

QUANTUM RESISTANT ENCRYPTION PLATFORM

Reference to Related Application(s)

[0001] This application claims the benefit of Convention priority from Australian Provisional Patent Application No. 2023902937, filed 12 September 2023, the contents of which are incorporated herein by reference in their entirety.

Technical Field

[0002] present invention relates generally to a system and method for encryption of data and, in particular, to a system and method for dynamically applying quantum resistant encryption to portions of data within a data structure or document. The present invention also relates to an apparatus for encryption of data, and to a computer program product including a computer readable medium having recorded thereon a computer program for encryption of data.

Background

[0003] In the context of increasing data security concerns and rapid advancements in quantum computing, traditional encryption methods are becoming increasingly vulnerable to attack. Traditional cryptographic methods, although robust against classical computing, are vulnerable to potential decryption by quantum computers. To address this vulnerability, quantum-resistant cryptography is a growing area of development. However, integrating quantum-resistant cryptography into existing systems, including legacy infrastructure, presents a significant technical problem.

[0004] The complexity of quantum-resistant cryptography is a significant hurdle in itself. The intricate nature of quantum-resistant cryptographic techniques demands a deep understanding of quantum principles, making implementation complex. As a result, the process of correctly integrating quantum-resistant cryptography into existing systems faces difficulty. Some traditional encryption solutions use plugins to provide encryption, with a different plugin often required for each different user application. Having a siloed or proprietary service for individual end points or applications can quickly become, or even untenable, to manage or to scale. Additionally, where data is often required to be accessed by numerous services or applications, a plugin or proprietary application specific solution often does not allow access to encryption keys to be brokered on an as-needed basis to decrypt the data. The computational requirements for some quantum-resistant encryption methods are also computationally

complex, and may not be suitable for implementation on end user devices or edge devices. The broad range of applications where data security is required, and the computational requirements for implementing quantum resistant encryption methods mean that deployment of quantum-resistant encryption faces complexities in practical implementation.

[0005] Furthermore, the integration of quantum-resistant encryption needs to be accompanied by the establishment of access controls and policies, adding further implementation difficulties. Errors in integration due to characteristics of legacy systems can leave data vulnerable to attacks. The challenge may be particularly pronounced when securing legacy systems that were not originally designed with quantum-resistant encryption or modern access control mechanisms in mind.

[0006] As cloud storage increases, data ingestion systems are increasingly used. Existing data ingestion systems often perform essential operations such as virus scanning, indexing, and tagging after data encryption, which can lead to inefficiencies and potential security vulnerabilities.

[0007] A need exists to integrate quantum resistant encryption methods in a manner that can be used agnostic of a particular format or application in which a user intends to enter or retrieve data. A need further exists to avoid inefficiencies and potential security vulnerabilities. Another need exists to maintain access control when integrating quantum resistant encryption methods.

Summary

[0008] It is an object of the present invention to substantially overcome, or at least ameliorate, one or more disadvantages of existing arrangements, for example to address or ameliorate at least one of the needs outlined above.

[0009] The arrangements described in some embodiments provide a platform that allows quantum resistant encryption methods to be used across a wide range of applications through operation of an application program interface (API). The platform allows data to be encrypted and stored in a secure manner without requiring computational complexity at user device. The arrangements described can in some implementations be used to perform pre-processing before encryption to maintain efficiency and security at the point of encryption and/or storage of encrypted data. The arrangements described may be integrated with current access control system in some embodiments. Some arrangements described also afford a user, whether an owner of a data file, or a developer of a data interface, flexibility to dynamically allow different levels and types of

encryption. The arrangement also affords the owner of data an ability to dynamically manage third party access to the encryption keys for said data, agnostic of the application or service.

[0010] According to a first aspect of the present disclosure, there is provided a method for encrypting data, the method comprising: receiving, via an application executing on a client device, a request to encrypt a user selection of a portion of data using a quantum-resistant algorithm; transmitting a message containing information for the selected portion of data of the request to a cloud server via an application program interface (API); encrypting a pair of keys associated with the request at the cloud server; encrypting the portion of data using the quantum-resistant algorithm and the encrypted keys at the cloud server; and updating, via the API display of the portion of data on the client device to replace the portion of data with an indication the portion has been encrypted.

[0011] According to another aspect of the present disclosure, there is provided a computer-readable storage medium which stores a program for executing a method for encrypting data, the method comprising: receiving, via an application executing on a client device, a request to encrypt a user selection of a portion of data using a quantum-resistant encryption algorithm; transmitting a message containing information for the selected portion of data of the request to a cloud server via an application program interface (API); encrypting a pair of keys associated with the request at the cloud server; encrypting the portion of data using the quantum-resistant algorithm and the encrypted keys at the cloud server; and updating, via the API display of the portion of data on the client device to replace the portion of data with an indication the portion has been encrypted.

[0012] Other aspects are also disclosed.

Brief Description of the Drawings

[0013] At least one embodiment of the present invention will now be described with reference to the drawings and appendices, in which:

[0014] Fig. 1 shows a top-level view of a system for encrypting data using quantum resistant encryption methods;

[0015] Figs. 2A and 2B form a schematic block diagram of a general purpose computer system upon which arrangements described can be practiced;

[0016] Fig. 3 shows an example software architecture for use in the system of Fig. 1;

[0017] Fig. 4 shows a method of encrypting data using one or more quantum resistant encryption methods;

[0018] Fig. 5 shows an example presentation of an online document in which data can be encrypted using the method of Fig. 4;

[0019] Fig. 6A shows an example presentation of an online document in which data can be encrypted using the method of Fig. 4;

[0020] Fig. 6B shows an example expanded drop-down menu of Fig. 6A;

[0021] Fig. 7 shows an example presentation of an online form in which data can be encrypted using the method of Fig. 4;

[0022] Fig. 8 shows an example presentation of HTML form in which data can be encrypted using the method of Fig. 4;

[0023] Fig. 9A shows an example JSON payload generated by operation of an API before the encryption methods described are implemented;

[0024] Fig. 9B shows an example updated version of the JSON payload of Fig. 9A after the encryption methods described are implemented;

[0025] Fig. 10 shows a method decrypting data, the encrypted using the method of Fig. 4, based on access controls; and

[0026] Fig. 11 shows an example method 1100 of implementing a HTTP POST command using the arrangements described.

Detailed Description including Best Mode

[0027] Where reference is made in any one or more of the accompanying drawings to steps and/or features, which have the same reference numerals, those steps and/or features have for the purposes of this description the same function(s) or operation(s), unless the contrary intention appears.

[0028] It is to be noted that the discussions contained in the "Background" section and that above relating to prior art arrangements relate to discussions of documents or devices which form public knowledge through their respective publication and/or use. Such should not be interpreted as a representation by the present inventor(s) or the patent applicant that such documents or devices in any way form part of the common general knowledge in the art.

[0029] Fig. 1 shows an example top-level system 100 on which the methods of encrypting data described herein may be practiced. The system 100 includes a client device 110, a communication network 120, an application program interface (API) 130 and a cloud server 140.

