

NON-CUSTODIAL TOOL FOR BUILDING DECENTRALIZED COMPUTER APPLICATIONS

BACKGROUND

[0001] Keeping user data secure has been an important issue for a long time. Most applications today are built with username/email/phone + password security model, which are obsolete for many reasons. For example, password leaks have become prevalent. If there exists a database with passwords, its often only a matter of time before they get stolen by sophisticated hackers that will attempt to monetize the password data.

[0002] In fact, even hashed and salted passwords are insecure. Once a database of hashed passwords is stolen, hackers can direct immense distributed computing resources at that database of passwords, utilizing parallel GPUs or botnets with hundreds of thousands of nodes to attempt hundreds of billions of password combinations per second in hopes of recovering plaintext identifier-password pairs.

[0003] Once attackers discover a password that hashes to the same hash as the one stored in the database, they'll take that identifier-password pair and try it on other applications like bank accounts. In many cases, a salted and hashed password database can give up another valid identifier-password pair around every minute. This results in about half a million leaked passwords per year on its own - doubling every few years. For companies using passwords as secrets to encrypt sensitive data, it's very much like symmetric encryption where the encryption key is a weak password that can be easily cracked by brute-force.

[0004] Additionally, user sessions are often hijacked after successful authentication, allowing attackers to exploit that user's application resources. In order to prevent such access, a system would need to re-authenticate a user with every user request. This creates an extremely cumbersome user experience if users have to type in their password at every single request.

[0005] What is needed is an improved system for securing data.

SUMMARY

[0006] The present technology, roughly described, provides a non-custodial authentication system for building decentralized computer applications that execute on a distributed computing system. The present technology works within a web browser, client application, or other software and provides access to decentralized computer applications through the browser. The present technology is non-custodial, wherein a public-private key pair, which represents user identity, is created on a client machine and then directly encrypted by a third-party platform without relying on one centralized computing system.

[0007] The present system uses public-private key pairs, generated for each user and application combination, to perform an act related to identity authentication and authorization. The non-custodial key management system of the present technology uses a hardware security module (HSM), located over a network and remote from the user's device, to store key information and perform encryption and decryption using a master key that cannot be exported.

[0008] In some instances, a method performs non-custodial authentication for a client. The method may begin with generating a private key by the client, wherein the client is associated with a user. A decentralized identifier token (DIDT) can be generated, at a client, from the private key. The user can then be authenticated for an application based on the DIDT.

[0009] In some instances, a non-transitory computer readable storage medium can have embodied thereon a program. The program can be executable by a processor to perform a method for performing non-custodial authentication for a client. The method may begin with generating a private key by the client, wherein the client is associated with a user. A decentralized identifier token (DIDT) can be generated, at a client, from the private key. The user can then be authenticated for an application based on the DIDT.

[0010] In some instances, a system can perform non-custodial authentication for a client. The system can include a server having memory and a processor. The memory can include one or more modules stored in the memory and executed by the processor. When executed by the processor, the system can generate a private key by the client, wherein the client is associated with a user. The system may further generate, at the client, a decentralized identifier token

(DIDT) from the private key. The system can then authenticate the user for an application based on the DIDT.

BRIEF DESCRIPTION OF FIGURES

- [0011] FIGURE 1 is a block diagram of a delegated key management system.
- [0012] FIGURE 2A illustrates a process for generating a private key in a non-custodial system.
- [0013] FIGURE 2B illustrates a method for signing a transaction by a user.
- [0014] FIGURE 2C illustrates a method for recovering a private key.
- [0015] FIGURE 3 is an exemplary method of a login process for a delegated key management system.
- [0016] FIGURE 4 is an exemplary method for requesting an access token.
- [0017] FIGURE 5 is an exemplary method for determining scoped credentials.
- [0018] FIGURE 6 is an exemplary method for determining permission.
- [0019] FIGURE 7 is an exemplary method for sending a decentralized application transaction.
- [0020] FIGURE 8 is an exemplary method for sending a decentralized contract function.
- [0021] FIGURE 9 is an exemplary method for performing a decentralized user signing.
- [0022] FIGURE 10 is an exemplary method for generating a decentralized identifier token.
- [0023] FIGURE 11 is an exemplary method for performing application authentication with a decentralized identifier token.
- [0024] FIGURE 12 is an exemplary method for performing account recovery.
- [0025] FIGURE 13 is an exemplary method for logging out a user.
- [0026] FIGURE 14 is a block diagram of a computing environment for implementing the present technology.

