

SYSTEMS FOR QUANTUM CYBER RESILIENCE OF DIGITAL ASSETS

TECHNICAL FIELD

The present disclosure relates generally to protecting data at rest and/or in transit to assure data integrity. More particularly, the present disclosure relates to providing quantum cyber resilience protection of digital assets to be stored in non-volatile memory of a computer system and/or communicated over a secure or unsecure network including from quantum computer cryptographic attacks.

BACKGROUND

Cryptography is used to protect the secrecy, integrity and authenticity of messages and information. In modern computer and information technology (IT) systems, numerous cryptographic standards and techniques have been developed for digital information (data and/or code) that employ digital encryption, signatures, and authentication to secure and protect this information. See, Schneier, B., *Applied Cryptography* (1993); Wong, D., *Real World Cryptography*, (2021).

Most cryptography techniques are used to protect digital messages and information in transit, typically by using an encryption key protocol. One of the critical challenges with use of encryption key protocols is that the level of security for such protocols is directly related to both the length of the message and the length of the encryption key. As computing power has increased, brute force attacks to crack encryption keys have resulted in the need to increase the length of encryption keys to increase the number of possible key combinations. Current encryption protocols are referred to as standard ciphers because they are vetted by organizations or agencies such as the National Institute of Standards and Technology (NIST). Examples of current standard encryption algorithms are the Advanced Encryption Standard (AES) protocols which typically use keys with either 128 bits ($2^{128} = 3.4e^{+38}$ possibilities) or 256 bits ($2^{256} = 1.2e^{+77}$ possibilities). The total time needed to try all such possibilities for AES 128-bit keys would require at least tens to thousands of years using conventional computer systems, thereby making such protocols well-suited for protecting relatively shorter messages and data in transit from attacks by classical computers.

Various cryptographic solutions have been developed to protect data at rest from tampering including data in non-volatile disk storage systems for a computer system. Hardware approaches for encrypting the entire storage system are described, for example, in U.S. Patent Nos. 8,473,754, 9,575,903, 9,785,801, 9,996,479, 10,691,837, 10,992,453 and 11,243,893, as well as U.S. Publ. Nos. 2020/0159888 and 2020/0117810. Other approaches for protecting data at rest are described, for example, in U.S. Patent Nos. 8,782,436, 9,419,796, 9,767,306, 10,360,395, 10,476,664, 10,860,744, 10,860,745, 11,297,166, 11,641,347, 11,777,724, and 11,550,883, as well as U.S. Publ. No. 2020/0195446, 2021/0306145, and 2023/0291545.

Non-volatile data storage systems have evolved from single disk drives to various configurations of array storage including redundant arrays of independent drives (RAID) systems and cloud storage servers. Examples of efforts to improve security for these array storage systems are described, for example, in U.S. Patent Nos. 8,468,244, 10,735,137, 11,281,790, and 11,516,201, as well as U.S. Publ. Nos. 2021/0234682 and 2023/0267054, and disk encryption software such as Azure and Truecrypt, and cloud storage system improvements as described in Celesti, A., "Adding long-term availability, obfuscation, and encryption to multi-cloud storage systems" Science Direct, Vol. 59, pg. 208-218 (Jan. 2016).

Over the last fifty years as the performance and power of computer systems has continued to increase, the sizes of digital data assets and of non-volatile storage systems have increased from kilobytes to gigabytes or even terabytes and the speed and bandwidth of digital communications have increased from 56-bit baud rates dial-up phone connections to the capacity of the modern internet and cellular networks to operate at gigahertz speeds and capable of transmitting gigabytes of data. This increase in speed and size can present security challenges when using conventional cryptographic techniques as the larger the size of an encrypted file, the easier it can be to attack or hack such a file. Such attacks are often based on finding patterns and relationships in such large files when an encryption key is not truly random or when arrangements of data can be inferred.

In recent years, a different kind of challenge to conventional encryption techniques has arisen based on the potential use of quantum computing and artificial intelligence powered attacks. The potential power of quantum decryption is encouraging hackers to employ a harvest-now-decrypt-later tactic to steal sensitive conventionally encrypted data today using classical computers and wait until Cryptanalytically Relevant Quantum Computers (CRQCs) coupled with Artificial Intelligence (AI) soon will be able to successfully hack encrypted data stolen today. Both Congress

and the White House have recognized the significant cyber threats posed by the rapid advance of CRQCs and AI. *See*, “Quantum Computing Cybersecurity Preparedness Act,” Publ. Law 117-220 (Dec. 21, 2022); “National Cybersecurity Strategy,” White House (Mar. 1, 2023); and “National Cybersecurity Strategy Implementation Plan” (Jul. 13, 2023). Quantum-based attacks could dramatically reduce the time and resources needed for brute force attacks to succeed in hacking data at rest and data in transit. This potential vulnerability is especially problematic in situations where the data/code at cold rest is used in computer systems that could be vulnerable to capture and physical breach by an adversary. *See*, “Understanding Quantum Computing White-Paper,” ID Quantique SA (May 2020); and “Post-Quantum Cryptography for Engineers,” IETF (Aug. 30, 2023).

Traditional cybersecurity approaches have focused on preventing and deterring unauthorized access to digital systems by using standard cryptography encryption algorithms to encrypt data. New encryption standards for the algorithms that may be used in post-quantum or quantum-resistant encryption are being evaluated by NIST. *See*, “Post-Quantum Cryptography,” NIST (Aug. 24, 2023). However, these standards focus on encryption for data in transit and will require years to be fully tested and evaluated for potential weaknesses and exploits. Even then, the challenges of protecting data/code at rest or in transit in the face of hacking efforts using CRQC and AI will remain. There is a recognized need to develop complementary mitigation strategies to provide cryptographic agility in the face of unknown future risks with new systems that can provide quantum cyber resilience for protection of digital assets that are more efficient and effective in protecting the integrity of such digital assets from prolonged and well-resourced brute force attacks, including quantum computing attacks.

SUMMARY OF THE DISCLOSURE

Embodiments of the present disclosure provide computer-implemented digital asset protection systems configured to generate packages of digital assets that are quantum cyber resilient at rest or in transit. Instead of using just improved quantum encryption algorithms to protect digital assets, disclosed embodiments augment the effectiveness of such enhanced quantum encryption algorithms by utilizing hybrid cryptographic techniques for Packers/unPackagers in a process referred to as “incryption/dicryption.”

In various embodiments of incryption, intrinsic information derivable from the protected data that may have discernable patterns or properties is rendered effectively invisible by obfuscating any patterns or organizations of potentially meaningful information content which could be used to facilitate brute force attacks against the quantum cyber resilient packages of digital assets. The results of such quantum cyber resilient protection systems that employ incryption are packages of digital assets that appear as a sea of nothingness produced by enhancing the noise and entropy of such quantum cyber resilient packages to make prolonged and well-resourced brute force hacking, reverse engineering, and even quantum computing attacks effectively unfeasible.

Unlike the current NIST initiatives to improve quantum-resistant encryption algorithms, the present disclosure assumes that encryption alone, even using post quantum keys (PQK), will not result in truly quantum resilient protection of digital information, especially larger amounts of digital information at rest or in transit. Instead, embodiments of the present disclosure effectively increase security strength by using combinations of hybrid cryptographic techniques and randomization to break the digital information into protected puzzle pieces where none of the puzzle pieces make sense and where there are no boxes or locations that differentiate different puzzle pieces so that it is effectively impossible to reassemble the puzzle or even know which puzzle pieces are valid pieces.

For purposes of the present disclosure, “incryption” is defined as a set of computer-implemented processes, protocols and/or techniques used to construct and/or deconstruct a representation of digital assets by encrypting/decrypting the digital assets as encrypted data using at least one protection/encryption algorithm and by using multiple obfuscation/invisibility processes, protocols and techniques to make any intrinsic information derivable from the encrypted data that may have discernable patterns or properties effectively invisible.

In various embodiments, different protection/encryption algorithms and obfuscation/invisibility protocols and techniques may be utilized depending upon different desired levels of resilience and threat protection. In some embodiments, quantum resilience is provided for digital assets by utilizing a quantum entropy random source to generate a quantum seed or key used with a quantum-resistant encryption algorithm. In other embodiments, brute force hacking protection is provided for digital assets by utilizing either a conventional or quantum random seed or key and either a set of conventional and/or quantum-resistant encryption algorithms together

with obfuscation/invisibility protocols and techniques that will generate an encrypted representation of the digital assets that effectively ensures that a total number of possible decryptions and reconstructions is significantly larger than a given amount of computing resources projected to be used by a particular brute force attack. In various embodiments, a level of cyber protection of a given size of digital assets can be achieved with differing levels of incryption such that the digital assets are sufficiently protected even if the media on which such assets are stored is hacked, stolen, or compromised or the communication channel over which such assets are transmitted is intercepted or hacked because there is no value in such stolen digital assets as the incrypted digital assets cannot be recovered.

In various embodiments of the present disclosure, a computer-implemented digital asset protection system is configured to protect a set of original files/data of the digital assets as quantum incrypted data that is quantum cyber resilient at rest or in transit. The system comprises a Packager system operably connected to a randomization entropy source and configured to protect the set of original files as quantum incrypted data by a set of protocols. In some embodiments, the Packager system is configured to generate a protected package to be securely delivered to and stored by a storage system of a target computing system, and the digital asset protection system further comprises an unPackager system configured to be installed in a secure computer node of the target computing environment to unpackage the protected package. In embodiments, the protected package includes a set of quantum incrypted data and a corresponding set of quantum containment units uniquely generated for the original digital assets.

In embodiments, the set of protocols and techniques used by the Packager system to protect the set of original files as quantum incrypted data uses the randomization entropy source to seed a set of Quantum Hybrid Ciphers (QHCs) to generate a quantum incrypted data (QUID) representation for the set of original digital assets. In some embodiments, the QHCs generate a set of quantum elements (QEs) that may be packaged or organized into a QUID. In other embodiments. Unlike conventional encryption techniques that include only a single standard encryption algorithm or a limited and fixed number and order of non-standard cipher techniques, the QHCs in various embodiments are a stack of multiple different ciphers that used in an order and number which is randomized to at least some degree to further enhance the strength and cryptographic agility of the quantum cyber resilient security. In various embodiments, the QHC stack can include multiple non-standard cryptographic techniques. In some embodiments, the non-

standard cryptographic techniques can include obfuscation ciphers for fragmenting each digital asset into fragments each being a quantum element, injecting each quantum element with a set of random noise, manipulating the bits in each quantum element using masking, substitution, shifting, permutation, and/or positional translation, shuffling the set of quantum elements into a random order, and/or introducing a set of decoy digital assets. In some embodiments, the non-standard cryptographic techniques can include protection ciphers for protecting portions of the digital assets using various publicly known or unknown non-standard cryptographic protection ciphers. In embodiments, the QHC stack can include one or more optional standard ciphers based on standard algorithms that have been publicly vetted and approved as current NIST encryption standards to meet applicable cryptographic compliance requirements.

In various embodiments, the output of the QHC stack is a paired Quantum Containment Unit (QCU) and QuID package. The term “incryption” is somewhat analogous to “encryption,” but encompasses an expanded cryptographic process that can include one or more standard encryption algorithms plus non-standard QHC processes. A QCU is an amalgamated codex of various incryption-related information, data, and/or metadata to be used by the unPackager for the “decryption” process that reverses the methodology and sequencing of the paired QCU/QuID package. The non-standard QHC processes are used to protect and/or obfuscate the data such that any intrinsic information, patterns, or properties derivable from the encrypted data are effectively rendered useless to adversaries. The QuID is the collection of digital assets (e.g., data, message, and/or code) that has been protected and is maintained in an incryption state either as data at rest or data in transit. In some embodiments, the QCU and QuID pairs are tightly coupled to their associated unique Packager/unPackager methodology.

In some embodiments, the QHC stack includes fragmenting each file into fragments each being a QE. In some embodiments with larger files or data objects, for example, the file may be split into data chunks with each data chunk then being split into fragments each being a QE. In various embodiments, a QHC stack may comprise a random number, selection and/or ordering of standard and/or non-standard ciphers, including obfuscation ciphers for injecting each QE with a set of random noise, encrypting each QE using an encryption protocol and a random seed or key, and shuffling a set of QEs into a random order such that the set of QEs is configured to be stored in a non-volatile memory as a QuID for the set of one or more original files. In some embodiments, each QE has a random size that exceeds a threshold size with a total number of QEs for each file

exceeding a threshold number. In various embodiments, the set of random noise for each QE is a set of random sizes of noise at a set of random locations in the QE whereby an attempted decryption of the QE would produce corrupted data if the set of random noise is not correctly removed before decryption. In various embodiments, each QE has an element name that is a random string of alphanumeric characters having a common length.

In some embodiments, some of the non-standard ciphers have a controllable Cipher Efficacy (CE) from a minimum CE_{MIN} to a maximum CE_{MAX} . In various embodiments, this CE can be tuned to dictate the overall latency, file size, and QHC efficacy of the unique Packager/unPackager methodology.

In some embodiments, the Packager system creates a QCU for the QuID having a quantum element map and a quantum cryptography map. In some embodiments, the quantum element map and the quantum cryptography map are separate files generated as part of the QCU. In other embodiment, the quantum element map and the quantum cryptography map are represented not as separate files, but rather as a single merged file of manipulated strings of information that protect and hide the encryption-related information, data and/or metadata relating to the incryption/dicryption processes and methodologies.

In some embodiments, the QCU is created as an amalgamated cryptex of various incryption-related information, data, and/or metadata combined into a single encrypted data object that serves as a metakey for each plain text package to be protected. An advantage of these embodiments is that the QCU can provide a package uniqueness level of protection unmatched by AES even in a situation where unique symmetric AES keys are used for each encryption of different plain text packages. In embodiments, not only are the QHC protection and obfuscation ciphers different for each package to be protected, the QHC protection and obfuscation ciphers order and efficacy can also vary on a per package basis.

In some embodiments, a quantum element map or meta data includes for each QE the corresponding quantum element name together with the random order and random size of that QE and the set of random sizes and the set of random locations of the set of random noise in that QE. The quantum cryptography map can include the random seed or key and a nonce value and for each file of the set of original files a set of element names of the QEs that correspond to each of the fragments of that file together with an original file name and a set of metadata for that file. In some embodiments, the quantum cryptography map includes a variable and/or random order of

the set of protocols, including at least some of the QHCs, including an order of at least some of the QHCs, used by the Packager system to incrypt and create the QuID. In some embodiments, the quantum cryptography map further includes meta data for a set of decoy digital assets in the set of original digital assets. In some embodiments, the quantum element map and the quantum cryptography map are created as separate files or data objects within the QCU. In other embodiments, the information for mapping elements and correlating element names to file names may be integrated into the QCU as one or more sets of manipulated strings of information that may be merged or integrated into a single data object.

