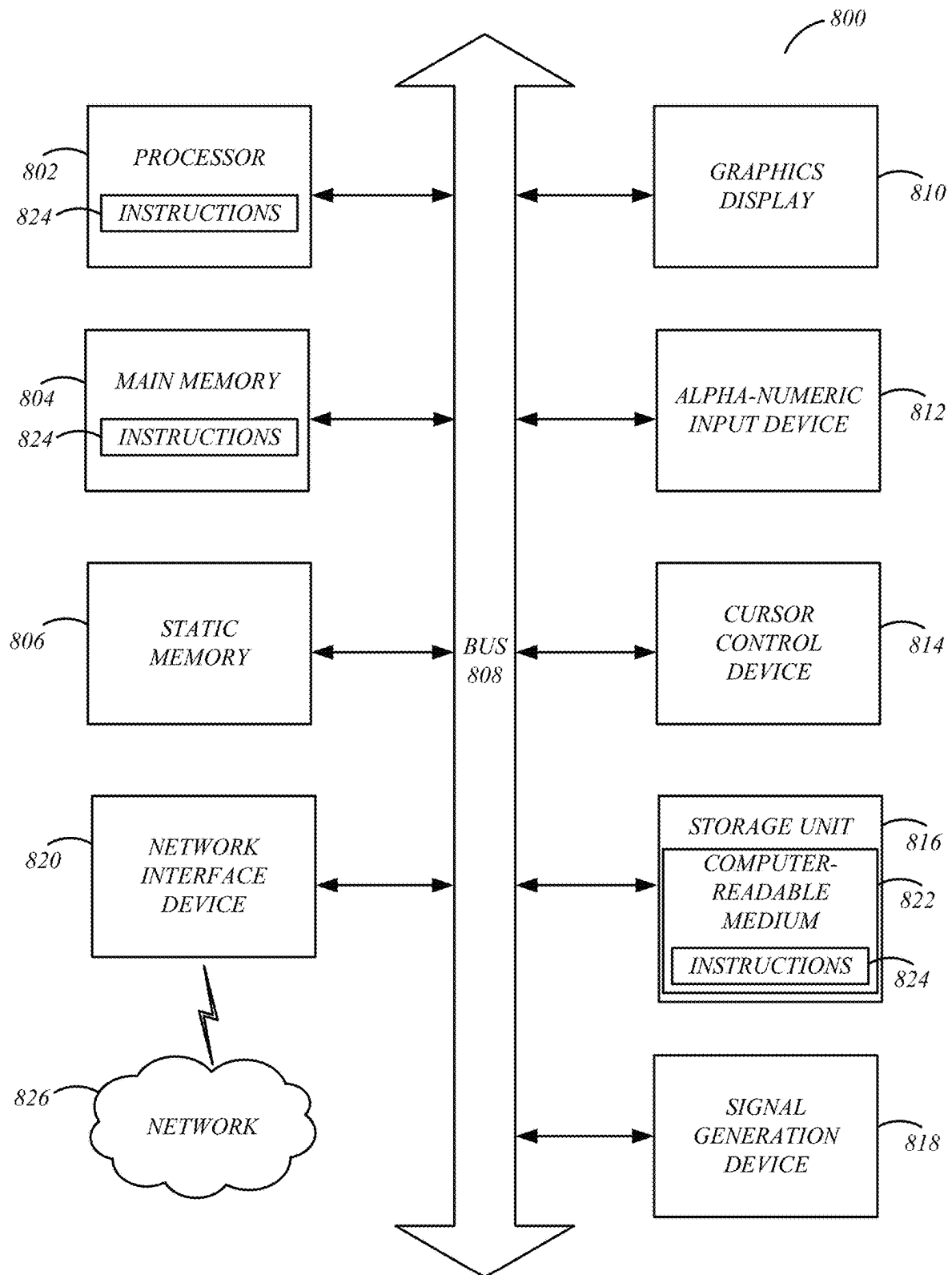


FIG. 6

**FIG. 7A****FIG. 7B**

**FIG. 8**

1

PROXY AUTONOMOUS PROTOCOL FOR BLOCKCHAIN ACCESS CONTROL

CROSS REFERENCE TO RELATED APPLICATIONS

The application claims benefit to U.S. Provisional Application No. 63/487,566, filed on Feb. 28, 2023, which is incorporated by reference herein for all purposes.

FIELD

The disclosure generally relates to access control security and, more specifically, to the architecture of access control for program protocols recorded on a blockchain.

BACKGROUND

The blockchain ecosystem currently does not provide an efficient and secure solution to extending blockchain built-in security. Application developers have minimal control over their application after it has been published and do not have the chance to protect them if this is not pre-coded in the contract. Smart contracts are applications inside blockchains which means that smart contracts inherit the security capabilities and features of the underlying blockchain. However, an application recorded on a blockchain lacks various security features that may protect the application from malicious attacks.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a block diagram that illustrates a system environment of an example computing server, in accordance with an embodiment.

FIG. 2 is a block diagram representing an example access control server, in accordance with an embodiment.

FIG. 3 is a block diagram illustrating an example access control system and the message control flow of the system, in accordance with some embodiments.

FIG. 4A is a block diagram illustrating an example access control system using an access control protocol, in accordance with some embodiments.

FIG. 4B is a block diagram illustrating an example access control system using an access control protocol, in accordance with some embodiments.

FIG. 4C is a block diagram illustrating an example access control system using an access control protocol, in accordance with some embodiments.

FIG. 5 is a message flowchart depicting an access control process for an autonomous program protocol, in accordance with some embodiments.

FIG. 6 is a conceptual diagram illustrating the structure of an example neural network is illustrated, in accordance with some embodiments.

FIG. 7A is a block diagram illustrating a chain of transactions broadcasted and recorded on a blockchain, in accordance with an embodiment.

FIG. 7B is a block diagram illustrating a connection of multiple blocks in a blockchain, in accordance with an embodiment.

FIG. 8 is a block diagram illustrating components of an example computing machine that is capable of reading instructions from a computer-readable medium and executing them in a processor (or controller).

2

The figures depict, and the detailed description describes, various non-limiting embodiments for purposes of illustration only.

DETAILED DESCRIPTION

The figures (FIGs.) and the following description relate to preferred embodiments by way of illustration only. One of skill in the art may recognize alternative embodiments of the structures and methods disclosed herein as viable alternatives that may be employed without departing from the principles of what is disclosed.

Reference will now be made in detail to several embodiments, examples of which are illustrated in the accompanying figures. It is noted that wherever practicable similar or like reference numbers may be used in the figures and may indicate similar or like functionality. The figures depict embodiments of the disclosed system (or method) for purposes of illustration only. One skilled in the art will readily recognize from the following description that alternative embodiments of the structures and methods illustrated herein may be employed without departing from the principles described herein.

System Overview

FIG. 1 is a block diagram that illustrates a system environment 100 of an example computing server, in accordance with an embodiment. By way of example, the system environment 100 includes a user device 110, an application publisher 120, an access control server 130, a data store 135, a blockchain 150, an autonomous program protocol 152, an access control autonomous protocol 154, and a virtual machine 156. The entities and components in the system environment 100 communicate with each other through the network 160. In various embodiments, the system environment 100 may include different, fewer, or additional components. The components in the blockchain system environment 100 may each correspond to a separate and independent entity or may be controlled by the same entity. For example, in one embodiment, the access control server 130 may control the data store 135.

While each of the components in the system environment 100 is often described in disclosure in a singular form, the system environment 100 may include one or more of each of the components. For example, there can be multiple user devices 110 communicating with the access control server 130 and the blockchain 150. Also, the access control server 130 may provide service for multiple application publishers 120, each of whom has multiple end users that may operate different user devices 110. While a component is described in a singular form in this disclosure, it should be understood that in various embodiments the component may have multiple instances. Hence, in the system environment 100, there can be one or more of each of the components.

A user device may also be referred to as a client device. A user device 110 may be controlled by a user who may be the customer of the application publisher 120, the access control server 130, or a participant of the blockchain 150. In some situations, a user may also be referred to as an end user, for example, when the user is the application publisher's customer who uses applications that are published by the application publisher 120. The user device 110 may be any computing device. Examples of user devices 110 include personal computers (PC), desktop computers, laptop computers, tablet computers, smartphones, wearable electronic devices such as smartwatches, or any other suitable electronic devices.

The user device 110 may include a user interface 115 and an application 122. The user interface 115 may be the interface of the application 122 and allow the user to perform various actions associated with application 122. For example, application 122 may be a distributed application and the user interface 115 may be the frontend. The user interface 115 may take different forms. In one embodiment, the user interface 115 is a software application interface. For example, the application publisher 120 may provide a front-end software application that can be displayed on a user device 110. In one case, the front-end software application is a software application that can be downloaded and installed on a user device 110 via, for example, an application store (App store) of the user device 110. In another case, the front-end software application takes the form of a webpage interface of the application publisher 120 that allows clients to perform actions through web browsers. The front-end software application includes a graphical user interface (GUI) that displays various information and graphical elements. In another embodiment, user interface 115 does not include graphical elements but communicates with the application publisher 120 via other suitable ways such as command windows or application program interfaces (APIs).

An application publisher 120, such as a software company, may be an entity that provides various types of software applications. The application publisher 120 may publish and/or operate various types of applications, such as application 122 that is installed on a user device 110, an autonomous application 124 that may be a decentralized application that is run on a decentralized network or blockchain, and the autonomous program protocol 152 that is recorded on a blockchain 150. The autonomous program protocol 152 may take the form of a smart contract or another type of autonomous algorithm that operates on a blockchain. The autonomous application 124 and autonomous program protocol 152 may be applications that have similar natures. In some embodiments, the autonomous application 124 may also operate on a blockchain and the autonomous application 124 is an example of autonomous program protocol 152. In some embodiments, the autonomous application 124 may serve as an interface of the autonomous program protocol 152. For example, the autonomous application 124 may allow a user to access one or more functions of the autonomous program protocol 152 through the interface of autonomous application 124. In some embodiments, the application publisher 120 may record a fully autonomous application on the blockchain 150 as the autonomous program protocol 152 and operate different applications, such as the application 122 and autonomous application 124 to allow a user, a device, or an automated agent to interact with the autonomous program protocol 152. In some embodiments, as discussed in further detail below throughout this disclosure, the autonomous program protocol 152 published by the application publisher 120 may incorporate certain protocols (e.g., access control protocols) of the access control server 130 to provide security and access control to the autonomous program protocol 152.

An access control server 130 may be a server that provides various access control services to provide security to an autonomous program protocol 152 recorded on the blockchain 150 and protect the autonomous program protocol 152 from malicious attacks. The access control server 130 may be centralized or decentralized. In some embodiments, the access control server 130 may not directly carry out access control but may publish an access control autonomous

protocol 154 to provide access control to an autonomous program protocol 152. The specification and configuration of the access control autonomous protocol 154 may be configured by a customer of the access control server 130 (e.g., an application publisher 120) such that the application publisher 120 may control how the access control of an autonomous program protocol 152 published by the application publisher 120 is managed by the access control autonomous protocol 154. In some embodiments, the access control server 130 and the access control autonomous protocol 154 may cooperate to provide access control features to an autonomous program protocol 152. The access control features may be distributed between the access control server 130 and the autonomous program protocol 152. For example, the access control server 130 may carry out some steps in granting access while the autonomous program protocol 152 may carry the remaining steps.

The services provided by the access control server 130 may include firewall, access control, sandbox testing environment, authentication (e.g., two-factor authentication), authorization, and other suitable cybersecurity services and compliance (e.g., Know Your Customers KYC) services. In one embodiment, the access control server 130 may be partially centralized and partially decentralized. For example, certain access control policies (e.g., who may access the autonomous program protocol 152) may be specified by an application publisher 120 and centrally enforced by the access control server 130. In some embodiments, the access control server 130 may also be decentralized and certain services such as authentication services can be carried out autonomously. The details of the operations and sub-components of the access control server 130 will be further discussed in association with FIG. 2.

U.S. Patent Application Publication No. US2024/0031146, entitled "Access control interfaces for blockchains," published on Jan. 25, 2024, is incorporated by reference herein for all purposes.

In various embodiments, the access control server 130 may take different suitable forms. For example, while the access control server 130 is described in a singular form, the access control server 130 may include one or more computers that operate independently, cooperatively, and/or distributively. In some embodiments, the access control server 130 may be a server computer that includes one or more processors and memory that stores code instructions that are executed by one or more processors to perform various processes described herein. In some embodiments, the access control server 130 may be a pool of computing devices that may be located at the same geographical location (e.g., a server room) or be distributed geographically (e.g., cloud computing, distributed computing, or in a virtual server network). In some embodiments, the access control server 130 may be a collection of servers that independently, cooperatively, and/or distributively provide various products and services described in this disclosure. The access control server 130 may also include one or more virtualization instances such as a container, a virtual machine, a virtual private server, a virtual kernel, or another suitable virtualization instance. The access control server 130 may provide various access control services as a form of cloud-based software, such as software as a service (SaaS), through the network 160. The access control server 130 may include the entity that operates the access control server 130.

The data store 135 includes one or more storage units such as memory that takes the form of a non-transitory and non-volatile computer storage medium to store various data.

The computer-readable storage medium is a medium that does not include a transitory medium such as a propagating signal or a carrier wave. The data store **135** may be used by the access control server **130** to store data related to the access control server **130**, such as access control policies of various autonomous program protocols **152** and associated authentication criteria. In one embodiment, the data store **135** communicates with other components by the network **160**. This type of data store **135** may be referred to as a cloud storage server. Examples of cloud storage service providers may include AMAZON AWS, DROPBOX, RACKSPACE CLOUD FILES, AZURE BLOB STORAGE, GOOGLE CLOUD STORAGE, etc. In another embodiment, instead of a cloud storage server, the data store **135** is a storage device that is controlled and connected to the access control server **130**. For example, the data store **135** may take the form of memory (e.g., hard drives, flash memory, discs, ROMs, etc.) used by the access control server **130** such as storage devices in a storage server room that is operated by the access control server **130**.