DETAILED DESCRIPTION

[0027] The present technology, roughly described, provides a non-custodial authentication system for building decentralized computer applications that execute on a distributed computing system. The present technology works within a web browser, client application, or other software and provides access to decentralized computer applications through the browser. The present technology is non-custodial, wherein a public-private key pair, which represents user identity, is created on a client machine and then directly encrypted by a third-party platform without relying on one centralized computing system

[0028] The present system uses public-private key pairs, generated for each user and application combination, to perform an act related to identity authentication and authorization. The non-custodial key management system of the present technology uses a “hardware wallet” system, located over a network and remote from the user’s device, to store key information and perform encryption and decryption.

[0029] Decentralized authentication is provided using key pairs generated by the present system. The system provides a key pair to a user, which can be used to sign an authentication token (decentralized identifier token, or DIDT). The key represents the user’s identity. The generated token can be used to authenticate and authorize user actions. The key pairs can be used to implement delegated key management, or non-custodial key management.

[0030] The present mechanism can be used with third-party systems that provide a platform to implement serverless architecture. The platform may be a global platform for serverless application, such as for example Amazon, Inc. Other third-party platforms may be used as well.

[0031] The present technology described herein addresses technical problems related to managing user identify and protecting the authentication and authorization process. Mistakes lead to unauthorized access and even financial loss from compromised user privacy or identity. Most application use username and password combinations to determine a user identify. These methods are obsolete at least because passwords leaks are prevalent, and resources can be used to recover hashed passwords that have been stolen from company resources. Cryptographic standards can also be used to protect data, and have been used in state-of-the-art security systems.

However, consumer deployment of cryptography-based security has failed to provide an acceptable user experience. For example, cryptographic keys are computer-generated, long strings of text that can't be changed, so they're hard for users to work with. Further, it can be very difficult, often impossible, to recover from loss or theft of a private key.

[0032] The present technology solves the technical problem related to managing user identity and protecting the authentication and authorization process. The present technology uses key management techniques and a decentralized identifier token (DIDT) to manage user identity. The key management involves using remote hardware security modules (HSMs) to encrypt a private key generated at a client. The present technology provides a decentralized solution with protection that is more secure than passwords and usernames, but provides an additional benefit of being able to decrypt a private key if the key is lost on the client device.

[0033] FIGURE 1 is a block diagram of a delegated key management system. Block diagram 100 of FIGURE 1 includes client 110, server 125, and third-party system 155. Client 110 includes browser 115 and user identity 120. User identity may include user identification information, metadata, and other data associated with the user of the overall system. Browser 115 may include an application that renders network content, including content pages, through an interface.

[0034] Server 125 includes web content 130, server engine 135, user table 140, key table 145, and user database 150. Web content 130 is content provided by server 125 to browser 115 at client 110. The web content includes content pages, interfaces, and other information for allowing a user at client 110 to access a delegated key management system. Server engine 135 includes an API for connecting to server 125. User table 140 includes user metadata and identification information for the user. Key table 145 includes an encrypted key that is stored for each user. The user database includes identification information for each user.

[0035] Client 110 and server 125 may be used with a variety of different third-party systems to provide a delegated key management system. The modules discussed with respect to third-party 155 are exemplary, and other types of third-party systems are intended to be compatible with server 125 and client 110.