In some embodiments, the set of protocols used by the Packager system to incrypt and create the QuID further includes a protocol order for the set of protocols that is variable. In other embodiments, the set of protocols further includes adding a set of decoy files to the set of original files. In some embodiments, the unPackager system is configured to access the protected package stored and for each computing node of the target computing system, unpackage one of the set of QuIDs to be accessed by that computing node and use a corresponding one of the set of QCUs to recover one or more of the set of original files. In embodiments, the unPackager system unpackages each file of the set of original files by reconstructing the file from the QuID using the quantum element map, removing the set of random noise from the file using the quantum element map, and configuring the file using the quantum cryptography map to be dicrypted and stored in a volatile memory of the computing node as the corresponding original file under the corresponding original file name.

In some embodiments, at least the set of QCUs are stored in a secure removable storage system such as a biometric access protected removable drive. In various embodiments, storing the set of original files in volatile memory of the computing node further protects the original files as they are effectively erased when the computing node is powered off.

In some embodiments, the Packager/unPackager system includes a separate Quantum Symmetric Secret (QSS) produced by the quantum random number generator. The QSS bears resemblance to a symmetric encryption key, yet it differs from conventional cryptographic symmetric keys by the way it is applied and used by one or more of the QHCs. In some embodiments, the QSS is used to provide an additional element of security for the Packager/unPackager system. In some embodiments, the QSS is injected at locations into the binary executable of the Packager system and the string of QSS bits is extracted and used to

confirm the proper relationship of a QCU/QuID pair and/or the proper relationship of the unPackager to the secondary node.

In some embodiments, the target computing environment for the digital asset protection system is an embedded system securely connected to a secure computer node and isolated from any external connections. In various embodiments, the embedded device includes a primary microcontroller having a volatile memory, a non-volatile memory, and a secure internal communication channel to the secure computer node, and at least one secondary microcontroller having at least a volatile memory and a secure internal communication channel to the primary microcontroller. In embodiments, the unPackager system utilizes at least the secure node to authorize and unpackage the unique one of the set of QuIDs generated for each of the primary microcontroller and the at least one secondary microcontroller to be loaded over the secure internal communication channels into the volatile memory of that microcontroller.

In some embodiments, the target computing environment for the digital asset protection system is a networked device securely connected to the secure computer node via an external network. In various embodiments, the networked device includes as computing nodes of the target computing environment at least one set of servers, each server having a volatile memory, a non-volatile memory, and a secure external communication channel to the secure computer node. In embodiments, the unPackager system utilizes at least the secure node to authorize and unpackage the unique one of the set of QuIDs generated for each of the server to be loaded into the volatile memory of that server utilizing double ratchet transmission control protocol communications over the external network.

The summary above is not intended to describe each illustrated embodiment or every implementation of the present disclosure. The figures and the detailed description that follow more particularly exemplify these embodiments.

BRIEF DESCRIPTION OF THE DRAWINGS

The disclosure can be more completely understood in consideration of the following detailed description of various embodiments of the disclosure, in connection with the accompanying drawings, in which:

FIG. 1 is a high-level schematic of an embodiment of a technology architecture for a quantum cyber resilient digital asset protection system.

FIGs. 2A and 2B are simplified high-level flow charts showing a comparison of the prior art process for standard AES encryption/decryption with the process for incryption/dicryption using an embodiment of the quantum cyber resilient digital asset protection system.

FIG. 3 is a flow chart of embodiments of the incryption process of the quantum cyber resilient digital asset protection system.

FIG. 4 is a flow chart of embodiments of the dicryption process of the quantum cyber resilient digital asset protection system.

FIGs. 5A and 5B are simplified block diagrams showing a comparison of the prior art process for standard AES encryption with the incryption process of an embodiment of the Vault.

FIG. 6 is a block diagram of the Vault embodiment of the quantum cyber resilient digital asset protection system.

FIG. 7A is a process flow diagram of an embodiment of the configuration manager for the Vault embodiment shown in FIG. 6.

FIG. 7B is a flow diagram of an embodiment of TCP communications for the Vault embodiment shown in FIG. 6.

FIG. 7C is a flow diagram of an embodiment of a Packager for the Vault embodiment shown in FIG. 6.

FIG. 8 is a dependency diagram of an embodiment of the digital asset protection system.

FIG. 9 is a high-level schematic diagram of an embodiment of the digital asset protection system.

FIG. 10 is a schematic diagram of one embodiment of a Quantum Hybrid Cipher (QHC) stack in accordance with an embodiment of the digital asset protection system.

FIG. 11 is a schematic diagram of an embodiment of a Transit configuration of the quantum cyber resilient digital asset protection system.

FIG. 12 is a process flow diagram of an embodiment of the Transit configuration embodiment shown in FIG. 11.

FIG. 13 is a schematic diagram of an embodiment of a Storage configuration of the quantum cyber resilient digital asset protection system.

FIG. 14A is a process flow diagram of a Distribute process of an embodiment of the Storage configuration embodiment shown in FIG. 13.

FIG. 14B is a process flow diagram of a Retrieve process of an embodiment of the Storage configuration embodiment shown in FIG. 13.

FIG. 14C is a process flow diagram of a Purge process of an embodiment of the Storage configuration embodiment shown in FIG. 13.

While embodiments of the disclosure are amenable to various modifications and alternative forms, specifics thereof shown by way of example in the drawings will be described in detail. It should be understood, however, that the intention is not to limit the disclosure to just the embodiments described. On the contrary, the intention is to cover all modifications, equivalents, and alternatives falling within the spirit and scope of the subject matter as defined by the claims.

DETAILED DESCRIPTION

For purposes of this specification, the following terms and acronyms are specifically defined as follows:

Amalgamated Codex or Cryptex refers to a collection of various information, data and/or metadata elements that are combined into a single data object. The term "amalgamated" suggests a fusion of different elements, while "codex" is a collection of information. So, an amalgamated codex refers to a unified data object that brings together diverse pieces of information, typically in a structured or organized manner. In the context of quantum cyber resilience and incryption, this data object would include incryption-related information and may be protected or encrypted.

Anti-Tampering (AT) includes engineering and human activities and capabilities that are intended to prevent or delay access to or exploitation of critical program information.

Ciphertext refers to encrypted/incrypted text/data transformed from plain text/data using a cryptographic process.

Critical Program Information (CPI) refers to the cyber capabilities and elements of a device or system that contribute to technical advantage of such devices or systems.

Cyber is a term that indicates relationship or relatedness to the field of computer science, including computers, information technology and virtual reality.

Cyber Resilience is the ability to anticipate, withstand, recover from, and adapt to adverse conditions, stresses, attacks, or compromises on cyber systems that use or are enabled by cyber resources.

Cybersecurity is a field of computer science focused on protecting computer and information technology networks, systems, and data from unauthorized access and cyber-attacks.

Digital Assets includes data, code, databases, text, training sets and/or weighting sets for neural networks, ciphertext that is hardware and/or software encrypted data, digital content, or digitally personal identifiable information, or any electronic or optical information in the form of a file, a record, or an element as one of a set of digital assets that is at rest, in use or in transit. Digital Assets are sometimes referred to as intellectual property.

Double Ratchet Encryption is a kind of algorithm used to exchange encrypted messages based on an asymmetric secret key.

Encrypt/Decrypt/Encryption generally refers to use of a single Standard (NIST approved and/or publicly available) ciphers (ex: AES-256).

Incryption/Incrypt/Dicrypt refers to a set of computer-implemented processes, protocols and/or techniques used to construct and/or deconstruct an object of digital assets by encrypting/decrypting the digital assets as encrypted data using at least one Standard protection/encryption algorithm, and using one or more non-Standard protection/encryption ciphers and multiple invisibility/obfuscation processes, protocols, and techniques to make any intrinsic information derivable from the encrypted data that may have discernable patterns or properties effectively invisible.

Obfuscate/Deobfuscate refers to the cryptographic processes performed by the Non-standard obfuscation ciphers in the QHC Stack.

Package/Unpackage refers to the execution of the QuID/QCU Packager/unPackager.

Protect/Restore refers to the cryptographic processes performed by the Non-standard protection ciphers in the QHC stack.

Processor is used in this disclosure to represent specific hardware computing structures and is synonymous with terms like controller and computer and should be understood to encompass not only computers having different architectures such as single/multi-processor architectures and sequential/parallel architectures as well as quantum computers, but also specialized circuits such as field-programmable gate arrays (FPGA), application specific circuits (ASIC), graphic processing units (GPU), signal processing devices and other devices.

Quantum Computer: refers to a type of advanced computational device that makes use of the principles of quantum mechanics, a fundamental theory in physics that describes the behavior

of energy and material on atomic and subatomic levels. Unlike classical computers, which use bits as the basic unit of information (represented as either 0 or 1), quantum computers use quantum bits, or qubits.

Quantum Computing is a field of computer science focused on the development of computer and information technologies based on the principles of quantum theory that uses the unique properties of quantum physics to solve problems that are too complex and/or too massive for solution by classical computing.

Quantum Containment Unit (QCU) is a protected package that contains an amalgamated codex of various incryption-related information, data, and/or metadata combined in a single data object for a data asset package to be protected.

Quantum Cryptography Map (QCM) is a digital map/file/data object/structured object associated with a QuID that contains information about cryptography key, nonce, and associated information, and metadata for the set of digital assets that are protected as ciphertext in the QuID.

Quantum Cyber Resilience or Quantum Resilience are cyber resilience solutions and techniques applicable to adverse conditions, stresses, attacks, or compromises on cyber systems where the cyber resources used include quantum-computing.

Quantum Element (QE) is a single quantum incrypted piece or shard that corresponds to a fragment of a file, a record, or an element in a set of original digital assets.

Quantum Element Map (QEM) is a digital map/file/data object/structured object that contains information and associated metadata about the quantum elements (QEs) in a QuID including relationships and correlations to the original set of digital assets.

Quantum Element (QE) is a single quantum incrypted piece or shard that corresponds to a fragment of a file, a record, or an element in a set of original digital assets.

Quantum Hybrid Cipher (QHC) is a set of different protection and obfuscation ciphers applied in a random order and/or at a variable efficacy.

Quantum Incrypted Data (QuID) is an incrypted package of a set of digital assets protected as ciphertext quantum elements for a computing node in a target computing system.

Representational State Transfer Application Programming Interface (RESTful API) is a web service implementation that adheres to the principles of REST, a software architectural style used for creating scalable web services. RESTful APIs are used for building web services that are lightweight, maintainable, and scalable, often used for internet applications.

Software Anti-Tampering (SAT) includes anti-tampering designed or intended to prevent reverse engineering and exploitation of critical software and firmware technologies to deter technology transfer, alteration of system capability, or the development of countermeasures.

Security Strength Bits (k) is a value used to estimate an end-to-end security strength represented in effective key length bits of security strength.

Tape Archive (TAR) command refers to the Tape Archive tool in Unix and Linux used for creating and manipulating archive files. The basic function of tar is to bundle a collection of files and directories into a single archive file (commonly known as a tarball), which can be easier to transport and store.

Transmission Control Protocol (TCP) is one of the main protocols of TCP/IP networks that enables two nodes to establish a connection and exchange streams of data.

As the field of quantum computing continues to advance at a rapid pace, the encryption standards that were once considered unbreakable and have long been endorsed by NIST may soon be vulnerable to new forms of cryptographic attack. This vulnerability presents the risk of sensitive data being exposed and/or compromised, a risk that was previously thought to be nearly impossible.

In addition to the need to shift to post-quantum cryptography (PQC) as the standard for asymmetric encryption algorithms, there is a pressing need for the adoption of more advanced cryptanalytic agile processes and technologies that can keep up with the accelerating advances of CRQCs. Embodiments of the present disclosure provides a quantum resilience solution that is engineered to specifically address and neutralize the heightened threats posed to symmetric encryption by the exponentially exploding computational capabilities inherent in AI-enhanced CRQCs.

Unlike standard encryption algorithms which have relied on linearly increasing key size as the way to make the algorithms stronger, the incryption security framework for a quantum cyber resilient digital asset protection system can be designed for cryptanalytic agility by using randomized processes that allow for an exponential increase in a theoretical maximum symmetric security strength as an average estimated number of bits of security (k). The value of k for various embodiments uses an estimated effective key size per cipher as part of the basis for calculating the effectiveness of various embodiments as described. In embodiments, k may be designated in an

equivalent key size in bits to permit a comparison to both the AES key sizes and the current export license key size reporting threshold for a symmetric non-standard encryption cipher.

In embodiments, estimating k is broken out by ciphers that are categorized as either obfuscation ciphers, which are non-standard, and protection ciphers, which are either standard encryption or non-standard encryption. It will be understood that the obfuscation ciphers will have less individual effective key size impact as the power of the obfuscation derives from the lack of knowledge of whether/how the cipher is being used. Likewise, the non-standard encryption ciphers can also have some effective key length benefit beyond just the key size, especially if the non-standard encryption cipher is not publicly known. In addition to an evaluation of the individual ciphers, k can include the factorial/exponential impact of the random order and cipher efficacy of the ciphers in a QHC stack. In various embodiments, a new QCU is created as an amalgamated codex or cryptex of various incryption-related information, data, and/or metadata combined into a single encrypted data object representing a metakey for each plain text package to be protected. In such embodiments, the QCU provides a package uniqueness level of protection unmatched by AES even if unique symmetric AES keys are used for each encryption; and not only is the QHC stack different for different packages, the incryption protection process also varies per protected package.

In some embodiments, the incryption scheme of the digital asset protection system can be analogized as generating random puzzle pieces that simply do not make sense and cannot be reassembled without the information in the QCU. When none of the puzzle pieces make sense, it is impossible to reassemble the puzzle. In various embodiments, the incryption schemes as disclosed have too many non-algorithmic options and configurations that cannot be evaluated using brute force attacks. Because the number of permutations of a given QuID is so unimaginably large there are no known mathematical curves that could be determined to represent the QuID.

Referring now to FIG. 1, a computer-implemented digital asset protection system 100 is shown, that is configured to protect a set of original files of digital assets 102 as protected packages 104 of quantum incrypted data that is quantum cyber resilient. In various embodiments, the set of original files of the digital assets 102 can include a set of files having a single original file for a single computing node in a target computing environment, a set of multiple original files for a single computing node, or multiple sets of original files each set for a different computing node in a target computing environment. Each original file of the digital assets can include digital

information in the form of a file of data or code, for example, or a record or an element of a database, for example, as one of a set of digital assets that is at rest, in use, or in transit.