[0030] The client device 110 may be any device that a user or developer may use to access, manipulate and/or generate data. For example, the client device 110 may be a personal computer, a laptop, a workstation, a smartphone or the like. The cloud server 140 can be any server or other storage device that is usually a different device to the client device 110 and is typically remote from the client device 110. In the arrangements described, the cloud server 140 can be used implement encryption operations in addition to storing data. The client device 110 communicates with the cloud server 140 via the communications network 120 and the API 130. The API 130 provides an interface to collect and transmit data between the client device 110 and the cloud server 140 via the network 120.

Computer Description

[0031] Figs. 2A and 2B depict a general-purpose computer system 200, upon which the various arrangements described can be practiced.

[0032] As seen in Fig. 2A, the computer system 200 includes: a computer module 201. The computer module 201 provides an implementation of the cloud server 140. In some implementations, multiple instances of the module 201 may form the cloud server 140. The client device 110 can also be implemented in a similar manner to the module 201, or as a device that operated in a similar manner to the module 201 but with limited processing resources.

[0033] The module 201 also includes input devices such as a keyboard 202, a mouse pointer device 203, a scanner 226, a camera 227, and a microphone 280; and output devices including a printer 215, a display device 214 and loudspeakers 217. An external Modulator-Demodulator (Modem) transceiver device 216 may be used by the computer module 201 for communicating

to and from a communications network 120 via a connection 221. The communications network 120 may be a wide-area network (WAN), such as the Internet, a cellular telecommunications network, or a private WAN. Where the connection 221 is a telephone line, the modem 216 may be a traditional "dial-up" modem. Alternatively, where the connection 221 is a high capacity (e.g., cable) connection, the modem 216 may be a broadband modem. A wireless modem may also be used for wireless connection to the communications network 120.

[0034] The computer module 201 typically includes at least one processor unit 205, and a memory unit 206. For example, the memory unit 206 may have semiconductor random access memory (RAM) and semiconductor read only memory (ROM). The computer module 201 also includes a number of input/output (I/O) interfaces including: an audio-video interface 207 that couples to the video display 214, loudspeakers 217 and microphone 280; an I/O interface 213 that couples to the keyboard 202, mouse 203, scanner 226, camera 227 and optionally a joystick or other human interface device (not illustrated); and an interface 208 for the external modem 216 and printer 215. In some implementations, the modem 216 may be incorporated within the computer module 201, for example within the interface 208. The computer module 201 also has a local network interface 211, which permits coupling of the computer system 200 via a connection 223 to a local-area communications network 222, known as a Local Area Network (LAN). As illustrated in Fig. 2A, the local communications network 222 may also couple to the wide network 220 via a connection 224, which would typically include a so-called "firewall" device or device of similar functionality. The local network interface 211 may comprise an Ethernet circuit card, a Bluetooth[®] wireless arrangement or an IEEE 802.11 wireless arrangement; however, numerous other types of interfaces may be practiced for the interface 211.

[0035] The I/O interfaces 208 and 213 may afford either or both of serial and parallel connectivity, the former typically being implemented according to the Universal Serial Bus (USB) standards and having corresponding USB connectors (not illustrated). Storage devices 209 are provided and typically include a hard disk drive (HDD) 210. Other storage devices such as a floppy disk drive and a magnetic tape drive (not illustrated) may also be used. An optical disk drive 212 is typically provided to act as a non-volatile source of data. Portable memory devices, such optical disks (e.g., CD-ROM, DVD, Blu-ray Disc[™]), USB-RAM, portable, external hard drives, and floppy disks, for example, may be used as appropriate sources of data to the system 200.

[0036] The components 205 to 213 of the computer module 201 typically communicate via an interconnected bus 204 and in a manner that results in a conventional mode of operation of the

computer system 200 known to those in the relevant art. For example, the processor 205 is coupled to the system bus 204 using a connection 218. Likewise, the memory 206 and optical disk drive 212 are coupled to the system bus 204 by connections 219. Examples of computers on which the described arrangements can be practised include IBM-PC's and compatibles, Sun Sparcstations, Apple MacTM or like computer systems.

[0037] The methods of data encryption and decryption may be implemented using the computer system 200 wherein the processes of Figs. 4-10 to be described, may be implemented as one or more software application programs 233 executable within the computer system 200. In particular, the steps of the method of Figs. 4 and 10 are effected by instructions 231 (see Fig. 2B) in the software 233 that are carried out within the computer system 200. The software instructions 231 may be formed as one or more code modules, each for performing one or more particular tasks. The software may also be divided into two separate parts, in which a first part and the corresponding code modules performs the described methods and a second part and the corresponding code modules manage a user interface between the first part and the user.

[0038] The software may be stored in a computer readable medium, including the storage devices described below, for example. The software is loaded into the computer system 200 from the computer readable medium, and then executed by the computer system 200. A computer readable medium having such software or computer program recorded on the computer readable medium is a computer program product. The use of the computer program product in the computer system 200 preferably effects an advantageous apparatus for implementing the encryption platform and methods described herein.

[0039] The software 233 is typically stored in the HDD 210 or the memory 206. The software is loaded into the computer system 200 from a computer readable medium, and executed by the computer system 200. Thus, for example, the software 233 may be stored on an optically readable disk storage medium (e.g., CD-ROM) 225 that is read by the optical disk drive 212. A computer readable medium having such software or computer program recorded on it is a computer program product. The use of the computer program product in the computer system 200 preferably effects an apparatus for implementing the methods described.

[0040] In some instances, the application programs 233 may be supplied to the user encoded on one or more CD-ROMs 225 and read via the corresponding drive 212, or alternatively may be read by the user from the networks 120 or 222. Still further, the software can also be loaded into the computer system 200 from other computer readable media. Computer readable storage media refers to any non-transitory tangible storage medium that provides recorded

instructions and/or data to the computer system 200 for execution and/or processing. Examples of such storage media include floppy disks, magnetic tape, CD-ROM, DVD, Blu-ray™ Disc, a hard disk drive, a ROM or integrated circuit, USB memory, a magneto-optical disk, or a computer readable card such as a PCMCIA card and the like, whether or not such devices are internal or external of the computer module 201. Examples of transitory or non-tangible computer readable transmission media that may also participate in the provision of software, application programs, instructions and/or data to the computer module 201 include radio or infra-red transmission channels as well as a network connection to another computer or networked device, and the Internet or Intranets including e-mail transmissions and information recorded on Websites and the like.

[0041] The second part of the application programs 233 and the corresponding code modules mentioned above may be executed to implement one or more graphical user interfaces (GUIs) to be rendered or otherwise represented upon the display 214. Through manipulation of typically the keyboard 202 and the mouse 203, a user of the computer system 200 and the application may manipulate the interface in a functionally adaptable manner to provide controlling commands and/or input to the applications associated with the GUI(s). Other forms of functionally adaptable user interfaces may also be implemented, such as an audio interface utilizing speech prompts output via the loudspeakers 217 and user voice commands input via the microphone 280.