A blockchain **150** may be a public blockchain that is decentralized, a private blockchain, a semi-public blockchain, an execution layer settling data on a public blockchain (e.g., Layer 2 blockchains, rollups), or an application-specific chain. A public blockchain network includes a plurality of nodes that cooperate to verify transactions and generate new blocks. In some implementations of a blockchain, the generation of a new block may also be referred to as a proposal process, which may be a mining process or a validation process. Some of the blockchains **150** support smart contracts, which are a set of code instructions that are stored on a blockchain **150** and are executable when one or more conditions are met. Smart contracts may be examples of autonomous program protocols **152**. When triggered, the set of code instructions of a smart contract may be executed by a computer such as a virtual machine of the blockchain **150**. Here, a computer may be a single operation unit in a conventional sense (e.g., a single personal computer) or may be a set of distributed computing devices that cooperate to execute the code instructions (e.g., a virtual machine or a distributed computing system). A blockchain **150** may be a new blockchain or an existing blockchain such as BITCOIN, ETHEREUM, EOS, NEO, SOLANA, AVALANCHE, etc.

The autonomous program protocols **152** may be token contracts, smart contracts, Web3 applications, autonomous applications, distributed applications, decentralized finance (DeFi) applications, protocols for decentralized autonomous organizations (DAO), non-fungible tokens (NFT), decentralized exchanges, identity services, blockchain gaming, metaverse protocols, and other suitable protocols and algorithms that may be recorded on a blockchain. The autonomous program protocol **152** may take the form of a set of code instructions that are stored on a blockchain **150** and are executable when one or more conditions are met. Smart contracts may be examples of autonomous program protocols **152** that may be executable by a computer such as a virtual machine **156** of the blockchain **150**. Here, a computer may be a single operation unit in a conventional sense (e.g., a single personal computer), a resource of the blockchain such as a virtual machine **156**, or a set of distributed computing devices that cooperate in executing the code instructions (e.g., a distributed computing system). An autonomous program protocol **152** includes a set of instructions. The instructions, when executed by one or more processors, cause one or more processors to perform steps specified in the instructions. The processors may correspond to a blockchain node of the blockchain **150** or may be

distributed among various nodes of the blockchain **150**. Each autonomous program protocol **152** may be associated with its own blockchain address so that the autonomous program protocol **152** is identifiable on the blockchain. Requests for actions to be performed by the autonomous program protocol **152** may be sent to the blockchain address. Each autonomous program protocol **152** may also be associated with a ledger that records the transfer of tokens that are associated with the autonomous program protocol **152** and records other actions that are associated with the autonomous program protocol **152**. The ledger may be retrievable using the blockchain address of the autonomous program protocol **152**.

An access control autonomous protocol **154** may be a special type of autonomous program protocol **152** that serves as a proxy protocol used to provide access control features of another autonomous program protocol **152**. As such, before interacting with the autonomous program protocol **152**, a party may provide information to the access control autonomous protocol **154** to allow the access control autonomous protocol **154** to determine whether the party has the authority to interact with the autonomous program protocol **152**. The autonomous program protocol **152** may be referred to as a protected autonomous program protocol **152** and the access control autonomous protocol **154** may also be referred to as a proxy autonomous program protocol.

The access control protocol **154** may store one or more verification information of the user in the ledger of the access control autonomous protocol **154** so that a protected autonomous program protocol **152** may review the verification information to determine whether a request has proper access rights to the protected autonomous program protocol **152**. For example, the access control autonomous protocol **154** may store a version of the transaction request (e.g., a hash), a version of the identity of the access requester (e.g., public key corresponding to the access requester), and/or a version of the access requester's signature in the ledger so that the protected autonomous program protocol **152** may retrieve the information on the ledger access control autonomous protocol **154** to verify that the access requester can access the autonomous program protocol **152**.

A virtual machine **156** is a resource unit of a blockchain **150**. A virtual machine **156** may be a standardized software execution environment that emulates the functionality of a physical machine and allows for the execution of autonomous program protocol **152** on the virtual machine **156**. A virtual machine **156** may be run by any blockchain node. In some embodiments, a virtual machine **156** may take the form of a sandboxed environment that is created within the blockchain network to execute autonomous program protocol **152**. The autonomous program protocols **152** are compiled into bytecode that can be executed by the virtual machine. One example of the virtual machine **156** is Ethereum Virtual Machine (EVM) that executes the programming language SOLIDITY. In some embodiments, a virtual machine **156** may operate based on binary instruction language such as WEBASSEMBLY that can be executed in a variety of environments, such as web browsers. An example of such a virtual machine **156** is Ethereum WebAssembly (EWASM) which may allow programmers to build autonomous program protocols **152** using various common programming languages.

In some embodiments, one or more access control protocols **154**, one or more access control servers **130**, one or more application publishers **120**, and other components may serve as multi-party computation nodes for carrying out any access control procedures that are described in this disclo-

sure. Hence, even though in various locations of this disclosure a feature is described as being carried out by a particular component, in a multi-party computation embodiment, the feature may be carried out cooperatively or distributively by any multi-party computation nodes. By way of example, an access control server **130** and an access control autonomous protocol **154** may serve as multi-party computation nodes to carry out various steps that are described in FIG. 3 through FIG. 5.

The communications among the user device **110**, the access control server **130**, the autonomous application **124**, the application publisher **120** and the blockchain **150** may be transmitted via a network **160**, for example, via the Internet. In one embodiment, the network **160** uses standard communications technologies and/or protocols. Thus, the network **160** can include links using technologies such as Ethernet, 802.11, worldwide interoperability for microwave access (WiMAX), 3G, 5G, LTE, 5G, digital subscriber line (DSL), asynchronous transfer mode (ATM), InfiniBand, PCI Express Advanced Switching, etc. Similarly, the networking protocols used on the network **160** can include multiprotocol label switching (MPLS), the transmission control protocol/Internet protocol (TCP/IP), the User Datagram Protocol (UDP), the hypertext transport protocol (HTTP), the simple mail transfer protocol (SMTP), the file transfer protocol (FTP), etc. The data exchanged over the network **160** can be represented using technologies and/or formats including the hypertext markup language (HTML), the extensible markup language (XML), etc. In addition, all or some of the links can be encrypted using conventional encryption technologies such as secure sockets layer (SSL), transport layer security (TLS), virtual private networks (VPNs), Internet Protocol security (IPsec), etc. The network **160** also includes links and packet-switching networks such as the Internet. Example Computing Server

FIG. 2 is a block diagram representing an example access control server **130**, in accordance with an embodiment. In the embodiment shown in FIG. 2, the access control server **130** includes configuration and policy engine **210**, account store **215**, access control engine **220**, cryptographic key management engine **225**, firewall engine **230**, machine learning model **235**, sandbox engine **240**, authentication engine **245**, autonomous program protocol building engine **250**, front-end interface **255**, communication terminals **260**, and blockchain interfacing engine **275**. The functions of the access control server **130** may be distributed among different components in a different manner than described below. Also, in various embodiments, the access control server **130** may include different, fewer, and/or additional components.

While the access control server **130** is used in a singular form, the access control server **130** may include one or more computers that include one or more processors and memory. The memory may store computer code that includes instructions. The instructions, when executed by one or more processors, cause the processors to perform one or more processes described herein. The access control server **130** may take different forms. In one embodiment, the access control server **130** is a single computer that executes code instructions directly. In another embodiment, the access control server **130** is a group of computing devices that communicate with each other. The computing devices may be located geographically at the same (e.g., a server room) or in different locations. In yet another embodiment, the access control server **130** includes multiple nodes that operate in a distributed fashion such as in cloud computing or distributed computing. Each node may include one or more computing devices operating together. For example, in

some embodiments, the access control server **130** is decentralized and is operated by different nodes cooperatively to form the access control server **130**. In some cases, the access control server **130** may also include virtual machines. Any computing devices, nodes, or virtual machines, singular or plural, may simply be referred to as a computer, a computing device, or a computing server. Components of the access control server **130** shown in FIG. 2, individually or in combination, may be a combination of hardware and software and may include all or a subset of the example computing system illustrated and described in FIG. 8.

The configuration and policy engine **210** may store and determine rules for various participants in the application environment **100**. A policy may be defined and initiated by an application publisher **120** or automatically added or defined by the access control server **130**. An application publisher **120** may transmit the policy setting to, or build the policy at, the access control server **130**. The configuration and policy engine **210** translates the policy to one or more configurations in the system environment **100**. A policy may be an access control policy for an autonomous program protocol **152**. The access control server **130** provides security, protection, and access control to an autonomous program protocol **152**. An application publisher **120** may specify one or more access control settings that define various criteria for granting access to an autonomous program protocol **152**. For example, the access control settings may define who can gain access to an autonomous program protocol **152** and the manner in which a party may access the autonomous program protocol **152**. The settings may also define trusted entities in authentication and various security rules in controlling the traffic related to the autonomous program protocol **152**. The settings may further define authorization and an access control list that may be specific to an autonomous program protocol **152**. The settings may further define the protocols of an access control autonomous protocol **154**.

A policy may be generic or specific. A specific policy may be a policy that is customized or specified by an application publisher **120** who published an autonomous program protocol **152**. A specific policy defines a special rule with respect to the security or access control of the autonomous program protocol **152**. For example, an application publisher **120** may define a context-specific policy on the access control of the autonomous program protocol **152**. In contrast, a generic policy may be a policy that is commonly beneficial to many autonomous program protocols **152** and may be automatically enforced by the access control server **130** upon request without having the application publisher **120** specifically define the rules in the generic policy. For example, a generic policy may be a policy to prevent the autonomous program protocol **152** from a denial-of-service attack or a policy that detects fraudulent transactions. The configuration and policy engine **210** may include default rules for a generic policy and may enforce a generic policy for various autonomous program protocols **152** that are vulnerable to common security threats.

The data store **215** is a database that stores various information with respect to settings provided by customers, such as application publishers **120**, of the access control server **130**. The data stored may include a profile of the customer, applications operated by the customer, autonomous program protocols **152** published by the customer, and various access control settings associated with an autonomous program protocol **152**. The data store **215** may also include or be in communication with a credential vault that stores user identifiers and passwords and the access control

server **130** may perform authentication on behalf of a customer. The data store **215** may also store data and metadata related to various transactions involving an autonomous program protocol **152**. The transaction records may be used as training samples in one or more machine learning models for identifying normal usage patterns of an autonomous program protocol **152** in distinguishing normal operations from potentially fraudulent operations or malicious activities.

The access control engine **220** manages the access control of an autonomous program protocol **152** based on the policy settings specified by an application publisher **120**. The access control engine **220** may deploy other engines, such as the firewall engine **230**, the machine learning model **235**, the sandbox engine **240**, and the authentication engine **245** to manage the access of an autonomous program protocol **152**. The access control engine **220** may control traffic, identify threats, and enforce authentication and authorization for an autonomous program protocol **152**. For example, the access control engine **220** may control whether a request to access autonomous program protocol **152** is valid and authorized. The request to access may include a function call of the autonomous program protocol **152**. If a request is valid and authorized, the access control engine **220** may generate a digital signature for the request. For example, the access control engine **220** may use a private cryptographic key of the access control server **130** to sign the payload of the request. The private cryptographic key may be specific to the particular autonomous program protocol **152**. The digital signature may be a requirement for autonomous program protocol **152** to recognize the request. In some embodiments, the autonomous program protocol **152** may store the public cryptographic key of the access control server **130** that corresponds to the private cryptographic key. The autonomous program protocol **152** may be configured to use the public cryptographic key to verify the digital signature before a function call may be invoked. If the access control engine **220** determines that the request is not valid or authorized, a digital signature is not generated and the access control engine **220** may block the request and log the request as part of the record. The access control engine **220** may also return an error message to the requester. If the autonomous program protocol **152** is configured to require the digital signature of the access control server **130** before a function is invoked, the party sending the request will not be able to gain access to autonomous program protocol **152** without authorization from the access control engine **220**.

The access control for an autonomous program protocol **152** may be function-specific. For example, an autonomous program protocol **152** may include more than one function call that can be invoked. The application publisher **120** may specify different access control policies for each of the function calls. For example, for certain function calls, the application publisher **120** may allow the general public to use those functions and the access control policies may be more lenient, such as allowing the use without authentication. In some cases, the application publisher **120** may request the access control server **130** to generate digital signatures for all of the requests for those public functions so long as the requests are not malicious. In some cases, the application publisher **120** may even allow the access control server **130** to sign all of the requests regardless of the situation so that access control is essentially bypassed in this type of situation. In some cases, function variants with different methods of authorization may be available on the autonomous program protocols **152**. Multiple entry functions may use a combination of signature requirement,

oracle list checking, allowlist checking, or no additional security checks. These functions may call the same private function for the autonomous program protocol logic. For other function calls, such as those related to premium functions that are offered to only certain users, such as paid subscribers, the application publisher **120** may specify access control policies that require authentication and authorization before the access control server **130** generates a digital signature for a request that tries to invoke one of those function calls. Restricted functions may also be available only for verified customers such as compliant users for compliance and “know your customer” purposes.

The nature of access control may vary for different autonomous program protocols **152**, depending on how an application publisher **120** specifies the policies. For example, in some cases, the access control engine **220** may provide firewall service to an autonomous program protocol **152** and protect the autonomous program protocol **152** from malicious attacks. In some cases, the access control engine **220** may provide an authentication service to limit access to another autonomous program protocol **152**. In some cases, the access control engine **220** may use one or more machine learning models **235** to identify abnormal patterns and traffic related to an autonomous program protocol **152** and may react to any potential malicious attack such as by blocking access attempts (e.g., not generating digital signatures) from parties that are identified as potential malicious parties. The type of suitable access controls may vary among embodiments and may be decided by an application publisher **120** who specifies various policies for an autonomous program protocol **152**.