In embodiments, the EnQuanta™ digital asset protection system 100 includes a QuantaPack™ Packager/unPackager system 110 configured to protect the set of original digital assets 102 as protected packages 104 by incryption and recover the set of original digital assets 102 by dicryption. In some embodiments, the functionality of the Packager/unPackager system 110 may be divided into separate programs/binaries for a Packager 112 and an unPackager 114.

In some embodiments, aspects of the Packager system 110 are within a secure computing environment that is isolated and/or secured for both physical and network access. In some embodiments, the Packager system 112 is configured to generate a protected package 104 to be securely delivered to and stored by a storage system of a target computing environment. In some embodiments, the unPackager 114 includes a QuantaPack™ QHC binary 116 to be downloaded into the target computing environment. In other embodiments, the functionality of the Packager/unPackager system 110 is a combined program/binary that is executed in a secure development or factory computing environment. In other embodiments, the unPackager 114 is configured to be installed in a secure computing node of the target computing environment to unpackage the protected package 104. In other embodiments, the combined program/binary for the Packager/unPackager system 110 is installed at each of a plurality of nodes connected by a secure/unsecure network in the computing environment. In these embodiments, the Packager/unPackager system 110 takes in the digital assets 102 that are to be protected and creates protected package 104 that is safe to be sent from one node as a transmit node over an unsecure/secured network to one or more other nodes as a receive node that is also equipped with a Packager/unPackager system 110 to unpackage the protected package 104 and recover the original digital assets 102.

In embodiments, aspects of Packager/unPackager system 110 are operably provided with a random number generator entropy source such as a hardware and/or software quantum random number generator or a hardware and/or software random number generator as an entropy source 106 in various forms which will be described, some of which may be referred to as a quantum random number generator (QRNG). In some embodiments, entropy source 106 of the digital asset protection system 100 is configured to generate a random seed and/or key to be used by the Packager/unPackager system 110 and in some embodiments by other software binaries that are

part of the digital asset protection system 100. In some embodiments, entropy source 106 also provides a seed to a QuantaSecret™ quantum symmetric secret (QSS) 118 that is integrated into various data objects as a further layer of protection as will be described.

In embodiments, the set of protocols and techniques used by the Packager 112 to protect the set of original files as quantum encrypted data uses the random number generator entropy source to seed different variable and randomized aspects of a set of Quantum Hybrid Ciphers (QHCs) 108 to generate a set of QuantaElements™ quantum elements (QEs) 122 that are combined together as a QuantaSafe™ quantum encrypted data (QuID) 124 representation for the set of original digital assets 102. Unlike conventional encryption techniques that include only a single standard encryption algorithm or use a limited and fixed number and order of non-standard cipher techniques, the QHCs 108 in various embodiments are a stack of multiple different ciphers that used in an order and number which is variable to at least some degree to further enhance the strength and cryptographic agility of the quantum cyber resilient security.

In various embodiments, the output of the QHC stack 108 is a paired QuantaKey™ Quantum Containment Unit (QCU) 120 and QuantaSafe™ QuID 124 as a protected package 104. The term “incryption” is somewhat analogous to “encryption,” but encompasses an expanded cryptographic process that can include one or more standard encryption algorithms plus non-standard QHC processes. A QCU 120 is an amalgamated codex or cyptex of various incryption-related information, data, and/or metadata to be used by the unPackager for the “dicryption” process that reverses the methodology and sequencing of the paired QCU 120 and QuID 124 as a protected package 104. The QuID 124 is the collection of digital assets (e.g., data, message, and/or code) that has been protected and is maintained in an incryption state either as data at rest or data in transit.

In some embodiments, the QCU 120 and QuID 124 pairs are tightly coupled to their associated unique Packager/unPackager methodology. In some embodiments, the QHC binary 116 and the QCU 120 are merged together as a QuantaKey™ + QuantaPack™ binary - QCU+ 126 that provides both the amalgamated codex or cyptex and the QHC binaries needed to decrypt the QuID 124. In some embodiments, a single pair of the QCU 120 and QuID 124 form the protected package 104. In other embodiments, a set of multiple pairs of QCUs 120 and QuIDs 124 are merged together into a QuID/QCU stack 128.

In some embodiments, there is a one-to-one relationship of a protected package 104 of the incryption protected QEs 122 bundled together in a QuID 124 and a corresponding QCU 120 as the amalgamated codex or cyptex of incryption-related information that serves as a kind of key ring that is itself locked in a lock box to securely hold the multiple, randomly ordered cipher keys and associated meta information for the QHCs 108 used to incrypt a QuID 124. In other embodiments there can be a one-to-many relationship or a many-to-one relationship between a single QCU 120 or a set of QCUs 120 and a set of QuIDs 124 or a single QuID 124 in the incryption protected packages 104.

In some embodiments, the QCU 120 includes information and meta data representing a set of random keys, nonces, and/or values is used for all randomization in the encryption and incryption in which at least some of non-standard ciphers are used in a random order and/or at a selective cipher efficacy as part of a QHC stack 108 used for quantum resilience protection of a given protected package 104. In other embodiments, some randomization values in the encryption and incryption of a given protected package may use a pseudo random value instead of a quantum random key. In some embodiments, the QCUs 120 are encrypted with the same or a different encryption key for one of a set of standard protection ciphers or encrypted using a different encryption key or a different cryptographic approach than the standard encryption ciphers used on the QuIDs 124.

In embodiments, the QCU 120 is used by a computing node in a target computing system to decrypt the corresponding QuID 124. In various embodiments, the QCU 120 can be an obfuscated JSON file containing all metadata related to a single associated QuID 124 or, in alternate embodiments, a set of associated QuIDs 124. The metadata consists of information about the incryption process used by the Packager system 112 to generate the QuID and enables a corresponding unPackager system 114 to unpackage the QuID 124. In embodiments, a unique “fingerprint” may be generated based on the QCU 124, and this fingerprint is injected into the unPackager binary of the unPackager system 114 or the QHC binary 116 to prevent other binaries besides the intended one from using the QCU 120 associated with that fingerprint. In some embodiments, the QCU 120 is optionally protected using information inherent to the QHC process to provide encryption, incryption, and/or additional security without having to separately store/share a unique key.

The EnQuanta™ hybrid cryptography QuantaPack™ process uses a dynamic combination of standard and non-standard ciphers to create a unique QuantaKey™ cryptex for each QuantaSafe™ ciphertext as protected packages for data-at-rest or data-in-transit applications. The protected packages can be safely sent over the internet and/or stored in permanent memory devices without any weaknesses. They look like a "sea of nothingness" that cannot be understood or hacked. By generating unique keys for each package, the EnQuanta hybrid cryptography provides stronger security than AES256-bit encryption.

The built-in key management simplifies encryption for data-in-transit, without need for complex public/private keys. The software-only solution is agile, ensuring compatibility with current and future NIST standards. Because cryptography is used in every aspect of our modern interconnected digital world, we need to start using better types of cryptography now to stay ahead of future threats. The EnQuanta solution is a new type of hybrid cryptography with four main benefits:

- **Better Protection:** EnQuanta provides stronger protection against today's cyber threats than 256-bit standard encryption by generating a unique, sophisticated key (cryptex) for each piece of protected information. Like a Shannon one-time pad, the cryptex is never reused but is smaller and easier to conceal. Using EnQuanta's hybrid cryptographic solution ensures no weaknesses for hackers to exploit.
- **Integrated Hybrid Key Establishment:** EnQuanta's amalgamated cryptex is an integrated approach to hybrid key establishment. This is superior to using just the recently NIST-approved PQC asymmetric encryption for public/private key exchanges. Using the integrated hybrid QuantaKey cryptex reduces the complexity of key exchanges for data-at-rest and eliminates the need for asymmetric key exchanges for data-in-transit.
- **Cryptographic Agility:** EnQuanta uses various encryption algorithms within its sophisticated and dynamic processes, and supports the latest available NIST standard algorithms, whatever they may be.
- **Quantum Cyber Resilience:** EnQuanta's flexibility allows it to keep ahead of both current cyber threats and future attacks from quantum computers and artificial intelligence.

As described herein, the EnQuanta solution is a new way to keep information safe from today's hackers and tomorrow's unknown challenges from quantum computers and artificial intelligence. It's a sophisticated software-only process called QuantaPack™ that does not require any special hardware. The EnQuanta solution enables quantum cyber resilience by implementing the QuantaPack process to produce QuantaKey/QuantaSafe pairs of protected cryptext/ciphertext:

- **Cryptoagility:** The hybrid cryptography solution combines standard encryption with unique multi-layered cryptographic methods in which both standard and non-standard ciphers in the QuantaPack process are easily adaptable/replaceable as soon as vulnerabilities are identified.
- **Increased key size:** The QuantaKey cryptext is dynamically generated for each ciphertext as a smaller version of a Shannon one-time pad and is variable in length based on the size and structure of the plaintext.
- **Post-quantum cryptography:** The EnQuanta Solutions can utilize PQC asymmetric algorithms for TCP Double Ratch Communications. The QuantaPack Cipher Stack can be upgraded to incorporate current and any future PQC symmetric encryption algorithms as they are approved by NIST.
- **Hybrid approaches:** The increased effectiveness of the EnQuanta hybrid cryptographic solution can be demonstrated in terms of comparing bits of security for EnQuanta's products to the 256 bits of security for AES.

EnQuanta's hybrid cryptographic solutions can deliver more than additional bits of security beyond what AES-256 alone provides. In some embodiments for commercial products for data-in-transit products, the QuantaPack process can provide more than 350 additional bits of security for data-in-transit products, and more than 500 additional bits of security for data-at-rest products. These figures represent a googol (10^{100}) times more resistant to brute-force attacks than AES alone for data-in-transit products, and a googol (10^{100}) multiplied by the number of atoms in the Earth (10^{150}) times more resistant to brute-force attacks than AES alone for data-at-rest products. Such estimates of the number of security bits for various embodiments of the QuantaPack process are based on a mathematical analysis of estimated averages of the outputs of security bits provided by a set of QuantaKey cryptextes and QuantaSafe ciphertexts. With the QuantaKey cryptext being a variable length one-time pad uniquely associated with the QuantaSafe ciphertext, no two members of the set of outputs will have the same estimated number of security bits. Hence, the estimated

averages are used in the mathematical model as a reasonable way to approximate the number of security bits provided by different specifications and modes of the QuantaPack process.

FIGS. 2A and 2B show a high-level comparison of the prior art process for standard AES encryption/decryption with the process for incryption/dicryption using the quantum cyber resilient digital asset protection system. In embodiments of the quantum cyber resilient digital asset protection system 100 shown in FIG. 2B, the Packager system 112 is configured to use incryption to generate the QCU 120 and the corresponding QuID 124 that are packaged to form the incryption protected package 104. The unPackager system 114 is configured to unpackage the protected package 104 using the QCU 120 to dicrypt the QuID 124 to reconstitute the original digital assets 102 as plaintext data. This process is somewhat like the high-level process for standard AES encryption 130 as shown in FIG. 2A where the plaintext data 102 is fed into the AES encryption algorithm 132 along with the AES key 136 (e.g. AES256 bit key) to generate the ciphertext 134. The AES decryption algorithm 138 can decrypt the ciphertext 134.

There are numerous differences, however, in these two processes. Foremost among these differences, the incryption process of the digital asset protection system 100 is a process in which the QCU as a variable metakey is an output of the incryption process and an input to the decryption process. In contrast, the standard AES process 130 is a process in which the same externally provided fixed key 136 is used as an input to the same algorithm used in both encryption 132 and decryption 138. Other differences in AES encryption versus embodiments of the digital asset protection system 100 include:

- Quantum protection can be facilitated by using QRNG entropy randomization throughout the QHC processes, instead of relying on just antiquated pseudo-random number generators and widely published ciphers, such as AES.
- The QCUs are exponentially more complex and obscure than current symmetric encryption keys.
- QCUs can be selectively separated from the QuIDs but remain matched sets.
- The QHC processes are cryptographically agile as they are not static; compare this to static keys that are stored with the encrypted data and the standard approved algorithm using those static keys never changes.

Table 1 below shows a side-by-side comparison of various features of each of these processes for various embodiments of the present disclosure.

