



(51) International Patent Classification:

G06F 21/12 (2013.01) G06F 15/167 (2006.01)
G06F 21/64 (2013.01) G06F 9/54 (2006.01)
G06F 21/60 (2013.01) H04L 9/32 (2006.01)

(21) International Application Number:

PCT/RU2015/000845

(22) International Filing Date:

4 December 2015 (04.12.2015)

(25) Filing Language:

English

(26) Publication Language:

English

(71) Applicant: INTEL CORPORATION [US/US]; 2200 Mission College Blvd., California, Santa Clara, 95054 (US).

(72) Inventors: SOMSIKOV, Andrey Viktorovich; Rodionova str., house 193/2, room 136, Nizhny Novgorod, 603163 (RU). KAZANTSEV, Roman Aleksandrovich; 150 Yasnaya Str., Bldg. 1, Moscow, 142770 (RU).

(74) Agent: LAW FIRM "GORODISKY & PARTNERS" LTD.; MITS Alexander Vladimirovich, POPOVA Elizaveta Vitalievna, B. Spasskaya str., 25, bldg. 3, Moscow, 129090 (RU).

(81) Designated States (unless otherwise indicated, for every kind of national protection available): AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IR, IS, JP, KE, KG, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SA, SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

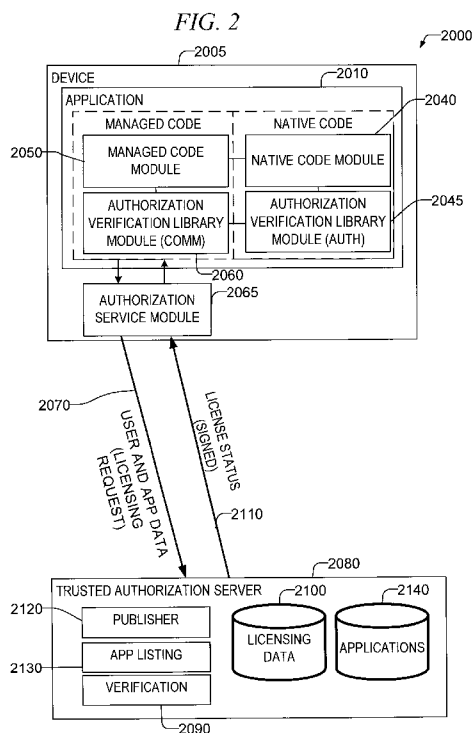
(84) Designated States (unless otherwise indicated, for every kind of regional protection available): ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SD, SL, ST, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

Declarations under Rule 4.17:

— of inventorship (Rule 4.17(iv))

[Continued on next page]

(54) Title: NATIVE CODE AUTHORIZATION VERIFICATION LIBRARY



(57) Abstract: Disclosed in some examples are methods, machine readable mediums, and systems for protecting applications from unauthorized access. In some examples, this may be accomplished by implementing authorization verification tools that takes advantage of both managed code and native code. Native code is easier to protect from hackers as more advanced obfuscation tools may be utilized than with managed code. Once native code is compiled it is typically compiled into machine code instructions that are executed directly by a processor. These instructions are much harder to reverse engineer (e.g., decompile) and even if decompiled, are harder to modify.



Published:

— *with international search report (Art. 21(3))*

NATIVE CODE AUTHORIZATION VERIFICATION LIBRARY

5

COPYRIGHT NOTICE

[0001] A portion of the disclosure of this patent document contains material that is subject to copyright protection. The copyright owner has no objection to the facsimile reproduction by anyone of the patent document or the patent disclosure, as it appears in the Patent and Trademark Office patent files or records, but otherwise reserves all copyright rights whatsoever. The following notice applies to the software and data as described below and in the drawings that form a part of this document: Copyright INTEL, All Rights Reserved.

15

TECHNICAL FIELD

[0002] Embodiments pertain to security. Some embodiments relate to securing applications against unauthorized access.

BACKGROUND

20 [0003] Software application developers have long battled with users who attempt to access their product in an unauthorized manner. Examples of unauthorized access include executing the application without a required license. Various authorization verification tools have been developed in order to thwart these attempts. Such verification tools are subject to attack themselves.

25

BRIEF DESCRIPTION OF THE DRAWINGS

[0004] In the drawings, which are not necessarily drawn to scale, like numerals may describe similar components in different views. Like numerals having different letter suffixes may represent different instances of similar components. The drawings illustrate generally, by way of example, but not by way of limitation, various embodiments discussed in the present document.

30

[0005] FIG. 1 shows a diagram of an authorization verification system according to some examples of the present disclosure.

[0006] FIG. 2 shows a diagram of an improved authorization verification system according to some examples of the present disclosure.

[0007] FIG. 3 shows a detailed diagram of an improved authorization verification system according to some examples of the present disclosure.

5 [0008] FIG. 4 shows a flowchart of a method of performing an authorization verification according to some examples of the present disclosure.

[0009] FIG. 5 shows a flowchart of a method of performing an authorization lookup on a trusted authorization server according to some examples of the present disclosure.

10 [0010] FIG. 6 is a block diagram illustrating an example of a machine upon which one or more embodiments may be implemented.

DETAILED DESCRIPTION

[0011] Application developers may wish to place constraints to the utilization of their applications. Based on these constraints, developers may prohibit certain
15 usages of the application by blocking features, functionality, or even execution of the entire application. Constraints may include one or more of: requiring the purchase or possession of a license to run all or part of the application, time-based constraints (e.g., the application, or certain features of the application are only active
20 for certain time periods or during certain periods of time), location based constraints (e.g., the application, or certain features of the application are only active within certain locations), user based constraints (e.g., ensuring that only certain users of a device access the application, such as by authenticating), and the like. An application usage that obeys these constraints may be deemed an authorized use of
25 the application. Application usage that does not obey these constraints may be deemed an unauthorized usage of the application.

[0012] Authorization verification tools are tools that application developers can utilize to enforce these constraints and ensure that only authorized users can execute an application or part of an application. Some authorization verification tools make
30 use of managed code libraries that provide functionality for applications to check a

user's authorization (e.g., a license status) with a trusted authorization server (e.g., a trusted licensing server). To use these authorization verification tools, an application developer includes the managed code libraries in the application package (e.g., an ".apk" file). These libraries are then called by the application code
5 when needed to verify authorization status.

[0013] For example, to check a licensing status, an application calls functions provided by an authorization verification library to check the licensing status of the computing device on which the user is trying to run the application. The trusted authentication servers retrieve licensing details of the application for the particular
10 device on which the application is running. The application then determines whether the licensing details satisfy the licensing constraints. Other types of authorization and constraint checking may function similarly. For example, for temporal constraints, the authorization verification library may contact the authorization server to determine whether the temporal constraints are satisfied.
15 The temporal server may return back a yes or no answer, or may return a trusted time with which the application may determine whether the temporal constraints are satisfied. In still other examples, for geolocation-based constraints the authorization verification library may contact the authorization server to determine whether location based constraints have been satisfied. The request to the authorization
20 verification library may include a device's current location, or may include information about the device that is then used by the authorization verification library to obtain the location of the device (e.g., from a cellular network provider). The location may then be returned to the device.

[0014] The trusted authorization server is a network-based server that stores
25 data about and helps to validate application constraints. For example, for licensing constraints the trusted authorization server may return licensing details (e.g., whether the device or user is authorized, what level they are authorized to, and the like) to the application on the device. The application then uses the licensing details to determine whether the use of the application is to be allowed. One example of a
30 managed code authorization library is the Licensing Verification Library (LVL)

provided by GOOGLE ® for applications running on its ANDROID® operating system. The LVL verifies licensing status for applications on ANDROID® through the GOOGLE® PLAY® application on the device. The authorization server is a GOOGLE® PLAY® server.

5 **[0015]** As previously discussed, these authorization verification libraries utilize managed code. Managed code relies upon an additional process at runtime or just before runtime that allows it to run on a particular platform - for example, Just In Time (JIT) compilation at run time, an interpreter (e.g., a Java run-time environment), or the like. In contrast, native code is compiled into machine
10 executable format for direct execution on a particular platform. For example, code compiled specifically for devices running an ANDROID® operating system. One problem with the use of managed code libraries for license verification is that managed code is easy to decompile to its source code. Once a hacker decompiles the managed code libraries, the hackers may insert code that bypasses authorization
15 checks.

[0016] To combat this, application developers have recently begun to turn to code obfuscation tools that intentionally turn source code into code that is much more difficult to understand. To bypass the authorization checks on obfuscated code, hackers have to figure out the obfuscated code in order to circumvent the
20 authorization checks. These tools still leave clues that can allow a hacker to rewrite the code. For example, message strings (e.g., "Access Granted") or certain methods like MessageBox and Show() are typically not obfuscated. These unobfuscated code portions may give a hacker the contextual clues they need to modify the portion of the code that controls access.