[0042] Fig. 2B is a detailed schematic block diagram of the processor 205 and a “memory” 234. The memory 234 represents a logical aggregation of all the memory modules (including the HDD 209 and semiconductor memory 206) that can be accessed by the computer module 201 in Fig. 2A.

[0043] When the computer module 201 is initially powered up, a power-on self-test (POST) program 250 executes. The POST program 250 is typically stored in a ROM 249 of the semiconductor memory 206 of Fig. 2A. A hardware device such as the ROM 249 storing software is sometimes referred to as firmware. The POST program 250 examines hardware within the computer module 201 to ensure proper functioning and typically checks the processor 205, the memory 234 (209, 206), and a basic input-output systems software (BIOS) module 251, also typically stored in the ROM 249, for correct operation. Once the POST program 250 has run successfully, the BIOS 251 activates the hard disk drive 210 of Fig. 2A. Activation of the hard disk drive 210 causes a bootstrap loader program 252 that is resident on the hard disk drive 210 to execute via the processor 205. This loads an operating system 253 into the RAM memory 206, upon which the operating system 253 commences operation. The

operating system 253 is a system level application, executable by the processor 205, to fulfil various high level functions, including processor management, memory management, device management, storage management, software application interface, and generic user interface.

[0044] The operating system 253 manages the memory 234 (209, 206) to ensure that each process or application running on the computer module 201 has sufficient memory in which to execute without colliding with memory allocated to another process. Furthermore, the different types of memory available in the system 200 of Fig. 2A must be used properly so that each process can run effectively. Accordingly, the aggregated memory 234 is not intended to illustrate how particular segments of memory are allocated (unless otherwise stated), but rather to provide a general view of the memory accessible by the computer system 200 and how such is used.

[0045] As shown in Fig. 2B, the processor 205 includes a number of functional modules including a control unit 239, an arithmetic logic unit (ALU) 240, and a local or internal memory 248, sometimes called a cache memory. The cache memory 248 typically includes a number of storage registers 244 - 246 in a register section. One or more internal busses 241 functionally interconnect these functional modules. The processor 205 typically also has one or more interfaces 242 for communicating with external devices via the system bus 204, using a connection 218. The memory 234 is coupled to the bus 204 using a connection 219.

[0046] The application program 233 includes a sequence of instructions 231 that may include conditional branch and loop instructions. The program 233 may also include data 232 which is used in execution of the program 233. The instructions 231 and the data 232 are stored in memory locations 228, 229, 230 and 235, 236, 237, respectively. Depending upon the relative size of the instructions 231 and the memory locations 228-230, a particular instruction may be stored in a single memory location as depicted by the instruction shown in the memory location 230. Alternately, an instruction may be segmented into a number of parts each of which is stored in a separate memory location, as depicted by the instruction segments shown in the memory locations 228 and 229.

[0047] In general, the processor 205 is given a set of instructions which are executed therein. The processor 205 waits for a subsequent input, to which the processor 205 reacts to by executing another set of instructions. Each input may be provided from one or more of a number of sources, including data generated by one or more of the input devices 202, 203, data received from an external source across one of the networks 120, 202, data retrieved from one of the storage devices 206, 209 or data retrieved from a storage medium 225 inserted into the

corresponding reader 212, all depicted in Fig. 2A. The execution of a set of the instructions may in some cases result in output of data. Execution may also involve storing data or variables to the memory 234.

[0048] The disclosed arrangements use input variables 254, which are stored in the memory 234 in corresponding memory locations 255, 256, 257. The described arrangements produce output variables 261, which are stored in the memory 234 in corresponding memory locations 262, 263, 264. Intermediate variables 258 may be stored in memory locations 259, 260, 266 and 267.

[0049] Referring to the processor 205 of Fig. 2B, the registers 244, 245, 246, the arithmetic logic unit (ALU) 240, and the control unit 239 work together to perform sequences of micro-operations needed to perform “fetch, decode, and execute” cycles for every instruction in the instruction set making up the program 233. Each fetch, decode, and execute cycle comprises:

- a fetch operation, which fetches or reads an instruction 231 from a memory location 228, 229, 230;

- a decode operation in which the control unit 239 determines which instruction has been fetched; and

- an execute operation in which the control unit 239 and/or the ALU 240 execute the instruction.

[0050] Thereafter, a further fetch, decode, and execute cycle for the next instruction may be executed. Similarly, a store cycle may be performed by which the control unit 239 stores or writes a value to a memory location 232.

[0051] Each step or sub-process in the processes of Figs. 4-10, to be described, is associated with one or more segments of the program 233 and is performed by the register section 244, 245, 247, the ALU 240, and the control unit 239 in the processor 205 working together to perform the fetch, decode, and execute cycles for every instruction in the instruction set for the noted segments of the program 233.

[0052] The methods of encrypting and decrypting data described herein may alternatively be implemented in dedicated hardware such as one or more integrated circuits performing the functions or sub functions of the methods described. Such dedicated hardware may include

graphic processors, digital signal processors, or one or more microprocessors and associated memories.

Platform

[0053] Fig. 3 shows an example software architecture 300 for a platform for implementing quantum-resistant encryption. The architecture 300 includes the API 130 and an encryption operations architecture 301. The architecture 301 relates to data processing associated with encryption and includes a data ingestion module 310, a pre-encryption operation module 320, a data restructuring module 330 (also referred to as a polymorphic restructuring data module), a hash encryption keys module 340 and a quantum resistant encryption module 350. The modules 310, 320, 330, 340 and 350 typically form sub-modules of the software application 211, are stored in one or more memories of the module 201 (such as 206) and execute on the processor 205 of the cloud server 140.

[0054] The API 130 provides a software interface between the architecture 301 and applications executing on user devices, such as the client device 110. The API 130 implements a set of rules for transmission of data between the architecture 301 and, the example of Fig. 1, the client device 110. The API 130 transmits messages including data between the client device 110 and the cloud server 140. The messages are typically in JSON (JavaScript Object Notation) but may use other formats such as plain txt, html, XML and the like. The API 130 can provide information presentable as an encryption menu to an application executing on the user device, as described in relation to examples below. As the API 130 provides an interface only, the functionality provided by the API 130 to the client device in terms of menus can be easily implemented on a wide variety of applications, such as browsers, word processors, document viewers, text files, audio streaming applications, video streaming or video playing applications and the like.

[0055] While the arrangements described relate to encryption being performed on the cloud server 140, the API 130 may be configured to additionally support encryption processes on the client device 110, for example classical encryption algorithms rather than quantum-resistant algorithms.