Table 1: Comparison of Incryption/Dicryption vs. AES Encryption/Decryption

Feature	Incryption/Dicryption	AES Encryption/Decryption
Ciphertext	<u>Incrypted Data (i.e. QuID)</u> A QuID is produced by a randomized cipher stack of QHCs consisting of standard and non-standard ciphers that provide encryption capabilities along with alternative data transformations. Embodiments combine the QHCs into a dynamic incryption process utilizing QRNG that produces a QuID and a generated QCU used as a metakey. A QuantaSafe ciphertext is produced by a randomized and hybrid Cipher Stack consisting of standard and non-standard ciphers that provide encryption capabilities along with alternative data transformations. The QuantaPack process produces a protected package of a QuantaSafe ciphertext and a package-unique QuantaKey cryptex.	<u>Encrypted Data (i.e. ciphertext)</u> AES ciphertext is produced by repeated application of a one-way mathematical function based on a constant-length key as input. Full diffusion in AES takes 2 rounds, meaning that after 2 rounds every bit in the output is dependent on every bit in the input. AES256 runs for 14 rounds. AES ciphertext is produced by repeated application of a round function based on a 256 bit key as input. Full diffusion in AES takes 2 rounds, so after 2 rounds every bit in the output is dependent on every bit in the input. AES256 runs for 14 rounds. See : The Design of Rijndael: AES - The Advanced Encryption Standard SpringerLink
Unpackaged Data	Dicrypted QuID is written to a RAMFS volatile memory mounted drive, from which it is automatically cleared if power is lost or other kill parameters are met, leaving no trace of its existence on the system. Dicrypted QuantaSafe ciphertext is written to a RAMFS volatile memory mounted drive, from which it is automatically cleared if power is lost or other kill parameters are met,	Decrypted ciphertext is typically written directly to persistent memory (RAMFS by default is not provided to AES). Even if sensitive decrypted information is deleted from the system, it may still be recoverable by system forensics tools. Decrypted ciphertext is typically written directly to persistent memory (RAMFS by default is not provided to AES). Even if sensitive decrypted information is deleted from the system,

	leaving no trace of its existence on the system.	it may still be recoverable by system forensics tools.
Process	<u>Non-deterministic</u> Randomized process that is easily reversible given access to the QCU and encryption algorithm. The process changes every time incryption is performed. Multiple ciphers (both standard and non-standard) ordered with true quantum randomization. Additional security framework processes are typically needed to properly use the incryption/dicryption core capabilities. The randomized non-deterministic process makes it impractically difficult to reverse the QuantaSafe ciphertext without access to the QuantaKey cryptex and QuantaPack process. The QuantaPack process changes every time incryption is performed and includes multiple ciphers (both standard and non-standard) applied in a hybrid and random order (using RNG or QRNG). Additional cyber resilience framework processes are required to properly use the QuantaPack incryption / dicryption core capabilities.	<u>Deterministic</u> Well-defined process that is easily reversible given access to the key and decryption algorithm. The process is the same every time encryption is performed. 14 rounds with 3 operations/round: SubBytes, ShiftRows, MixColumns. No additional framework processes are needed to use AES; the encryption/decryption algorithm is public knowledge. Well-defined deterministic process that can be susceptible to future attacks by CRQC coupled with AI and by AI-enhanced Grover's Adaptive Search. The process is the same every time encryption is performed. 14 rounds with 3 operations/round: SubBytes, ShiftRows, MixColumns. No additional framework processes are needed to use AES. The encryption/decryption algorithm is public knowledge.
Key	The QCU is an amalgamated codex generated by the incryption process, using a QRNG entropy source that is effectively a metakey larger than the AES key. The QCU is an <u>output</u> from the incryption process. Because the QHCs use a QRNG, all random processes can	An external source provides the AES key to the encryption algorithm. The initialization vector sizes used by AES are typically generated from a pseudo-random entropy source. The key is an <u>input</u> to the encryption process. If the key was not properly generated, it could undermine the AES encryption security. Using password-

	be seeded properly and without the possibility of weak keys. QCU size is variable, directly related to input data size. Example: 1.3Mb protected data → 12Kb QCU 20.6Mb protected data → 150Kb QCU The QuantaKey cryptex is generated by the QuantaPack process. The QuantaKey is an output from the QuantaPack process with all random processes properly seeded with true random numbers and without the possibility of weak keys. QuantaKey size is variable, directly related to input data size, number of data objects and number of data files. In typical situations not involving large numbers of data files and directories, the QuantaKey size can range from 0.5% - 5.0% of a plaintext input data size. QuantaKey File Size Examples: 1KB-4KB plaintext data → 100-200 bytes 1.3 MB plaintext data → 12 KB bytes 20.6 MB plaintext data → 150 KB bytes	based keys or low quality pseudo-random keys can lead to this issue. Key size is predetermined, not related to input data size. Example: 1.3Mb protected data → 0.032Kb key 20.6Mb protected data → 0.032Kb key An external source provides the AES key to the encryption algorithm. The key is an input to the encryption process. If the key was not properly generated, it could undermine the AES encryption security. Using password-based keys or a low quality pseudorandom number generator can lead to this issue. Key size is predetermined, not related to input data size. AES Key File Size Examples: 1KB-4KB plaintext data → 32 bytes 1.3 MB plaintext data → 32 bytes 20.6 MB plaintext data → 32 byte
Comparison Results	k is in a range dependent on the Specification and number of data objects. k DUAL-USE: $\cong 356 \leq k \leq 896$ Δk DUAL-USE: $\cong 100 \leq \Delta k \leq 640$ DUAL-USE = Base Specification	k for AES is a fixed value equal to the number of bits in the AES key, which for AES256 is 256. k AES256: = 256 Δk AES256: = 1

FIGS. 5A and 5B show a comparison of the prior art process for standard AES encryption with the incryption process of an embodiment of the Vault. As described with respect to FIGS. 2A and 2B, for example, standard AES encryption is a process in which a same fixed key is used as an input to a same algorithm used in encryption and decryption, whereas an incryption process used when performing a secure vault data transaction is a process that employs a variable metakey and a variable set of algorithms. Further distinctions between the standard AES encryption and the incryption process and additional details of the processes will now be discussed.

Referring now to FIG. 5A, a typical AES data transaction is shown. To initiate the AES data transaction, for example, a system user can interact with a user interface of computing device 152 to specify the key 136 (e.g., a public (asymmetric)/private (symmetric) key) to be used for encrypting and/or decrypting data. The computing device 152, for example, can transmit the key 136 to a system that is configured to perform standard AES encryption 130, over secure/unsecure channel(s) 158. After receiving the key 136, for example, the encryption-enabled system can use the key 136 as an input to the encryption/decryption method 132 (e.g., an AES encryption/decryption method), which can operate on plaintext data to generate encrypted data 134 (e.g., ciphertext), which can be stored in memory. Similarly, the same key 136 that was used to encrypt the plaintext data can be used as an input to the encryption/decryption method 132, which can operate on the encrypted data 134 (e.g., ciphertext) to decrypt and restore the plaintext data.

Referring now to FIG. 5B, a Vault data transaction is shown. To initiate the Vault data transaction, for example, a system user can interact with a user interface of computing device 150 to provide commands to a Packager/unPackager system 110 that is in communication with the computing device 150. In the present example, the Packager/unPackager system 110 includes an entropy source 106 that can generate a seed for various aspects of the cipher and other protection processes of the Packager/unPackager system 110, including the various QHCs 108 that are to be randomly selected from and randomly sequenced when incrypting data (e.g., plaintext data). As part of an incryption process, for example, a QHC stack 108 can be generated by the Packager/unPackager system 110 and can be used to create QEs 122 from the plaintext data. As shown in Fig. 6, a Quantum Containment Unit bundle (QCU+) 126 that includes a QHC binary 116 and a QCU 120 can also be generated and packaged together. The QCU+ 126 and the QuID 124 can be provided, together or separately, over secure channel(s) 156 to a Vault system 140. After receiving the QCU+ 126 and the QuID 124, for example, the Vault system 140 can employ

an unPackager App Stub 117 that extracts the QHC binary 116 from the QCU+ 126 to operate on the QuID 124 according to the information included in the QCU 120 that is also included in the QCU+ 126 to restore the plaintext data. Rather than merely being an input to an encryption process like the key 136 for decryption in the AES process 130, the QCU 120 is generated as one of the outputs of the incryption process. Thus, the format, size, and content of the QCU is variable, adding another layer of complexity and security.

FIGS. 3 and 4 are more detailed flow charts of embodiments of an incryption process 300 (shown in FIG. 3) and a decryption process 350 (shown in FIG. 4) of the quantum cyber resilient digital asset protection system 100. The processes 300, 350 can be performed by components of the digital asset protection system 100, for example, and will be described with reference to FIG. 1. However, other systems may be used to perform the same process or a similar process. Operations of the processes 300, 350, for example, may occur in the illustrated sequence, or may occur in a sequence that is different than in the illustrated sequence. In some embodiments, two or more operations of the processes 300, 350 may be concurrent.

Referring now to FIG. 3A, at 302, the incryption process 300 starts. For example, the Packager/unPackager system 110 of the digital asset protection system 100 (shown in FIG. 1) can receive as input the set of original digital assets 102 (also shown in FIG. 1), and a command to encrypt the original digital assets 102. The original digital assets 102, for example, can represent one or more data units, including files, records, application code, or other sorts of digital information.

At 304, a determination is performed of whether there is more data to package. During the incryption process 300, for example, the Packager 112 of the Packager/unPackager system 110 can track a current status of the process 300 over time, including an indication of which portions (e.g., files or file chunks) of the original digital assets 102 have already been encrypted, and an indication of which portions of the original digital assets 102 have yet to be encrypted. In the present example, as the process 300 has just started, the original digital assets 102 have yet to be encrypted.

At 306, a portion of the data (e.g., a file or file chunk) is read. For example, the Packager 112 of the Packager/unPackager system 110 can reference its internal tracking of the current status of the incryption process 300 for the digital assets and can select and read the next portion of the data for incryption. In general, the incryption process 300 can proceed iteratively through the

original digital assets 102, with a first portion of the original digital assets 102 being encrypted first, with a second portion of the assets 102 being encrypted second, and so forth, until all portions of the assets 102 have been encrypted. In some embodiments, the data portions can be of a uniform size. In some embodiments, the data portions can be of a variable size. For example, each file of a set of data files can be of a different size, and each data file can be separately read and encrypted. As another example, a single data file can be partitioned into a set of differently sized chunks, which are separately read and encrypted.

At 308, a randomized Quantum Hybrid Cipher (QHC) stack (e.g., selected from the QHCs 108, shown in FIG. 1) is generated, based on a configuration mode. For example, the Packager 112 of the Packager/unPackager system 110 can reference random data generated by the entropy source 106 (or another source of random data) to randomly select from a set of possible ciphers (e.g., including both NIST-based ciphers and non-standard protection/obfuscation ciphers), and to randomly sequence the selected ciphers. In general, the QHC stack includes at least two different ciphers, and can possibly include many ciphers. In some embodiments, all of the ciphers in the generated QHC stack can be different ciphers. In some embodiments, some of the ciphers in the generated QHC stack can be the same cipher, repeated at different positions in the stack, and optionally with different configuration parameters. Further details related to the ciphers and their configurations are provided below with respect to FIGs. 9 and 10, as well as elsewhere in this document and the supporting documentation.

At 310, a next QHC input is initialized, based on source data. For example, the Packager 112 of the Packager/unPackager system 110 can provide the portion of the data that was previously read (at 306) as the next QHC input to a process that executes the QHC stack 108. In general, to execute the QHC stack 108, the Packager 112 can sequentially execute each cipher according to the determined (at 308) random order of ciphers, with the first cipher operating on the portion of the data that was read, with a second cipher operating on the output of the first cipher, and so forth, until each of the ciphers in the QHC stack 108 have been executed.

At 312, an individual cipher is executed. For example, the Packager 112 of the Packager/unPackager system 110 can track a progress of an incryption process for the previously read data (e.g., the portion of data that was read at 306), including a reference to a position in the QHC stack 108 of a currently executed cipher. In the present example, the first cipher in the QHC stack 108 can be executed on the portion of the data that was previously read. Executing the

individual cipher, for example, can include executing the cipher code for the cipher according to the processing parameters that have been defined for the cipher (e.g., during a previously performed configuration operation, during a randomized parameter selection process, or at another time).

At 314, QHC processing parameters are stored in a Quantum Containment Unit (QCU) data structure 120. For example, the Packager 112 of the Packager/unPackager system 110 can store the QHC processing parameters for the executed individual cipher in the QCU data structure 120, along with an identifier of the executed cipher. As the incryption process 300 progresses, for example, the identifier of each executed cipher and its corresponding processing parameters can be sequentially appended to the QCU data structure 120. In some embodiments, QCU data structure 120 can be considered as a conceptual map that is generated for use in later dicrypting the incrypted data.

At 316, a determination is performed of whether there are additional ciphers to execute. For example, the Packager 112 of the Packager/unPackager system 110 can reference the position of the previously executed cipher in the QHC stack 108 and can determine whether any subsequent ciphers exist in the QHC stack 108 (e.g., ciphers that have yet to be executed), according to the current position and the previously determined random order of ciphers. In the present example, the Packager 112 determines that several additional ciphers exist in the QHC stack 108 after the first executed cipher.

At 318, if there are additional ciphers to execute, the next QHC input is initialized based on previous QHC output. In the present example, the Packager 112 of the Packager/unPackager system 110 can initialize the next QHC input based on the output of the previously executed cipher (e.g., the first cipher). Operations for incrypting the portion of the data can iteratively continue, with the next individual cipher in the QHC stack (according to its sequential position in the QHC stack 108) being selected and then executed (at 312), the QHC processing parameters for the next individual cipher being stored in the QCU data structure 120 along with an identifier of the next individual cipher (at 314), and the determination of whether additional ciphers exist in the QHC stack 108 being performed (at 316), until all of the ciphers in the QHC stack 108 have been executed, and all of the QHC processing parameters for the executed ciphers have been stored in the QCU data structure 120 along with the corresponding cipher identifiers. As the output of an executed cipher is being provided as an input to a next cipher in the QHC stack 108, it will be

appreciated how the resulting incrypted data becomes more and more indecipherable with each iteration.

At 320, if there are no additional ciphers to execute, a set of quantum elements (QEs) 122 is generated based on the final output of execution of the QHC stack 108. For example, the Packager 112 of the Packager/unPackager system 110 can generate the set of QEs 122 (e.g., data fragments), from the output of the last executed cipher in the QHC stack 108 and can optionally shuffle the QEs in a random order. In the present example, the Packager 112 can also generate a unique identifier (e.g., a QE name) for each QE in the set of QEs.

At 322, the unique identifiers (e.g., the QE names) are stored in the QCU data structure 120. In some embodiments, the QE names are standardized and obfuscated to remove any discernable metadata information. For example, the Packager 112 of the Packager/unPackager system 110 can store, for each QE 122 in the set of QEs, the unique identifier (e.g., the QE name) that had been generated for the QE 122, in the QCU data structure 120, along with the optional information that indicates its sequence among the set of QEs 122. The information pertaining to the QEs 120 can be stored in the QCU data structure 120 along with the information that pertains to the executed ciphers in the QHC stack 108 (e.g., the cipher identifiers and associated configuration parameters).

In the present example, the incryption process 300 loops back to 304, where another determination is made of whether there is more data to package. If so (e.g., one or more files remain unpackaged, one or more data chunks remain unpackaged, etc.), the next portion of the data (e.g., file or file chunk) can be read (at 306) and can be independently incrypted. Incryption of the next portion of the data, for example, can be performed according to a new randomized QHC stack 108 that is different from the previously executed QHC stack 108, and which results in an additional set of QEs 122 and an additional QCU data structure 120 for the next portion of data. The process 300 iterates thusly, until all the data has been incrypted.

At 324, if no more data is to be packaged (e.g., all of the data files or chunks have been packaged), a determination is performed of whether Quantum Incrypted Data (QuID) 124 bundling is enabled (e.g., based on a configuration setting of the Packager/unPackager system 110). If so, at 326, a QuID bundle 126 of incrypted QEs 122 is created and then encrypted. For example, the Packager 112 of the Packager/unPackager system 110 can bundle all of the previously generated

QEs for all of the incrypted portions of data into a single data package for secure storage and/or transmission.

At 328, a QCU 120 is generated and encrypted from the stored QCU data structure(s) (e.g., either in response to determining that QuID bundling is not enabled, or in response to determining that QuID bundling is enabled and upon creating and encrypting the QuID bundle). For example, the Packager 112 of the Packager/unPackager system 110 can create and encrypt the QCU 120 (shown in FIG. 1), using one or more standard and/or non-standard encryption techniques. At 330, after creating and encrypting the QCU 120, the incryption process 300 ends.