The cryptographic key management engine **225** stores and manages one or more keys of the access control server **130** to allow the access control server **130** to participate in various blockchains and to generate digital signatures for requests to access autonomous program protocols **152**. The cryptographic key management engine **225** stores various private cryptographic keys of the access control server **130**. In some embodiments, the access control server **130** may use master private cryptographic keys for different autonomous program protocols **152**. In some embodiments, for each autonomous program protocol **152**, the cryptographic key management engine **225** may generate a new pair of private and public cryptographic keys. The cryptographic key management engine **225** keeps the private cryptographic key secret and may publish the public cryptographic key to be included in the autonomous program protocol **152**, at a location on the blockchain **150**, or at a certificate authority. In some embodiments, upon a request from an application publisher **120**, the cryptographic key management engine **225** generates a pair of private-public cryptographic keys and sends the public cryptographic key to be incorporated in an autonomous program protocol **152** to be recorded on a blockchain **150**. The autonomous program protocol **152** may be configured to require verification of the digital signature using the public cryptographic key before all or certain functions in the autonomous program protocol **152** may be called. In some embodiments, multiple private-public cryptographic key pairs may be generated and multiple public cryptographic keys may be saved in the autonomous program protocol **152**. Aggregated signatures may be used for certain functions.

In some embodiments, the access control server **130** may also participate in activities of various blockchains **150**, such as performing transactions on blockchains **150**. For a blockchain **150**, the cryptographic key management engine **225** may maintain one or more private keys to allow the access

control server **130** to generate blockchain addresses of the access control server **130** and to validate that access control server **130** owns the blockchain-based units that are connected to one or more public cryptographic keys of the access control server **130**. In various embodiments, a blockchain address of the access control server **130** may be generated by a series of one or more one-way functions from the public key, which is generated from the private key. The cryptographic key management engine **225** may derive a blockchain address by hashing the public key, adding prefixes, suffixes, and/or versions to the hash or the public key, creating a checksum, encoding a derived result, and truncating the address.

The firewall engine **230** may be part of the access control engine **220** and provide network security for an autonomous program protocol **152** by monitoring and controlling incoming and outgoing network traffic based on one or more security rules that are specified by an application publisher **120**. For example, the autonomous program protocol **152** may be configured to require a digital signature or another suitable authorization label from the access control server **130** in order to invoke a function of the autonomous program protocol **152**. In some embodiments, the network traffic related to the autonomous program protocol **152** is routed to the access control server **130** first for the access control server **130** to monitor the traffic. The access control server **130** may track and filter network traffic based on rules that are determined by the application publisher **120**. The access control server **130** may maintain, in the data store **215**, an access control list that contains a list of permissions associated with the autonomous program protocol **152**. The firewall engine **230** may implement various existing firewall techniques in controlling access to the autonomous program protocol **152**. The firewall engine **230** may implement one or more Internet security protocols, such as transport layer security, and may flag or isolate requests that do not pass the security protocols.

The access control server **130** may include one or more machine learning models **235** that are trained to identify potentially malicious activities, threats, fraudulent transactions, or otherwise non-compliant activities that attempt to access an autonomous program protocol **152**. The access control server **130** may rely on both predetermined security rules that are specified by an application publisher **120** to identify any invalid or unauthorized requests and a machine learning model **235** that predicts whether a request may be non-compliant even if the request complies with the security rules. How the application publisher **120** or the access control server **130** may define what activity is non-compliant may depend on the context of the autonomous program protocol **152**. For example, if the autonomous program protocol **152** is a DeFi application, a machine learning model **235** may be trained to identify potentially fraudulent transactions that may involve maximal extractable value (MEV) transactions, money laundering transactions, or other illegal business activities. In another instance, the autonomous program protocol **152** may be an application that provides utility to a company. A machine learning model **235** may be trained to identify potential Internet attacks such as denial-of-access attacks so that the autonomous program protocol **152** is protected from malicious activities.

A machine learning model **235** may be part of the access control engine **220** and may receive various data and contextual information related to an attempted request for accessing an autonomous program protocol **152** to predict whether the request may be non-compliant. The input of the machine learning model **235** may include the IP address of

the request, the function call in the request, the purported identity of the requestor, parameters used in the request, date and time of the request, frequency of the request, usage patterns of the autonomous program protocol **152**, authentication information of the request, past activities of the requester, past activities of other relevant users, client data (e.g., wallet data, browser data, operating system data), cookies, user behavior on an application frontend, other activities by other users on the blockchain (e.g., to detect correlated attacks), smart contract code (e.g., both source code, if available, and binary code), geographical location estimations from IP addresses, and other suitable information. A machine learning model **235** may be trained using past transaction instances as training samples. For example, the data and contextual information related to past transaction instances may be stored in the data store **215**. Each training sample may be stored as a feature vector that includes the data and contextual information as the dimensions of the vector. Each of the past transactions may be labeled as compliant or noncompliant. In some cases, the training samples may also be multi-classes and are labeled with different noncompliant activities. Each training label may have multiple dimensions. Based on the feature vectors and the training labels of past transaction instances, machine learning model **235** may be trained to predict whether a future request is compliant or non-compliant.

A sandbox engine **240** may be part of the access control engine **220** and may allow a party that attempts to invoke one or more function calls of the autonomous program protocol **152** to simulate the transaction at the access control server **130** first before actually invoking the autonomous program protocol **152**. For example, a party may have a request that is part of a larger algorithm. The request is to be sent to the autonomous program protocol **152** to carry out. The party may use the sandbox engine **240** to simulate the result of the autonomous program protocol **152** carrying out the request and determine whether the result generates the desirable outcome and/or whether the result generates any undesirable side effects. If the result is satisfactory, the party may request the access control server **130** to digitally sign the actual request and have the request sent to the autonomous program protocol **152**.

The authentication engine **245** may be part of the access control engine **220** and may allow an application publisher **120** to request the access control server **130** to carry out authentication procedures before a request for accessing an autonomous program protocol **152** is authorized by the access control server **130**. For example, the application publisher **120** may design and publish an autonomous program protocol **152** that is reserved for only certain account holders of the application publisher **120**. To prevent an unauthorized party from gaining access to the autonomous program protocol **152**, the access control server **130** may carry an authentication process such as verifying the credential of the requester before the access control server **130** authorizes a request for the autonomous program protocol **152**. The authentication engine **245** may provide any suitable types of authentication procedures such as two-factor authentication. For example, upon a request is received and the credential is verified, the authentication engine **245** may generate a token code for the requester. The authentication engine **245** may set a time limit for the requester to enter the token code before the authentication engine **245** generates a digital signature to authorize the request.

The autonomous program protocol building engine **250** may be an engine that assists an application publisher **120** in building an autonomous program protocol **152** that incor-

13

porates various access control features of the access control server **130** into the autonomous program protocol **152**. The autonomous program protocol building engine **250** may also be used to construct an access control autonomous protocol **154**. The autonomous program protocol building engine **250** may allow the application publisher **120** to build an autonomous program protocol **152** such as a smart contract or a Web3 application on the platform provided by the access control server **130** and automatically generate the code that enables the autonomous program protocol **152** to incorporate the access control feature. The autonomous program protocol building engine **250** may include compiler, simulation, and debugging features that allow the application publisher **120** to test and simulate the autonomous program protocol **152** before the autonomous program protocol **152** is recorded on a blockchain. The autonomous program protocol building engine **250** may also publish the finalized autonomous program protocol **152** on behalf of the application publisher **120** on a blockchain **150**. In some embodiments, after the code for autonomous program protocol **152** is written, the autonomous program protocol building engine **250** may cause the cryptographic key management engine **225** to generate a new pair of private-public cryptographic keys and store the public cryptographic key as part of the code or a mutable portion (e.g., a variable) of the autonomous program protocol **152**. The application publisher **120** may design the autonomous program protocol **152** with multiple function calls. The application publisher **120** may specify which function calls are subject to the access control of the access control server **130**. The autonomous program protocol building engine **250** may incorporate the code that requires the autonomous program protocol **152** to use the public cryptographic key to verify the digital signature of the access control server **130** before a function call is invoked. The access control part of the code may be generated automatically by autonomous program protocol building engine **250** or by having the application publisher **120** include a code library published by the access control server **130** and inserting the access control code in the source code of the autonomous program protocol **152**. Likewise, an access control autonomous protocol **154** may be published and may be used to store a public cryptographic key of the access control server **130** in a manner similar to the discussion above.

In various embodiments, the public cryptographic key may be stored in the autonomous program protocol **152** or the access control autonomous protocol **154** in different manners. While an autonomous program protocol **152** is used in the discussion below, similar features may be applied to an access control autonomous protocol **154**. In some embodiments, the public cryptographic key may be stored as part of the immutable code of the autonomous program protocol **152**. In some embodiments, the public cryptographic key may be stored as a variable that can only be changed by the original owner who published the autonomous program protocol **152**. For example, the autonomous program protocol **152** may include an initial function such as a constructor function that is only called when the autonomous program protocol **152** is first recorded on a blockchain **150**. The constructor function may define the original owner that is tractable to a wallet address. The original owner, who possesses the wallet address, may have the authority to upload a public cryptographic key and modify the public cryptographic key for key rotation purposes or for mitigation of providers of access control server **130**. An example relevant part of the pseudocode of the autonomous program

14

protocol **152** for implementing the public cryptographic key as a variable for the autonomous program protocol **152** is shown below.

```

contract TestContract {
  address signingAuthorityKey;
  // Signing Authority Key is the public cryptographic key of access
  control
  // server 130.
  address owner;
  modifier onlyOwner( ) {
    require(msg.sender == owner, "Action is not permitted.");
  }
  constructor( ) {
    owner = msg.sender;
  }
  // This function would be called by the owner (developer / deployer)
  // when the contract is deployed.
  // The parameter would be the public cryptographic key.
  function setSigningAuthorityKey(address signingAuthority) public
  onlyOwner {
    signingAuthorityKey = signingAuthority;
  }
}

```

The autonomous program protocol building engine **250** may generate the access control part of the autonomous program protocol **152** and also an interface for accessing the autonomous program protocol **152**. The interface for accessing the autonomous program protocol **152** may be an application **122**, an autonomous application **124**, an oracle machine, or another suitable way to interact with the autonomous program protocol **152**. For example, the interface may include code that routes any request attempting to reach the autonomous program protocol **152** to access control server **130** first to receive a digital signature from the access control server **130** that indicates the request is authorized by the access control server **130**. Upon the receipt of the digital signature, the interface may forward the request for the requester to sign. The user's application (e.g., a wallet) may then send the request to autonomous program protocol **152**. In some embodiments, an access control autonomous protocol **154** may be used for the access control verification. The access control autonomous protocol **154** may include the control component that is used to verify the digital signature of the access control server **130**. In turn, the access control autonomous protocol **154** may store verified proof of authorization in its ledger and the autonomous program protocol **152** may retrieve the proof from the ledger of the access control autonomous protocol **154** to determine access privileges.

The access control server **130** may include one or more front-end interfaces **255**. A front-end interface **255** allows application publishers **120** to manage their profiles, build autonomous program protocol **152**, and manage settings related to access control and security level of the autonomous program protocol **152** published by the application publisher **120**. The front-end interface **255** may take different forms. A first example of front-end interface **255** is a software application interface that is installed on a user device **110** such as a smartphone and computer. A second example of front-end interface **255** is a webpage interface of the access control server **130** that allows users to manage their accounts through web browsers. A third example front-end interface **255** is an application program interface (API) of the access control server **130** that allows users to perform actions through program codes and algorithms.

15

The communication terminal **260** of the access control server **130** provides network and blockchain connections between the access control server **130** and various entities that communicate with the access control server **130**. The access control server **130** may serve as a node of various public blockchains to provide up-to-date information about the state of the blockchain. The access control server **130** may include different terminals such as blockchain terminal, asset exchange terminal, and messaging application terminal. Each terminal may manage a data feed or a webpage that publishes information regarding the related services and server status. Each terminal may also include its individual API.

The blockchain interfacing engine **275** provides various functionalities for the access control server **130** to perform activities on different blockchains **150** that may have their own standards and protocols. The access control server **130** may serve as a node of a blockchain **150** to participate in the mining and data validation process. The blockchain interfacing engine **275** allows access control server **130** to broadcast various transactions to a blockchain network for recordation. For example, the blockchain interfacing engine **275** may publish autonomous program protocol **152** on behalf of an application publisher **120**, such as in the situation where the application publisher **120** uses autonomous program protocol building engine **250** to build the autonomous program protocol **152**. The blockchain interfacing engine **275** also routinely checks new blocks generated in various blockchains to check whether pending blockchain transactions or actions have been confirmed on the blockchains **150**. The blockchains **150** may include public blockchains, consortium blockchains, and private blockchains. The degree of decentralization of various blockchains **150** may vary. In one embodiment, the access control server **130** may set the standard and publish its own blockchain **150** that allows the public to participate in the blockchain network.

The blockchain interfacing engine **275** may include a smart contract engine that manages the generation and triggering of various smart contracts that are recorded on different blockchains. A smart contract may be created through a particular programming language that is compatible with a blockchain **150**. A smart contract is recorded on a block of the blockchain and may be immutable. The recorded smart contract may include executable code instructions that are triggered by a certain condition. When the condition is met and verified, the code instructions are executed by a computer to automatically execute the contract terms that take the form of code instructions. The computer that executes the smart contract may take various forms. For example, a computer described herein may be a conventional personal computer, a virtual machine for the blockchain, or even a collection of distributed nodes in distributed computing. When the code instructions of the smart contract are executed, the code instructions may cause certain events (e.g., a transaction, a generation of a token, creation of new information) to be recorded on a blockchain. In some embodiments, after a request to access an autonomous program protocol **152** is authorized by the access control server **130**, instead of transmitting the digital signature back to the requester, the access control server **130** may directly communicate to the autonomous program protocol **152**, such as a smart contract, to initiate the request.