25 **[0017]** Disclosed in some examples are methods, machine-readable mediums, and systems for protecting applications from unauthorized access. In some examples, this may be accomplished by implementing authorization verification tools that takes advantage of both managed code and native code. Native code is easier to protect from hackers as more advanced obfuscation tools may be utilized
30 than with managed code. Once native code is compiled it is typically compiled into

machine code instructions that are executed directly by a processor. These instructions are much harder to reverse engineer (e.g., decompile) and even if decompiled, are harder to modify.

[0018] Turning now to FIG. 1, a diagram of an authorization verification system 5 1000 is shown according to some examples of the present disclosure. Application 1010 may execute on one or more computing devices 1005, such as a mobile device (e.g., a smartphone, tablet, laptop computer, desktop computer, or the like). Application 1010 comprises both managed code and native code. Native code module 1040 comprises one or more portions of the application code that is in a native format. Managed code module 1050 comprises the portion of code that is in 10 a managed code format. Application 1010's managed code also includes authorization verification library module 1060. Managed code module 1050 interacts with authorization verification library module 1060 to verify that the application 1010 is authorized before allowing the application 1010, or a portion of 15 the application 1010, to execute or continue executing to provide the corresponding functions of the application (e.g., a gaming application, a productivity application, and the like).

[0019] In the case of FIG. 1, authorization verification library module 1060 20 verifies a licensing status of device 1005 to determine that application 1010 is authorized. For example, authorization verification library module 1060 implements a number of functions that, when called by managed code module 1050, sends data comprising one or more of user, device and application data in a licensing request 1070 over a network to trusted authorization server 1080. Trusted authorization server 1080 determines a license status of the device. Verification 25 module 1090 of trusted authorization server 1080 looks up the application in a licensing database 1100 using the application data included in licensing request 1070 and determines a licensing status of the device 1005 (or a user of the device) from licensing database 1100. In some examples, device 1005 may be uniquely identified using one or more unique identification numbers (e.g., a UUID).

[0020] In some examples, the license may be per- device – that is, all users of the device are licensed to use the application. In other examples, the license may be per-user where only certain users may be licensed. And in yet other examples, the license may be per- user and per- device, that is, only certain users are licensed on
5 certain devices. In examples in which the license is per- device, the authorization verification library module 1060 may send a unique identifier of the device 1005 to the trusted authorization service 1080. In other examples, where licensing is per-user, the authorization verification library module 1060 may send information (e.g., credentials) about the user trying to access the application 1010. In examples in
10 which the licensing is per-user-per-device, the authorization verification library module 1060 may send both a device and user identifier.

[0021] The licensing status is returned in license status message 1110 to authorization verification library module 1060, which then communicates that information to managed code module 1050. In some examples, this happens
15 through a callback function in managed code module 1050 that is registered with authorization verification library module 1060. In some examples, authorization verification library module 1060 communicates with trusted authorization server 1080 by issuing a command to an authorization service module 1065 which may then handle the communications with trusted authorization server 1080.
20 Authorization service module 1065 may be a separate application, an operating system service, or a module within another application.

[0022] Trusted authorization server 1080 may have a publisher module 1120 which allows developers to upload applications to trusted authorization server 1080. The process of publishing an application may include one or more quality assurance
25 checks, application registration, entering licensing terms such as price and duration of the license, and the like. Application listing module 1130 may create one or more network-based application “stores” with which users may connect, obtain licenses for, and download one or more applications that were published by developers through publisher module 1120. Applications uploaded by publisher

module 1120 and provided by the application listing module 1130 may be stored in an applications database 1140.

[0023] The managed code portions of the application 1010 are easily decompiled and may be modified by an unscrupulous individual. This individual may then post the modified version of the application online where others may download it and use the software in an unauthorized manner. Worse yet, the unscrupulous individual may plant malware or other undesirable code, tricking others into believing it's a legitimate version and destroying the reputation of the application developer.

10 [0024] Turning now to FIG. 2, a diagram of an improved authorization verification system 2000 is shown according to some examples of the present disclosure. Application 2010 may execute on one or more computing devices 2005, such as a mobile device (e.g., a smartphone, tablet, laptop computer, or the like). Application 2010 comprises both managed code and native code. Native code module 2040 comprises the portion of the application code that is in a native format. Managed code module 2050 comprises the portion of code that is in a managed code format.

[0025] In FIG. 2, authorization verification library module is split into two parts. One part, authorization verification library module (communications) 2060 is managed code. The other part, the authorization verification library module (authorization context) 2045 is native code. Authorization verification library module (communications) 2060 provides library functions that are resident in managed code portions of the application and handle communications with trusted authorization server 2080 (in some examples via authorization service module 2065). These functions include gathering user, device, and application information, forming a licensing request 2070, sending the request, receiving licensing status 2110, and generating a callback to the callback function of the application to pass back the licensing status to the application.

[0026] Authorization verification library module (authorization context) 2045 may contain code which initializes, queries, and sets an authorization context for an

application. In some examples, for licensing based authorization, the authorization context may be a licensing context. In these cases, the authorization verification library module (authorization context) 2045 may set the authorization context based upon information about the licensing status obtained by authorization and verification library module (communications) 2060. Authorization verification library module (authorization context) 2045 may have code that stores and retrieves the licensing status and passes this status to native code module 2040 which makes decisions about whether to continue executing application 2010 based upon the licensing status. For example, if device 2005 or the user of the device is not licensed, the native code module 2040 may perform an unauthorized device or user function. Example unauthorized device or user functions include one or more of: terminating execution of the application, displaying a message, taking the user to an app listing page for the application on trusted authorization server 2080 to purchase a license, or take any other action desired. In some examples, the evaluation of the license status 2110 may be done in native code module 2040, but in other examples, authorization verification library module (authorization context) 2045 may verify that the use of application 2010 is authorized and licensed based upon licensing constraints communicated to authorization verification library module (authorization context) 2045 by the application 2010, or by the application's use of overloaded functions of authorization verification library module (authorization context) 2045.

[0027] Trusted authorization server 2080 may include verification module 2090 that looks up the application in a licensing database 2100 using the application data included in licensing request 2070 and determines a licensing status of the device 2005 (or a user of the device) from licensing database 2100. This licensing status may be returned in licensing status 2110. In addition to providing authorization services, trusted authorization server 2080 may have a publisher module 2120 which allows developers to upload applications to trusted authorization server 2080. The publishing process may include one or more quality assurance checks, application registration, entering licensing terms such as price and duration of the license, and

the like. Application listing module 2130 may create one or more network-based application “stores” with which users may connect, obtain licenses, and download one or more applications that were uploaded by developers through publisher module 2120. Applications uploaded by publisher module 2120 and provided by application listing module 2130 may be stored in an applications database 2140.

[0028] Turning now to FIG. 3, a detailed diagram of an improved authorization verification system 3000 is shown according to some examples of the present disclosure. FIG. 3 presents additional details of the system of FIG. 2. FIG. 3 will be discussed in conjunction with FIGs. 4 and 5. FIG. 4 shows a flowchart of a method of performing authorization verification according to some examples of the present disclosure. FIG. 5 shows a flowchart of a method of performing an authorization lookup on a trusted authorization server according to some examples of the present disclosure. At operation 4010 managed code portions of application code module 3050 calls an initialize_authorization_context function 3030 of authorization verification library (authorization context) module 3020. This function initializes an authorization context 3040. Authorization context 3040 may be stored as a data structure in memory (volatile or non-volatile) of the computing device. Authorization context 3040 may store information needed to authorize the application. In some examples, this includes licensing information such as a licensing status and contextual information such as one or more of: the application name or id, user name or id, device name or id, nonce, and licensing status. An initialize_authorization_context function 3030 initializes the authorization context with one or more of the application name or id, device name or id, user names or ids. The function also generates and stores in the authorization context a nonce, initializes the license state to unlicensed, and computes a keyed-hash message authentication code (HMAC) digest using an HMAC function. The inputs to the HMAC function include the initialized authorization context 3040 and the HMAC private key 3150.

[0029] HMAC is a specific construction for calculating a message

authentication code (MAC) involving a cryptographic hash function in combination

with a secret cryptographic key. Example cryptographic hash functions include MD5 and SHA-1. One implementation of HMAC is defined by the Internet Engineering Task Force (IETF) Request for Comment (RFC) 2104.

[0030] The integrity of the authorization context 3040 can be verified by any module possessing HMAC private key 3150. HMAC private key 3150 is a cryptographic key used in an encryption algorithm. The module calculates that HMAC digest of the authorization context 3040 using the values of the authorization context 3040, the HMAC function, and HMAC private key 3150. If the calculated HMAC digest matches the HMAC digest stored with authorization context 3040, then authorization context 3040 is valid. Otherwise, authorization context 3040 is not valid and tampering is indicated.