For a more detailed schematic of the data and process flow from the input data of a digital asset split into multiple data chunks and processed by multiple Cipher Stacks to generate QuantaElements and corresponding cryptex information and meta data for each Cipher Stack that are then arranged into a QuantaSafe QuID and bundled with QuantaKey QCU as a protected package, refer to FIG. 15.

Referring now to FIG. 4, at 352, the dicryption process 350 starts. For example, the Packager/unPackager system 110 of the digital asset protection system 100 (shown in FIG. 1) can receive the QCU 120 (also shown in FIG. 1) as input, along with the QEs 122 (also shown in FIG. 1), which can optionally be bundled in a QuID bundle 124. The QCU 120 and the QEs 122, for example, can together be used to restore one or more data units (e.g., including files, records, application code, or other sorts of digital information) that have previously been incrypted (e.g., using the incryption process 300, shown in FIG. 3).

At 354, a QCU 120 is decrypted. For example, the unPackager 114 of the Packager/unPackager system 110 can decrypt the QCU 120 (shown in FIG. 1), using one or more standard and/or non-standard decryption techniques. The unPackager 114, for example, can receive information that pertains to the encryption technique(s) that were previously used to encrypt the QCU 120, and can decrypt the QCU 120 based on the received information.

At 356, a determination is performed of whether Quantum Incrypted Data (QuID) bundling is enabled (e.g., based on a configuration setting of the Packager/unPackager system 110 and/or information included in the QCU 120). If QuID bundling is enabled, at 358, a QuID bundle is decrypted and a set of quantum elements (QEs) is extracted. For example, the unPackager 114 of the Packager/unPackager system 110 can decrypt the QuID bundle and extract the QEs 122 (e.g., a batch of files or other data elements).

At 360, a determination is performed of whether there is more data to unpackage (e.g., either in response to determining that QuID bundling is not enabled, or in response to determining that QuID bundling is enabled and upon decrypting the QuID bundle 124 and extracting the QEs 122). During the dicryption process 350, for example, the unPackager 114 of the Packager/unPackager system 110 can track a current status of the process 350 over time, including an indication of which batches of QEs 122 (e.g., files or other data elements) have already been dicrypted, and an indication of which batches of QEs 122 have yet to be dicrypted. In the present example, as none of the batches of QEs 122 have yet been dicrypted, the process 350 can proceed.

At 362, a batch of QEs 122 is read for a next file or chunk to dicrypt. For example, the unPackager 114 of the Packager/unPackager system 110 can reference its internal tracking of the current status of the decryption process 350 and can select and read a next batch of QEs 122 (e.g., a batch of QEs for a next file or chunk) for dicryption. In general, the dicryption process 350 can proceed iteratively through batches of the QEs 122, with a first batch of the QEs 122 being dicrypted first, with a second batch of the QEs 122 being dicrypted second, and so forth, until all batches of the QEs 122 have been dicrypted. The batches of QEs, for example, can correspond to the sets of QEs 122 that were previously generated through incrypting the original data assets 102, and can be identified by the unPackager 114 from information included in the corresponding QCU 120 (e.g., the unique identifiers that have been maintained for each QE 122 in association with each incrypted file or chunk).

At 364, a next QHC 120 input is initialized, based on the QEs 122. For example, the unPackager 114 of the Packager/unPackager system 110 can provide the batch of QEs 122 for the next file or chunk that was previously read (at 362) as the next QHC input to a process that reverses the operations specified in the QHC stack 108. In general, to reverse the operations of the QHC stack, the unPackager 114 can, in reverse sequential order according to the random order of ciphers in which a portion of data was incrypted, perform operations that reverse the effect of each cipher in the QHC stack. For example, the effect of a last cipher on the QEs 122 can be reversed first, with an output of the last cipher reversal being passed to an operation for reversing the second to last cipher, and so forth, until the effect of each of the ciphers in the QHC stack has been reversed and the data file or chunk is dicrypted.

At 366, QHC parameters are initialized from a QCU data structure 120. For example, the unPackager 114 of the Packager/unPackager system 110 can reference information in the QCU

120 that pertains to the QHC stack 108 that was used to encrypt the file or chunk (e.g., including the ordered list of ciphers and parameters for each cipher), and can identify the processing parameters that were used when encrypting data using each of the ciphers). As the decryption process 350 progresses, for example, the unPackager 114 can apply the parameters that were previously used for executing each cipher, when performing a reversal of the cipher.

At 368, an individual cipher is executed in a way such that its effect is reversed. For example, the unPackager 114 of the Packager/unPackager system 110 can perform operations that reverse the individual cipher and can track a progress of a decryption process for the previously read QEs (e.g., the QEs that were read at 362), including a reference to a position in the QHC stack of a cipher that is currently being reversed. In the present example, the effect of the last cipher in the QHC stack on the QEs can be reversed first, by executing a reversal cipher that corresponds to the last cipher. Executing the reversal cipher, for example, can include executing the reversal cipher code according to the processing parameters that were used when previously applying the cipher during encryption.

At 370, a determination is performed of whether there are additional ciphers to execute (such that the effects are reversed). For example, the unPackager 114 of the Packager/unPackager system 110 can reference the position of the previously reversed cipher in the QHC stack 108 and can determine whether any preceding ciphers exist in the QHC stack 108 (e.g., ciphers that were applied during encryption before the cipher that has just been reversed), according to the current position and the order of ciphers in the QHC stack 108. In the present example, the unPackager 114 determines that several preceding ciphers exist in the QHC stack 108.

At 372, if there are additional ciphers to execute (reverse), a next QHC input is initialized, based on previous QHC output. In the present example, the unPackager 114 of the Packager/unPackager system 110 can initialize the next QHC input based on the output that results from the reversal of the last cipher in the QHC stack 108. Operations for decrypting the QEs 122 can iteratively continue, with the QHC parameters for the next individual cipher in the QHC stack 108 according to a reversed order of the stack being initialized (at 366), the next individual cipher being executed in a way such that the effect of the cipher is reversed (at 368), and the determination of whether additional ciphers exist being performed (at 370), until all of the ciphers in the QHC stack 108 have been reversed, starting from the last cipher and ending with the first cipher that were originally used to encrypt the file or chunk.

At 374, if there are no additional ciphers to execute, a final output is written to a configured location. For example, the unPackager 114 of the Packager/unPackager system 110 can write the decrypted file or chunk to a volatile memory location where additional files are appended as the dicryption process 350 continues. In the present example, the dicryption process 350 loops back to 360, where another determination is made of whether there is more data to unpackage. If so (e.g., one or more batches of QEs remain packaged), the next batch of QEs 122 can be read (at 362) and can be independently encrypted. Dicryption of the next batch of QEs 122, for example, can be performed by reversing the effect of the randomized QHC stack 108 that was used to encrypt the QEs 122, which can be different from the previously reversed QHC stack 108. The process 350 iterates thusly, until all of the batches of QEs 122 have been dicrypted.

At 376, if there is not more data to unpackage, the dicryption process 350 ends. For example, the unPackager 114 of the Packager/unPackager system 110 can assemble the results of decrypting each batch of QEs and can provide the final result of the process 350 as a restored copy of the set of original digital assets 102.

Configurations

Referring again to FIG. 1, the computer-implemented digital asset protection system 100 can be implemented in various configurations and/or instantiations. Some of these are shown in FIG. 1 for Vault 140, Transit 142, including different modes for Comms 142-a, Transfer 142-b, and Agent 142-c, Storage 144, and Lite 146. As shown schematically in FIG. 16, these different configurations can include different features and options directed to different computer system and digital asset configurations and/or instantiations. In general, these configurations and/or instantiations include the core features of the digital asset protection system 100 and may also have add or drop features or have variations. It will be understood that various hybrid or combinations of these configurations and/or instantiations may also be implemented or utilized.

Vault 140 is generally directed to hardened quantum cyber resilience protection of static digital assets stored in long-term non-volatile memory that may not be secure where the digital assets are only periodically updated or changed (e.g., embedded microcode for computer systems potentially susceptible to capture and reverse engineering of the microcode). Transit 142 is generally directed to quantum cyber resilience of digital assets in transit where the digital assets are potentially subject to snooping or capture on computer network channels that may be unsecure

(e.g., transmissions of wireless or internet networks). Storage 146 is generally directed to quantum cyber resilience protection of dynamic digital assets in non-volatile memory in a presumably secure facility where the digital assets are frequently or continually updated or changed (e.g., networked server storage or RAID servers).

Vault

The Vault configuration 140 is a digital data storage product that can be used to protect data objects/assets such as firmware, dataset(s), and other CPI. Vault offers exceptional cyber resilience against data theft by storing sensitive information in an encrypted state that cannot be hacked. In some embodiments, the Vault technology uses the power of quantum entropy randomization, combined with proprietary processes and QHC stacks to provide state-of-the-art quantum cyber resilience protection of static digital assets stored in long-term non-volatile memory that may not be secure where the digital assets are only periodically updated or changed.

Electronic information in the form of digital data/assets can be protected and deployed to remote nodes securely. A QCU as the unique amalgamated codex/cryptex is used to incrypt a corresponding QuID and both are packaged to create the set of encrypted data objects being protected. The set of one or more QCUs and the corresponding set of one or more QuIDs are then physically separated once deployed to a target server (node) of a system. The encrypted data in the QuIDs is stored in a non-volatile memory and is protected through 'render useless' tactics at cold rest, ensuring the security and integrity of the information even if the system is compromised. The amalgamated codex in the QCUs is stored in a secured computer storage or a secured removable drive and only distributed to nodes in the system to enable access and/or installation of the incrypted data in the QuIDs. Incrypted data is unpackaged only into volatile memory and used for specific operations before being discarded. In the event a system is captured or compromised, Vault features a Software Anti-Tamper (SAT) solution that guarantees all data stored in volatile memory is lost and/or purged to prevent unauthorized access to the sensitive information.

Features of Vault

- Quantum resilience data protection in cold storage.
- Physical separation between QuID (locked/incrypted data/protected package) and QCU (locked key ring/amalgamated codex/cryptex).

- Transmission of the QCU is performed over encrypted TCP to unlock the QuID.
- Versatile software-only design allows Vault to operate within various network topologies.
- Could be used in conjunction with Storage.
- Quantum Resilience
 - Randomized QHC stack and processes unique to each Packager execution.
 - Volatile memory working area for unpackaged sensitive information; cleared when power is removed.
 - unPackager binaries that control the QHCs dicryption process are stored in a physically separate location from the corresponding QuIDs and transferred to any secondary nodes at unpackaging time. This sensitive code is stored only in volatile memory and immediately discarded after unpackaging.
- Anti-Tamper
 - The QCU is tightly coupled to the associated unPackager binaries via a unique fingerprint, resulting in unpackaging failure if an invalid QCU fingerprint is detected in the unPackager binaries.
 - Attempts to modify or tamper with the QuIDs, QCUs, or unPackager binaries will be detected and cause subsequent unpackaging attempts to fail.
 - Option to have QuIDs and QCUs “shuffled” after each execution so stale copies of QuIDs and/or QCUs are invalidated, making replay attacks infeasible and providing extra security for boot operations.

Use Cases for Vault

- Standalone Primary: Any data can be protected with Vault if separation is maintained between a QuID and the corresponding QCU, and the QCU is kept secure. The simplest use case is a standalone primary node that has a QuID in non-volatile memory, and a QCU on a secure external device acting as Primary System Storage (PSS). The secure device is plugged into the primary node, the QuID is dicrypted using the QCU and unpackaged into volatile memory, then the system continues to boot and/or use the sensitive data as needed.

- **Primary-Secondary:** A distribution package (containing a QuID) is deployed to non-volatile memory on a secondary node. Additional distribution packages can be deployed to as many secondary nodes as needed. The corresponding QCU or QCU's are deployed to a PSS (ideally some form of removable media that can be stored securely when not in use). The primary node acts as the "head of the snake" - without the primary node, none of the secondary nodes can unpack their QuIDs. As secondary nodes boot, each retrieves its QCU from PSS via secure TCP, each uses the retrieved QCU to decrypt and unpack the corresponding QuID into volatile memory, then each continues to boot and/or use the sensitive data as needed.
- **Bridging:** Systems that do not allow direct communication between secondary and primary nodes can use bridge node(s) to facilitate QCU/unPackager binary retrieval.

Storage

The Storage configuration 144 provides secure network storage for data-at-rest and fortified network communications for data-in-transit. The system's primary server/node is responsible for encrypting sensitive data, which is then broken into pieces and distributed across multiple secondary servers/nodes in the Storage network. Distributing encrypted data to the networked servers/nodes results in random pieces of the full dataset being physically separated and stored on random servers/nodes, so sensitive information remains safe even if a hacker manages to gain access to or hijacks some of the servers/nodes. The Storage system includes a software-only RAID capability specifically designed to provide resilience against ransomware. In some embodiments, the system has a quantum cyber resilience capable of maintaining operations even if up to 40% of the network servers/nodes are compromised by ransomware attacks.

Features of Storage

- Fortified network communications for data-in-transit to networked data-at-rest servers/nodes.
- Adjustable system resilience based on network server/node configuration.
- Software-only RAID10 capability with per-distribution dynamic mirrors.
- Hardware fault tolerance via data redundancy.

- Stored information is protected against ransomware attacks, data breaches, insider threats, and other forms of malicious cyber activities.
- Data-at-rest protection with physical separation between QuID and QCU data objects.
- Memory-only primary node avoids writing QuID or QCU to disk before distribution, improving both security and performance characteristics.
- Simple software installation via an Application Program Interface (API) that supports Distribute (create), Retrieve (read), and Purge (delete) operations to manage encrypted data stored in Storage.
- Keys and metadata are managed automatically, so the user only needs to specify the correct Global Universal Identification (GUID) when making API requests.

Use Cases

- Alternative to traditional long-term persistent storage solutions.
- Can be used in conjunction with Vault and/or Transfer.
- Can be used to store/maintain any database that contains sensitive information.
- Can be installed to support a variety of access levels, including:
- Public-facing cloud storage service.
- VPN/Firewall restricted service, accessible only to users within a specific organization.
- Installed on a local network (LAN) and only available to users with physical access to the network.