The blockchain interfacing engine **275** may also include an oracle machine that may serve as a data feed for an autonomous program protocol **152**. The oracle machine may receive different data from various sources. For example,

16

different parties may provide information and data to the oracle machine. When relevant information is obtained by the oracle machine, some code instructions of the autonomous program protocol **152** may be triggered if certain conditions are met.

Example Access Control System

FIG. 3 is a block diagram illustrating an example access control system **300** and the message control flow of the system, in accordance with some embodiments. The access control system **300** may be an example of the system environment **100**. The access control system **300** may include an application **310**, the access control server **130**, and an autonomous program protocol **152** recorded on the blockchain **150**. The access control system **300** may also include other applications **320** and other program protocols **330** recorded on the blockchain **150**.

The application **310** may be an example of application **122** or autonomous application **124**, such as a Web3 application. The application **310** may serve as an interface for a party to interact with the autonomous program protocol **152**. For example, a user may manually request to initiate an action at the autonomous program protocol **152** through an application **122**. An autonomous agent may initiate a request through the autonomous application **124**. The application **310** may include the core code **312** which is largely designed by the application publisher **120** and serve as the primary features of the application **310**. The application **310** may also include access control code **314** that may be generated by the access control server **130** and control the routing of requests so that the application **310** can communicate with the autonomous program protocol **152** under the access control framework designed by the access control server **130**.

The core code **312** may generate a request **316** (e.g., “SmartContracts.methods.setName(“NewName”).send()”) directed to the autonomous program protocol **152**. The request may also be referred to as an interaction request. The request **316** may include a specific function call of the autonomous program protocol **152** such as “setName” in this example. The access control code **314** may package the function call data together with client data (e.g., user’s behavior data, etc.), route the request **316** to the access control server **130** and request the information and digital signature from the access control server **130**. Packaging the function call data may include extracting the functions and the parameters included in the functions and hashing the information. In some embodiments, Packaging the function call may also include adding context metadata to the request **316**. The access control code **314** causes the application **310** to route the request **316** to the access control server **130**.

Upon receiving the request **316**, the access control server **130** may analyze the request **316** using the access control engine **220** to determine whether the request **316** is in compliance with access control policies set by the application publisher **120**. The analysis may include determining whether the request **316** is authenticated and authorized. The types of analyses that may be performed by access control engine **220** are discussed in further detail in FIG. 2. The access control engine **220** may deploy the firewall engine **230**, the machine learning model **235**, the sandbox engine **240**, the authentication engine **245** and any other suitable access control protocols to analyze the request **316**. The access control engine **220** in turn determines whether to authorize the request.

If the access control engine **220** authorizes the request, the access control server **130** may use the cryptographic key management engine **225** to generate a digital signature **340**

17

of the access control server 130 to signify the authorization. The access control server 130 may use a private cryptographic key to sign a version of the request 316. The version of the request 316 may be the request 316 itself, a hash of the request 316, the request 316 with context data. For example, the access control server 130 may use the private cryptographic key to encrypt a version of the request 316 to generate the digital signature 340. The access control server 130 may generate a response 350 for the authorization. The response 350 may include the request 316, context data 352, and the digital signature 340. The response 350 may be transmitted back to the application 310 or transmitted directly to the blockchain 150 to serve as an authorized request. If the response 350 is returned to the application 310, the access control code 314 of the application 310 may cause the response 350 to be transmitted to the blockchain 150.

If the access control engine 220 does not authorize the request 316, the access control server 130 may simply ignore the request 316 or send a simple response to the application 310 that the request 316 is denied. In some cases where the access control server 130 determines that the request 316 may be transmitted by a malicious party, the access control server 130 may also add the requester or an identifier of the application 310 (e.g., IP address, application identifier) to a blocked list.

Upon receiving the response 350 that includes the digital signature 340, the autonomous program protocol 152 may verify the digital signature 340 and execute the function call specified in the request 316. For example, the autonomous program protocol 152 may include core code 360 and access control code 362. Similar to the core code 312, the core code 360 may be largely designed by the application publisher 120 and serve as the primary function of the autonomous program protocol 152. The access control code 362 may be generated by the access control server 130 and enable the access control of the autonomous program protocol 152. For example, the access control code 362 may store a copy of the public cryptographic key that corresponds to the private cryptographic key used to generate the digital signature 340. The access control code 362 uses the public cryptographic key to decrypt the digital signature 340 and verify the digital signature 340. If the digital signature 340 is verified, the autonomous program protocol 152 will carry out the function call and execute the function in the core code 312 in response to the request 316.

The access control system allows an application publisher 120 to control the access to the autonomous program protocol 152 stored on the blockchain 150. Other applications 320 or other program protocols 330 recorded on the blockchain 150 may not be able to directly communicate or cause the autonomous program protocol 152 to perform any actions without authorization from the access control server 130.

Example on-Chain Proof

FIG. 4A through FIG. 4C illustrate various embodiments of using an access control autonomous protocol 154 to provide an access control feature for a protected autonomous program protocol 152. The protected autonomous program protocol 152 may be referred to as a protected smart contract. While one or more access control features may be hosted by the access control autonomous protocol 154, in some embodiments, the protected autonomous program protocol 152 may also be accessed through mechanisms other than the access control feature provided by the access control autonomous protocol 154. An example of a protected autonomous program protocol 152 may be any token con-

18

tract. In this way, the protected autonomous program protocol 152 may be used by users who do not use access control provided by the access control server 130. Optionally, the protected autonomous program protocol 152 may include the access control features provided by the access control server 130.

In some embodiments, the use of a protected autonomous program protocol 152 may include the access verification process and the transaction. In some embodiments, both the access verification process and the transaction may be conducted through the protected autonomous program protocol 152. This type of embodiment is illustrated in FIG. 4A. In some embodiments, the access verification process may be conducted through the access control autonomous protocol 154. After verification, the user may access the protected autonomous program protocol 152 through another mechanism outside of the access control autonomous protocol 154. This type of embodiment is illustrated in FIG. 4B. In some embodiments, two or more protected autonomous program protocols 152 may perform transactions. Each of the protected autonomous program protocols 152 may be protected by an access control autonomous protocol 154. This type of embodiment is illustrated in FIG. 4C.

Various embodiments illustrate protocols for requesting approvals via signature and hash validation in a secure and reliable way. The solution leverages client-side features provided by the access control server 130 and contract integrations to communicate and interact with any type of contract on a Marketplace, DEX, and DeFi to build and verify the transactions without impeding normal or expected operational workflow of transactions for these platforms.

In some embodiments, a standard smart contract (an example of protected autonomous program protocol 152) is extended with modifier calls within the functions of the smart contract to validate signatures and hashes. By updating these functions, the smart contract remains compatible with existing platforms. External verification of the transactions being generated against the smart contract can be added using a proxy smart contract (an example of access control autonomous protocol 154).

In some embodiments, the components illustrated in FIG. 4A through FIG. 4C may include an access requester 402, an application 122, the access control server 130, a protected autonomous program protocol 152 and the access control autonomous protocol 154. In various embodiments, one or more components may be added, removed, or modified.

The access requester 402 may be the requester who intends to use the protected autonomous program protocol 152 to perform a transaction. The transaction can be of any nature, depending on the functionality of the protected autonomous program protocol 152. The access requester 402 can be a natural person such as an end user who intends to use a smart contract that resides on the blockchain 150, a robotic agent, an organization, a blockchain protocol, or any suitable entity.

An application 122 may be an application that is published and provided by the access control server 130 for providing one or more access control features. The application 122 may serve as an interface among the access requester 402, the blockchain 150, and the access control server 130. For example, the application 122 may be equipped with typical functionalities of a blockchain node so that the application 122 may be used to broadcast transactions on behalf of the access requester 402. The application 122 may also include access control features of the access control server 130 so that the access requester 402 may be authenticated and/or authorized through the application 122

before the access requester **402** may access the protected autonomous program protocol **152**. The extent of access control management provided by the application **122** may vary in different embodiments. For example, in some embodiments, the application **122** may serve as an authenticator to authenticate the access requester **402**. In some embodiments, the application **122** may provide wallet features and help the access requester **402** to manage blockchain identity such as the blockchain address and, in some cases, the private key of the access requester **402**. In some embodiments, the application **122** may not provide any wallet features and route any identity and authorization requests to the access control server **130** for the determination of access control.

The access control server **130** may provide various access control features that are further discussed in FIGS. **1** and **2**. In some embodiments, the access control may be as simple as an authentication so that the protected autonomous program protocol **152** is no longer permissionless and requires authentication verification before the protected autonomous program protocol **152** can be accessed. In some embodiments, the access control may include authorization based on policies that are set by the publisher or administrator of the protected autonomous program protocol **152**. For example, the access control server **130** may provide a user interface for the administrator to set up policies on how the protected autonomous program protocol **152** may be accessed. The policies may include access privileges, restrictions, conditions, and other parameters. The policies may be role based, identity based, transaction parameter specific, temporal based, context based, or any other factors based, whether the factors are static or dynamic, deterministic or probabilistic, domain specific or not. In some embodiments, the access control server **130** may also apply a machine learning model discussed in FIG. **6** to determine whether an access requester **402** is authorized to access the protected autonomous program protocol **152**.

A protected autonomous program protocol **152** may be any suitable autonomous program protocol **152** that is recorded on the blockchain **150**. The protected autonomous program protocol **152** may include two or more components, such as an access control component **420** and a protected functionality component **422**. The access control component **420** may be a component that is included to allow the autonomous program protocol **152** to limit access. This component may be added to the protected autonomous program protocol **152** after the autonomous program protocol **152** is first published. For example, the autonomous program protocol **152** may be a common smart contract recorded on a public blockchain **150** such as ETHERUM. The autonomous program protocol **152** may be updated to add the access control component **420** to allow the autonomous program protocol **152** to include some of the access control features provided by the access control server **130**. In some embodiments, the access control component **420** may be an updated component that is added after the first launch of the protected autonomous program protocol **152**. The updated component may include a protocol that is configured to communicate with the access control autonomous protocol **154**.

The access control autonomous protocol **154** may be a proxy autonomous protocol recorded on the blockchain **150** for providing access control features for one or more protected autonomous program protocols **152**. In some embodiments, the access control autonomous protocol **154** may include access control features for multiple autonomous program protocols **152**. In some embodiments, both the

access control autonomous protocol **154** and the protected autonomous program protocol **152** may be smart contracts that are recorded on a public blockchain such as ETHERUM.

The access control autonomous protocol **154** may be associated with a blockchain address such that the access control autonomous protocol **154** is identifiable and accessible by other autonomous program protocols **152** using the blockchain address. The access control autonomous protocol **154** may include two or more components, such as a checker **410** and a ledger **412**. The checker **410** may include a set of protocols, when executed by a node of the blockchain **150**, such as a virtual machine **156** of the blockchain **150**, to verify proofs of authorizations that are transmitted to the access control autonomous protocol **154**. For example, the proof of authorization may include a cryptographic signature of the access control server **130** so that the checker **410** may verify the proof of authorization. The proof of authorization may also be time-bound or may include a hash of the transaction as part of the cryptograph signature so that the specific transaction requested by the access requester **402** may also be verified. In some embodiments, the checker **410** may store the public cryptographic key corresponding to the private cryptographic key possessed by the access control server **130** and use the public cryptographic key to verify any digital signatures of the access control server **130**.

The ledger **412** may include the proofs of authorization **416** of various access requesters **402**. The ledger **412** may be accessible by the blockchain address of the access control autonomous protocol **154**. The ledger **412** may include sub-ledgers that each corresponds to a protected autonomous program protocol **152**. For example, in some embodiments, the ledger **412** may store a first set of records of proofs of authorization **416** for a first protected autonomous program protocol **152** and a second set of records of proofs of authorization **416** for a second protected autonomous program protocol **152**. The protected autonomous program protocols **152** may use the blockchain address of the access control autonomous protocol **154** to gain access to the ledger **412** to retrieve proofs of authorization **416** to confirm that a transaction request from an access requester **402** is authorized before carrying out the transaction.

In various embodiments, proofs of authorization **416** may take different forms and may include different data. The proofs of authorization **416** may also be referred to as verification proofs, which provide proofs that the transaction requests or the access requesters **402** have been verified. In some embodiments, proof of authorization **416** may include a cryptographic signature of the access control server **130** and a hash of a requested transaction. The cryptographic signature may be signed by a private cryptographic key possessed by the access control server **130**. The hash of the requested transaction may be included in the cryptographic signature so that the requested transaction may be verified. In some embodiments, proof of authorization **416** may include a cryptographic signature of the access control server **130** and an identifier of the access requester **402**, such as a blockchain address or a version of a public cryptographic key of the access requester **402**. In this type of proof of authorization **416**, the access requester **402** may have access to the protected autonomous program protocol **152** for multiple transactions until the access privilege is revoked. In some embodiments, proof of authorization may be time-bound (e.g., having a timestamp) so that the requested transaction is required to be completed within a limited time or the access requester **402** may conduct any transactions within the limited time. In some embodiments,

21

proof of authorization may be a zero-knowledge proof that can be verified by a protected autonomous program protocol 152.

FIG. 4A is a block diagram illustrating a process 400 for verifying access control using an access control autonomous

protocol 154, in accordance with some embodiments. In step 1, an access requester 402, such as a wallet user, may send a transaction request to an application 122, which may be a software application, a distributed application or another suitable application with integration of features provided by the access control server 130.