[0056] The data ingestion module 310 operates to receive an ingest data for encryption and other information. The data is received in response to operations of a user of the client device 110, to be described in relation to Fig. 4. The data ingestion module can receive data in a variety of data types and formats, based on the system used by the user. The data types

received may include text, images (for example in JPEG, GIF or TIFF formats), audio data (for example in WAV or MP4 format), video data (for example in MPEG format) and other types and formats of data. The data ingestion module 310 typically undertakes three steps, being extraction of the relevant data, as received via operation of the API 130, loading or transfer of data to the cloud device 140 and transforming the data to a suitable format (for example JSON format or any format suitable for storage and encryption). The data ingestion module 310 can use standard ingestion techniques such as streaming, batch ingestion and hybrid techniques. In order to secure the data for loading to the cloud server, security methods such as encryption and support for protocols such as Secure Sockets Layer and HTTP over SSL may be implemented by the module 310.

[0057] The pre-encryption operation module 320 operates to implement a pipeline of operations to prepare the data received from the data ingestion module 310. The operations implemented by the module 320 can vary between implementations based on factors such as data application, expected security levels and the like. The pipeline of operations generally operates to assess if any operational security risk is associated with the ingested data and/or to prepare and organise data for database storage. The pre-encryption operation module typically implements operations of (i) virus scanning, (ii) malicious payload scanning, (iii) indexing, and (iv) tagging. The module 320 can also perform compression in some implementations.

[0058] Virus scanning is implemented to check the received ingested data is devoid of malicious content. The module 320 can implement virus scanning techniques such as heuristic-based detection, signature-based detection, dynamic malware analysis, static file analysis and the like. Malicious payload scanning typically involves the module 320 implementing techniques such as checking strings for common attacks such as SQL injection and other known vulnerabilities based on the Common Vulnerability and Exposures (CVE) repository. Additionally or alternatively, the malicious payload scanning can be based on Open Worldwide Application Security Project (OWASP) SQL injection testing guidelines. Virus scanning and malicious payload scanning each provide methods to assess if any operational security risk is associated with the ingested data.

[0059] The pre-encryption operation module 320 can also operate to perform indexing to index each received portion of data for efficient data retrieval and management. The module 320 typically implements existing indexing techniques and supports external index caching tools. For example, the Meilisearch library or other similar resources can be used for local indexing. Alternatively, the indexing may be performed on external third party tools.

[0060] In some implementations, the pre-encryption operation module 320 operates to tag the data received via the data ingestion module 310. Tagging operations typically involve appending metadata to the data in the form of a key-value pair. The module 320 can implement known tagging techniques. Tagging the ingested data can operate to simplify classification and organisation of data for more efficient data handling on the cloud server 140.

[0061] Tagging and indexing the ingested data each provide methods to prepare or organise the ingested data for database storage.

[0062] In existing arrangements, the pipeline of operations implemented by the module 320 are typically performed after data encryption is performed. The example arrangements described implement the pipeline of operations before encryption. While the pipeline of operations could be performed after encryption, performing before can provide advantages such as one or more of decreasing security vulnerabilities and improving computational and organisational efficiencies.

[0063] The polymorphic data restructuring module 330 operates to restructure the data output by the pre-encryption operation module 320 into a format suitable for integration with typical third party products using polymorphic data restructuring techniques. The module 330 operates to generate placeholders for encrypted strings of data (for example data to later be encrypted in the workflow). The placeholders can include a unique identifier for future lookup and decryption, if required. The format of the ingested data is dynamically determined using the appropriate HTTP header or MIME type of the data.

[0064] The hash encryption keys module 340 operates to generate cryptographic hashes of the encryption keys to be utilized for data encryption. Use of the hashed keys is integral for establishing access control policies and verifying the identities of authorized users and devices. For example, the module 340 may operate to generate encryption keys for later use in applying quantum encryption. The keys may be generated using existing a key generation protocol. Each of the public and private keys can be encrypted using techniques such as Secure Hash Standards (SHA), MD-5, RIPEMD-160 and the like. The resultant encrypted keys may be stored in a memory of the cloud server 140 such as the memory 206, or in a memory of an external device. The encrypted keys may be indexed or otherwise associated with a portion of data and/or an encryption instruction for ease of identification. Public and private keys of a pair are typically stored in separate databases to one another, even if encrypted.

[0065] The quantum resistant encryption module 350 operates to encrypt the data output by the polymorphic data restructuring module 330 using the encryption keys hashed by the module 340 and one or more quantum-resistant algorithms. Quantum-resistant algorithms are typically algorithms engineered to withstand attacks from both existing quantum computers and developing quantum computing technology. Examples of quantum-resistant encryption algorithms are Advanced Encryption Standard (AES) algorithm AES-256, Cryptographic Suite for Algebraic Lattices (CRYSTALS)-Kyber and the like. In some implementations, the arrangements described may allow a user of the client device to select one of a plurality of different encryption algorithms from different providers, for example OpenSSL algorithms or proprietary algorithms. The module 350 can be configured to implement several encryption algorithms, including classical as well as quantum-resistant algorithms, if such varying levels of encryption are to be selectable by the user of the client device. The quantum-resistant algorithms may be executed on the cloud server 140 or across several servers, depending on factors such as computational requirements, data for encryption, or the like. The modular design of the module 350 allows multiple encryption algorithms to be used simultaneously, depending on the requirements of the system(s) using the arrangements described, with a simple selection from a drop-down menu. Furthermore, some implementations can limit the use of only approved algorithms when running in a regulated area (i.e. FedRAMP High) whereby unapproved algorithms or cryptographic implementations cannot be used under any circumstances. For example, the module 350 can be assigned permissions associated with a user indicating permissions for particular algorithms. The permissions associated with the user may be governed based on parameters such as predetermined permissions, geographic location, a status or title associated with the user and the like.

[0066] In numerous regulated environments, data processing locations must adhere to strict geographic requirements. The module 350 is designed to be adaptable, capable of operating across geographically distributed Linux, BSD, and Windows-based operating systems. The module 350 supports the isolation of workloads to specific regions or data centers across multiple cloud or on-premises providers. By maintaining provider agnosticism, the module 350 enables users to configure and execute encryption processes globally, ensuring compliance with regional data sovereignty requirements while offering flexibility in deployment

Implementation Example

[0067] Fig. 4 shows an example method 400 of encrypting data using a platform according to the arrangements described. Some steps of the method 400 are implemented on the client device 110, by execution of software controlled by a processor of the client device 110 (similar

to the processor 205.). Other steps of the method 400 are executed on the cloud server 140 using software (such as the software 211) stored in the memory 206 under control of the processor 205.

[0068] Operation of the method 400 will be described in part with reference to examples in Figs. 5-9.

[0069] The method 400 starts a user access step 405. At the step 405, a user of the client device accesses data for which encryption may be required. For example, Fig. 5 shows a window 500 presenting a data in form of a document 505 (an online text document such as a document in MS Word format), as reproduced on a display (similar to the display 214) of the client device 110. In another example, Fig. 6A shows a window 600 presenting data in form of a document 605 (an online spreadsheet), as reproduced on the display of the client device 110. In another example, Fig. 7 shows a window 700 presenting data 705 (a Google™ form), as reproduced on the display of the client device 110. In yet another example, Fig. 8 shows content 800 presented as a document 805 (an online web form, for example presented in a web browser for a health prover's website), as reproduced on the display of the client device 110. The user can interact with the data in each case by manipulation of inputs of the client device 110, such as using a mouse or a touch-screen.