Transit

Comms

The Comms configuration mode 142-a provides a secure real time communication channel between two or more parties. Every message is protected to provide state-of-the-art quantum cyber resilience protection for real time digital communications using an optimized set of QHCs that combine exceptional performance with industry leading security capabilities. Digital data is protected by incryption with a unique amalgamated codex/cryptex, referred to as a QCU, and a corresponding QuID created as a protected package for the incrypted data. Data is also protected

in transit with fortified network communication. Comms also uses a Packager/UnPackager system that incorporates a unique Quantum Symmetric Secret (QSS) generated by an RNG or QRNG to bolster security and defend against man-in-the-middle attacks.

This configuration mode facilitates low-latency Point-to-Point or Point-to-Multipoint communications. It is designed to be impervious to any tools or brute force methods that hackers might use to decipher or unlock the QCU/QuID, which are available on the dark web. Offering double-layered protection, it secures the transfer of data over unsecured channels. This system is especially effective in providing real-time protection for conversations, messages, and emails. In cases where data is intercepted, the encrypted data remains incomprehensible, appearing as a 'sea of nothingness' to unauthorized interceptors.

Features

- Protection for real time data in transit.
- Low-latency secure communications.
- Scalable bandwidth via multi-socket capability.
- Versatile design allowing centralized or point-to-point communication.
- Multi-layer security provided by a non-deterministic incryption process operating within state-of-the-art public key cryptography.
- QSS ensures only related nodes (same QSS) can communicate with each other and also provides interception security.
- QuID is processed and transmitted without ever needing to be written to non-volatile memory.

Use Cases

- Email.
- Audio streaming.
- Sensor logging updates.
- Text messaging, including small media attachments.
- Real-time communications of relatively small datasets (<1Mb).

Transfer

The Transfer configuration mode 142-b facilitates the transfer of large datasets for non-real time scenarios. It is designed to be resilient against data interception, ensuring that even the most sophisticated brute force methods or tools available on the dark web are ineffective. Every transfer is protected by state-of-the-art quantum cyber resilience protection for real time digital communications using an optimized set of QHCs that combine exceptional performance with industry leading security capabilities. Digital data is protected by incryption with a unique amalgamated codex/cryptex, referred to as a QCU, and a corresponding QuID created as a protected package for the incrypted data. Transfer features a Packager/UnPackager system that employs a unique Quantum Symmetric Secret (QSS) generated by an RNG or QRNG for enhanced security. Both Point-to-Point and Point-to-Multipoint file sharing are supported.

This configuration mode offers a dual layer of protection for the transfer of data. The first layer of protection packages all sensitive data into a QuID/QCU pair and blends them together to achieve a single bundle only separable by a party's privy to the QSS. In the event of interception, the combined pair appears as an indecipherable "sea of nothingness". The second layer of security uses state-of-the-art public key cryptography to guarantee privacy while the combined pair is in transit. This allows Transfer to be used in a wide range of operating environments, including over unsecured networks. This deployment flexibility coupled with quantum cyber resilience protection of data integrity effectively eliminates the need for physical hand couriers.

Features

- Protection for non-real time data in transit.
- Large dataset support.
- Scalable bandwidth via multi-socket capability.
- Versatile design allowing centralized or point-to-point transfers.
- QSS ensures only related nodes (same QSS) can communicate with each other and provides interception security.
- Multi-layer security provided by a non-deterministic incryption process operating within state-of-the-art public key cryptography.
- Non-volatile memory only QCU.

- QuID is processed and transmitted without ever needing to be written to non-volatile memory, although QuID can be unpackaged to non-volatile memory by the receiver, depending on data size and intended use.
- Non-real-time, allowing more sophisticated QHCs to be used for maximum security.

Use Cases

- Video streaming.
- Transfer database dump file.
- Send media from a remotely controlled vehicle to a base station upon returning to origin.
- Non-real-time communications of relatively large datasets (>5Mb).

Quantum Hybrid Ciphers

In some embodiments, a minimal set of incryption protocols and techniques representing the ciphers in the QHC stack 108 used by the Packager system 112 to protect the set of original digital assets 102 as a QuID 124 includes:

- An obfuscation cipher that fragments each file into fragments each being a QE 122, injecting each QE 122 with a set of random noise,
- One or more protection ciphers that encrypts each QE 122 using a standard encryption protocol protection cipher and a random key or seed, and at least one non-standard encryption protection cipher, and
- An obfuscation cipher that shuffles the set of QEs 122 into a random order such that the set of QEs 122 is configured to be stored in a non-volatile memory as a QuID 124 for the set of original digital assets 102.

In some embodiments, the set of protocols used by the Packager system 112 to create the QuID 124 further include a protocol order for the set of protocols that is variable. In other embodiments, the set of protocols further includes adding a set of decoy files to the set of original digital assets 102.

In embodiments, the incryption process of the Packager/unPackager system 110 uses a random set of ciphers that are executed in random sequence, which is referred to as the “cipher

stack” of QHCs 108. Security of the cipher stack is primarily based on the security of each cipher in the stack. The QHC ciphers are analyzed individually so that the security contribution of each can be quantified. In embodiments, the available QHC ciphers provide a variety of different operations that are combined in a cipher stack to protect digital assets 102. There are two main cipher types:

- **Protection Ciphers:** These ciphers are designed to directly protect (i.e. encrypt) the input data. Various embodiments can use standard ciphers such as AES256, as well as non-standard ciphers that apply various data transformations.
- **Obfuscation Ciphers:** These ciphers are designed to add complexity to the packaged data by injecting decoy data or chunks of noise. There are also ciphers that can blend data together or split data apart to allow protected packages 104 to appear as a “sea of nothing” to an attacker. Note that many of these ciphers do not individually contribute a large number of bits of security, but they provide desirable properties when combined.

An embodiment of the stack of QHCs 108 can include the following ciphers:

- **Protection – Standard Encryption Ciphers:** An encryption cipher configured to encrypt the set of incrypted data.
 - AES256-GCM
 - ChaCha20-Poly1305
- **Protection Non-standard Ciphers**
 - **Permutation:** a permutation cipher configured to transform bits in the set of incrypted data.
 - **Bit Shift/Mask cipher:** a cipher configured to rotate 4 byte blocks and applies an XOR operation using a randomly chosen value.
 - **Substitution cipher** configured to substitute bits in the set of incrypted data.
- **Obfuscation Ciphers**
 - **Sprinkling:** Packager applies each character of a QSS to random locations within a QuID ciphertext (injection locations are dynamically determined to avoid having to store in the QCU). unPackager collects QSS characters from the QuID ciphertext,

compares against configured QSS value, and only reconstitutes protected data if extracted and configured QSS match.

- Fragmentation: a fragmentation cipher configured to fragment the set of incrypted data.
- Noise
 - Padding: an augmentation cipher configured to pad the set of incrypted data to the random size.
 - Decoys: a fabrication cipher configured to add a set of decoy data to the set of incrypted data.
 - Injection: an injection cipher configured to inject a set of random noise into the set of incrypted data.
 - Multi-Chunk Noise (larger pieces of random noise)
 - Multi-Character Noise (smaller pieces of random noise)
- Shuffle: a randomization cipher configured to shuffle the set of QE incrypted data.

In other embodiments, the stack of QHCs 108 can include subsets of randomly or quasi-randomly selected and ordered.

In some embodiments as indicated in the table below, quantum encryption includes a variable combination and order of encryptions and ciphers, together with a static combination and order of quantum protection. In some embodiments, the stack of QHCs 108 can include a subset of quasi-randomly selected and ordered of the following protection and obfuscation ciphers for both a real-time embodiment and a non-real-time embodiment as listed in the Table below. Note that the random selection of available ciphers is made as a selection of a subset of the indicated ciphers, with the order as indicated in the table and with the Decoy Cipher and the Fragmentation Cipher not being a randomly selected cipher:

Table: QHC Process Ordering

(Real-time Embodiment)	(Non-Real-time Embodiments)
AES256 and 1 Protection Cipher	AES256 and 1-2 Protection Ciphers
Decoy Cipher	
	Shuffle Cipher
Fragmentation Cipher	
	1 - 2 Protection Ciphers
1 Noise Cipher	1 - 2 Noise Ciphers
	Shuffle Cipher

Digital Asset Protection Features

In various embodiments, each QE 122 has a random size that exceeds a threshold size with a total number of QEs 122 for each QuID 124 exceeding a threshold number. In various embodiments, each QE 122 has an element name that is a random string of alphanumeric characters having a common length. In various embodiments, the set of random noise for each QE 122 is a set of random sizes of noise at a set of random locations in the QE 122 whereby decryption of the QE 122 would produce corrupted data if the set of random noise is not correctly removed before decryption/decryption. In certain embodiments, the quantity of random noise added is determined by a randomly selected amount within a range of noise sizes, or as a proportion of the overall size of the protected package104, the QuID 124, or the given QE 122.

In embodiments, the Packager system 112 creates a QCU 120 for the QuID 124 having a quantum element map (QEM) and a quantum cryptography map (QCM). The QEM includes for each QE 122 and the corresponding quantum element name for a QE 122 together with the random order and random size of that QE 122 and the set of random sizes and the set of random locations of the set of random noise in that QE 122. The QCM includes the random key or seed and a nonce

value and for each file of the set of original digital assets 102 a set of element names of the QEs 122 that correspond to each of the fragments of that file together with an original file name and a set of metadata for that file. In some embodiments, the QEM and the QCM are created as separate files or data objects within the QCU 120. In other embodiments, the information for mapping elements and correlating element names to file names represented by the QEM and QCM is integrated into the QCU 120 as one or more sets of manipulated strings of information that may be merged or integrated into a single data object.

In some embodiments, an element name is a File Name Length (i.e. length of the original file stem + extension) that is used to extract the original file name from the decrypted ciphertext. A file name (ex: test.txt) is a concatenation of a file stem (ex: test) and a file extension (ex: .txt). An Output File Stem is a set of random alpha-numeric characters used to name the generated ciphertext and key/metadata files for each original file being protected. The original file name is prepended to the original file data before encrypting/incrypting (and extracted after dicrypting/decrypting) to avoid including the original file name in human readable format. The ciphertext file stems generated during encryption (i.e. Output File Stem) are reflected in the QEM as the File Stem.

In various embodiments, a QCM may include one or more of the following:

- Key
- Nonce
- File name length
- Key length
- Nonce length
- Message length
- Additional data
- Ciphertext length
- Output file stem
- Additional data length
- File decoy flag

In various embodiments, a QEM may include one or more of the following for each file in an original set of files:

- File stem

- File decoy flag
- File noise length
- File noise location
- File size
- Array of QE metadata, with each QE metadata containing:
 - QE identifier
 - QE order
 - QE noise length
 - QE noise location
 - QE decoy flag
 - Array of QE locations

In various embodiments the digital asset protection system 100 utilizes the entropy source 106 to provide true quantum entropy randomization. In some embodiments, the entropy source 106 is a software-only quantum entropy source as discussed herein with respect to FIG. 9. In other embodiments, the entropy source 106 captures photons as quantum particles and translates a quantum event into a binary code that is used to create a quantum random key. In some embodiments, the entropy source 106 is physically within a secure computing environment or is connected to the secure computing environment by a secure network connection. In some embodiments, the secure network connection uses transmission control protocol and a double ratchet encryption.

FIG. 9 shows a high-level schematic diagram of an embodiment of the digital asset protection system. At this high level, the system starts with a true quantum random seed from entropy source 106 that is paired with an entropy extractor. In one embodiment, the quantum entropy source for the QRNG is a software-only quantum random seed source such as the QuantinumTM) Origin Onboard seed licensed from Quantinuum Ltd, as generally described at www.quantinuum.com/cybersecurity/quantumoriginonboard and in U.S. Patent No. U.S. 11,256,477 and U.S. Publ. Patent Appl. No 2023/0291555. The Quantum Origin software-only approach provides a randomness source that has been mathematically shown to be equivalent to a true quantum source. It provides an unparalleled level of verifiable randomness, eliminates any

hardware tether, and enables embodiments of the digital asset protections system of the present disclosure to be installed on any computer system with no hardware or external reach back needed.

In another embodiment, the entropy source 106 is sourced from a quantum random generator photon card that is either directed connected to a computer system executing the Packager/unPackager system 110, or network connected to the computer system executing the Packager/unPackager system 110 over a secure network, or sent to the computer system executing the Packager/unPackager system 110 with double ratchet encryption or using a secure comms embodiment of the digital asset protection system 100 of the present disclosure. In some embodiments the quantum random seeds/key are also fed into some or all the QHC processes in a QHC stack 108. In embodiments utilizing the software-only QRNG entropy source 106 compared to a hardware tethered photon noise QRNG entropy source 106, the software-only QRNG entropy source 106 can significantly reduce latency for the Packager/unPackager system while increasing overall efficacy. In some embodiments the QRNG seeds are also fed into some or all the QHC processes in a QHC stack 108.

In embodiments, the digital asset protection system 100 as disclosed herein is entropy-agile, meaning that one or more active entropy source providers can be specified at compile time. In embodiments, entropy source providers can either be QRNG or pseudo random number generator. True random number generation is important in cryptography applications (specifically in the area of key generation) to avoid exposing predictable patterns to attackers. Some of the best available random number generators are QRNGs based on quantum technology, such as Quantis™, Quintessence Labs™, and Quantinuum®. Quantis is a PCIe hardware cards solution, Quintessence Labs offers both a PCIe hardware cards solution and qStream™ a Entropy-as-a-Service (EAAS) web call, while Quantinuum Quantum Origin™ is a software-only solution.

In embodiments, Pseudo Random Number Generators and pseudo random entropy may be used as a fallback for QRNG failure, or for certain cases where true QRNG is not strictly necessary or where potentially cheaper options are desirable. There are numerous pseudo random generators available, including several being built into modern operating systems. In embodiments, the digital asset protection system 100 as disclosed herein may leverage pseudo random generation available from the cryptography libraries discussed below:

- Quantis - The Quantis dependency consists of a single static precompiled header file, in addition to a Dynamic Linked Library (DLL) or Shared Object (SO) file. More

information about this QRNG is available at www.idquantique.com/random-number-generation/products/quantis-random-number-generator/.

- Quintessence Labs - The Quintessence dependency consists of static header files, as well as separate so-called “stream services” that must be installed on the system and started to allow access to random data from the PCIe hardware card. More information about this QRNG is available at www.quintessencelabs.com/news/quintessencelabs-qstream-entropy-as-a-service-eaas-solution-delivers-truly-random-numbers-for-encryption-keys.
- Quantinuum - the Quantinuum dependency can be either Command Line Interface (CLI) or static header. To access entropy from the CLI, code for embodiment of system 100 must make a system call to the CLI and retrieve the process output. More information about this QRNG is available at www.quantinuum.com/cybersecurity/quantumorigin.