In step 2, the application 122 may send the parameters of the request to the access control server 130 for verification. The parameters of the request may include the parameters of the access requester 402 and/or the parameters of the requested transactions. In some embodiments, the process 400 may grant the access requester 402 repeated access privileges based on one request, and the access privilege may not be transaction-specific. In such embodiments, the request parameters may primarily focus on the access requester 402, such as including the authentication credentials of the access requester 402, the roles of the access requester 402, and other suitable identifiers. In some embodiments, the process 400 may grant a transaction-specific access privilege to the access requester 402. In such embodiments, the parameters may also include the parameters of the transactions, such as the nature of the transactions, metadata, etc.

The access control server 130 may evaluate the parameters and determine whether the access requester 402 is authorized to access the protected autonomous program protocol 152. The review process may depend on the access policies specific to the protected autonomous program protocol 152. In some embodiments, the access control server 130 may use one or more machine learning models to determine the access authority of the access requester 402, including determining the security risk of a transaction. If the request is found valid and/or there are no security issues for compliance, the access control server 130 may provide approval. The approval may include a cryptographic signature of the access control server 130 and, in some situations, a hash of the transaction.

In step 3, the application 122 may transmit the approval 404 of the access control server 130 to the checker component 410 of the access control autonomous protocol 154.

In step 4, the access control autonomous protocol 154 may validate on-chain the signature of the access control server 130. The on-chain validation may be conducted through the block generation process of the blockchain 150. The protocols of the checker 410 are executed in a new block generation process to validate the signature. In some embodiments, the access control autonomous protocol 154 may include the public cryptographic key of the access control server 130. Here, the inclusion of the public cryptographic key may refer to the actual copy of the public cryptographic key or a version of the public cryptographic key, such as a pointer of the public cryptographic key. The protocols of the checker 410 may use the public cryptographic key to decrypt the digital signature in the approval 404 to verify the signature of the access control server 130. Upon verification of the signature, the access control autonomous protocol 154 may store proof of authorization 416 in the ledger 412. The storage of the proof of authorization 416 may be carried out as a new block of the blockchain 150 is generated.

In step 5, the application 122 may send an access request 424 to the protected autonomous program protocol 152. The

22

access request 424 may include a transaction, a hash of the transaction, the blockchain address associated with the access requester 402, and other related data. In some embodiments, the hash of the transaction may be identical to the hash in the proof of authorization 416 stored in the ledger 412 of the access control autonomous protocol 154 so that the transaction can be verified. In some embodiments, the access request 424 may be transmitted in different manners. For example, in some embodiments, the access request 424 may be transmitted directly to the protected autonomous program protocol 152, as shown in the step 5a. In some embodiments, the access request 424 may be transmitted to another autonomous program protocol 152, such as a standard autonomous program protocol 152 recorded on the blockchain 150, as shown in step 5b. The other autonomous program protocol 152 will in turn route the 424 to the protected autonomous program protocol 152.

In step 6, the access request 424 may be transmitted to the access control component 420 of the autonomous program protocol 152.

In step 7, the access control component 420 may ping the access control autonomous protocol 154 using the blockchain address of the access control autonomous protocol 154 to retrieve the proofs of authorization 416 stored in the ledger 412. The access control component 420 may go through the retrieved proofs of authorization 416 to find and verify whether there is proof of authorization 416 that grants access to the access requester 402. For example, if the proof of authorization is transaction-specific, the access control component 420 may verify that the transaction hash received by the access control component 420 matches the transaction hash stored in proof of authorization 416. The autonomous program protocol 152 may also review its own ledger to determine if the transaction has previously been executed. If the transaction hashes match and the transaction has not been executed, the access control component 420 sends an approval of the transaction to the protected functionality component 422 of the protected autonomous program protocol 152. In another example, if the proof of authorization 416 grants a general access privilege to the access requester 402 for a limited period, the proof of authorization 416 may store the public cryptographic key or a version of the key of the access requester 402. The access request 424 may include a digital signature of the access requester 402 who possesses the corresponding private cryptographic key. The access control component 420 may use the public cryptographic key retrieved from the ledger 412 to decrypt the digital signature in the access request 424 to verify that the access requester 402 is authorized to use the protected autonomous program protocol 152. Such authorization may be time bound and the access control component 420 may also verify the time limit.

In step 8, the protected autonomous program protocol 152, upon verification of the access request 424, may execute the protected functionality 422. The protected functionality 422 can be of any nature and will depend on the code and protocols within the protected autonomous program protocol 152. In some embodiments, any smart contracts recorded on the blockchain 150 can be the protected autonomous program protocol 152.

FIG. 4B is a block diagram illustrating a process 440 for verifying access control using an access control autonomous protocol 154 and for carrying out a transaction using a different application 122, in accordance with some embodiments. The process 440 differs from the process 400 in that the access requester 402 uses an access control application 442 to verify the access privilege to the protected autonomous

mous program protocol 152 and uses another application 122 to carry out the transaction. The access control application 442 may be an example of a special type of application 122.

In step 1, an access requester 402 may send a transaction request to the access control application 442, which may be a software application, a distributed application or another suitable application with integration of features provided by the access control server 130.

In step 2, the access control application 442 may send the parameters of the request to the access control server 130 for verification. The parameters of the request may include the parameters of the access requester 402 and/or the parameters of the requested transactions. In some embodiments, the process 400 may grant the access requester 402 repeated access privileges based on one request, and the access privilege may not be transaction-specific. In such embodiments, the request parameters may primarily focus on the access requester 402, such as including the authentication credentials of the access requester 402, the roles of the access requester 402, and other suitable identifiers. In some embodiments, the process 400 may grant a transaction-specific access privilege to the access requester 402. In such embodiments, the parameters may also include the parameters of the transactions, such as the nature of the transactions, metadata, etc.

The access control server 130 may evaluate the parameters and determine whether the access requester 402 is authorized to access the protected autonomous program protocol 152. The review process may depend on the access policies specific to the protected autonomous program protocol 152. In some embodiments, the access control server 130 may use one or more machine learning models to determine the access authority of the access requester 402, including determining the security risk of a transaction. If the request is found valid and/or there are no security issues for compliance, the access control server 130 may provide approval. The approval may include a cryptographic signature of the access control server 130 and, in some situations, a hash of the transaction.

In step 3, the application 122 may transmit the approval 404 of the access control server 130 to the checker component 410 of the access control autonomous protocol 154.

In step 4, the access control autonomous protocol 154 may validate on-chain the signature of the access control server 130. The on-chain validation may be conducted through the block generation process of the blockchain 150. The protocols of the checker 410 are executed in a new block generation process to validate the signature. In some embodiments, the access control autonomous protocol 154 may include the public cryptographic key of the access control server 130. Here, the inclusion of the public cryptographic key may refer to the actual copy of the public cryptographic key or a version of the public cryptographic key, such as a pointer of the public cryptographic key. The protocols of the checker 410 may use the public cryptographic key to decrypt the digital signature in the approval 404 to verify the signature of the access control server 130. Upon verification of the signature, the access control autonomous protocol 154 may store proof of authorization 416 in the ledger 412. The storage of the proof of authorization 416 may be carried out as a new block of the blockchain 150 is generated.

While the process 440 may be used for a transaction-specific proof of authorization 416, the process 440 is particularly suitable for granting a general access privilege to an access requester 402. For example, in step 5, the access

requester 402, after access is granted and recorded on the ledger 412, may now use any application 122 to conduct transactions with the protected autonomous program protocol 152. For example, the access requester 402 may use an application 122 to compose a transaction.

In step 6, the application 122 may send an access request 424 to the protected autonomous program protocol 152. The access request 424 may include a transaction, a hash of the transaction, the blockchain address associated with the access requester 402, and other related data. In some embodiments, the hash of the transaction may be identical to the hash in the proof of authorization 416 stored in the ledger 412 of the access control autonomous protocol 154 so that the transaction can be verified. In some embodiments, the access request 424 may be transmitted in different manners. For example, in some embodiments, the access request 424 may be transmitted directly to the protected autonomous program protocol 152, as shown in the step 6a. In some embodiments, the access request 424 may be transmitted to another autonomous program protocol 152, such as a standard autonomous program protocol 152 recorded on the blockchain 150, as shown in the step 6b. The other autonomous program protocol 152 will in turn route the 424 to the protected autonomous program protocol 152.

In step 7, the access request 424 may be transmitted to the access control component 420 of the autonomous program protocol 152.

In step 8, the access control component 420 may ping the access control autonomous protocol 154 using the blockchain address of the access control autonomous protocol 154 to retrieve the proofs of authorization 416 stored in the ledger 412. The access control component 420 may go through the retrieved proofs of authorization 416 to find and verify whether there is proof of authorization 416 that grants access to the access requester 402. For example, if the proof of authorization is transaction-specific, the access control component 420 may verify that the transaction hash received by the access control component 420 matches the transaction hash stored in proof of authorization 416. The autonomous program protocol 152 may also review its own ledger to determine if the transaction has previously been executed. If the transaction hashes match and the transaction has not been executed, the access control component 420 sends an approval of the transaction to the protected functionality component 422 of the protected autonomous program protocol 152. In another example, if the proof of authorization 416 grants a general access privilege to the access requester 402 for a limited period, the proof of authorization 416 may store the public cryptographic key or a version of the key of the access requester 402. The access request 424 may include a digital signature of the access requester 402 who possesses the corresponding private cryptographic key. The access control component 420 may use the public cryptographic key retrieved from the ledger 412 to decrypt the digital signature in the access request 424 to verify that the access requester 402 is authorized to use the protected autonomous program protocol 152. Such authorization may be time bound and the access control component 420 may also verify the time limit.

In step 9, the protected autonomous program protocol 152, upon verification of the access request 424, may execute the protected functionality 422. The protected functionality 422 can be of any nature and will depend on the code and protocols within the protected autonomous program protocol 152. In some embodiments, any smart contracts recorded on the blockchain 150 can be the protected autonomous program protocol 152.

FIG. 4C is a block diagram illustrating a process 450 for verifying access control using an access control autonomous protocol 154 for transactions that involve multiple protected autonomous program protocols 152, in accordance with some embodiments. The process 450 differs from the process 400 in that two or more protected autonomous program protocols 152 are involved in carrying out the transaction.

In step 1, an access requester 402 may send a transaction request to an access control application 442, which may be a software application, a distributed application or another suitable application with integration of features provided by the access control server 130.

In step 2, the access control application 442 may send the parameters of the request to the access control server 130 for verification. The parameters of the request may include the parameters of the access requester 402 and/or the parameters of the requested transactions. In some embodiments, the process 400 may grant the access requester 402 repeated access privileges based on one request, and the access privilege may not be transaction-specific. In such embodiments, the request parameters may primarily focus on the access requester 402, such as including the authentication credentials of the access requester 402, the roles of the access requester 402, and other suitable identifiers. In some embodiments, the process 400 may grant a transaction-specific access privilege to the access requester 402. In such embodiments, the parameters may also include the parameters of the transactions, such as the nature of the transactions, metadata, etc.

The access control server 130 may evaluate the parameters and determine whether the access requester 402 is authorized to access the protected autonomous program protocol 152. The review process may depend on the access policies specific to the protected autonomous program protocol 152. In some embodiments, the access control server 130 may use one or more machine learning models to determine the access authority of the access requester 402, including determining the security risk of a transaction. If the request is found valid and/or there are no security issues for compliance, the access control server 130 may provide approval. The approval may include a cryptographic signature of the access control server 130 and, in some situations, a hash of the transaction.

In some embodiments, since two protected autonomous program protocols 152 are involved, two approvals 404 are generated, each may include a transaction hash and a digital signature.

In step 3, the application 122 may transmit the approvals 404 of the access control server 130 to the checker component 410 of the access control autonomous protocol 154.

In step 4, the access control autonomous protocol 154 may validate on-chain the signatures of the access control server 130. The on-chain validation may be conducted through the block generation process of the blockchain 150. The protocols of the checker 410 are executed in a new block generation process to validate the signature. In some embodiments, the access control autonomous protocol 154 may include the public cryptographic key of the access control server 130. Here, the inclusion of the public cryptographic key may refer to the actual copy of the public cryptographic key or a version of the public cryptographic key, such as a pointer of the public cryptographic key. The protocols of the checker 410 may use the public cryptographic key to decrypt the digital signature in the approval 404 to verify the signature of the access control server 130. Upon verification of the signature, the access control autonomous protocol 154 may store proofs of authorization

416 in the ledger 412. The storage of the proofs of authorization 416 may be carried out as a new block of the blockchain 150 is generated. In some embodiments, since two protected autonomous program protocols 152 are involved, two proofs of authorization 416 may be stored.

In step 5, the access requester 402, after access is granted and recorded on the ledger 412, may now use an application 122 to conduct a transaction that involves two or more protected autonomous program protocols 152. For example, the access requester 402 may use an application 122 to compose a transaction. The application 122 may be a different application than the access control application 442 or may be the same application.

In step 6, the application 122 may send an access request 424 to the protected autonomous program protocol 152. The access request 424 may include a transaction, a hash of the transaction, the blockchain address associated with the access requester 402, and other related data. In some embodiments, the hash of the transaction may be identical to the hash in the proof of authorization 416 stored in the ledger 412 of the access control autonomous protocol 154 so that the transaction can be verified. In some embodiments, the access request 424 may be transmitted in different manners. For example, in some embodiments, the access request 424 may be transmitted directly to a first protected autonomous program protocol 152, as shown in the step 6a. In some embodiments, the access request 424 may be transmitted to a second protected autonomous program protocol 152, as shown in the step 6b. The second protected autonomous program protocol 152 will in turn route the 424 to the first protected autonomous program protocol 152.

In step 7, the access request 424 transmitted to the second protected autonomous program protocol 152 may be verified by the second autonomous program protocol 152. The verification step conducted by the second protected autonomous program protocol 152 may be very similar to the verification step 9 conducted by the first protected autonomous program protocol 152.

In step 8, the access request 424 may be transmitted to the access control component 420 of the first protected autonomous program protocol 152.