[0070] The method 400 continues from step 405 to a present API-enabled menus step 410. The step 410 executes present a menu enabled by the API 130 to allow the user of the client device to specify which portions of the document 500 are to be encrypted (or to request decryption). The content of the menu will depend on operations to be made selectable to the user of the client device. The minimum operation to be provided will be to allow the user to select a data, or a portion of data and indicate the action required, i.e., that the selected data is to be encrypted. Additionally, menus functions may include:

[0071] Selection of a particular level of encryption (e.g., high, medium, low).

[0072] Selection of a desired quantum-resistant algorithm from a listed plurality of quantum-resistant algorithms. Access to some algorithms may be limited using access controls as described below.

[0073] Selection of access controls associated with the encrypted data. Access controls can relate to traditional access control parameters, such as identifying a username allowed access, applying geo-fence constraint, a time constraint or other constraint associated with permission

to the encrypted data. Access controls can be policy-based, for example adding classification indicators (for example and extra classification indicating an information classification level for military applications or identification of a time window during which information is restricted). Other access controls may relate to managed storage, for example indicating that the encrypted data is to be stored in a particular database or on a particular server, for example a military database or server.

[0074] Indication or selection of policies associated with encryption, as discussed below.

[0075] For example, the window 500 includes a menu area 510 presented based on information received via the API 130. The menu area includes an "Encrypt Content" button 515, a "View Protected Content" button 520 and an "Encryption Scheme" drop-down menu 525. The button 515 is selectable by the user to request encryption of a selected portion of the data of the document, for example text at a highlighted area 550 for encryption. The user can use the menu 525 when dropped down (not shown) to select a particular quantum-resistant encryption algorithm from a list of available algorithms. Other example menus are shown in Fig. 6A and 7.

[0076] Different menus may have different access or policy controls presented, depending on user requirements or subscription to the encryption service provided by the cloud server 140. For example, Fig. 5 includes a drop-down menu 560 relating to managed storage options and a drop-down menu 565 for indicating or selecting AI scanning policies, whereas the menu 610 includes an additional menu 670 for classification and a menu 630 for geofencing.

[0077] As the menu functionality is provided by the API, flexibility is available in terms of the end application being used, the parameters such as access controls made available to the user, and the like.

[0078] The step 410 operates to present, at the application executing on the client device, a menu enabled by the API 130. The menu can in some embodiments provide an interface by which one of a plurality of quantum-resistant algorithms may be selected by the user. The menu in some arrangements additionally or alternatively provides an interface by which one or more access controls may be selected by the user. The menu in some arrangements provides an interface by which the user can actively or directly request encryption or enter an encryption request.

[0079] Once the API-enabled menu is presented, the method 400 continues from step 410 to an identify encryption required step 415. At step 415, the API 130 receives information that an

encryption instruction has been identified, in response to an operation of the user of the client device 110. The minimum information required is identification of the portion of data and the action (encryption) required. For example, the user may select the highlighted portion of text 550, in which case the information received may be the textual content. In some arrangements, a default encryption algorithm may be used if an algorithm is not specified. In other arrangements, the user may use the drop-down menu 525 to select a desired encryption algorithm and the information received at the API will include identification of the algorithm. In the example of Fig. 6, the user has selected a portion 650. In the example of Fig. 7, as the user builds a form, encryption requirements for different form fields (such as Option 1, field 750) may be specified.

[0080] In yet other arrangements, the information identifying that encryption is required can include access constraints set by the user. The access constraints can include information identifying who may view the portion of data after encryption and any associated constraints. In some arrangements, the user may be able to enter an identification and/or a password (not shown). In Fig. 6A, similarly to Fig. 5, the window 600 includes a menu area 610. The menu area 610 includes an "Encrypt Content" button 615, a "View Protected Content" button 620 and an "Encryption Scheme" drop-down menu 625.

[0081] Additionally, the menu area 610 includes the "Geofencing" drop down menu 630 and a map area 635 (for example provided by a Google Maps™ API). The user can expand the menu 630 to specify access controls based on a geofence. For example, the user may enter a country name (such as "Australia") or city name ("such as Sydney") to indicate that the information is only allowed to be viewed in that country or city. Additionally or alternatively, the user may use the map area 635 to select an area or geofence. The geofence information is included in the information that an encryption instruction has been identified.

[0082] Fig. 6B shows an example menu 610b, corresponding to a portion of the menu 610 where the geofencing drop-down menu has been expanded. An extended menu area 630b includes a number of selectable zones 631 (Zone 1, Zone 2 and Zone 3) in which Zone 2 is selected. As Zone 2 is selected, updated map plugin area 635b (corresponding to 635) shows a geographic area corresponding to Zone 2. Other menu options may be shown or available for selection, for example based on country, state, postcode, latitude and longitude boundaries or any other indication that can be used as a geofence.

[0083] In the examples of Figs. 5 and 6 above, an encryption instruction is detected by the user actively selecting an encryption instruction via one of the buttons 515 and 615. In other

arrangements, the encryption instruction may be detected without an active encryption request by the user but rather based on a predetermined characteristic of an interactive document, as set by a developer for example. For example, Fig. 8 shows the content 800, which is an online form to be completed by the user of the device 110 to provide personal information to a health provider, "SimpleHealth". The form includes fields 810, 815, 820 and 825 for a user to provide information (full name, email address, password and social security number respectively). The window 800 also includes a "Securely submit button" 830. When the user enters their details and to the eilds 810-825 and presses the submit button 830, the API 130 will receive information identifying that an encryption operation is required based on parameters or characteristics predetermined by a developer with respect to the buttons 810-820. For example, based on predetermined settings in the html description of the page showing the form, the content entered to 815, 820 and 825 may be marked for encryption but the content of 810, the full name, may not. Accordingly, selection of one or more portions of data for encryption may be set by an author of the document and in some cases may be predetermined. The one or more portions of data for encryption may be actively selected by the user (as in Figs. 5 and 6) or embedded in a structure of the document (for example a html description of a web form or web page). In some instances, the predetermined information may include access controls and/or a selected encryption algorithm, as identified by information embedded in the document description.

[0084] In addition to being directly or actively requested by the user (e.g., using buttons 515 or 625), an instruction or request for encryption may also be received at the cloud server 140 based on a data structure associated with data presented to the user, as described in relation to the form 800. A policy or logic can be implemented flexibly by a software developer to include conditions specific to an implementation. Further, the request for encryption of a portion of data may be received at the cloud server 140 based on a policy associated with presentation of data to the user policy, or logic associated with a data file structure or a process associated with the user subscription. In the example of the web form 800, the request is identified via the structure of the associated data, being embedded in the document description such as by html. In other arrangements, the encryption instruction may be due to a policy set by the user or by the system. For example, the menu 510 includes the drop-down menu 565 entitled AI Scanning. The menu 565 can provide an interface to an AI-based function that can scan large quantities of documents as they are being ingested/proxied and determine sensitive data segments/lines/fields. In one implementation, if the AI scanning function (for example, RapidMiner or DataRobot) is enabled, all data is transmitted via the API 130 for ingestion by the module 310 and input to AI scanning module (not shown). The AI scanning module may form part of the architecture 301 or stored in a separate architecture or device to the cloud server

140. AI scanning will result in identification of relevant data for sending to the pre-processing module 320 generating keys and performing the encryption in association with any determined policies. Other policy-based operations may be available.