FIG. 10 shows a schematic diagram of one embodiment of a randomized stack QHCs 108 that provides cryptographic agility. In this instance of randomly ordered ciphers cipher₀ is shown as a standard NIST-based cipher within the QHC stack. This optional standard NIST-based cipher relies on standard algorithms in line with NIST encryption standards, such as AES. Because cipher₀ adheres to NIST standards, the complete QHC process meets all cryptographic compliance requirements. Unlike cipher₀ which has a fixed efficacy, some of the other QHC non-standard protection/obfuscation ciphers 1-N (108-a, 108-b, to 108-c) have a respective ‘slider-bar’ that controls a variable cipher efficacy (CE) from a minimum CE_{MIN} to a maximum CE_{MAX}. This CE can be tuned to dictate the overall latency, file size, and combined security strength efficacy of an embodiment of the QHC stack 108.

In various embodiments, baseline latency testing for a given network configuration via ping tests at the Presentation Layer (OSI 6) can be utilized to identify an average latency to be used for variable CE tuning. In various embodiments, the average latency can represent a one-time measurement, a single average over a period, or a dynamic average over one or more periods. In some embodiments, an automated process included, for example, as a selectively callable routine in the Packager/unPackager system 110 can be provided for tuning each of ‘slider-bars’ (109-a, 109-b, and 109-c) from -CE to +CE. In other embodiments, a one-time or recurring process can be utilized to adjust the QHC non-standard protection/obfuscation ciphers 1-N ‘slider-bars’ (109-a,

109-b, and 109-c) like the way an equalizer for a sound system is balanced for a given environment and type of music, with the optimized settings then hardcoded into the code for the Packager/unPackager system 110. In some embodiments, an automatic process can be utilized to dynamically adjust the slider-bars (109-a, 109-b, and 109-c) to target certain selective parameters, such as a target effective k or a target latency for incryption/dicryption, or a target network bandwidth for a desired set of digital assets in a given computing environment.

In embodiments, in addition to a tunable variable CE for one or more of the QHCs 108, the total number (N) of all the QHCs 108 in a stack and an order in which some or all of the ciphers are applied is randomized to further enhance the strength and cryptographic agility of the QHC stack 108.

In embodiments, the QCU 120 and QuID 124 pairs are tightly coupled to their associated unique Packager/unPackager system 110. In various embodiments, the Packager/unPackager 110 can be installed and executed as part of the operating system kernel or as part of a boot process, as a separate application program stored in the non-volatile memory of a given computer system under control of the operating system, or as an executable program that can be periodically downloaded to and non-volatile memory and then executed by the given computer system. In embodiments, the Packager/unPackager 110 can be fingerprinted to a given computer system or computer network such that the Packager/unPackager 110 is effectively tied to a given hardware, processor, or network environment. In embodiments, the Packager/unPackager 110 is a compiled binary executable code program that includes the code for each of QHC of the stack of QHCs 108 as a separably callable software routine executed in a sequence determined by a control routine. In embodiments, a random sequence of the control routine is hard coded into a control routine in the Packager 112 and unPackager 114 system(s). In other embodiments, the random sequence is dictated by parameters and/or meta data that may be included in the QCU 120 and/or in some embodiments may be randomized based on the entropy source 106.

In embodiments, the Packager/unPackager 110 can also utilize a separate Quantum Symmetric Secret (QSS) 118. In some embodiments, the QSS 118 is produced based on the entropy source 106. The QSS 118 bears resemblance to a symmetric encryption key, yet it differs from conventional cryptographic symmetric keys by the way it is conveyed, applied and/or used by the QHCs 108 and the Packager/unPackager 110. In some embodiments, a QSS 118 is added to reduce the risk of transmitting a QCU 120 simultaneously with an associated QuID 124. If the

communications are intercepted by a nefarious actor, and the reconstitution algorithm becomes known, the QuID 124 would potentially be at risk if AES encryption on the QCU is broken. Having an additional layer of security (i.e. the QSS 118) leaves the QuID 124 corrupted even if an AES encryption is broken if the QSS 118 is unknown.

In some embodiments such as Comms 142-a as described in FIG. 11, the QSS is implemented by updating the Packager 112 to apply each character of QSS 118 to random locations within the QuID 124. In various embodiments, the Packager/unPackager system 110 can dynamically calculate injection locations to avoid having to store the QSS 118 in the QCU 120. The unPackager 114 can be updated to collect QSS 118 characters from the QuID 124, compare against injected QSS 118 values, and only reconstitute protected data if the extracted and injected QSS's 118 match. If the Receiver node 172 corresponding to the Transmitter node 170 that protected the intercepted data is stolen, it could potentially be used to reconstitute the data despite using a QSS 118. However, compromise of a Receiver node 172 would only affect that particular set of nodes configured with the same QSS 118, and periodic updating of the Packager/unPackager system 110 on the Receiver nodes 172 can address this potential concern.

In other embodiments, such as Vault 140 as described in FIG. 6, the QSS 118 can be used to improve overall security by directly coupling a QuID 124 to a QSS 118 injected into the QHC binary file 116 for a given unPackager system 114. In various embodiments, this may be done in a manner similar to how the QCU fingerprint ties a QCU 120 to a QHC binary file 116 for a given unPackager system 114. In various embodiments, aspects of the Packager/unPackager system 110 can be fingerprinted to be executed only on a given target computer or node using system license keys, token keys or machine IDs, and/or system IDs injected into the executable binaries. Examples of fingerprinting techniques as used in the field of computers, see [en.wikipedia.org/wiki/Fingerprint_\(computing\)](https://en.wikipedia.org/wiki/Fingerprint_(computing)).

FIG. 8 is a dependency diagram of an embodiment of the digital asset protection system 100 as disclosed herein. In various embodiments, the code for the digital asset protection system is developed to have minimal third-party dependencies as security of code produced by third parties is more difficult to analyze and guarantee security compared to custom code, and custom code provides more complete control over design and implementation details, allowing for more precise control of features and capabilities. Even so, there may still be several third-party dependencies such as those facilitating QRNG hardware/software interfacing, standard

cryptographic capabilities, Windows-specific services such as WinSock TCP, and miscellaneous others for C++ logging, JSON, HTTP, and code packing support which may be useful to integrate.

In embodiments, some third-party software dependencies (e.g., Quantis, Libcurl) have dynamic linking requirements, which means that even if a particular library is not in use due to configuration variations, the dynamic linking must still take place during program initialization. Dynamic linking is one reason for requiring certain dependencies to be specified at compile time instead of run time. Another benefit of compile-time dependency specification is that only code actually in use is included in the compiled binary, which reduces the overall attack surface area.

In various embodiments, these dependencies provide Application Programming Interfaces (APIs) for access to standard encryption/decryption capabilities, as well as hashing, key derivation, and pseudo random number generation, among other things. In various embodiments, system 100 is crypto-agile, meaning that the active cryptography source provider can be specified at compile time.

- Libsodium - Libsodium provides a static header and excellent documentation for relatively easy implementation and use. However, it is not currently FIPS140 compliant, so alternative cryptography source providers have also been implemented.
- Cryptopp - Alternative to Libsodium, providing additional alternative encryption methods. This library is also currently not FIPS140 compliant.
- OpenSSL - FIPS140-compliant library providing necessary cryptography capabilities described in the introduction to this section.
- Windows Dependencies - Some capabilities such as TCP socket communications and pseudo random number generation depend on dynamic libraries available in the Windows operating system. These capabilities are available by default in Unix operating systems.
- Dokan - This dependency is a Windows driver application facilitating mounting/unmounting a drive as a RAM filesystem (RAMFS). In embodiments, RAMFS is used to contain sensitive information that is cleared automatically or manually without residue remaining anywhere in non-volatile memory.
- WinSock - Enables use of Windows TCP socket capabilities necessary for embodiment of system 100 inter-node and external stream service communications.

Vault

Referring now to FIG. 6, embodiments of the Vault configuration 140 are described. Unlike current encryption techniques that might use such a quantum random key for a known quantum-resistant encryption algorithm, embodiments of the present disclosure integrate variable and multi-faceted incryption protocols in building a protected package 104 for the set of digital assets 102 such that there can be no reassembly of the QuIDs (e.g., QuIDs 124 or QCU/QuID stack 128) without the corresponding QCUs (e.g., QCU 120 or QCU/QuID stack 128).

In some embodiments, Vault 140 is configured to create, store, and distribute the digital assets 102 to each computing node of a target computing environment. In some embodiments, the Vault 140 runs as an executable set of files on the secure computing node(s). In other embodiments, Vault 140 is integrated into a BIOS of the secure computing node(s). In some embodiments, Vault 140 runs on an embedded computer system 160 with the digital assets 102 for each computing node protected as incrypted data in individual QuIDs 124 for a given computer node 164, 166.

In some embodiments, the separation between the QCU 120 and QuID 124 is further augmented by using an unPackager Stub App 117 that is configured to request an upload of a QHC binary 116 for a given order and selection of QHC ciphers in the QHC stack. In some of these embodiments, the QHC binary 116 is combined with the QCU 120 by a string merge operation to form a QCU+ 128 that contains both. In some embodiments (not shown in Fig. 4), the QHC binary 116 is separately transferred to and stored in the target computer systems 160 such that only the QCUs 120 are distributed to the main processors 164 and secondary processor 166 in the embedded computer system 160. In other embodiments, the target computer systems 160 may be servers/nodes on a networked computer system.

In embodiments, the unPackager system 114 for the primary microcontroller/secondary microcontrollers construct utilizes techniques in which a unique QuID 124 is unpackaged and distributed to each microcontroller device. For example, QuIDs 124 are unpackaged and installed as boot images in volatile memory in microcontrollers. In some embodiments, the unPackager system 114 has an embedded QEM/QCM as a fingerprint to link the unPackager 114 and the given microcontroller to the respective QuID files 124 protecting the original set of files as a boot image for that microcontroller. In some embodiments, the embedding of a fingerprint from the QCU 120 inside of the unPackager system 120 to coordinate distribution of protected packages individually to each of the nodes allows each node to be hot swappable capable. In some embodiments, this is

facilitated by having a QCU 120 in the same non-volatile memory device as the QuID 124. In some embodiments, the unPackager system 124 is embedded in microcode.

In some embodiments of an embedded system, a post-boot shuffle may be provided that includes shuffling order, update element names in the QEMs and update all fields in the QCMs that are used to correlate element names to original file names. In some embodiments, the shuffle is a dynamic randomization shuffle. In some embodiments, the set of original file names are directly related to the element identifiers. In other embodiments, the set of original files 102 are indirectly related to corresponding ciphertext and corresponding QCMs. In some embodiments, a correlation of the set of original file names is indirectly linked to/built based on key pairs of a file stem value and multi-step linking with other fields in the corresponding QCM.

Referring now to the flowchart of FIG. 7A, an embodiment of the end-to-end process of the Vault configuration 140 will be described. In this embodiment, the secured computing facility 150 is a development facility where sensitive data identification and protection takes place. Sensitive information must be identified for each node in the target system 160, and a distribution package is subsequently generated for each target node.

In Vault, packaging is accomplished standalone or as additional steps in a traditional build process. Once configured, the Packager binary can be called to transform input data into the protected package. The result of the Packager is one or more distribution packages depending on the use case. One distribution package is generated for each node in the target system. Generated files can be managed by external scripting as needed to prepare for deployment.

In embodiments, before placing distribution package(s) on portable secure storage device(s) for transport to the target computing system, testing should be performed at the Development Facility on a replica of the actual remote system 160 to ensure proper functionality.

In embodiments, a Primary System Storage (PSS) 154 can be either a secure removable storage device, or a Storage network. In these embodiments, the QSS 118 is transported to the remote facility, so in preparation for doing so the data must be placed on a secure storage device (could be the same device actually used at the remote facility if not using Storage). PSS is secured by way of password protection or other type of authentication mechanism (CAC card, biometrics, etc.)

In embodiments, the PSS 154 may be physically transported to a remote site and placed in secure storage (physical vault) in accordance with the security, access and/or secrecy protocols and standards of the user or entity operating the target computing system.

In embodiments, a protected package is placed on a portable storage device that can subsequently be used at the remote facility to deploy to the target device. In the Embedded Mode, for Existing Hardware (Firmware Update) protected data is placed on a portable secure storage device that can subsequently be used at the remote facility to deploy to the target device. For New Hardware, protected data is placed on disk images that are installed (EPROM'd) on the embedded device(s). These digital assets can subsequently be used at the remote facility to deploy to the target device.

In embodiments, after testing the development facility has a set of one or more distribution devices ready for transfer to the target node(s) at the remote facility. The devices, consisting of a secure PSS device, along with one or more target node devices must be physically transported to the remote facility. Confidentiality of the PSS device must be maintained to ensure security of the protected data on the target node devices. The PSS device must be stored in appropriate secure storage such as a vault or similar upon arrival at the remote facility.

In embodiments, deployment at a Remote Facility having the target computing system can be accomplished in various ways. As used in the context of Vault, the term “deployed” is equivalent to “transferred from a portable transport device to the target device”. In some embodiments, for a Primary node the PSS is kept in a secure physical vault at the Remote Facility. Before each use of the Primary node, the PSS must be removed from a vault or secure location and physically installed into the Primary node. If any sensitive data was protected for use on the Primary node, the protected data must be deployed to the Primary node. For a Secondary node, the deployment will differ depending upon the device and process being performed:

- Regular Device - Protected data is deployed to the target device.
- Embedded Device
 - Existing Hardware (Firmware Update) -A firmware update is performed to deploy the protected data to the target device.
 - New Hardware - Same steps as for Existing Hardware, but the steps can be performed either at the Development Facility, or the Remote Facility if desired/necessary.

In embodiments, an unPackager operation follows a similar but reversed process. After successful User authentication, the protected package must be unpackaged into original file format to enable access to the sensitive information. To ensure system security, the sensitive information is only ever placed in volatile memory.

- Primary node
 - Single Node - unPackager binary retrieves QCU from local PSS and unpackages protected data into volatile memory.
 - Multi Node - unPackager binary starts a TCP server and listens for connections from dependent nodes.
- Secondary node
 - Secondary node(s) wait for the primary system to become available. Once the Primary node is available, the Secondary node(s) are then able to retrieve their QCU(s) from remote PSS and unpackage protected data into volatile memory.

In some embodiments, Vault can be configured to perform a Post-Boot Shuffle. After a successful boot of a computing node, if configured to perform post-boot shuffle, the local QCU and QuID data is shuffled and the updated QCU is pushed to PSS, thereby invalidating stale copies of the QCU and QuID.