[0036] Third-party system 155 includes one or more servers that provide access token service 160, credentials service 165, permission service 170, and hardware security module resource 175.

Access token service 160 can generate and provide a token to a user. Credentials service 165 handles credentials for the particular user. Permission service 170 is used to process a received request with a credential. For example, the permission service can perform checks to see if the user has permission to access a resource. Hardware security module resource 175 provides security for users to encrypt and decrypt data. Credentials service 165 provides direct access into the third-party system bypassing server 125.

[0037] In operation, a user at client 110 may perform authentication through server 125. The server 125 may then communicate with the third-party system, and the third-party system may then generate a token for the end-user. The token can be used to exchange for a specific credential. For example, the credential can be used to access the hardware security module resource 175 to encrypt and decrypt on the client side. Providing a token, obtaining a credential, and performing transactions are discussed in more detail below with respect to FIGURES 2-10.

[0038] FIGURE 2A illustrates a process for generating a private key in a non-custodial system. FIGURE 2 includes client 110, server 125, third party service 155, and hardware security module 175. Operations 202-220 occur between the elements 110, 125, 155, and 175. First, a sign-up event is detected at client 202. A sign-up event may include a new account request initiated by a user at the client device. A sign-up request message 202 is transmitted from the client 110 to server 125. The server 125 receives the sign-up message 205 and authenticates the user. Authentication can include sending an email to an email account associated with the user that initiated the sign-up event.

[0039] After authenticating the user, the server generates a new user request message 204. The new user request message is transmitted from server 125 to third party service 155. The third-party service receives the new user request, generates a time bound access token, and transmits the token to server 125 at 206. The time bound access token may include time-to-live (TTL) data embedded within the token. The server forwards the time bound token to client 110 at 208. The time bound token allows client 110 to directly communicate with third party service 155.

[0040] The client 110 transmits the time bound access token to third party service 155 at 210. The third party service receives the access token (e.g., time bound access token), generates a master key and scoped credential (e.g., a record that contains authentication information), and transmits

the scoped credential to the user at step 212. By providing the scoped credentials to the user, the user is enabled to work with the third-party service to access their master keys stored on the third party service, for example for encryption and decryption. The server is bypassed in this step, and cannot forge or intercept the scoped credentials. In some instances, the access token and scoped credentials can be created dynamically by the third-party service, with audit logs and with a time-to-live (TTL) enabled.

[0041] The client generates a public-private key pair based on the scoped credentials at step 214. The public-private key pair is generated within a JavaScript iframe implemented within a network browser at the client. The iframe is inaccessible to the application which may be integrating the present technology. The public-private key pair can be generated using cryptography, for example cryptographically secure pseudo-random 256-bit entropy. In some instances, the public-private key pair can be generated using ECDSA (secp256k1) algorithm, which can also be used to sign a payload and validate a DIDT. Once generated, the generated key pair and the scoped credentials are transmitted to the third-party service at step 216. The third-party service may then send the generated key pair and the scoped credentials to the hardware security module at step 217. The hardware service module is dedicated for the user and the particular application. The hardware service module receives the private key and scoped credentials, encrypts the private key using a master key, and transmits the encrypted private key back to the third-party service at step 218. The third-party service then transmits the encrypted private key to the client at step 218. After the keys are encrypted, the key is stored on the iframe of the client network browser and a copy of the encrypted key is uploaded to server 125 at step 220. The uploaded copy of the key can be used to recover the user's identity. When a user subsequently authenticates, the encrypted private key is accessed by the client from the iframe, and the users can decrypt the encrypted key directly with the third-party service.