In step 9, the access control component 420 may ping the access control autonomous protocol 154 using the blockchain address of the access control autonomous protocol 154 to retrieve the proofs of authorization 416 stored in the ledger 412. The access control component 420 may go through the retrieved proofs of authorization 416 to find and verify whether there is proof of authorization 416 that grants access to the access requester 402. For example, if the proof of authorization is transaction-specific, the access control component 420 may verify that the transaction hash received by the access control component 420 matches the transaction hash stored in proof of authorization 416. The autonomous program protocol 152 may also review its own ledger to determine if the transaction has previously been executed. If the transaction hashes match and the transaction has not been executed, the access control component 420 sends an approval of the transaction to the protected functionality component 422 of the protected autonomous program protocol 152. In another example, if the proof of authorization 416 grants a general access privilege to the access requester 402 for a limited period, the proof of authorization 416 may store the public cryptographic key or a version of the key of the access requester 402. The access request 424 may include a digital signature of the access requester 402 who possesses the corresponding private cryptographic key. The access control component 420 may use the public crypto-

27

graphic key retrieved from the ledger **412** to decrypt the digital signature in the access request **424** to verify that the access requester **402** is authorized to use the protected autonomous program protocol **152**. Such authorization may be time bound and the access control component **420** may also verify the time limit.

In step **10**, the protected autonomous program protocol **152**, upon verification of the access request **424**, may execute the protected functionality **422**. The protected functionality **422** can be of any nature and will depend on the code and protocols within the protected autonomous program protocol **152**. In some embodiments, any smart contracts recorded on the blockchain **150** can be the protected autonomous program protocol **152**.

Example Access Control Process

FIG. **5** is a flowchart depicting an example process **500** for providing access control on a protected autonomous program protocol **152**, in accordance with some embodiments. The process **500** may be performed by a computing device, such as an access control server **130**, and an access control autonomous protocol **154**. The process **500** may be embodied as a software algorithm that may be stored as computer instructions that are executable by one or more processors. The instructions, when executed by the processors, cause the processors to perform various steps in the process **500**. The processes **400**, **440**, and **450** are examples of the process **500**.

In some embodiments, the access control server **130** may store **510** a private cryptographic key. The private cryptographic key possessed by the access control server **130** may correspond to a public cryptographic key that may be stored in an access control autonomous protocol **154**. For example, the access control server **130** may generate a pair of cryptographic keys using the Elliptic Curve Digital Signature Algorithm (ECDSA). The private cryptographic key is kept secret by the access control server **130**. The public cryptographic key is published. In some cases, a copy of the public cryptographic key may be stored on the blockchain **150** as part of the access control autonomous protocol **154**. In some embodiments, the access control server **130** may use a multi-party computation technique to store the private cryptographic key. For example, a key shard may be stored and may be used to compute the private cryptographic key when used.

In some embodiments, the access control server **130** may also receive access control settings related to a protected autonomous program protocol **152**. The access control setting may specify one or more policies in granting and access requester **402** access to the protected autonomous program protocol **152**. The access control setting may include settings with respect to different function calls of the autonomous program protocol **152**. For example, in some embodiments, the autonomous program protocol **152** provides different functions and each function may be associated with a different access control setting so that the security level for each function may be different. In some embodiments, an application publisher **120**, through the interface provided by the access control server **130**, may specify the access control settings for various functions associated with the autonomous program protocol **152**. In some cases, the access control setting with respect to one or more function calls may be unrestricted so that any party can gain access to those unrestricted functions. Additional examples of access control settings are discussed in FIG. **2** in association with the access control engine **220**.

In some embodiments, the access control server **130** may receive **520**, from an access requester **402**, an access request

28

for accessing the protected autonomous program protocol **152** stored on a blockchain **150**. The request may include a function call to the autonomous program protocol **152** and parameters related to the function call. The access request may include various parameters. The request may be routed to the access control server **130** for the access control server **130** to determine whether to authorize the request. Additional information such as context information of the request, and metadata of the request (e.g., IP address of the request sender, parameters of the request) may also be received by the access control server **130**. In some embodiments, the access control server **130** may also receive (e.g., load from storage) previous function call requests transmitted by the same requester and previous function call requests by other users that are relevant to the instant transaction.

In some embodiments, the access control server **130** may analyze **530** parameters related to the access requester to determine whether the access requester is authorized to access the protected autonomous program protocol. For example, the access control server **130** may determine, based on the parameters related to the access requester, that the access request is in compliance with one or more policies specified in the access control setting. In some embodiments, the parameters may be as simple as authentication credentials. In those situations, the access requester **402** may access the protected autonomous program protocol **152** as long as the access requester **402** is authenticated. In some embodiments, the parameters related to the access requester may also include a parameter related to a transaction that the access requester wants to be processed by the protected autonomous program protocol **152**.

The access control server **130** may use tools associated with the access control engine **220**, including firewall engine **230**, machine learning model **235**, sandbox engine **240**, and authentication engine **245** to review the request. The type of review may depend on the situation and the access control policies specified. For example, the access control server **130** may train a machine learning model **235** to identify potential non-compliant items. In some embodiments, the trace of previous calls may be used to identify a malicious party. For example, a fraudulent person who is to commit a fraudulent activity may follow certain patterns. The access control server **130** may identify the sequence of actions. For example, the access control server **130** may identify past function calls by the requester to holistically determine whether the request may be malicious or otherwise non-compliant. For example, the access control server **130** may keep a stack trace of previous function calls per application **310**. In some cases, requests to call a particular function with certain parameters are not inherently malicious but, when combined with another request to another function call with additional parameters, the collection of the requests can become malicious. The access control server **130** may use rule-based models or machine learning models to identify or predict requests that may be non-compliant.

In some embodiments, the application publisher **120** may specify that the autonomous program protocol **152** or certain function calls in the autonomous program protocol **152** are unrestricted or permissionless. This may occur when the autonomous program protocol **152** is first launched or the application publisher **120** adjusts the security and authorization level of the autonomous program protocol **152** through the access control server **130**. For example, the application publisher **120** may decide to open the blockchain **150** to the general public.

The determination may include a strictly rule-based approach (e.g., whether a request passes the authentication

process, whether the request is from an allowlist of IP addresses), a heuristic approach (e.g., using an algorithm that analyzes metadata to predict the nature of the request), a contextual approach (e.g., using contextual data, prior requests, and other factors to determine the nature of the request), a predictive approach (using one or more machine learning models), a simulation approach, or any combination of various approaches. The access control server **130** may train machine learning models to help various publishers identify malicious attacks. An application publisher **120** may also set up rules using the configuration and policy engine **210** to design the determination process.

The access control server **130** may make a determination from a list of possible determination outcomes. For example, the list of possible outcomes may include “allow,” “block,” “suspicious,” “unknown,” and “ignored.” In some embodiments, the possible outcomes may also depend on the kinds of attacks. In some embodiments, for some attacks, the outcomes are binary (sign or not sign) while in other attacks the outcomes may include different possibilities such as likelihoods of fraud, Sybil, side-effects, etc. Depending on the rating of the request determined by the access control server **130**, the access control server **130** may take different actions. For example, an application publisher **120** may specify what actions the access control server **130** should take for rating such as “suspicious” and “unknown.” For example, in some cases, the access control server **130** may be the unknown request as allowed or ignored, depending on the preference of the application publisher **120**. In some cases, for suspicious requests, the access control server **130** may flag the request and notify the application publisher **120** for manual review. Again, how the access control server **130** may handle a rating may be selectable by the application publisher **120**.

In some embodiments, the access control server **130** may generate **540** a cryptographic signature for the access request. The cryptographic signature may be generated by the private cryptographic key. In some embodiments, the access request may include a transaction payload to be transmitted to the protected autonomous program protocol **152** for processing. The cryptographic signature of the proof of authorization may include a hash of the transaction payload.

In some embodiments, the access control server **130** may transmit **550** the cryptographic signature to the blockchain **150** to be recorded. For example, the access control server **130** may broadcast, directly or via an application controlled by the access control server **130**, a blockchain transaction request that asks the blockchain **150** to record the cryptographic signature to the ledger of the access control autonomous protocol **154**. In some embodiments, the access control autonomous protocol **154** may receive **560** the cryptographic signatures.

In some embodiments, the access control autonomous protocol **154** may verify **570** the cryptographic signature. For example, the access control autonomous protocol **154** may be configured to, upon receiving the digital signature, hash the payload of the request (or part of it), and use the public cryptographic key of the access control server **130** to decrypt the digital signature. In some embodiments, the decryption of the digital signature generates the hash of the payload of the request that was hashed by the access control server **130**. The autonomous program protocol **152** compares the hash generated from hashing the payload of the request and the hash generated from the digital signature. If the hashes match, the digital signature is verified.

In some embodiments, the access control autonomous protocol **154** may store **580** proof of authorization in association with the protected autonomous program protocol **152**. For example, the proof of authorization may be stored in a ledger of the access control autonomous protocol **154**. The ledger may be associated with a blockchain address of the access control autonomous protocol **154**.

In some embodiments, the proof of authorization is accessible by the protected autonomous program protocol **152** to verify the access requester **402** is authorized to access the protected autonomous program protocol **152**. In some embodiments, the access control autonomous protocol **154** may be configured to store the hash of the transaction payload. The protected autonomous program protocol **152** is configured to verify the hash of the transaction payload before processing the transaction payload.

In some embodiments, the access control autonomous protocol **154** is configured to store a plurality of proofs of authorization. A first proof of authorization may be associated with authorization for a first protected autonomous program protocol **152** and a second proof of authorization may be associated with authorization for a second protected autonomous program protocol **152** that is different from the first autonomous program protocol **152**. The access control autonomous protocol **154** may be associated with a blockchain address. The first autonomous program protocol **152** and the second autonomous program protocol **152** are configured to retrieve the proofs of authorization stored in the access control autonomous protocol **154** at the blockchain address.

In some embodiments, the proof of authorization may include a second cryptography signature associated with the access requester. In some embodiments, the proof of authorization is timestamped so that the proof of authorization is valid for a predetermined period.

Example Machine Learning Models

In various embodiments, a wide variety of machine-learning techniques may be used. Examples include different forms of supervised learning, unsupervised learning, and semi-supervised learning such as decision trees, support vector machines (SVMs), regression, Bayesian networks, and genetic algorithms. Deep learning techniques such as neural networks, including convolutional neural networks (CNN), recurrent neural networks (RNN) long short-term memory networks (LSTM), transformers, attention models, and generative adversarial networks (GANs) may also be used. For example, various machine learning models **235** that are used to predict whether a request is non-compliant (e.g., malicious, unauthorized, fraudulent) may apply one or more machine learning and deep learning techniques.

In various embodiments, the training techniques for a machine learning model may be supervised, semi-supervised, or unsupervised. In supervised learning, the machine learning models may be trained with a set of training samples that are labeled. For example, for a machine learning model trained to predict if a request is noncompliant, the training samples may be past transactions labeled with compliant or noncompliant. The labels for each training sample may be binary or multi-class. Labels may be used to indicate which threat(s) are connected to the request: drain, sybil, etc. Binary (has vulnerability or not) and composite multi-class (binary: yes; vulnerabilities: drain) labels may be used. In training a machine learning model for identifying malicious activities, the training samples may be past transactions with contextual data of those transactions. In some cases, an unsupervised learning technique may be used. The samples used in training are not labeled. Various unsuper-

vised learning techniques such as clustering may be used. For example, non-compliant requests may follow certain patterns and may be clustered together by an unsupervised learning technique. In some cases, the training may be semi-supervised with the training set having a mix of labeled samples and unlabeled samples.

A machine learning model may be associated with an objective function, which generates a metric value that describes the objective goal of the training process. For example, the training may intend to reduce the error rate of the model in predicting whether a request is non-compliant. In such a case, the objective function may monitor the error rate of the machine learning model. Such an objective function may be called a loss function. Other forms of objective functions may also be used, particularly for unsupervised learning models whose error rates are not easily determined due to the lack of labels. In transaction prediction, the objective function may correspond to the difference between the model's predicted outcomes and the manually recorded outcomes in the training sets. In various embodiments, the error rate may be measured as cross-entropy loss, L1 loss (e.g., the sum of absolute differences between the predicted values and the actual value), and L2 loss (e.g., the sum of squared distances).

Referring to FIG. 6, the structure of an example neural network is illustrated, in accordance with some embodiments. The neural network 600 may receive an input and generate an output. The neural network 600 may include different kinds of layers, such as convolutional layers, pooling layers, recurrent layers, fully connected layers, custom layers, and different nodes 610. A convolutional layer convolves the input of the layer (e.g., an image) with one or more kernels to generate different types of images that are filtered by the kernels to generate feature maps. Each convolution result may be associated with an activation function. In some embodiments, a pair of convolutional layers may be followed by a recurrent layer that includes one or more feedback loops. The feedback may be used to account for spatial relationships of the features in text or temporal relationships of objects. The layers may be followed by multiple fully connected layers that have nodes connected to each other. The fully connected layers may be used for classification and object detection. In one embodiment, one or more custom layers may also be presented for the generation of a specific format of output. For example, a custom layer may be used for image segmentation for labeling pixels of an image input with different segment labels.

The order of layers and the number of layers of the neural network 600 may vary in different embodiments. In various embodiments, a neural network 600 includes one or more layers 602, 604, and 606, but may or may not include any pooling layer or recurrent layer. If a pooling layer is present, not all convolutional layers are always followed by a pooling layer. A recurrent layer may also be positioned differently at other locations of the CNN. For each convolutional layer, the sizes of kernels (e.g., 3×3, 5×5, 7×7, etc.) and the numbers of kernels allowed to be learned may be different from other convolutional layers.