[0085] The user action of selecting any of the buttons 515, 615 and 830 results in a message being generated for communication through the API 130 to the cloud server 140 (in particular to the module architecture 301). The message is in a format such as JSON and include fields or parameters for each item of information associated with the user action. For example, the message may contain objects for (i) each portion of data selected or, in the case of a form, entered, (including content, location within the overall document) and (ii) the action required (encryption or decryption). The information in the message also includes JSON objects for other optional parameters that are in use, such as

[0086] Selection of a particular level of encryption (e.g., high, medium, low).

[0087] Selection of a desired quantum-resistant algorithm from a listed plurality of quantum-resistant algorithms.

[0088] Selection of access controls associated with the encrypted data such as identifying a username allowed access, applying geo-fence constraint, a time constraint or other constraint associated with permission to the encrypted data.

[0089] Selection or identification of policy controls, such as an AI scanning policy.

[0090] Some simplistic examples of messages for the API 130 in JSON format are included in Table 1 below.

<pre>{ "body": "this is a string to be encrypted" }</pre>	An object "body" indicates the body of the text. "This is a string to be encrypted" is the text string that will be encrypted by the module 350. As no action or algorithm are presented, an encryption request using a default algorithm can be assumed.
<pre>{ "body": "this is a string to be encrypted" "algorithm": "AES-256" }</pre>	An object "body" indicates the body of the text and an object "algorithm" indicates a quantum-resistant algorithm to be used. "This is a string to be encrypted" is the text string that will be encrypted by the module 350 using an algorithm indicated by the "algorithm" object, in this case using AES-256.

Table 1 – API message examples

[0091] The effect of the step 415 is that the cloud server 140 receives, via an application executing on a client device 110, such as the application on which the data and the API menu are presented) a user selection of a portion of data and a request to encrypt the data using a quantum-resistant algorithm. A message containing information for the selected portion of data (for example as shown in Table 1) and the request are transmitted to a cloud server via the API 130.

[0092] The method 400 continues from step 415 to an ingest data step 420. At step 420, the information associated with the user request is transmitted, via the network 120 to the cloud server 140. The data ingestion module 310 operates to ingest the information received via the JSON message, including the data to be encrypted and any associated information, such as access constraints, selected encryption algorithm and the like.

[0093] The method 400 continues under control of the processor 405 from the step 420 to a pre-processing step 425. At step 425, the pre-encryption operation module 320 operates to implement a pipeline of operations such as (i) virus scanning, (ii) malicious payload scanning, (iii) indexing, and (iv) tagging as described in relation to Fig. 3. For example, any associated access controls may be tagged as associated with the data. If any step of the pipeline indicates

that the data presents a security risk, the module 320 may operate to transmit an error message via the API 130 to the client device 110 and terminate the method 400, deleting the risk-associated data.

[0094] The method 400 continues under execution of the processor 205 from step 425 to a restructure data step 430. At the step 430, the polymorphic data restructuring module 330 operates to restructure the data output by the pre-encryption operation module 320 into a format suitable for integration with typical third-party products using polymorphic data restructuring techniques.

[0095] The method 400 continues under execution of the processor 205 from step 430 to a hash cryptographic key step 435. At step 435, the hash encryption keys module 340 operates to generate cryptographic hashes of the encryption keys to be utilized for data encryption, for example a pair of asymmetrical keys.

[0096] The method 400 continues under execution of the processor 205 from step 435 to an encrypt data step 440. At step 440, the software 211 operates to encrypt the portion of data for the detected encryption instruction using a quantum-resistant encryption algorithm, for example using the module 350. The quantum-resistant encryption algorithm may be a default algorithm or an only algorithm stored in a memory of the cloud server 140. Alternatively, the algorithm may be one of a plurality of stored algorithms identified in the information in the JSON message generated by operation of step 415. For example, the user may have selected AES from a menu for a first portion of data and CRYSTALS-Kyber for a second selected portion of data.

[0097] The method 400 continues operation under control of the processor 205 from step 440 to a store encrypted data step 445. The step 445 operates to store the encrypted data, for example in the memory 206 of the cloud server 140 or in a memory of a separate device and to store the associated keys and metadata.

[0098] The step 435 operates to firstly encrypt a pair of keys associated with the request at the cloud server 140 and store the keys, for example a public/private key pair stored in 206. The step 440 operates to encrypt the portion of data using a quantum-resistant algorithm and the encrypted keys at the cloud server 140, providing a full encryption solution.

[0099] The method 400 continues under control of the processor 205 from step 445 to an update document or associated information step 450. The step 450 operates to transmit, via the API, updated data to the document. For example, data in the areas 550 and 555 and 650 are updated to be highlighted and to indicate that the data is restricted in the examples of Figs. 5

and 6. Restricted means that a user cannot view the data without a decryption request being allowed. In some implementations, the replaced data can be in form of selectable text or other selectable data. In other implementations, the replaced data can be a selectable URL that will lead to a browser page for entering a decryption request. In other words, the step 445 updates, via the API 130, display of the selected portion of data on the client device 110 to replace the portion of data with an indication the portion has been encrypted.

[0100] In implementations where the encrypted data relates to a predetermined setting, for example for the web form 800, associated information is updated at the step 450. For example, an account for the user who filled in the form may be updated to include the encrypted data and, in some instances, an associated key.

[0101] Through operation of the API, data associated with objects in the document can be updated. For example, Fig. 9A shows a set 900 of data objects in JSON format associated with two accounts, one for John Doe and one for Jane Smith. As shown in Fig. 9A, personal information such as date of birth, driver license number, phone number and address are stored for each account. Fig. 9B shows a set 950 of data objects corresponding to the set 900 after the method 400 has been implemented for each account. As shown in Fig. 9B, the personal information of date of birth, driver license number, phone number and address in each instance has been replaced with an encrypted value.

[0102] The arrangements described provide a platform which, though use of the API 130 to allow one or more portions of data to be encrypted using one or more types of quantum-resistant algorithms based on a single user instruction and effectively one line of code. As described above, the user instruction can relate to an active request to encrypt, such as using the buttons 515 and 615 to expressly request encryption. Alternatively, the user instruction may be indirect and relating to data structure, for example via submission of an online form and pressing the button 830.

[0103] The portion of data can relate to a whole set of data, for example a whole document or full account details or a subset of data, for example one or more areas of a document (such as 550 and 555) or a subset of account details (for example relating to 815, 820 and 825 only). The arrangements described can be used for multiple data format types and multiple types of algorithms. The platform allows dynamic and adaptable encryption whereby different subsets of data (for example first portion 550 and second portion 555) can be assigned different algorithms or different levels of security and/access constraints.