[0042] FIGURE 2B illustrates a method for signing a transaction by a user. In FIGURE 2B, the communicating elements are a client with a user logged in 110, a server 125, a third-party service 155, a hardware security module 175, and a blockchain service 185. The client with the user logged in provides a time bounded access token to third-party service at step 230. The third-party service receives the access token, generates a scoped credential, and transmits the scoped

credential back to the client at step 232. The scoped credentials are then transmitted to the hardware security module by the client at step 234. The scoped credentials are transmitted to decrypt an encrypted private key. The hardware security module receives the scoped credentials, decrypts the encrypted private key using the scoped credentials, and returns the decrypted private key at step 236. Once the decrypted private key is received by the client, the client can use the decrypted private key to sign transaction data at step 238. The private key may be purged from memory at the client at step 240. The signed transaction data than may then be submitted to the server at step 242. Server 125 may then submit the signed transaction data to a blockchain service 185 at step 244. Once received, the sign transaction data is stored by the blockchain 185, and a confirmation or submission result is returned to the server 125 at step 246. Upon receiving the submission result, the result is returned to the client from the server at step 248.

[0043] FIGURE 2C illustrates a method for recovering a private key. The elements communicating in FIGURE 2C include client 110, server 125, third party servers 155, and hardware security module 175. First, the client may send a message to server 125 regarding a forgotten password request at step 250. Server 125 may send an email to reset the user's password as message 252. The messages sent to the user's email service 280. Users then notified at client 110 and that the server ascent the password reset email at step 254. A user may reset their password and a resent password message is sent from client 110 to server 125 at step 256. A time bound access token request is then sent by server 125 to third-party service 155 at step 258. The third-party service receives the request and generates a time bound access token, and transmits the token to server 125 at step 260. Server 125 receives the token and forwards the access token to client 110 at step 262.

[0044] Client 110 may transmit an access token to the third-party service at step 264. The third-party service 155 may receive the token and transmits scoped credentials for the user back to the client at step 266. Upon receiving the scoped credentials, the client may request an encrypted private key from the server at step 268. When the client initially generates the encrypted private key, a copy is sent to the server 125 in case the user needs to recover their credentials. The server provides encrypted private key in response to the client's request at step 270. The scoped credentials and the private key are transmitted to the hardware security module from client 110

at step 272, with a request to decrypt the private key. The hardware security module 175 receives requests, decrypts the private key, and returns the decrypted private key to client 110 at step 274.

[0045] FIGURE 3 is an exemplary method of a login process for a delegated key management system. User information, such as for example a phone number, email, or other information, is provided by the user and received at the client at step 310. Though a phone number is discussed with respect to the method of FIGURE 3, other information can be received from a user to perform login, such as an email, username, or other information. The phone number received by client 110 is provided to server 125. A determination is made at server 125 as to whether the phone number is associated with an existing account or is a new phone number at step 315. If the phone number is associated with an account and is not a new phone number, the user is authenticated using the phone number at step 355. The method of FIGURE 3 then returns to the login session at step 350 and the login process ends at step 360.

[0046] If the provided user information is a new phone number, email or other information at step 315, then a setup process is initiated for the user. First, authentication is performed at step 320. An access token is then received at step 330. The access token may be received by the user at client 110. More details for receiving an access token are discussed with respect to the method of FIGURE 3. The access token may then be exchanged for scoped credentials at step 335. The access token can only be exchanged for the credentials directly with the third-party system. More detail for step 335 is discussed with respect to the method of FIGURE 4.

[0047] Concurrently with step 320-335, a public key and private key are created at step 325. The public and private key may be generated in the browser of a client device 110. A hardware security module with scoped credentials is accessed to encrypt the private keys at step 340 on a client device 110, which completely bypasses server 125. In this manner, a user is required to interact with the hardware security module with the scoped credentials in order to encrypt the private keys. More detail for step 340 is discussed with respect to the method of FIGURE 5.

[0048] After encryption of the private keys, the encrypted private key is uploaded at step 345 and the user login session continues at step 350. The method of FIGURE 3 then ends at step 360.