[0031] Any time an authorized change is made, HMAC digest value is updated using the HMAC function, HMAC private key 3150, and the new values of authorization context 3040. In the example of FIG. 3, any time a change is made to authorization context 3040 by any of the authorization verification library (authorization context) module 3020 functions, the HMAC digest is recalculated and the new HMAC is stored. Additionally, when the authorization context 3040 is accessed (after the initialization), the integrity of the authorization context 3040 is verified. This protects authorization context 3040 from changes that were not made by functions of the authorization verification library (authorization context) module 3020. If hackers were to change any information in authorization context 3040, the HMAC digest associated with authorization context 3040 would not match the HMAC digest calculated by functions of the authorization verification library (authorization context) module 3020. The authorization verification library (authorization context) module 3020 may deem tampering to be an unauthorized access and perform an unauthorized device function.

[0032] At operation 4020, the application sends an authorization query (e.g., a licensing request) to the trusted authorization server 3080 to obtain an authorization status. Managed code portions of application code module 3050 calls an authorization_query_function 3070 of authorization verification library

(communications) module 3060 in the managed code portion of the application. Authorization_query_function 3070 contacts a trusted authorization server 3080. In some examples, it uses an intermediary service (such as an authorization service module – not shown for clarity). In some examples, the request includes context data include application names or ids, user names or ids, device name or id, other device information, a nonce, and the like.

[0033] Turning now to FIG. 5, the trusted authorization server 3080 receives the authorization query at operation 5010. At operation 5020 the trusted authorization server 3080 determines the authorization status for the given information (e.g., for a given user or device). For example, a response may include licensing information, such as the application name, user name, nonce, and license status. In other examples, other constraints may include other information in the response. For example, for temporal constraints, the response may include an indication as to whether the execution of the application (or a part of the application) meets the temporal constraints. In another example, the response may have a trusted time that is used by the application itself to determine whether the constraints are satisfied. The response from the authorization server may be signed with one of a pair of cryptographic keys 3090 (e.g., an RSA® private key) of the application vendor which may be created by the trusted authorization server 3080 during application publication. The corresponding other key may be provided to the application by the application vendor and used to verify the signature. The response is sent at operation 5030 to the requesting device.

[0034] Turning back to FIG. 4, the device receives the authorization response at operation 4030. The response is handled by an authorization_listener_function 3100 of authorization verification library (communications) module 3060. Authorization_listener_function 3100 calls the set_authorization_status function 3110 of authorization verification library (authorization context) module 3020 of the native code portion, passing in the response received from trusted authorization server 3080.

[0035] At operation 4040, using the other one of the pair of cryptographic keys 3120 (e.g., a public key), set_authorization_status function 3110 verifies the integrity of the response from trusted authorization server 3080 using the corresponding one of the pair of cryptographic keys. Set_authorization_status function 3110 verifies that the message is from trusted authorization server 3080 by matching the nonce in the message with the expected nonce sent to trusted authorization server 3080. If the message is from trusted authorization server 3080, the set_authorization_status function 3110 verifies that the value of the fields (user name, nonce, and application name) in the authorization context 3040 matches the expected values. If these values match, set_authorization_status function 3110 updates authorization context 3040 with the information sent from trusted authorization server 3080 in the response. For example, the licensing status from the response sent by trusted authorization server 3080 may be stored in the authorization context 3040. Prior to updating the authorization context 3040 with the new authorization information, set_authorization_status function 3110 may also re-verify the integrity of authorization context 3040 by calculating an HMAC digest (as described above) and comparing it to the stored HMAC digest. If the calculated HMAC digest matches the stored HMAC digest, then the integrity is validated. If the calculated HMAC digest does not match the stored HMAC digest, tampering is indicated and action may be taken (such as terminating the application, throwing an error message or exception, or the like). Once the current authorization context is verified, the set_authorization_status function 3110 may update the licensing status of authorization context 3040, recalculate the HMAC digest using the new value of authorization context 3040, and store the recalculated HMAC digest.

[0036] At operation 4050, native code portions of the application 3130 calls a get_authorization_status function 3140 of authorization verification library (authorization context) module 3020 that resides in native code portion of the application. Get_authorization_status function 3140 determines whether the current user or device is authorized. Using HMAC private key 3150 that was initialized by initialize_authorization_context function 3030, get_authorization_status function

3140 verifies that authorization context 3040 is not corrupted or modified inappropriately (as described above). Get_authorization_status function 3140 then returns the authorization status to the native code portions of the application 3130 that may either allow or disallow further application execution, or may take other
5 actions depending on the results including unauthorized device or user functions.

[0037] This solution is resistant to common attacks. For example, authorization context 3040 can be restricted to only be updatable by native functions initialize_authorization_context function 3030, get_authorization_status 3140, and set_authorization_status 3110 because they are the only functions with access to
10 HMAC private key 3150 and therefore are the only functions able to recalculate the HMAC digest. Moreover, these native functions and the keys may be protected against tampering and reverse engineering by native code obfuscation, which is more robust than managed code obfuscation. Native code obfuscation may also be applied to some of the native code portions of the application 3130 which call
15 get_authorization_status function 3140 to make it more difficult for attackers to remove license check points. In some examples, the obfuscation compiler, such as the Intel® Tamper Protection Toolkit produces code which is self-encrypted at run-time and protected against tampering and reverse engineering.

[0038] FIG. 6 illustrates a block diagram of an example machine 6000 upon
20 which any one or more of the techniques (e.g., methodologies) discussed herein may perform. In alternative embodiments, the machine 6000 may operate as a standalone device or may be connected (e.g., networked) to other machines. In a networked deployment, the machine 6000 may operate in the capacity of a server machine, a client machine, or both in server-client network environments. In an
25 example, the machine 6000 may act as a peer machine in peer-to-peer (P2P) (or other distributed) network environment. The machine 6000 may be a personal computer (PC), a tablet PC, a set-top box (STB), a personal digital assistant (PDA), a mobile telephone, a smart phone, a web appliance, a network router, switch or bridge, or any machine capable of executing instructions (sequential or otherwise)
30 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 a set (or multiple sets) of instructions to perform any one or more of the methodologies discussed herein, such as cloud computing, software as a service (SaaS), other computer cluster configurations.

5 [0039] Examples, as described herein, may include, or may operate on, logic or a number of components, modules, or mechanisms. Modules are tangible entities (e.g., hardware) capable of performing specified operations and may be configured or arranged in a certain manner. In an example, circuits may be arranged (e.g.,
10 internally or with respect to external entities such as other circuits) in a specified manner as a module. In an example, the whole or part of one or more computer systems (e.g., a standalone, client or server computer system) or one or more hardware processors may be configured by firmware or software (e.g., instructions, an application portion, or an application) as a module that operates to perform
15 specified operations. In an example, the software may reside on a machine readable medium. In an example, the software, when executed by the underlying hardware of the module, causes the hardware to perform the specified operations.

[0040] Accordingly, the term “module” is understood to encompass a tangible entity, be that an entity that is physically constructed, specifically configured (e.g.,
20 hardwired), or temporarily (e.g., transitorily) configured (e.g., programmed) to operate in a specified manner or to perform part or all of any operation described herein. Considering examples in which modules are temporarily configured, each of the modules need not be instantiated at any one moment in time. For example, where the modules comprise a general-purpose hardware processor configured
25 using software, the general-purpose hardware processor may be configured as respective different modules at different times. Software may accordingly configure a hardware processor, for example, to constitute a particular module at one instance of time and to constitute a different module at a different instance of time.

[0041] Machine (e.g., computer system) 6000 may include a hardware processor
30 6002 (e.g., a central processing unit (CPU), a graphics processing unit (GPU), a

hardware processor core, or any combination thereof), a main memory 6004 and a static memory 6006, some or all of which may communicate with each other via an interlink (e.g., bus) 6008. The machine 6000 may further include a display unit 6010, an alphanumeric input device 6012 (e.g., a keyboard), and a user interface (UI) navigation device 6014 (e.g., a mouse). In an example, the display unit 6010, input device 6012 and UI navigation device 6014 may be a touch screen display. The machine 6000 may additionally include a storage device (e.g., drive unit) 6016, a signal generation device 6018 (e.g., a speaker), a network interface device 6020, and one or more sensors 6021, such as a global positioning system (GPS) sensor, compass, accelerometer, or other sensor. The machine 6000 may include an output controller 6028, such as a serial (e.g., universal serial bus (USB), parallel, or other wired or wireless (e.g., infrared(IR), near field communication (NFC), etc.) connection to communicate or control one or more peripheral devices (e.g., a printer, card reader, etc.).