A machine learning model may include certain layers, nodes, kernels, and/or coefficients. Training of a neural network may include forward propagation and backpropagation. Each layer in a neural network may include one or more nodes, which may be fully or partially connected to other nodes in adjacent layers. In forward propagation, the neural network performs the computation in the forward direction based on the outputs of a preceding layer. The

operation of a node may be defined by one or more functions. The functions that define the operation of a node may include various computation operations such as convolution of data with one or more kernels, pooling, recurrent loop in RNN, various gates in LSTM, etc. The functions may also include an activation function that adjusts the weight of the output of the node. Nodes in different layers may be associated with different functions.

Each of the functions in the neural network may be associated with different coefficients (e.g. weights and kernel coefficients) that are adjustable during training. In addition, some of the nodes in a neural network may also be associated with an activation function that decides the weight of the output of the node in forward propagation. Common activation functions may include step functions, linear functions, sigmoid functions, hyperbolic tangent functions (tanh), and rectified linear unit functions (ReLU). After input is provided into the neural network and passes through a neural network in the forward direction, the results may be compared to the training labels or other values in the training set to determine the neural network's performance. The process of prediction may be repeated for other images in the training sets to compute the value of the objective function in a particular training round. In turn, the neural network performs backpropagation by using gradient descent such as stochastic gradient descent (SGD) to adjust the coefficients in various functions to improve the value of the objective function.

Multiple rounds of forward propagation and backpropagation may be iteratively performed. Training may be completed when the objective function has become sufficiently stable (e.g., the machine learning model has converged) or after a predetermined number of rounds for a particular set of training samples. The trained machine learning model can be used for performing prediction or another suitable task for which the model is trained.

Example Blockchain Architecture

FIG. 7A is a block diagram illustrating a chain of transactions broadcasted and recorded on a blockchain, in accordance with an embodiment. The transactions described in FIG. 7A may correspond to any of the transactions and the transfer of blockchain-based units described in previous figures.

In some embodiments, a blockchain is a distributed system. A distributed blockchain network may include a plurality of nodes. Each node is a user or a server that participates in the blockchain network. In a public blockchain, any participant may become a node of the blockchain. The nodes collectively may be used as a distributed computing system that serves as a virtual machine of the blockchain. In some embodiments, the virtual machine or a distributed computing system may be simply referred to as a computer. Any users of a public blockchain may broadcast transactions for the nodes of the blockchain to record. Each user's digital wallet is associated with a private cryptographic key that is used to sign transactions and prove the ownership of a blockchain-based unit.

The ownership of a blockchain-based unit may be traced through a chain of transactions. In FIG. 7A, a chain of transactions may include a first transaction 710, a second transaction 720, and a third transaction 730, etc. Each of the transactions in the chain may have a fairly similar structure except the very first transaction in the chain. The first transaction of the chain may be generated by a smart contract or a mining process and may be traced back to the smart contract that is recorded on the blockchain or the first block in which it was generated. While each transaction is

linked to a prior transaction in FIG. 7A, the transaction does not need to be recorded on consecutive blocks on the blockchain. For example, the block recording transaction 710 and the block recording transaction 720 may be separated by hundreds or even thousands of blocks. The trace-back of the prior block is tracked by the hash of the prior block that is recorded by the current block. In some embodiments, an account model is used and transactions do not have any references to previous transactions. Transactions are not chained and do not contain the hash of the previous transaction.

Referring to one of the transactions in FIG. 7A, for illustration, the transaction 720 may be referred to as a current transaction. Transaction 710 may be referred to as a prior transaction and transaction 730 may be referred to as a subsequent transaction. Each transaction includes transaction data 722, a recipient address 724, a hash of the prior transaction 726, and the current transaction's owner's digital signature 728. The transaction data 722 records the substance of the current transaction 720. For example, transaction data 722 may specify a transfer of a quantity of a blockchain-based unit (e.g., a coin, a blockchain token, etc.). In some embodiments, the transaction data 722 may include code instructions of a smart contract.

The recipient address 724 is a version of the public key that corresponds to the private key of the digital wallet of the recipient. In one embodiment, the recipient address 724 is the public key itself. In another embodiment, the recipient address 724 is an encoded version of the public key through one or more functions such as some deterministic functions. For example, the generation of the recipient address 724 from the public key may include hashing the public key, adding a checksum, adding one or more prefixes or suffixes, encoding the resultant bits, and truncating the address. The recipient address 724 may be a unique identifier of the digital wallet of the recipient on the blockchain.

The hash of the prior transaction 726 is the hash of the entire transaction data of the prior transaction 710. Likewise, the hash of the prior transaction 736 is the hash of the entire transaction data of transaction 720. The hashing of the prior transaction 710 may be performed using a hashing algorithm such as a secure hash algorithm (SHA) or a message digest algorithm (MD). In some embodiments, the owner corresponding to the current transaction 720 may also use the public key of the owner to generate the hash. The hash of prior transaction 726 provides a trace-back of the prior transaction 710 and also maintains the data integrity of the prior transaction 710.

In generating a current transaction 720, the digital wallet of the current owner of the blockchain-based unit uses its private key to encrypt the combination of the transaction data 722, the recipient address 724, and the hash of prior transaction 726 to generate the owner's digital signature 728. To generate the current transaction 720, the current owner specifies a recipient by including the recipient address 724 in the digital signature 728 of the current transaction 720. The subsequent owner of the blockchain-based unit is fixed by the recipient address 724. In other words, the subsequent owner that generates the digital signature 738 in the subsequent transaction 730 is fixed by the recipient's address 724 specified by the current transaction 720. To verify the validity of the current transaction 720, any nodes in the blockchain network may trace back to the prior transaction 710 (by tracing the hash of prior transaction 726) and locate the recipient address 714. The recipient address 714 corresponds to the public key of the digital signature 728. Hence, the nodes in the blockchain network may use

the public key to verify the digital signature 728. Hence, a current owner who has the blockchain-based unit tied to the owner's blockchain address can prove the ownership of the blockchain-based unit. In this disclosure, it can be described as the blockchain-based unit being connected to a public cryptographic key of a party because the blockchain address is derived from the public key. For example, the access control server 130 may own blockchain-based units in a machine learning model 235. The blockchain-based units are connected to one of the public cryptographic keys of the access control server 130.

The transfer of ownership of a blockchain-based unit may be initiated by the current owner of the blockchain-based unit. To transfer the ownership, the owner may broadcast the transaction that includes the digital signature of the owner and a hash of the prior transaction. A valid transaction with a verifiable digital signature and a correct hash of the prior transaction will be recorded in a new block of the blockchain through the block generation process.

FIG. 7B is a block diagram illustrating a connection of multiple blocks in a blockchain, in accordance with an embodiment. Each block of a blockchain, except the very first block which may be referred to as the genesis block, may have a similar structure. The blocks 750, 760, and 760 may each include a hash of the prior blockchain 752, a nonce 754, and a plurality of transactions (e.g., a first transaction 756, a second transaction 758, etc.). Each transaction may have the structure shown in FIG. 7A.

In a block generation process, a new block may be generated through mining or voting. For the mining process of a blockchain, any nodes in the blockchain system may participate in the mining process. The generation of the hash of the prior block may be conducted through a trial and error process. The entire data of the prior block (or a version of the prior block such as a simplified version) may be hashed using the nonce as a part of the input. The blockchain may use a certain format in the hash of the prior block in order for the new block to be recognized by the nodes as valid. For example, in one embodiment, the hash of the prior block needs to start with a certain number of zeroes in the hash. Other criteria of the hash of the prior block may also be used, depending on the implementation of the blockchain.

In a voting process, the nodes in a blockchain system may vote to determine the content of a new block. Depending on the embodiment, a selected subset of nodes or all nodes in the blockchain system may participate in the votes. When there are multiple candidates new blocks that include different transactions are available, and the nodes will vote for one of the blocks to be linked to the existing block. The voting may be based on the voting power of the nodes.

By way of example of a block generation process using mining, in generating the hash of prior block 762, a node may randomly combine a version of the prior block 750 with a random nonce to generate a hash. The generated hash is somewhat of a random number due to the random nonce. The node compares the generated hash with the criteria of the blockchain system to check if the criteria are met (e.g., whether the generated hash starts with a certain number of zeroes in the hash). If the generated hash fails to meet the criteria, the node tries another random nonce to generate another hash. The process is repeated for different nodes in the blockchain network until one of the nodes finds a hash that satisfies the criteria. The nonce that is used to generate the satisfactory hash is the nonce 764. The node that first generates the hash 762 may also select what transactions that are broadcasted to the blockchain network are to be included in block 760. The node may check the validity of the

transaction (e.g., whether the transaction can be traced back to a prior recorded transaction and whether the digital signature of the generator of the transaction is valid). The selection may also depend on the number of broadcasted transactions that are pending to be recorded and also the fees that may be specified in the transactions. For example, in some embodiments, each transaction may be associated with a fee (e.g., gas) for having the transaction recorded. After the transactions are selected and the data of block **760** is fixed, the nodes in the blockchain network repeat the trial and error process to generate the hash of prior block **772** by trying different nonce. In embodiments that use voting to generate new blocks, a nonce may not be needed. A new block may be linked to the prior block by including the hash of the prior block.

New blocks may continue to be generated through the block generation process. A transaction of a blockchain-based unit (e.g., an electronic coin, a blockchain token, etc.) is complete when the broadcasted transaction is recorded in a block. In some embodiments, the transaction is considered settled when the transaction is considered final. A transaction is considered final when there are multiple subsequent blocks generated and linked to the block that records the transaction.

In some embodiments, some of the transactions **756**, **758**, **766**, **768**, **776**, **778**, etc. may include one or more smart contracts. The code instructions of the smart contracts are recorded in the block and are often immutable. When conditions are met, the code instructions of the smart contract are triggered. The code instructions may cause a computer (e.g., a virtual machine of the blockchain) to carry out some actions such as generating a blockchain-based unit and broadcasting a transaction documenting the generation to the blockchain network for recordation.

Computing Machine Architecture

FIG. **8** is a block diagram illustrating components of an example computing machine that is capable of reading instructions from a computer-readable medium and executing them in a processor (or controller). A computer described herein may include a single computing machine shown in FIG. **8**, a virtual machine, a distributed computing system that includes multiple nodes of computing machines shown in FIG. **8**, or any other suitable arrangement of computing devices.

By way of example, FIG. **8** shows a diagrammatic representation of a computing machine in the example form of a computer system **800** within which instructions **824** (e.g., software, program code, or machine code), which may be stored in a computer-readable medium for causing the machine to perform any one or more of the processes discussed herein may be executed. In some embodiments, the computing machine operates as a standalone device or may be connected (e.g., networked) to other machines. In a networked deployment, the machine may operate in the capacity of a server machine or a client machine in a server-client network environment, or as a peer machine in a peer-to-peer (or distributed) network environment.

The structure of a computing machine described in FIG. **8** may correspond to any software, hardware, or combined components shown in FIGS. **1** and **2**, including but not limited to, the user device **110**, the access control server **130**, a computing node that operates a virtual machine **156**, a node of a blockchain network, and various engines, modules interfaces, terminals, and machines shown in FIG. **2**. While FIG. **8** shows various hardware and software elements, each of the components described in FIGS. **1** and **2** may include additional or fewer elements.

By way of example, a computing machine may be a personal computer (PC), a tablet PC, a set-top box (STB), a personal digital assistant (PDA), a cellular telephone, a smartphone, a web appliance, a network router, an internet of things (IoT) device, a switch or bridge, or any machine capable of executing instructions **824** that specify actions to be taken by that machine. Further, while only a single machine is illustrated, the term “machine” shall also be taken to include any collection of machines that individually or jointly execute instructions **824** to perform any one or more of the methodologies discussed herein.

The example computer system **800** includes one or more processors (generally, processor **802**) (e.g., a central processing unit (CPU), a graphics processing unit (GPU), a digital signal processor (DSP), one or more application-specific integrated circuits (ASICs), one or more radio-frequency integrated circuits (RFICs), or any combination of these), a main memory **804**, and a static memory **806**, which are configured to communicate with each other via a bus **808**. The computer system **800** may further include graphics display unit **810** (e.g., a plasma display panel (PDP), a liquid crystal display (LCD), a projector, or a cathode ray tube (CRT)). The computer system **800** may also include an alphanumeric input device **812** (e.g., a keyboard), a cursor control device **814** (e.g., a mouse, a trackball, a joystick, a motion sensor, or other pointing instrument), a storage unit **816**, a signal generation device **818** (e.g., a speaker), and a network interface device **820**, which also are configured to communicate via the bus **808**.

The storage unit **816** includes a computer-readable medium **822** on which is stored instructions **824** embodying any one or more of the methodologies or functions described herein. The instructions **824** may also reside, completely or at least partially, within the main memory **804** or within the processor **802** (e.g., within a processor's cache memory) during execution thereof by the computer system **800**, the main memory **804** and the processor **802** also constituting computer-readable media. The instructions **824** may be transmitted or received over a network **826** via the network interface device **820**.

The performance of certain operations may be distributed among more than one processor, not only residing within a single machine but deployed across a number of machines. In some example embodiments, one or more processors or processor-implemented modules may be located in a single geographic location (e.g., within a home environment, an office environment, or a server farm). In other example embodiments, one or more processors or processor-implemented modules may be distributed across a number of geographic locations. Even though the specification or the claims may refer to some processes to be performed by a processor, this may be construed to include a joint operation of multiple distributed processors. In some embodiments, a computer-readable medium comprises one or more computer-readable media that, individually, together, or distributively, comprise instructions that, when executed by one or more processors, cause a processor (including in the situation of one or more processors) to perform, individually, together, or distributively, the steps of the instructions stored on the one or more computer-readable media. Similarly, a processor comprises one or more processors or processing units that, individually, together, or distributively, perform the steps of instructions stored on a computer-readable medium. In various embodiments, the discussion of one or more processors that carry out a process with multiple steps does not require any one of the processors to carry out all of the steps. For example, a processor A can carry out step A,

a processor B can carry out step B using, for example, the result from the processor A, and a processor C can carry out step C, etc. The processors may work cooperatively in this type of situation such as in multiple processors of a system in a chip, in Cloud computing, or in distributed computing.