[0049] FIGURE 4 is an exemplary method for requesting a token. The method of FIGURE 4 provides more detail for step 230 of the method of FIGURE 2. First, a token request is received at

step 410. A determination is then made at step 415 as to whether the token request is associated with a new user. If the request is not from a new user, a user identity and resources retrieved at step 435. In some instances, the user identity and resource may be specific to the third-party. The method then continues to step 440 where the access token is returned to the user at step 440, and the method of FIGURE 4 ends at step 445.

[0050] If at step 415 the user is determined to be a new user based on the token request, a user account is created and added to an identity pool at step 420. Optionally, a dedicated hardware security module resource is then created at step 425. In some instances, the dedicated hardware security resource may be created for pool, rather than for a particular user. Hence, a dedicated hardware security module resource may only be created at step 425 if an additional pool is needed once the user is created. A policy is updated to restrict access and to link the user identity and the resource at step 430. The access token from the newly created user is then returned at step 440, and the method ends at step 445.

[0051] FIGURE 5 is an exemplary method for determining scoped credentials. First, a scoped credentials request is received at step 510. The supplied access token is then validated at step 515. A determination is made as to whether the token is valid at step 520. If the token is not valid, the scoped credentials request is rejected at step 535 and the process of FIGURE 5 ends at step 540.

[0052] If the token is determined to be valid at step 520, the token is exchanged for scoped credentials at step 525. In some instances, users are directed to a site, wherein the token is exchanged for scoped credentials associated with a particular user identity pool. A user will then have permissions associated with a pool to access particular resources. The scoped credentials are then returned to a user at step 530. The credentials are returned to a client by the third-party system. In some instances, to implement a delegated key management system, the credentials must be returned by the third-party system. The method of FIGURE 5 then ends at step 540.

[0053] FIGURE 6 is an exemplary method for determining permission. The method of FIGURE 6 provides more detail for step 240 the method of FIGURE 2 First, a dedicated hardware security module is accessed at step 610. User supplied credentials are then validated at step 615, and a determination is made as to whether the user credentials are valid at step 620. In some instances, when a user receives credentials back from a third-party service, the third-party service may

subsequently determine if the credentials are valid. If the credentials are not valid, the resource access is rejected at step 625 and the method of FIGURE 6 ends at step 645.

[0054] If the credentials are determined to be valid at step 620, the credentials are checked against a policy at step 630. The policy is applied to uses associated with a particular user pool. There is an assumed role, and anyone who authenticates against the specific user pool is checked with their credentials. If authentication is successful, a user can become the specified role. The specific role will have a specific policy attached to the role.

[0055] After checking credentials against the policy, a determination is made as to whether the user can access the resource at step 635. If a user does not have permission to access the resource, resource access is rejected at step 625 and the method continues to step 645. If a user can access the resource, as determined by the third-party system, resource access is allowed at step 640 and the method of FIGURE 6 ends at step 645.

[0056] In some instances, the present technology can be used to sign transactions. Transaction data can be signed based on an encrypted private key. The signed transaction data can be submitted for storage at a plurality of linked peer-to-peer computers that store data in blocks that are linked together.

[0057] FIGURE 7 is an exemplary method for sending a decentralized application transaction. First, a user is logged in at step 710. After login, a determination is made at step 715 as to whether encrypted private key exists on the client 110 for the login user. If an encrypted private key exists, the access token is received at step 725. If the encrypted private key does not exist for the user, encrypted private key is retrieved at step 720, and the method continues to step 725.

[0058] An access token is received at step 725. The access token is then exchanged for scoped credentials at step 730. Next, a hardware security module with scoped credentials is accessed to decrypt the encrypted private keys at step 735.

[0059] A payload is constructed at step 740. The payload is constructed for use on a platform for decentralized applications. In some instances, the platform may be a global platform for decentralized applications. Once the decryption and encryption are completed with a private key, user can use it on a client site for execution operations. A payload is then signed using the

private key at step 745. The sign payload is then sent to the decentralized application platform network at step 750. The method of FIGURE 7 then ends at step 755.