15 [0042] The storage device 6016 may include a machine readable medium 6022 on which is stored one or more sets of data structures or instructions 6024 (e.g., software) embodying or utilized by any one or more of the techniques or functions described herein. The instructions 6024 may also reside, completely or at least partially, within the main memory 6004, within static memory 6006, or within the hardware processor 6002 during execution thereof by the machine 6000. In an example, one or any combination of the hardware processor 6002, the main memory 6004, the static memory 6006, or the storage device 6016 may constitute machine readable media. The instructions in machine readable medium 6022 may include native instructions and managed instructions. Both native and managed instructions cause the machine 6000 to perform certain operations, including for example, the operations of FIGs. 4 or 5. As disclosed previously, native instructions are instructions that are in a form for execution on a particular platform. Managed instructions rely upon an additional process at runtime or just before runtime that allows it to run on a particular platform - for example, Just In Time (JIT)

25 30 compilation at run time, an interpreter (e.g., a Java run-time environment), or the

like. These instructions may be translated or converted into instructions that are in a form for execution on a particular platform.

[0043] While the machine readable medium 6022 is illustrated as a single medium, the term "machine readable medium" may include a single medium or
5 multiple media (e.g., a centralized or distributed database, and/or associated caches and servers) configured to store the one or more instructions 6024.

[0044] The term "machine readable medium" may include any medium that is capable of storing, encoding, or carrying instructions for execution by the machine 6000 and that cause the machine 6000 to perform any one or more of the techniques
10 of the present disclosure, or that is capable of storing, encoding or carrying data structures used by or associated with such instructions. Non-limiting machine readable medium examples may include solid-state memories, and optical and magnetic media. Specific examples of machine readable media may include: non-volatile memory, such as semiconductor memory devices (e.g., Electrically
15 Programmable Read-Only Memory (EPROM), Electrically Erasable Programmable Read-Only Memory (EEPROM)) and flash memory devices; magnetic disks, such as internal hard disks and removable disks; magneto-optical disks; Random Access Memory (RAM); Solid State Drives (SSD); and CD-ROM and DVD-ROM disks. In some examples, machine readable media may include non-transitory machine
20 readable media. In some examples, machine readable media may include machine readable media that is not a transitory propagating signal.

[0045] The instructions 6024 may further be transmitted or received over a communications network 6026 using a transmission medium via the network interface device 6020. The Machine 6000 may communicate with one or more
25 other machines utilizing any one of a number of transfer protocols (e.g., frame relay, internet protocol (IP), transmission control protocol (TCP), user datagram protocol (UDP), hypertext transfer protocol (HTTP), etc.). Example communication networks may include a local area network (LAN), a wide area network (WAN), a packet data network (e.g., the Internet), mobile telephone networks (e.g., cellular
30 networks), Plain Old Telephone (POTS) networks, and wireless data networks (e.g.,

Institute of Electrical and Electronics Engineers (IEEE) 802.11 family of standards known as Wi-Fi®, IEEE 802.16 family of standards known as WiMax®, IEEE 802.15.4 family of standards, a Long Term Evolution (LTE) family of standards, a Universal Mobile Telecommunications System (UMTS) family of standards, peer-to-peer (P2P) networks, among others. In an example, the network interface device 6020 may include one or more physical jacks (e.g., Ethernet, coaxial, or phone jacks) or one or more antennas to connect to the communications network 6026. In an example, the network interface device 6020 may include a plurality of antennas to wirelessly communicate using at least one of single-input multiple-output (SIMO), multiple-input multiple-output (MIMO), or multiple-input single-output (MISO) techniques. In some examples, the network interface device 6020 may wirelessly communicate using Multiple User MIMO techniques.

OTHER NOTES AND EXAMPLES

[0046] Example 1 is a non-transitory machine readable medium that stores instructions which when performed by a machine, cause the machine to perform operations comprising: querying, over a network, an authorization server for an authorization status of an application, the instructions for querying being managed instructions; receiving an authorization status response from the authorization server; responsive to receiving the authorization status response from the authorization server, updating an authorization context with the authorization status, the instructions for updating the authorization context being native instructions; and determining that the application is not authorized based upon the authorization status stored in the authorization context, and in response, performing an unauthorized device function, the instructions for determining being native instructions.

[0047] In Example 2, the subject matter of Example 1 optionally includes, wherein the authorization status comprises a licensing status and wherein the operations of determining that the application is not authorized based upon the authorization status comprises determining that the application is not licensed.

[0048] In Example 3, the subject matter of any one or more of Examples 1–2 optionally include, wherein the operations comprise initializing the authorization context by calculating a keyed hash message authentication code (HMAC) digest value based upon a private cryptographic key and the authorization context.

5 [0049] In Example 4, the subject matter of Example 3 optionally includes, wherein the operations of determining that the application is not authorized comprises: verifying the HMAC digest value by matching the HMAC digest value with an HMAC digest value that is calculated based upon the private cryptographic key and the authorization context; and determining that the authorization status
10 stored in the authorization context indicates that the device is not authorized to run the application.

[0050] In Example 5, the subject matter of any one or more of Examples 1–4 optionally include, wherein the operations of performing the unauthorized device function comprises terminating further execution of the application.

15 [0051] In Example 6, the subject matter of any one or more of Examples 1–5 optionally include, wherein the native instructions are obfuscated using a code obfuscation tool.

[0052] In Example 7, the subject matter of any one or more of Examples 1–6 optionally include, wherein the authorization status received from the authorization
20 server is signed with a first cryptographic key corresponding to the application, and wherein receiving the authorization status comprises verifying the authorization status using a second cryptographic key corresponding to the application and the first cryptographic key, the instructions for verifying the authorization status being native instructions.

25 [0053] In Example 8, the subject matter of any one or more of Examples 1–7 optionally include, wherein the instructions for querying the authorization server, receiving an authorization status, and updating the authorization context are implemented as a library.

[0054] In Example 9, the subject matter of any one or more of Examples 1–8 optionally include, wherein the authorization context is stored as a data structure in memory of the machine.

5 [0055] In Example 10, the subject matter of any one or more of Examples 1–9 optionally include, wherein the operations of querying, over the network, the authorization server and receiving the authorization status from the authorization server comprises the operations of communicating with an authorization service executing on the machine.

10 [0056] Example 11 is a device, comprising: a computer processor; a non-transitory memory, that stores instructions of an application, which when performed by the computer processor, cause the device to perform operations comprising: querying, over a network, an authorization server for an authorization status of an application, the instructions for querying being managed instructions; receiving an authorization status response from the authorization server; responsive to receiving
15 the authorization status response from the authorization server, updating an authorization context with the authorization status, the instructions for updating the authorization context being native instructions; and determining that the application is not authorized based upon the authorization status stored in the authorization context, and in response, performing an unauthorized device function, the
20 instructions for determining being native instructions.

[0057] In Example 12, the subject matter of Example 11 optionally includes, wherein the authorization status comprises a licensing status and wherein the operations of determining that the application is not authorized based upon the authorization status comprises determining that the application is not licensed.

25 [0058] In Example 13, the subject matter of any one or more of Examples 11–12 optionally include, wherein the operations comprise initializing the authorization context by calculating a keyed hash message authentication code (HMAC) digest value based upon a private cryptographic key and the authorization context.

[0059] In Example 14, the subject matter of Example 13 optionally includes,
30 wherein the operations of determining that the application is not authorized

comprises: verifying the HMAC digest value by matching the HMAC digest value with an HMAC digest value that is calculated based upon the private cryptographic key and the authorization context; and determining that the authorization status stored in the authorization context indicates that the device is not authorized to run the application.

[0060] In Example 15, the subject matter of any one or more of Examples 11–14 optionally include, wherein the operations of performing the unauthorized device function comprises terminating further execution of the application.

[0061] In Example 16, the subject matter of any one or more of Examples 11–15 optionally include, wherein the native instructions are obfuscated using a code obfuscation tool.

[0062] In Example 17, the subject matter of any one or more of Examples 11–16 optionally include, wherein the authorization status received from the authorization server is signed with a first cryptographic key corresponding to the application, and wherein receiving the authorization status comprises verifying the authorization status using a second cryptographic key corresponding to the application and the first cryptographic key, the instructions for verifying the authorization status being native instructions.

[0063] In Example 18, the subject matter of any one or more of Examples 11–17 optionally include, wherein the instructions for querying the authorization server, receiving an authorization status, and updating the authorization context are implemented as a library.

[0064] In Example 19, the subject matter of any one or more of Examples 11–18 optionally include, wherein the authorization context is stored as a data structure in memory of the machine.

[0065] In Example 20, the subject matter of any one or more of Examples 11–19 optionally include, wherein the operations of querying, over the network, the authorization server and receiving the authorization status from the authorization server comprises the operations of communicating with an authorization service executing on the machine.

[0066] Example 21 is a method comprising: using a computer processor: querying, over a network, an authorization server for an authorization status of an application, the querying being done using managed instructions; receiving an authorization status response from the authorization server; responsive to receiving
5 the authorization status response from the authorization server, updating an authorization context with the authorization status, the updating the authorization context being done using native instructions; and determining that the application is not authorized based upon the authorization status stored in the authorization context, and in response, performing an unauthorized device function, the
10 instructions for determining being done using native instructions.