[0104] The order of some steps in the method 400 may be changed, or some steps may run in parallel or different order (for example 435 may run earlier or at the same time as one of the 425 and 430). Depending on the implementations, some steps may be removed. In particular, the step 450 may not be used in cases such as the web form 800.

[0105] The arrangements described also allow decryption based on user permissions. Fig. 10 shows a method 1000 of decryption using the arrangements described. Some steps of the method 1000 are implemented on the client device 110, by execution of software controlled by a processor of the client device 110 (similar to the processor 205). Other steps of the method 1000 are executed on the cloud server 140 using software (such as the software 211) stored in the memory 206 under control of the processor 205.

[0106] The method 1000 starts at a step 1005. The step 1005 operates in a similar manner to step 405 whereby the user accesses data, for example by presenting the document 505 in the window 500. The method 1000 continues to a present API-enabled menu step 1010. The step 1010 presents the menu 510, in the same manner as the step 410.

[0107] The method 1000 continues to a decryption request received step 1015. For example, the user selects the portion 550 of the document 505 and presses the button 520. In other cases, the user may select a URL in the area 550. The API 130 receives the request and any data associated with the requestor, for example a use name, geographic information (if available) or the like. The information is transmitted in a message (for example using JSON format) via the API 130 to the cloud server 140.

[0108] An example JSON message may be:

```
{  
  "body": "xxx111yyy222"  
  "Action": "access"  
  "username": johndoe123  
}
```

[0109] In the example above, user "johndoe123" want access to (that is to see the decrypted version of) an encrypted string "xxx111yyy222".

[0110] The method 1000 continues to a check step 1020. At step 1020 the application 133 executes to check if the user is allowed access to a decrypted version of the information in the

request. The access controls restrict access to the encrypted keys for decrypting the data. At step 1020, the application 211 firstly operates to identify any access constraints associated with information, for example metadata stored in association with the tagged and indexed data at the pre-processing step 425. Once the access information has been access, the software 211 operates to obtain user data associated with the request (for example username, geographic area obtained from an IP address or a GPS chip of the user's device such as 110, a key associated with or provided by the user's device or the like). The module 211 determines if the user making the request is allowed access by comparison to metadata stored in association with the selected piece of data. For example, if a geofence is Australia and the user is identified as being in Sydney or Melbourne, access may be granted. In order to be allowed access, all access controls usually need to be met. The metadata reflects any access controls added by the user when requesting encryption. If the user is not allowed permission ('N' at 1020) an error message is presented to the user and the method 1000 ends.

[0111] If the user is allowed permission ('Y' at 1020), the method 1000 continues to an obtain key step 1030. In being allowed access the user requesting decryption is effectively allowed access to the encryption keys for the data. At step 1030 the application 211 retrieves the associated private key. The method 1000 continues from step 1030 to a decrypt data step. The software 211 operated to decrypt the data using the retrieved key and the algorithm used at the encryption stage. The arrangement described therefore affords the owner of data (the user who requests encryption) an ability to dynamically manage third party access to the encryption keys for said data, agnostic of the application or service.

[0112] The method 1000 continues to step 1040 and provides the data to the user. For example, the data is transmitted via the API 130 to update the content of the document 505 displayed in the window 500.

[0113] In some implementations, for example web document applications, a "Smart Proxy" may be used to act as an intermediary in providing the functionality of the methods of Figs. 4 and 10 seamlessly through use of HTTP headers. Fig. 11 shows an example method 1100 of implementing a HTTP POST command using the arrangements described. Some steps of the method 1100 are implemented on the client device 110, by execution of software controlled by a processor of the client device 110 (similar to the processor 205). Other steps of the method 1100 are executed on the cloud server 140 using software (such as the software 211) stored in the memory 206 under control of the processor 205.

[0114] The method 1100 starts at 1101 when the client device 110 sends a HTTP POST request with a JSON body. For example, this may correspond to user access at 405 where the user of the client device wants to send an update to content in a web page.

[0115] The method 1100 continues to step 1105. At step 1105 the API 130 receives the HTTP request with JSON payload. A Smart Proxy application included in the API 130 intercepts the POST request and reads the HTTP headers. The method 1100 continues from step 1105 to a step 1110. At step 1110 the API extracts the JSON payload received with the HTTP POST request at step 1150.

[0116] In one example, the POST Request Body received at step 1105 includes three example fields as below:

```
{  
  "field1": "value1",  
  "field2": "value2",  
  "field3": "value3"  
}
```

[0117] The example above includes three JSON keys (field1, field2 and field3) each having an associated payload or data (value1, value2 and value3 respectively).

[0118] The method 1100 continues from step 1110 to a step 1115. Step 1115 select one of the JSON keys present in the HTTP POST request as a current JSON key. The step 1115 is implemented using the Smart Proxy of the API 130.

[0119] The method 1100 continues from step 1115 to a check step 1120. The step 1120 executes to check a value type of a current selected JSON key extracted in operation of steps 1105 to 1115. The keys may have different value types, based on the POST request and the format of the data being prepared by the user. In the example described, the value types are (i) String/Number or (ii) Object/Array. However, different value types may be used or used for different functions.

[0120] In the example of Fig. 11, use of a String/Number as the value type indicates that the quantum resistant methods described herein are to be used. An instruction for quantum resistant encryption is effectively received, as at step 415. If step 1120 returns a string or a number as the value type, the method 1100 continues from step 1120 to an encryption step

1125. The step 1125 operates to provide the data to the cloud server 140, which implements the steps required for quantum resistant encryption. For example, the step 1125 implements the steps 420 to 445 of the method 400, generating an encrypted version of the data in the selected JSON data.

[0121] The method 1100 continues from step 1125 to a step 1130. At step 1130 the value of the JSON data before encryption is replaced with encrypted data. The method 1100 continues from step 1130 to a step 1135.

[0122] In the example of Fig. 11, use of an object or an array as the value type indicates that encryption is not required and that the quantum resistant methods described herein are not to be used. If step 1120 returns "Object/Array" the method 1100 continues to a step 1145. The step 1145 indicates that no encryption is required and the value of the JSON data associated with the selected JSON key is maintained. The method 1100 continues from step 1145 to the step 1135.

[0123] The step 1135 indicates that the method 1100 is ready to progress to a next JSON key. The method 1100 proceeds to a step 1140. At step 1140 a check is made if all JSON keys have been checked at step 1120 and corresponding following steps made. If all keys have not been processed ("No") at step 1140, the method 1100 returns to step 1115 to select a different JSON key as the current JSON key.

[0124] If all JSON keys and associated data have been processed ("Yes" at step 1140) the method 1100 continues from step 1140 to a reconstruction step 1150. The step 1150 reconstructs the JSON payload received at step 1105. The reconstructed JSON payload may include (i) data encrypted at step 1125 and replacing original data at step 1130, (ii) original data as maintained at step 1145, or (iii) a mixture of encrypted and original data if some value types indicated encryption and other value types did not. The step 1150 can be implemented at the cloud server 140 or at the API 130.