[0060] FIGURE 8 is an exemplary method for sending a decentralized contract function. A user logs in at step 810. The login process is discussed in more detail with respect to the method of FIGURE 2. After login, a determination is made as to whether an encrypted private key exists on the client 110 for the logged in user at step 815. If encrypted private key exists, the method of FIGURE 8 continues to step 825. If an encrypted private key does not already exist, the encrypted private key is retrieved at step 820, and the method continues to step 825.

[0061] An access token is received at step 825. The access token may then be exchanged for the scoped credentials at step 830. A hardware security module with scoped credentials can be accessed to decrypt the encrypted private keys at step 835. A contract call payload may then be constructed at step 840. In some instances, the payload can be construed for a decentralized application.

[0062] A payload can be signed with a private key at step 845, and the signed payload is then sent to the decentralized application platform at step 850. The method of FIGURE 8 then ends at step 855.

[0063] FIGURE 9 is an exemplary method for performing a decentralized user signing. A user is logged in at step 910. User login proceeds as discussed with respect to the method of FIGURE 2. After login, a determination is made as to whether a private key exists on the client 90 for the user at step 915. If a user key exists, the method of FIGURE 9 continues to step 925. If the private key does not exist, encrypted private key is retrieved for the logged in user at step 920 and the method of FIGURE 9 continues to step 925.

[0064] An access token is received at step 925. The access token may then be exchanged for the scoped credentials at step 930. A hardware security module with scoped credentials can be accessed to decrypt the encrypted private keys at step 935.

[0065] A payload can then be signed with a private key at step 945. The signed payload can then be sent to a decentralized application platform at step 950. The method of FIGURE 9 then ends at step 955.

[0066] FIGURE 10 is an exemplary method for generating a decentralized identifier token. A user is logged in at step 1010. At some point while the user is logged in, a determination is made as to whether an encrypted private key exists at step 1015. If encrypted private key does not exist, encrypted private key is retrieved at step 1020. The method of figure 10 then continues to step 1025. If the encrypted private key does exist, the system receives an access token at step 1025. The access token is exchanged for scoped credentials by the client from a hardware security module at step 1030. The hardware security module is accessed with the scoped credentials to decrypt the encrypted private keys at step 1035. A decentralized identifier token (DIDT) payload is constructed at step 1040. The DIDT payload is signed with a private key at step 1045. The method of figure 10 then ends at step 1050.

[0067] FIGURE 11 is an exemplary method for performing application authentication with a decentralized identifier token. A login request is received at step 1110. A decentralized identifier token (DIDT) is then generated at step 1115. The DIDT may be generated at the client, for example by a client application or a network browser iframe. The method of FIGURE 10 provides more detail for generating a DIDT token. The DIDT is then returned to the application at step 1120. The application can validate the decentralized identifier token at step 1125. A determination is then made as to whether the decentralized identifier token is validated at step 1130. If the token is not validated, then the method of FIGURE 11 ends at step 1140. If the token is validated by the application, the application successfully validates the user at step 1135. The method of FIGURE 11 then ends at step 1140.

[0068] FIGURE 12 is an exemplary method for performing account recovery. A request to initiate account recovery is received from a user at step 1210. The recovery emails are sent to the user at step 1215. A user may reset the password with a password reset link provided to the user via email and at step 1220. Once the password has been reset, user may be prompted to perform login again at step 1225. Encrypted private key may then be returned to user at step 1230. The method of FIGURE 12 then ends at step 1235.

[0069] FIGURE 13 is an exemplary method for logging out a user. A first, the user is logged out at step 1310. An encrypted private key for the user is cleared at the client-side for the user at step

1320. A user login session is then deactivated at step 1330. The method of FIGURE 13 then ends at step 1340.

[0070] FIGURE 14 is a block diagram of a computing environment for implementing in the present technology. System 1400 of FIGURE 14 may be implemented in the contexts of the likes of machines that implement battery client 140, server 145, and third-party system 155. The computing system 1400 of FIGURE 14 includes one or more processors 1410 and memory 1420. Main memory 1420 stores, in part, instructions and data for execution by processor 1410. Main memory 1420 can store the executable code when in operation. The system 1400 of FIGURE 14 further includes a mass storage device 1430, portable storage medium drive(s) 1440, output devices 1450, user input devices 1460, a graphics display 1470, and peripheral devices 1480.