While computer-readable medium **822** is shown in an example embodiment to be a single medium, the term “computer-readable medium” should be taken to include a single medium or multiple media (e.g., a centralized or distributed database, or associated caches and servers) able to store instructions (e.g., instructions **824**). The computer-readable medium may include any medium that is capable of storing instructions (e.g., instructions **824**) for execution by the machine and that causes the machine to perform any one or more of the methodologies disclosed herein. The computer-readable medium may include, but not be limited to, data repositories in the form of solid-state memories, optical media, and magnetic media. The computer-readable medium does not include a transitory medium such as a signal or a carrier wave.

Additional Configuration Considerations

Beneficially, with various embodiments described in this disclosure, in a cryptographically proofed, cost-efficient way, smart contract (or other Web3 application) owners could add an interface to their applications to have control over the applications after being deployed to the blockchain. In addition, the application publishers could also apply security technologies to control the applications in real time. Since the interactions would be vetted and signed by the access control system before the interaction request reaches the application on the blockchain, the access control server can block and prevent malicious or unwanted actions.

Certain embodiments are described herein as including logic or a number of components, engines, modules, or mechanisms, for example, as illustrated in FIG. 2. Engines may constitute either software modules (e.g., code embodied on a computer-readable medium) or hardware modules. A hardware engine is a tangible unit capable of performing certain operations and may be configured or arranged in a certain manner. In example embodiments, one or more computer systems (e.g., a standalone, client, or server computer system) or one or more hardware engines of a computer system (e.g., a processor or a group of processors) may be configured by software (e.g., an application or application portion) as a hardware engine that operates to perform certain operations as described herein.

In various embodiments, a hardware engine may be implemented mechanically or electronically. For example, a hardware engine may comprise dedicated circuitry or logic that is permanently configured (e.g., as a special-purpose processor, such as a field programmable gate array (FPGA) or an application-specific integrated circuit (ASIC)) to perform certain operations. A hardware engine may also comprise programmable logic or circuitry (e.g., as encompassed within a general-purpose processor or another programmable processor) that is temporarily configured by software to perform certain operations. It will be appreciated that the decision to implement a hardware engine mechanically, in dedicated and permanently configured circuitry, or in temporarily configured circuitry (e.g., configured by software) may be driven by cost and time considerations.

The various operations of example methods described herein may be performed, at least partially, by one or more processors, e.g., processor **802**, that are temporarily configured (e.g., by software) or permanently configured to perform the relevant operations. Whether temporarily or permanently configured, such processors may constitute

processor-implemented engines that operate to perform one or more operations or functions. The engines referred to herein may, in some example embodiments, comprise processor-implemented engines.

The performance of certain operations may be distributed among one or more processors, not only residing within a single machine but deployed across a number of machines. In some example embodiments, one or more processors or processor-implemented modules may be located in a single geographic location (e.g., within a home environment, an office environment, or a server farm). In other example embodiments, one or more processors or processor-implemented modules may be distributed across a number of geographic locations.

ADDITIONAL CONSIDERATIONS

The foregoing description of the embodiments has been presented for the purpose of illustration; it is not intended to be exhaustive or to limit the patent rights to the precise forms disclosed. Persons skilled in the relevant art can appreciate that many modifications and variations are possible in light of the above disclosure.

Any feature mentioned in one claim category, e.g. method, can be claimed in another claim category, e.g. computer program product, system, or storage medium, as well. The dependencies or references in the attached claims are chosen for formal reasons only. However, any subject matter resulting from a deliberate reference back to any previous claims (in particular multiple dependencies) can be claimed as well, so that any combination of claims and the features thereof is disclosed and can be claimed regardless of the dependencies chosen in the attached claims. The subject matter may include not only the combinations of features as set out in the disclosed embodiments but also any other combination of features from different embodiments. Various features mentioned in the different embodiments can be combined with explicit mentioning of such combination or arrangement in an example embodiment or without any explicit mentioning. Furthermore, any of the embodiments and features described or depicted herein may be claimed in a separate claim and/or in any combination with any embodiment or feature described or depicted herein or with any of the features.

Some portions of this description describe the embodiments in terms of algorithms and symbolic representations of operations on information. These operations and algorithmic descriptions, while described functionally, computationally, or logically, are understood to be implemented by computer programs or equivalent electrical circuits, microcodes, or the like. Furthermore, it has also proven convenient at times, to refer to these arrangements of operations as engines, without loss of generality. The described operations and their associated engines may be embodied in software, firmware, hardware, or any combinations thereof.

Any of the steps, operations, or processes described herein may be performed or implemented with one or more hardware or software engines, alone or in combination with other devices. In some embodiments, a software engine is implemented with a computer program product comprising a computer-readable medium containing computer program code, which can be executed by a computer processor for performing any or all of the steps, operations, or processes described. The term “steps” does not mandate or imply a particular order. For example, while this disclosure may describe a process that includes multiple steps sequentially with arrows present in a flowchart, the steps in the process

do not need to be performed in the specific order claimed or described in the disclosure. Some steps may be performed before others even though the other steps are claimed or described first in this disclosure. Likewise, any use of (i), (ii), (iii), etc., or (a), (b), (c), etc. in the specification or in the claims, unless specified, is used to better enumerate items or steps and also does not mandate a particular order.

Throughout this specification, plural instances may implement components, operations, or structures described as a single instance. Although individual operations of one or more methods are illustrated and described as separate operations, one or more of the individual operations may be performed concurrently, and nothing requires that the operations be performed in the order illustrated. Structures and functionality presented as separate components in example configurations may be implemented as a combined structure or component. Similarly, structures and functionality presented as a single component may be implemented as separate components. These and other variations, modifications, additions, and improvements fall within the scope of the subject matter herein. In addition, the term “each” used in the specification and claims does not imply that every or all elements in a group need to fit the description associated with the term “each.” For example, “each member is associated with element A” does not imply that all members are associated with an element A. Instead, the term “each” only implies that a member (of some of the members), in a singular form, is associated with an element A. In claims, the use of a singular form of a noun may imply at least one element even though a plural form is not used.

Finally, the language used in the specification has been principally selected for readability and instructional purposes, and it may not have been selected to delineate or circumscribe the patent rights. It is therefore intended that the scope of the patent rights be limited not by this detailed description, but rather by any claims that are issued on an application based hereon. Accordingly, the disclosure of the embodiments is intended to be illustrative, but not limited, of the scope of the patent rights.

What is claimed is:

1. A system comprising:

an access control server comprising a processor and memory configured to store code comprising instructions, wherein the instructions, when executed by the processor, cause the processor to:

store a private cryptographic key belonging to the access control server;

receive, from an access requester, an access request for accessing a protected autonomous program protocol stored on a blockchain;

analyze parameters related to the access requester to determine whether the access requester is authorized to access the protected autonomous program protocol;

generate, using the private cryptographic key, a cryptographic signature associated with the access request, the cryptographic signature generated by the access control server; and

transmit the cryptographic signature to the blockchain to be recorded; and

an access control autonomous protocol recorded on the blockchain, the access control autonomous protocol being a proxy protocol providing one or more access control features for the protected autonomous program protocol, wherein the access control autonomous protocol comprises (1) a checker that includes a set of protocols for verification and stores a public crypto-

graphic key corresponding to the private cryptographic key belonging to the access control server and (2) a ledger that stores proofs of authorization of a plurality of access requesters, and wherein the access control autonomous protocol is configured to:

receive the cryptographic signature generated by the access control server;

verify, by the set of protocols in the checker, the cryptographic signature using public cryptographic key stored in the checker; and

store a proof of authorization associated with the protected autonomous program protocol in the ledger of the access control autonomous protocol, wherein the ledger of the access control autonomous protocol is at a blockchain address of the access control autonomous protocol and is accessible by the protected autonomous program protocol to verify that the access requester is authorized to access the protected autonomous program protocol.

2. The system of claim 1, wherein the instructions of the access control servers further comprise instructions to:

receive access control setting related to the protected autonomous program protocol, the access control setting specifying one or more policies in granting access to the protected autonomous program protocol; and

determine, based on the parameters related to the access requester, that the access request is in compliance with one or more policies specified in the access control setting.

3. The system of claim 1, wherein the parameters related to the access requester comprise a parameter related to a transaction that the access requester wants to be processed by the protected autonomous program protocol.

4. The system of claim 1, wherein the parameters related to the access requester comprise an authentication credential of the access requester.

5. The system of claim 1, wherein a first proof of authorization of the proofs of authorization is associated with authorization for a first protected autonomous program protocol and a second proof of authorization of the proofs of authorization is associated with authorization for a second protected autonomous program protocol that is different from the first autonomous program protocol.

6. The system of claim 5, wherein the access control autonomous protocol is associated with the blockchain address and the first autonomous program protocol and the second autonomous program protocol are configured to retrieve the proofs of authorization stored in the access control autonomous protocol at the blockchain address.

7. The system of claim 1, wherein the access request comprises a transaction payload to be transmitted to the protected autonomous program protocol for processing, and the cryptographic signature of the proof of authorization comprises a hash of the transaction payload.

8. The system of claim 7, wherein the access control autonomous protocol is configured to store the hash of the transaction payload and the protected autonomous program protocol is configured to verify the hash of the transaction payload before processing the transaction payload.

9. The system of claim 1, wherein the proof of authorization comprises a second cryptography signature associated with the access requester, and the proof of authorization is timestamped so that the proof of authorization is valid for a predetermined period.

41

10. The system of claim 1, further comprising a first application that is configured to communicate with the access control server to transmit the access request on behalf of the access requester.

11. The system of claim 10, further comprising a second application that is configured to communicate with the protected autonomous program protocol to carry out transactions for the access requester using the protected autonomous program protocol, and the second application is different from the first application.

12. The system of claim 1, wherein the protected autonomous program protocol comprises an updated component that is added after a first launch of the protected autonomous program protocol, and the updated component comprises a protocol that is configured to communicate with the access control autonomous protocol.

13. The system of claim 1, wherein the access control autonomous protocol is a first smart contract recorded on Ethereum blockchain and the protected autonomous program protocol is a second smart contract recorded on the Ethereum blockchain.

14. A computer-implemented method, comprising:

storing a private cryptographic key belonging to an access control server;

receiving, from an access requester, an access request for accessing a protected autonomous program protocol stored on a blockchain;

analyzing parameters related to the access requester to determine whether the access requester is authorized to access the protected autonomous program protocol;

generating, using the private cryptographic key, a cryptographic signature associated with the access request; and

transmitting the cryptographic signature to the blockchain to be recorded on an access control autonomous protocol recorded on the blockchain, the access control autonomous protocol being a proxy protocol providing one or more access control features for the protected autonomous program protocol, wherein the access control autonomous protocol comprises (1) a checker that includes a set of protocols for verification and stores a public cryptographic key corresponding to the private cryptographic key belonging to the access control server and (2) a ledger that stores proofs of authorization of a plurality of access requesters, and wherein the access control autonomous protocol is configured to: receive the cryptographic signature generated by the access control server;

verify, by the set of protocols in the checker, the cryptographic signature using public cryptographic key stored in the checker; and

store a proof of authorization associated with the protected autonomous program protocol in the ledger of the access control autonomous protocol, wherein the ledger of the access control autonomous protocol is at a blockchain address of the access control autonomous protocol and is accessible by the protected autonomous program protocol to verify that the access requester is authorized to access the protected autonomous program protocol.

15. The computer-implemented method of claim 14, further comprising:

receiving access control setting related to the protected autonomous program protocol, the access control setting specifying one or more policies in granting access to the protected autonomous program protocol; and

42

determining, based on the parameters related to the access requester, whether the access request is in compliance with the one or more policies specified in the access control setting.

16. The computer-implemented method of claim 14, wherein the parameters related to the access requester comprise a parameter related to a transaction that the access requester wants to be processed by the protected autonomous program protocol.

17. The computer-implemented method of claim 14, wherein the parameters related to the access requester comprise an authentication credential of the access requester.

18. A non-transitory computer readable medium configured to store code comprising instructions, wherein the instructions, when executed by one or more processors, cause the one or more processors to:

store a private cryptographic key belonging to an access control server;

receive, from an access requester, an access request for accessing a protected autonomous program protocol stored on a blockchain;

analyze parameters related to the access requester to determine whether the access requester is authorized to access the protected autonomous program protocol;

generate, using the private cryptographic key, a cryptographic signature associated with the access request; and

transmit the cryptographic signature to the blockchain to be recorded on an access control autonomous protocol recorded on the blockchain, the access control autonomous protocol being a proxy protocol providing one or more access control features for the protected autonomous program protocol, wherein the access control autonomous protocol comprises (1) a checker that includes a set of protocols for verification and stores a public cryptographic key corresponding to the private cryptographic key belonging to the access control server and (2) a ledger that stores proofs of authorization of a plurality of access requesters, and wherein the access control autonomous protocol is configured to: receive the cryptographic signature generated by the access control server;

verify, by the set of protocols in the checker, the cryptographic signature using public cryptographic key stored in the checker; and

store a proof of authorization associated with the protected autonomous program protocol in the ledger of the access control autonomous protocol, wherein the ledger of the access control autonomous protocol is at a blockchain address of the access control autonomous protocol and is accessible by the protected autonomous program protocol to verify that the access requester is authorized to access the protected autonomous program protocol.

19. The non-transitory computer readable medium of claim 18, wherein the parameters related to the access requester comprise a parameter related to a transaction that the access requester wants to be processed by the protected autonomous program protocol.

20. The non-transitory computer readable medium of claim 18, wherein the parameters related to the access requester comprise an authentication credential of the access requester.

* * * * *