[0125] Using the example provided above, after implementation of the step 1150, field1 and field2 returned a value type of a string or a number, whereas field3 had a type of an object or array. The reconstructed JSON payload at step 1150 therefore included encrypted values for field1 and field2, as shown below.

```
{  
  "field1": "encrypted_value1",  
  "field2": "encrypted_value2",  
  "field3": "value3"  
}
```

[0126] The method 1100 continues from step 1150 to a step 1155. At step 1155 a HTTP response to the request sent at 1101 is transmitted to the client device 110. The response includes the reconstructed JSON payload generated at step 1150. The step 1155 can be considered equivalent to the step 450 of updating a document or information. The method 1100 ends after implementation of step 1155.

[0127] Encryption often increases the size (in bytes) of the plaintext payload. Increasing size of payload can be problematic in existing implementations when dealing with APIs that limit the length of a given string. The arrangements described can allow automatic and seamless compression before the payload is encrypted using compression tools (such as gzip or the like), for example at preprocessing at step 425. For example, the selected portion of data is compressed at step 425 (for example as a last step in the pipeline of operations of the module 320) or step 430 and the compressed data is encrypted using the quantum-resistant algorithm and the encrypted keys at the cloud server at step 440. The data can be uncompressed upon decryption, for example after decryption in implementation of step 1035. Ability to use compression tools maintains a smaller encrypted payload (in bytes) to support third party APIs and data storage systems.

[0128] Encrypting a payload is typically destructive in terms of preserving the format of the original payload data. For example, encrypting an entire JSON or XML payload in a HTTP POST request body would require the receiving endpoint (typically a server) to support the encrypted payload. In using JSON format, in some implementations the arrangements described can operate to encrypt the "value" in a key:value or object-based system and preserves the structure of the payload. Additionally, the encrypted value can encoded using base64 encryption to make the value universally readable, even if the payload value contains non-standard characters. Ability to use base64 encryption increases the compatibility with third party services, even legacy systems, services, datastores, databases and many more, that would not otherwise support storing or processing encrypted data.

Industrial Applicability

[0129] The arrangements described are applicable to the computer and data processing industries and particularly for the data security industries.

[0130] The present invention provides a platform by which quantum-resistant encryption is readily available to users in a dynamic and, on the client device, resource-efficient manner. In particular, use of the API 130 allows the user access to different types of quantum-resistant encryption algorithms without requiring plugins specific to the algorithm and/or a user application for viewing, manipulating or generating data content. The encryption can be implemented without requiring significant computation on the client side, rather only generation and transmission of a message with the required information via the API 130. The solution provided is accordingly application-agnostic and encryption-agnostic as the API 130 can be configured for a wide variety of applications. The platform provides flexibility to dynamically encrypt different portions of a data set with different levels of encryption. Through use of an API, the platform can utilise encryption algorithms resistant to both classical and quantum computers.

[0131] Additionally, the platform provides flexibility to be integrated with existing system including access control systems. Ability for integration with existing access control systems can support custom data classifications, suitable for use in any industry - including government, military and healthcare.

[0132] By conducting operations such as virus scanning, indexing, and tagging prior to data encryption, some embodiments of the system enhance processing efficiency while safeguarding data.

[0133] The foregoing describes only some embodiments of the present invention, and modifications and/or changes can be made thereto without departing from the scope and spirit of the invention, the embodiments being illustrative and not restrictive.

[0134] **(Australia Only)** In the context of this specification, the word “comprising” means “including principally but not necessarily solely” or “having” or “including”, and not “consisting only of”. Variations of the word "comprising", such as “comprise” and “comprises” have correspondingly varied meanings.

CLAIMS:

1. A method for encrypting data, the method comprising:
 - receiving, via an application executing on a client device, a request to encrypt a user selection of a portion of data using a quantum-resistant encryption algorithm;
 - transmitting a message containing information for the selected portion of data of the request to a cloud server via an application program interface (API);
 - encrypting a pair of keys associated with the request at the cloud server;
 - encrypting the portion of data using the quantum-resistant algorithm and the encrypted keys at the cloud server; and
 - updating, via the API display of the portion of data on the client device to replace the portion of data with an indication the portion has been encrypted.
2. The method according to claim 1, further comprising:
 - assessing, at the cloud server, if any operational security risk is associated with the selected portion of data before encrypting the portion of data.
3. The method according to claim 2, further comprising:
 - indexing and tagging at the cloud server the selected portion of data before encrypting the portion of data.
4. The method according to claim 1, further comprising:
 - presenting, at the application executing on the client device, a menu enabled by the API, the menu providing an interface by which one of a plurality of quantum-resistant encryption algorithms may be selected by the user.
5. The method according to claim 1, further comprising:
 - presenting, at the application executing on the client device, a menu enabled by the API, the menu providing an interface by which one or more access controls may be selected by the user, the access controls restricting access to the encrypted keys for decrypting the data and being included in the message information.
6. The method according to claim 5, wherein the access controls include one or more of a geofence, a user identification and a classification.
7. The method according to claim 1, wherein the request is received by the user directly entering an encryption request.

8. The method according to claim 1, wherein the request is received based on a structure associated with data presented to the user.

9. The method according to claim 1, wherein the request is received based on a policy associated with presentation of data to the user.

10. The method according to claim 1, further comprising compressing the selected portion of data, and wherein the compressed portion of data is encrypted using the quantum-resistant algorithm and the encrypted keys at the cloud server.

11. A computer-readable storage medium which stores a program for executing a method for encrypting data, the method comprising:

- receiving, via an application executing on a client device, a request to encrypt a user selection of a portion of data using a quantum-resistant encryption algorithm;

- transmitting a message containing information for the selected portion of data of the request to a cloud server via an application program interface (API);

- encrypting a pair of keys associated with the request at the cloud server;

- encrypting the portion of data using the quantum-resistant algorithm and the encrypted keys at the cloud server; and

- updating, via the API display of the portion of data on the client device to replace the portion of data with an indication the portion has been encrypted.

12. The computer-readable storage medium according to claim 11, wherein the method further comprises:

- assessing, at the cloud server, if any operational security risk is associated with the selected portion of data before encrypting the portion of data.

13. The computer-readable storage medium according to claim 12, wherein the method further comprises:

- indexing and tagging at the cloud server the selected portion of data before encrypting the portion of data.

14. The computer-readable storage medium according to claim 11, wherein the method further comprises:

presenting, at the application executing on the client device, a menu enabled by the API, the menu providing an interface by which one of a plurality of quantum-resistant encryption algorithms may be selected by the user.

15. The computer-readable storage medium according to claim 11, wherein the method further comprises:

presenting, at the application executing on the client device, a menu enabled by the API, the menu providing an interface by which one or more access controls may be selected by the user, the access controls restricting access to the encrypted keys for decrypting the data and being included in the message information.