[0067] In Example 22, the subject matter of Example 21 optionally includes, wherein the authorization status comprises a licensing status and wherein determining that the application is not authorized based upon the authorization status comprises determining that the application is not licensed.

15 [0068] In Example 23, the subject matter of any one or more of Examples 21–22 optionally include initializing the authorization context by calculating a keyed hash message authentication code (HMAC) digest value based upon a private cryptographic key and the authorization context.

[0069] In Example 24, the subject matter of Example 23 optionally includes,
20 wherein determining that the application is not authorized comprises: verifying the HMAC digest value by matching the HMAC digest value with an HMAC digest value that is calculated based upon the private cryptographic key and the authorization context; and determining that the authorization status stored in the authorization context indicates that the device is not authorized to run the
25 application.

[0070] In Example 25, the subject matter of any one or more of Examples 21–24 optionally include, wherein performing the unauthorized device function comprises terminating further execution of the application.

[0071] In Example 26, the subject matter of any one or more of Examples 21–25 optionally include, wherein the native instructions are obfuscated using a code obfuscation tool.

5 [0072] In Example 27, the subject matter of any one or more of Examples 21–26 optionally include, wherein the authorization status received from the authorization server is signed with a first cryptographic key corresponding to the application, and wherein receiving the authorization status comprises verifying the authorization status using a second cryptographic key corresponding to the application and the first cryptographic key.

10 [0073] In Example 28, the subject matter of any one or more of Examples 21–27 optionally include, wherein querying the authorization server, receiving an authorization status, and updating the authorization context are performed using a library.

15 [0074] In Example 29, the subject matter of any one or more of Examples 21–28 optionally include, wherein the authorization context is stored as a data structure in memory of the machine.

[0075] In Example 30, the subject matter of any one or more of Examples 21–29 optionally include, wherein querying, over the network, the authorization server and receiving the authorization status from the authorization server comprises
20 communicating with an authorization service executing on the machine.

[0076] Example 31 is a device comprising: means for querying, over a network, an authorization server for an authorization status of an application, the querying being done using managed instructions; means for receiving an authorization status response from the authorization server; means for updating an authorization context
25 with the authorization status responsive to receiving the authorization status response from the authorization server, the updating the authorization context being done using native instructions; and means for determining that the application is not authorized based upon the authorization status stored in the authorization context, and in response, means for performing an unauthorized device function, the
30 instructions for determining being done using native instructions.

[0077] In Example 32, the subject matter of Example 31 optionally includes, wherein the authorization status comprises a licensing status and wherein means for determining that the application is not authorized based upon the authorization status comprises means for determining that the application is not licensed.

- 5 [0078] In Example 33, the subject matter of any one or more of Examples 31–32 optionally include means for initializing the authorization context by calculating a keyed hash message authentication code (HMAC) digest value based upon a private cryptographic key and the authorization context.

- [0079] In Example 34, the subject matter of Example 33 optionally includes,
10 wherein means for determining that the application is not authorized comprises: means for verifying the HMAC digest value by matching the HMAC digest value with an HMAC digest value that is calculated based upon the private cryptographic key and the authorization context; and means for determining that the authorization status stored in the authorization context indicates that the device is not authorized
15 to run the application.

[0080] In Example 35, the subject matter of any one or more of Examples 31–34 optionally include, wherein means for performing the unauthorized device function comprises means for terminating further execution of the application.

- [0081] In Example 36, the subject matter of any one or more of Examples 31–35
20 optionally include, wherein the native instructions are obfuscated using a code obfuscation tool.

- [0082] In Example 37, the subject matter of any one or more of Examples 31–36 optionally include, wherein the authorization status received from the authorization server is signed with a first cryptographic key corresponding to the application, and
25 wherein means for receiving the authorization status comprises means for verifying the authorization status using a second cryptographic key corresponding to the application and the first cryptographic key.

[0083] In Example 38, the subject matter of any one or more of Examples 31–37 optionally include, wherein means for querying the authorization server, means for

receiving an authorization status, and means for updating the authorization context are performed using a library.

[0084] In Example 39, the subject matter of any one or more of Examples 31–38 optionally include, wherein the authorization context is stored as a data structure in
5 memory of the device.

[0085] In Example 40, the subject matter of any one or more of Examples 31–39 optionally include, wherein means for querying, over the network, the authorization server and means for receiving the authorization status from the authorization server comprises means for communicating with an authorization service executing on the
10 device.

WHAT IS CLAIMED IS:

1. A non-transitory machine readable medium that stores instructions which
when performed by a machine, cause the machine to perform operations
5 comprising:
 - querying, over a network, an authorization server for an authorization status
of an application, the instructions for querying being managed instructions;
 - receiving an authorization status response from the authorization server;
 - responsive to receiving the authorization status response from the
10 authorization server, updating an authorization context with the authorization status,
the instructions for updating the authorization context being native instructions; and
 - determining that the application is not authorized based upon the
authorization status stored in the authorization context, and in response, performing
an unauthorized device function, the instructions for determining being native
15 instructions.
2. The machine readable medium of claim 1, wherein the authorization status
comprises a licensing status and wherein the operations of determining that the
application is not authorized based upon the authorization status comprises
20 determining that the application is not licensed.
3. The machine readable medium of claim 1, wherein the operations comprise
initializing the authorization context by calculating a keyed hash message
authentication code (HMAC) digest value based upon a private cryptographic key
25 and the authorization context.
4. The machine readable medium of claim 3, wherein the operations of
determining that the application is not authorized comprises:

verifying the HMAC digest value by matching the HMAC digest value with an HMAC digest value that is calculated based upon the private cryptographic key and the authorization context; and

determining that the authorization status stored in the authorization context
5 indicates that the device is not authorized to run the application.

5. The machine readable medium of claim 1, wherein the operations of performing the unauthorized device function comprises terminating further execution of the application.

10

6. The machine readable medium of claim 1, wherein the native instructions are obfuscated using a code obfuscation tool.

7. The machine readable medium of claim 1, wherein the authorization status
15 received from the authorization server is signed with a first cryptographic key corresponding to the application, and wherein receiving the authorization status comprises verifying the authorization status using a second cryptographic key corresponding to the application and the first cryptographic key, the instructions for verifying the authorization status being native instructions.

20

8. The machine readable medium of claim 1, wherein the instructions for querying the authorization server, receiving an authorization status, and updating the authorization context are implemented as a library.

25 9. The machine readable medium of claim 1, wherein the authorization context is stored as a data structure in memory of the machine.

10. The machine readable medium of claim 1, wherein the operations of querying, over the network, the authorization server and receiving the authorization

status from the authorization server comprises the operations of communicating with an authorization service executing on the machine.

11. A device, comprising:

- 5 a computer processor;
a non-transitory memory, that stores instructions of an application, which when performed by the computer processor, cause the device to perform operations comprising:
- 10 querying, over a network, an authorization server for an authorization status of an application, the instructions for querying being managed instructions;
receiving an authorization status response from the authorization server;
responsive to receiving the authorization status response from the
15 authorization server, updating an authorization context with the authorization status, the instructions for updating the authorization context being native instructions; and
determining that the application is not authorized based upon the authorization status stored in the authorization context, and in response,
20 performing an unauthorized device function, the instructions for determining being native instructions.

12. The device of claim 11, wherein the authorization status comprises a licensing status and wherein the operations of determining that the application is not
25 authorized based upon the authorization status comprises determining that the application is not licensed.

13. The device of claim 11, wherein the operations comprise initializing the authorization context by calculating a keyed hash message authentication code

(HMAC) digest value based upon a private cryptographic key and the authorization context.

14. The device of claim 13, wherein the operations of determining that the application is not authorized comprises:
- 5 verifying the HMAC digest value by matching the HMAC digest value with an HMAC digest value that is calculated based upon the private cryptographic key and the authorization context; and
- 10 determining that the authorization status stored in the authorization context indicates that the device is not authorized to run the application.

15. The device of claim 11, wherein the operations of performing the unauthorized device function comprises terminating further execution of the application.
- 15

16. The device of claim 11, wherein the native instructions are obfuscated using a code obfuscation tool.

17. The device of claim 11, wherein the authorization status received from the authorization server is signed with a first cryptographic key corresponding to the application, and wherein receiving the authorization status comprises verifying the authorization status using a second cryptographic key corresponding to the application and the first cryptographic key, the instructions for verifying the authorization status being native instructions.
- 20
- 25

18. The device of claim 11, wherein the instructions for querying the authorization server, receiving an authorization status, and updating the authorization context are implemented as a library.

19. The device of claim 11, wherein the authorization context is stored as a data structure in memory of the machine.

20. The device of claim 11, wherein the operations of querying, over the
5 network, the authorization server and receiving the authorization status from the authorization server comprises the operations of communicating with an authorization service executing on the machine.