[0071] The components shown in FIGURE 14 are depicted as being connected via a single bus 1490. However, the components may be connected through one or more data transport means. For example, processor unit 1410 and main memory 1420 may be connected via a local microprocessor bus, and the mass storage device 1430, peripheral device(s) 1480, portable storage device 1440, and display system 1470 may be connected via one or more input/output (I/O) buses.

[0072] Mass storage device 1430, which may be implemented with a magnetic disk drive, an optical disk drive, a flash drive, or other device, is a non-volatile storage device for storing data and instructions for use by processor unit 1410. Mass storage device 1430 can store the system software for implementing embodiments of the present invention for purposes of loading that software into main memory 1420.

[0073] Portable storage device 1440 operates in conjunction with a portable non-volatile storage medium, such as a floppy disk, compact disk or Digital video disc, USB drive, memory card or stick, or other portable or removable memory, to input and output data and code to and from the computer system 1400 of FIGURE 14. The system software for implementing embodiments of the present invention may be stored on such a portable medium and input to the computer system 1400 via the portable storage device 1440.

[0074] Input devices 1460 provide a portion of a user interface. Input devices 1460 may include an alpha-numeric keypad, such as a keyboard, for inputting alpha-numeric and other information, a pointing device such as a mouse, a trackball, stylus, cursor direction keys,

microphone, touchscreen, accelerometer, and other input devices. Additionally, the system 1400 as shown in FIGURE 14 includes output devices 1450. Examples of suitable output devices include speakers, printers, network interfaces, and monitors.

[0075] Display system 1470 may include a liquid crystal display (LCD) or other suitable display device. Display system 1470 receives textual and graphical information and processes the information for output to the display device. Display system 1470 may also receive input as a touchscreen.

[0076] Peripherals 1480 may include any type of computer support device to add additional functionality to the computer system. For example, peripheral device(s) 1480 may include a modem or a router, printer, and other device.

[0077] The system of 1400 may also include, in some implementations, antennas, radio transmitters and radio receivers 1490. The antennas and radios may be implemented in devices such as smart phones, tablets, and other devices that may communicate wirelessly. The one or more antennas may operate at one or more radio frequencies suitable to send and receive data over cellular networks, Wi-Fi networks, commercial device networks such as a Bluetooth device, and other radio frequency networks. The devices may include one or more radio transmitters and receivers for processing signals sent and received using the antennas.

[0078] The components contained in the computer system 1400 of FIGURE 14 are those typically found in computer systems that may be suitable for use with embodiments of the present invention and are intended to represent a broad category of such computer components that are well known in the art. Thus, the computer system 1400 of FIGURE 14 can be a personal computer, handheld computing device, smart phone, mobile computing device, workstation, server, minicomputer, mainframe computer, or any other computing device. The computer can also include different bus configurations, networked platforms, multi-processor platforms, etc. Various operating systems can be used including Unix, Linux, Windows, Macintosh OS, Android, as well as languages including Java, .NET, C, C++, Node.JS, and other suitable languages.

[0079] The foregoing detailed description of the technology herein has been presented for purposes of illustration and description. It is not intended to be exhaustive or to limit the technology to the precise form disclosed. Many modifications and variations are possible in light

of the above teaching. The described embodiments were chosen to best explain the principles of the technology and its practical application to thereby enable others skilled in the art to best utilize the technology in various embodiments and with various modifications as are suited to the particular use contemplated. It is intended that the scope of the technology be defined by the claims appended hereto.