21. A method comprising:
10 using a computer processor:
 querying, over a network, an authorization server for an authorization status of an application, the querying being done using managed instructions;
 receiving an authorization status response from the authorization server;
 responsive to receiving the authorization status response from the
15 authorization server, updating an authorization context with the authorization status, the updating the authorization context being done using native instructions; and
 determining that the application is not authorized based upon the authorization status stored in the authorization context, and in response, performing an unauthorized device function, the instructions for determining being done using
20 native instructions.

22. The method of claim 21, wherein the authorization status comprises a licensing status and wherein determining that the application is not authorized based upon the authorization status comprises determining that the application is not
25 licensed.

23. The method of claim 21, comprising initializing the authorization context by calculating a keyed hash message authentication code (HMAC) digest value based upon a private cryptographic key and the authorization context.
30

24. The method of claim 23, wherein determining that the application is not authorized comprises:

verifying the HMAC digest value by matching the HMAC digest value with an HMAC digest value that is calculated based upon the private cryptographic key

5 and the authorization context; and

determining that the authorization status stored in the authorization context indicates that the device is not authorized to run the application.

25. The method of claim 21, wherein performing the unauthorized device

10 function comprises terminating further execution of the application.

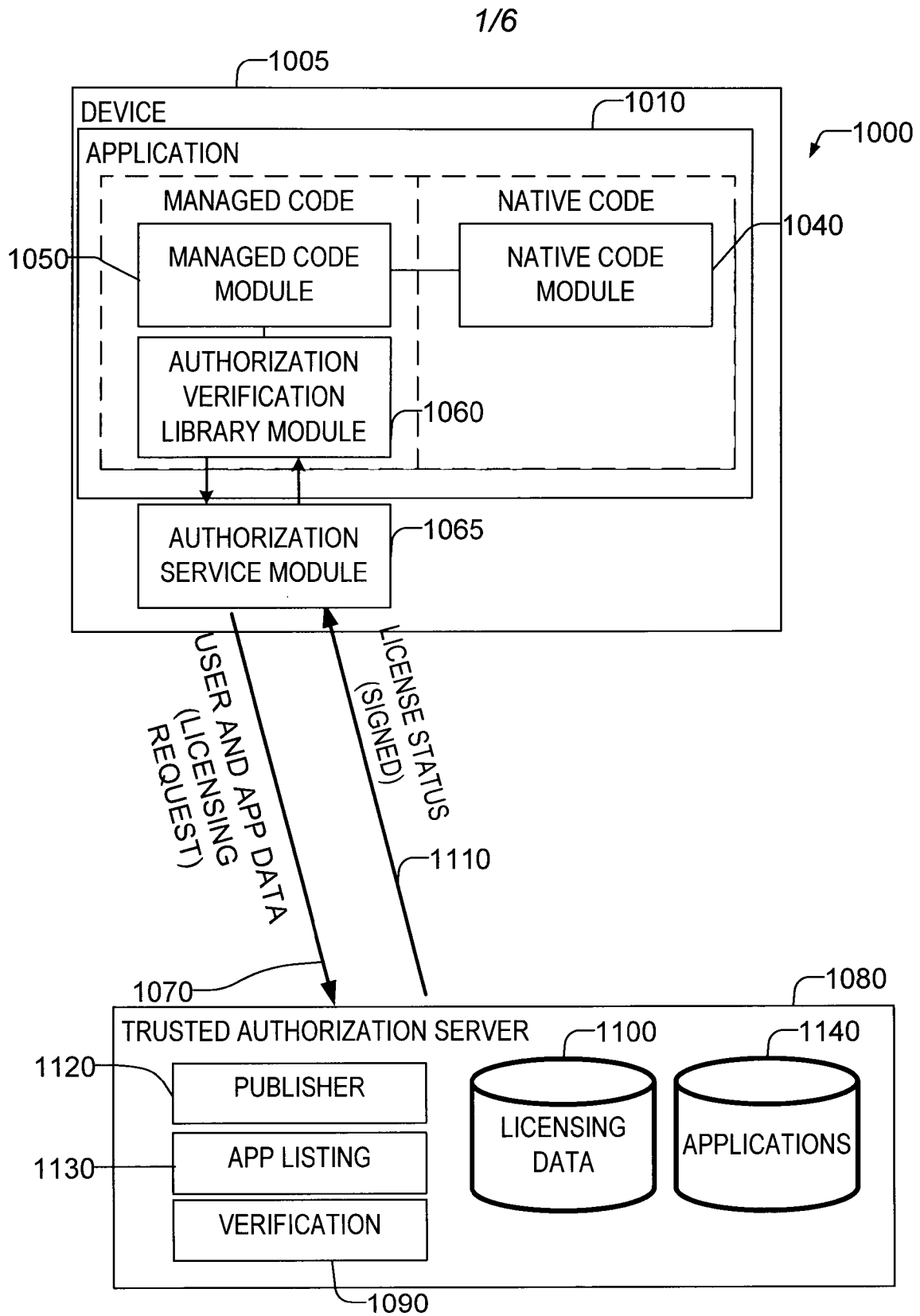


FIG. 1

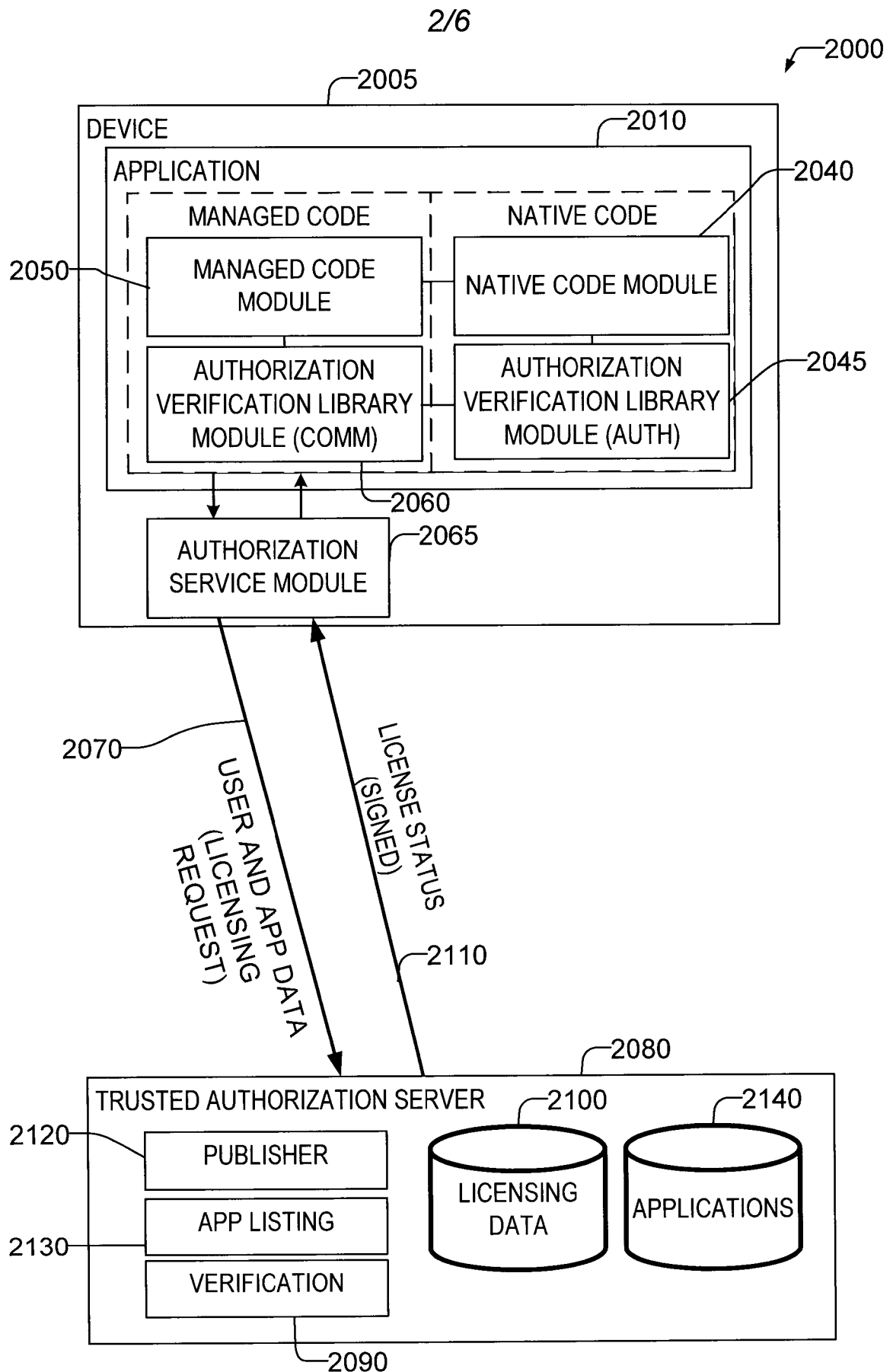
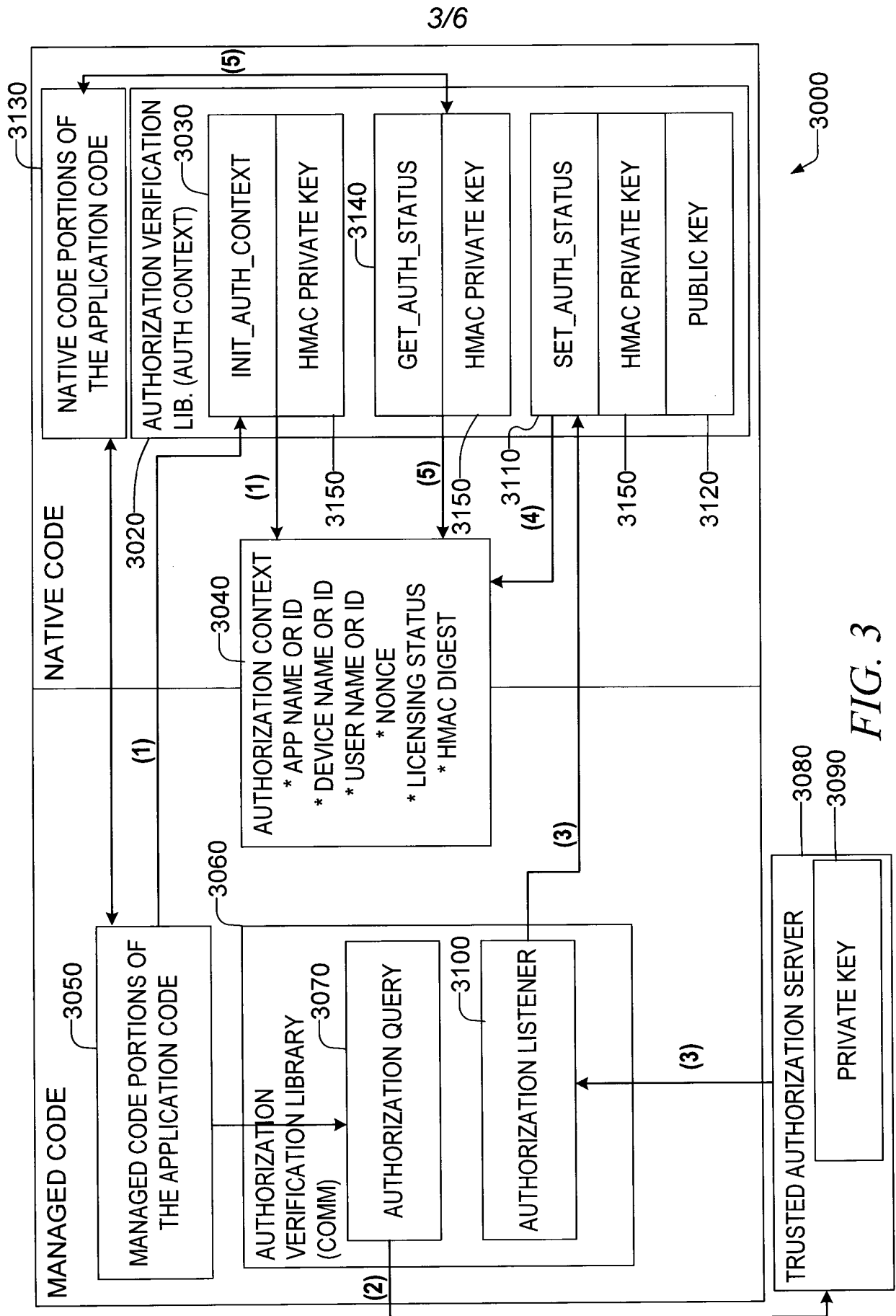
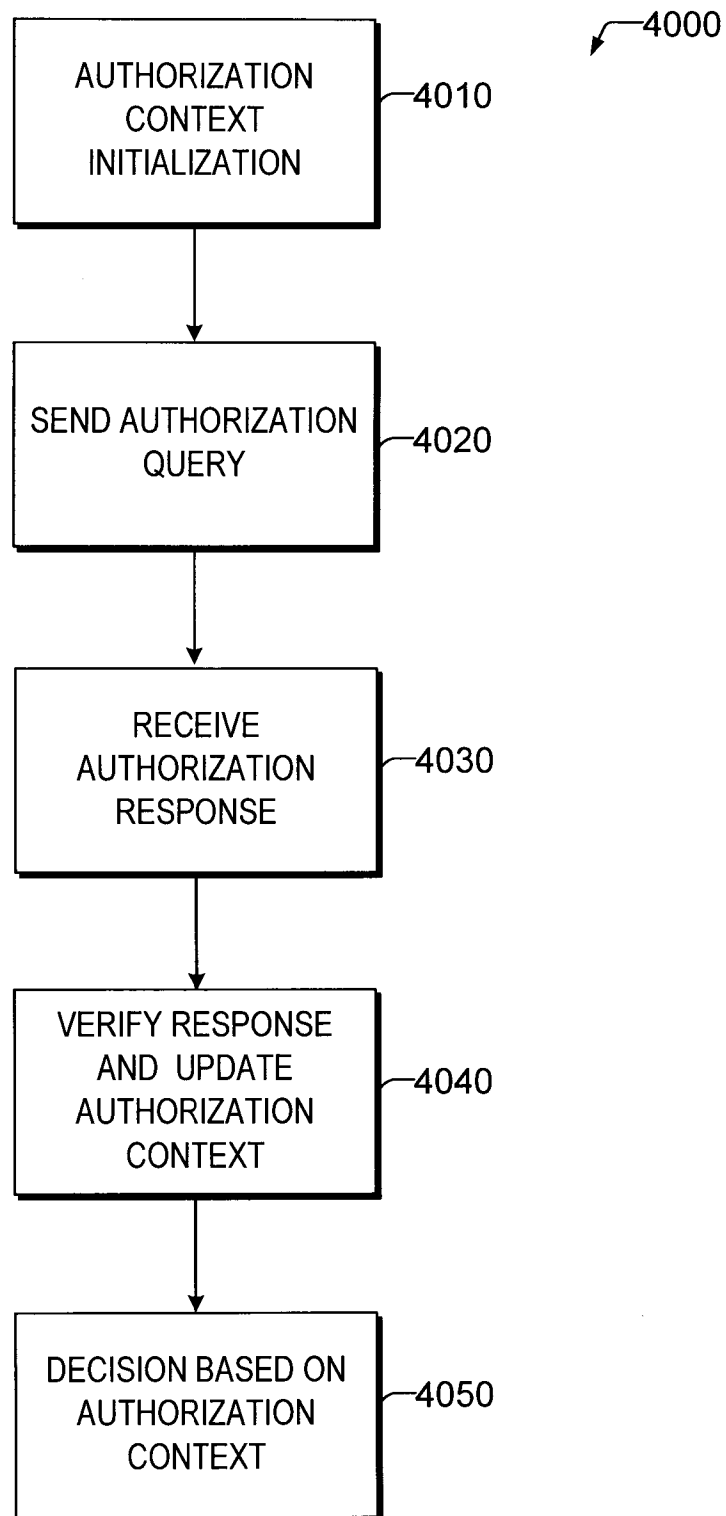


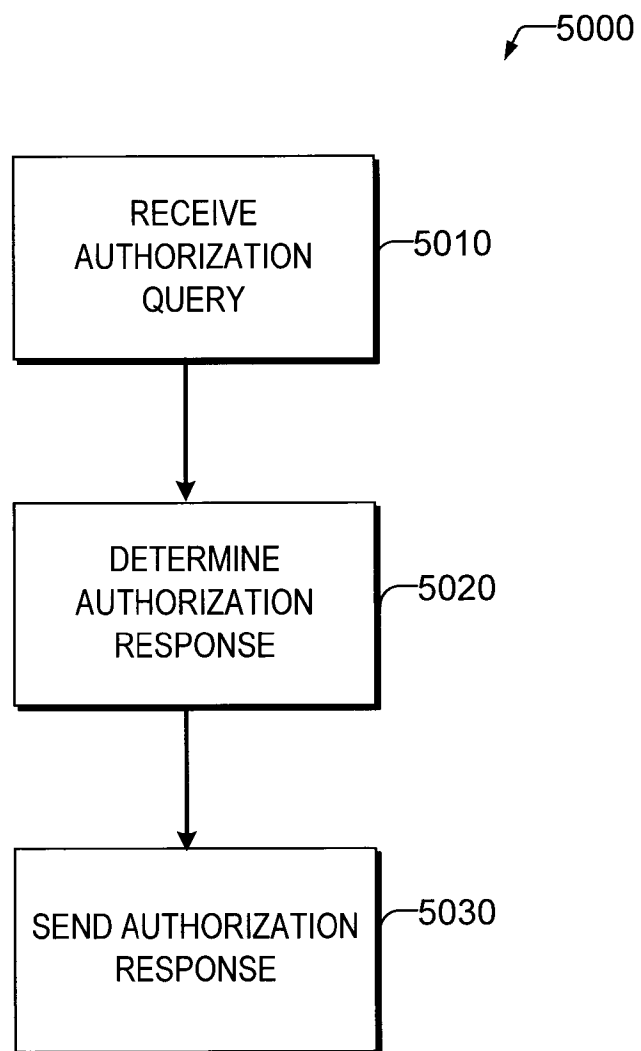
FIG. 2



4/6

*FIG. 4*

5/6

*FIG. 5*

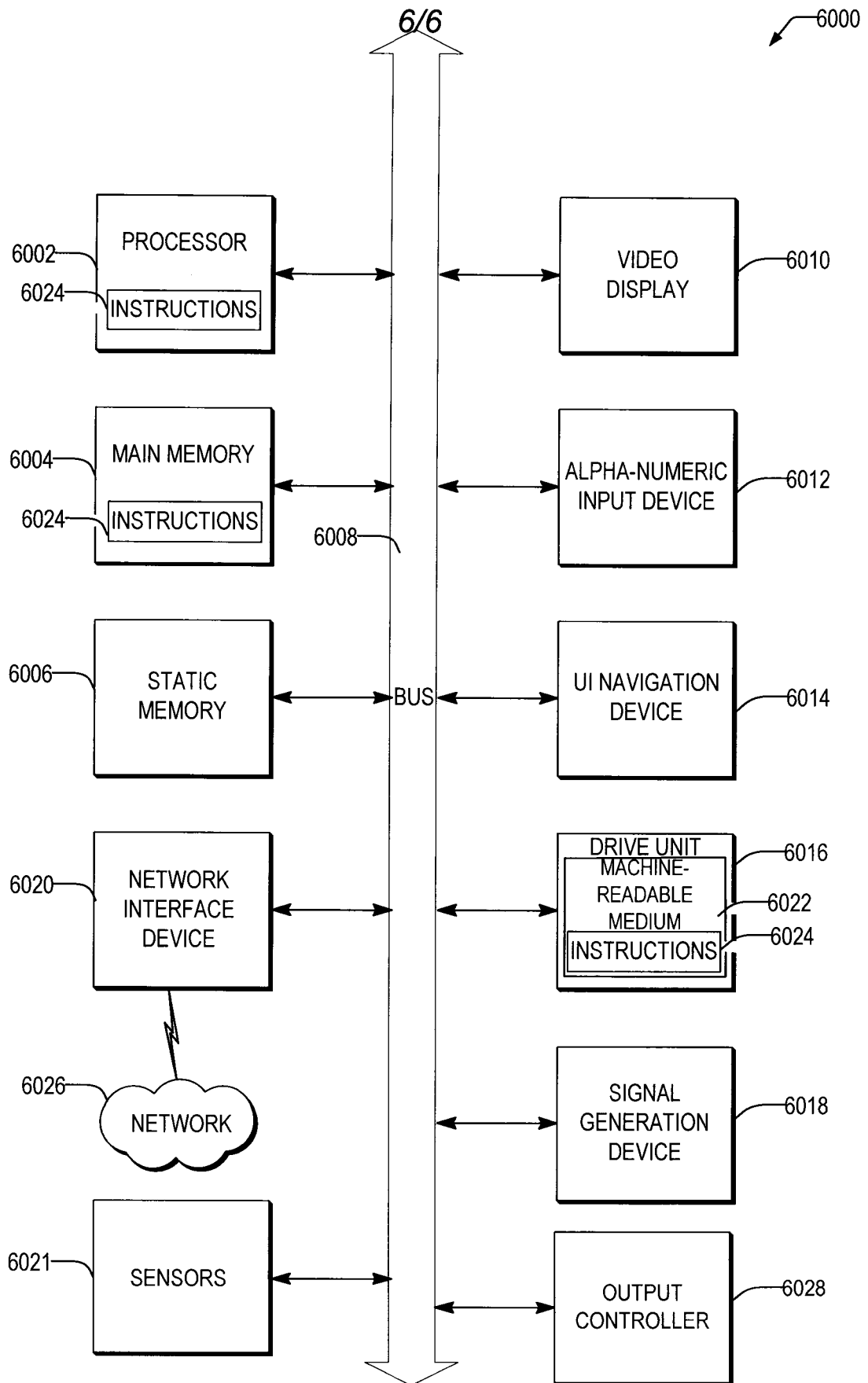


FIG. 6

INTERNATIONAL SEARCH REPORT

International application No
PCT/RU2015/000845

A. CLASSIFICATION OF SUBJECT MATTER INV. G06F21/12 G06F21/64 G06F21/60 G06F15/167 G06F9/54 ADD. H04L9/32		
According to International Patent Classification (IPC) or to both national classification and IPC		
B. FIELDS SEARCHED Minimum documentation searched (classification system followed by classification symbols) G06F H04L		
Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched		
Electronic data base consulted during the international search (name of data base and, where practicable, search terms used) EPO-Internal, WPI Data		
C. DOCUMENTS CONSIDERED TO BE RELEVANT		
Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	"Docand Reference Series Android 4 - Available Services - Nov 2013 - API 19", 1 November 2013 (2013-11-01), XP055210602, Retrieved from the Internet: URL:https://docand.com/wp-content/uploads/2013/10/Docand-Reference-Series-Android-AP119-Nov-Book4-Available-Services.pdf [retrieved on 2015-09-01] pages 229-231 pages 237-238 pages 241-249 page 256 ----- -/-	1-25
☒ Further documents are listed in the continuation of Box C. ☒ See patent family annex.		
* Special categories of cited documents : "A" document defining the general state of the art which is not considered to be of particular relevance "E" earlier application or patent but published on or after the international filing date "L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified) "O" document referring to an oral disclosure, use, exhibition or other means "P" document published prior to the international filing date but later than the priority date claimed "T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention "X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone "Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art "&" document member of the same patent family		
Date of the actual completion of the international search		Date of mailing of the international search report
9 September 2016		16/09/2016
Name and mailing address of the ISA/ European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Fax: (+31-70) 340-3016		Authorized officer Rey, Salvador

INTERNATIONAL SEARCH REPORT

International application No

PCT/RU2015/000845

C(Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT		
Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	PROTSENKO MYKOLA ET AL: "Protecting Android Apps Against Reverse Engineering by the Use of the Native Code", 5 August 2015 (2015-08-05), CORRECT SYSTEM DESIGN; [LECTURE NOTES IN COMPUTER SCIENCE; LECT.NOTES COMPUTER], SPRINGER INTERNATIONAL PUBLISHING, CHAM, PAGE(S) 99 - 110, XP047323382, ISSN: 0302-9743 ISBN: 978-3-642-34346-9 [retrieved on 2015-08-05] the whole document -----	1-25
A	"Handbook of Applied Cryptography, Chapter 1: 1.9 Hash functions; Chapter 9: Hash Functions and Data Integrity ED - Menezes A J; Van Oorschot P C; Vanstone S A", 1 January 1997 (1997-01-01), HANDBOOK OF APPLIED CRYPTOGRAPHY; [CRC PRESS SERIES ON DISCRETE MATHEMATICS AND ITS APPLICATIONS], CRC PRESS, BOCA RATON, FL, US, PAGE(S) 33,321 - 331,352, XP002699266, ISBN: 978-0-8493-8523-0 abstract pages 1-3 pages 14-16,19 -----	3,4,13, 14,23,24
A	MICHAEL DIETZ ET AL: "Quire: Lightweight Provenance for Smart Phone Operating Systems", ARXIV.ORG, CORNELL UNIVERSITY LIBRARY, 201 OLIN LIBRARY CORNELL UNIVERSITY ITHACA, NY 14853, 11 February 2011 (2011-02-11), XP080498615, sections 2, 2.3 and 3.3 -----	1-4, 11-14, 21-24
X	US 2015/149363 A1 (WOMACK MARK D [US] ET AL) 28 May 2015 (2015-05-28) the whole document -----	1-25

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No

PCT/RU2015/000845

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2015149363 A1	28-05-2015	AU 2011255512 A1	08-11-2012
		CA 2797131 A1	24-11-2011
		CN 103038775 A	10-04-2013
		CN 105760716 A	13-07-2016
		KR 20130114593 A	17-10-2013
		US 2011289003 A1	24-11-2011
		US 2012023596 A1	26-01-2012
		US 2015149363 A1	28-05-2015
		WO 2011146785 A1	24-11-2011