Embodiments as disclosed support both flat and hierarchical network architectures with one Primary node and 0-n Secondary nodes.

- Flat - A flat network is one where every node can communicate directly with every other node.
- Hierarchical - In a hierarchical network, in addition to Primary and Secondary nodes, there is also a third type of node called a Bridge node that forwards TCP message packets to the next node(s) in the hierarchy. For example, if a Secondary node needs to communicate with the Primary node, it sends a message to the closest upstream node; if this node is not the Primary node the message is forwarded to the closest upstream node until the Primary node is reached. If the Primary node must communicate with a Secondary node, the Bridge node(s) forward the message to the next downstream node until the Secondary node with system ID matching that of the message is found.

In some embodiments, a long-running connection to the upstream is maintained after successful unpackaging on all Secondary nodes. If upstream connection is lost and timeout threshold is enabled and reached, the volatile working area is automatically unmounted.

In various embodiments, features of Vault are provided to address potential Hardware Failure of one or more of the computing nodes.

- Primary node - The QCU in PSS houses all Primary and Secondary node QCU. If it is destroyed or becomes unusable in some way, all Primary and Secondary nodes must be re-deployed. For the Primary node, this would first require transport of a new PSS device from Development to Remote Facility.
- Secondary nodes must also be re-deployed, as their QCU data was located on the PSS device. Transport and re-deploy new deployment packages for all Secondary nodes.
- Secondary/Bridge nodes - Transport and re-deploy Secondary node distribution package. Protected data is deployed to the target device, and the associated QCU is deployed to the secure PSS device.

FIG. 7B shows a flow diagram of an embodiment of a TCP communication among Primary Nodes which generate/distribute protected packages, Bridging Nodes which convey or transfer but do not access a protected package, and Secondary Nodes which receive and utilize protected packages. In some embodiments, TCP/double ratchet encryption is used to transmit the QCU 120 from the primary node that has the RSD that stores the QCU 120 to the corresponding secondary node for that QCU. A description of how these kinds of encryptions operate can found at Perron, et al., “The Double Ratchet Algorithm,” (11/20/2016), available at signal.org/docs/specifications/doubleratchet/doubleratchet.pdf, and “Transmission Control Protocol,” available at en.wikipedia.org/wiki/Transmission_Control_Protocol.

Development Facility

The development facility is where sensitive data identification and protection takes place. Sensitive information must be identified for each node in the target system, and a distribution package is subsequently generated for each target node.

FIG. 7C shows a flow diagram of an embodiment of a Packager for the Vault embodiment. Vault protects data by creating a distribution package that is later deployed to a secondary node. When creating this distribution, the Packager is run on a development machine that can access the

digital assets to be protected. Figure 7C shows the process flow for creating this distribution package.

The packaging process is controlled by a configuration manager that provides an easy way for the user to control how the Vault system will be configured. Several of the options are:

- **Target OS / Architecture:** This information is used so that the correct form of the unPackager binary is included in the distribution.
- **Use Stub:** If the Use Stub option is true, two unPackager binaries will be included in the distribution. The first binary is a stub application that will be deployed on the secondary node and does not contain any code for processing QHCs. The QHC logic is fully contained in the second binary that will be deployed to the PSS and transmitted to the secondary node when unpackaging occurs.
- **Use HTTP:** This option indicates whether secondary nodes have a communication link to the internet. For some clients, the secondary nodes exist in an air gapped system and do not have connectivity beyond direct links to the local network.

The initial Packaging step can use a QRNG to randomly create a QHC stack that processes the original data object(s) to be protected. After the data has been encrypted by the QHC stack, the resulting set of QEs is bundled together, encrypted, and stored in the distribution area with a Stealthy String name that obfuscates the file type. In addition, the QCU is also encrypted at this point.

If the Stub option is enabled, a stub version of the unPackager application for the correct OS and architecture is copied to the distribution area. This binary is renamed to simply qsu and then injected with a QCU fingerprint so that an integrity check can be performed at the time of unpackaging to make sure that an attacker is not using an invalid binary for the target data assets. This injection area is protected by adding a checksum value that is also validated when the binary is executed.

The second binary in the Stub case is a version of unPackager that includes functionality for QHCs. This binary is then injected with a fingerprint and checksum value and then encrypted. Next, Packager will combine the resulting encrypted binary with the encrypted QCU into a single file using a blended strings functionality. This file is again encrypted and written into a PSS location inside the distribution area, using the System ID as the file name. When the distribution is deployed to the target secondary node, the contents of this directory will instead be moved to

the PSS location on the primary node. At runtime, the primary node will be responsible for decrypting, unblending, and decrypting the QCU and binary files to respond to requests from the secondary nodes.

If the Stub option is not enabled, the process is simplified because the full qsu binary can be added to the distribution area and deployed directly to the secondary node. In this case, the binary injection of the QCU fingerprint and checksum still occurs, but the resulting binary is not encrypted. To simplify the processing on the primary node, the encrypted QCU is still blended with an empty string, encrypted and named after the System ID.

In both the Stub and non-Stub options, separate binaries are available that do not include the dependencies for making external HTTP requests. Without these libraries being added, these alternative binaries are smaller.

The last step that Packager completes when building a distribution is to include additional dependencies that may need to be included. This may include assets such as dynamically linked library files, shared object files, executable files, driver files, or certificates. In some cases, these dependencies are conditional, based on the packaging options. For example, when HTTP is enabled, an OS-specific runtime library is included in the distribution area. In addition, when the target secondary node is configured for a Windows system, a set of dependencies are included that are needed to enable Vault to use a volatile memory for reading and writing data assets.

In some embodiments, the QCU 120 is stored a removable storage device (RSD) that is the only store for the QCU 120. In embodiments, the primary node can access the QCU 120 in the RSD and create a unique hash token for sending the QCU 120 to boot the QuID 124 based on the decrypted original set of files for that particular secondary node on the secondary node in response to a request from the secondary node.

The Vault product protects data at cold rest. Physical separation is maintained between protected data and the QCU required to unlock it. Without a QCU the protected data is rendered useless and impervious to brute force attacks, even from quantum computers. QCUs are retrieved from a Primary node by a Secondary node at runtime and used to reconstitute protected data into volatile memory. Bridging capability is also available to support hierarchical networks. The Vault can be used in any system composed of two or more nodes and is ideally suited for embedded systems where data residing on microcontrollers or other embedded devices must be protected in the field. Any environment where there is concern of attackers taking physical possession of a

device housing sensitive data can benefit from pre-emptively securing vulnerable devices with the Vault.

Technical Summary

Consists of one Primary node, and zero (0) or more Secondary node(s). Each Secondary node contains a QuID and a QSU binary with associated dependencies. Each secondary node calls the QSU binary on boot. The binary retrieves the QCU from the Primary node's PSS and reconstitutes the QuID with reconstituted data residing in memory (RAMFS). The system can also be configured to shuffle the protected data and generate a new QCU after each boot, which prevents replay attacks. Communication between all primary, bridge and secondary nodes are protected using a double ratchet algorithm to encrypt all messages.

Key Features

- Data Protection in cold storage
- Physical separation between QuID (set of lock) and QCU (locked key ring)

Binary Injection

Accomplished via overlay area attached to end of binaries by build scripts.

Packager/unPackager

unPackager is injected with the following items by Packager:

- `SYSTEM_TYPE`: Locks unPackager binary into running as either Primary, Secondary, or Bridge node to prevent attacker from attempting to circumvent Secondary node QuID/QCU physical separation requirements by running as standalone Primary node
- `FINGERPRINT_QCU`: Used to guarantee that a particular unPackager binary can only be used to unpack the associated QuID/QCU; thus even if an attacker gains access to an unPackager binary, it cannot be used to reconstitute protected data unless the attacker has gained access to the specific unPackager binary associated with the compromised QuID/QCU
- `AUTO_RAMFS_CLEANUP`: If true, Secondary and Bridge nodes will continuously monitor connection to upstream for interruption. Upon interruption, auto-reconnect will be

attempted until specified delay timeout is reached, after which RAMFS area will be cleaned up and unPackager process terminated

- `AUTO_RAMFS_CLEANUP_DELAY`: Amount of time in milliseconds to delay cleaning up the RAMFS area; auto reconnect to upstream is attempted until delay is reached

Obfuscation

- Configuration and QCU attribute keys are defined as arbitrary strings: attr-#
- Binary sensitive strings are defined as hexadecimal char vectors
- QCU values are encoded to base64

Sea of Nothing

- All Quantum Element identifiers are randomly generated alphanumeric characters of predefined length
- Directory names within the PSS directory are 64 byte hashed System IDs
 - The System ID is a 32 byte string (used as the hashing message), the QCU fingerprint (length can vary) is used as the hashing key, and the resulting hashed string is 64 bytes
- QCU file name is the same as the parent PSS directory name: hashed System ID

Validation

System Type

- unPackager binary is injected with User-selected System Type (i.e. Primary, Secondary, or Bridge).
- unPackager terminates if the injected System Type does not match the System Type as determined based on configured ip/port. This prevents a Secondary node from being configured as a standalone Primary node in an attempt to circumvent the Secondary node QuID/QCU physical separation requirements.

QCU Fingerprint

- QCU Fingerprint is unique to a given QCU
- QCU Fingerprint is designed so that QuID/QCU Shuffle does not change the fingerprint

- unPackager binary is injected with a QCU Fingerprint calculated from the QCU produced by Packager when protecting data
- unPackager terminates if the injected QCU Fingerprint does not match the fingerprint of the QCU retrieved from PSS (this ensures a different, perhaps compromised, binary is not being used to reconstitute protected data)

Security Analysis

Before executing the unpacking logic, the unPackager first checks if the fingerprint calculated from the QCU matches that which was injected into the unPackager at packaging time. If a mismatch is detected, the unPackager process exits and does not unpack the data. This prevents an attacker from attempting to unpack a QuID using a unPackager binary and/or QCU that was not originally packaged with the QuID.

QuID/QCU Shuffle

- QuID/QCU Shuffle prevents replay attacks by guaranteeing that QuID and QCU data are scrambled after each successful execution of unPackager

unPackager Stub

- The unPackager Stub application is stored on secondary nodes, and does not contain any sensitive information about the incryption process
- The QHC binary application is stored in PSS and contains sensitive code to incrypt and dicrypt packages. When data must be unpacked on the secondary node, the Stub application makes a request to the Primary node to retrieve this executable file.

QHC Binary Encryption

The QHC binary (the one with the “secret sauce”) is encrypted by Packager and unencrypted by the unPackager Stub prior to execution.

Auto-Cleanup

On Secondary nodes (including Bridge nodes), if unPackager stub connection to Primary node is lost, reconstituted data (if any) is automatically removed and the RAMFS area is unmounted

- User can specify length of timeout before RAMFS working area is sanitized

Versatile Packager

- Any file type can be protected
- Any file size can be processed
- Directories and nested directory structures are supported

Versatile unPackager

- unPackager can run in single node (i.e. Primary without TCP connections) or multiple node configuration, and can function either with or without protection.
 - A single node Primary is useful for situations where the QCU is stored on a removable Secure Key flash drive, and protected data resides on only one other device (standalone devices start TCP services; no associated Secondary or Bridge nodes)
 - A node with protection enabled will attempt to reconstitute protected data
- Primary
 - Head of the snake; without this node running none of the Secondary or Bridge nodes can retrieve QCUs to reconstitute protected data
 - Typically runs without incryption/dicryption enabled
- Secondary
 - Must run with incryption/dicryption enabled (otherwise would not serve any purpose)
 - QCU is retrieved from secure storage area (PSS) via Primary node and used to reconstitute protected data into RAMFS working area
- Bridge
 - Used in situations where direct connectivity to the Primary node is not possible from a Secondary node
 - TCP message forwarding is used to pass message packets up/down the connection chain

- Typically runs without incryption/dicryption enabled

Transit

FIG. 11 is a schematic diagram of an embodiment of a Comms mode 142-a of a Transit configuration 142 embodiment of the digital asset protection system 100 to provide quantum cyber resilience to a communication over an unsecured network 158. This diagram illustrates a simplified single data transaction using an embodiment of the digital asset protection technology to provide cyber resilience to hackers utilizing dark web tools and/or AI enabled CRQC to attempt to gain access to captured data during its transmission across unsecured/secured networks 158. Adversaries are going to capture data in transit over unsecured/secured networks, but the embodiments as disclosed herein effectively render useless any such captured data.

In embodiments, a Packager/unPackager system 110 takes in digital assets 102 that is to be protected and creates a unique QCU 120/QuID 124 pair as a quantum incrypted package 104 that is safe to be sent from a transmit node 170 over an unsecure/secured network 158 to a receive node 172 that is also equipped with a Packager/unPackager system 110. Once a quantum incrypted package 104 is received by the receive node, the Packager/unPackager system 110 uses the QCU 120 included in the package 104 and a local QSS 118 to de-incrypt the QuID 124. The QuID 124 is then reconstituted into the original data form 102 on the receive node 172. In some embodiments, the Packager/unPackager system 110, QCU 120 and QuID 124 are only stored and executed in volatile memory as a further cybersecurity protection. When the quantum incrypted package 102 is intercepted at some point 176 in transmission over the unsecured/secured network 158, both the QCU 120 and QuID 124 will appear as noise 178 to the hacker – effectively a “sea of nothingness.” Furthermore, no tools or knowledge available on the dark web, or even AI enabled CRQC’s can aid hackers in deciphering or unlocking either the QCU 120 or QuID 124 in quantum incrypted packages 104. The incryption produced by the various embodiments disclosed herein effectively leaves hackers with no idea of a starting point from which to begin hacking.

In embodiments of the digital asset protection system 100 referred to as Comms, a transmitter node protects data then sends it to a receiver where the protected data is converted back into original format. In these embodiments, Comms leverages core data protection capabilities to prevent an attacker from gaining any useful information by intercepting communication messages. Protected data if intercepted will appear to be noise, and even if collected would be impervious to

brute force attacks. This capability is suitable for environments where sensitive data must be transmitted over communication channels that may be insecure.

In various embodiments, the digital asset protection system 100 utilizes a Packager binary, unPackager binary, and generated QuID and QCU. To achieve secure communication, in an embodiment of Comms the Packager binary is run on a distributor node to protect a communication message, then the generated QuID (in encrypted tar archive/tape archive format) and QCU (in encrypted format) are sent to a receiver node via double ratchet TCP where unPackager is run to reconstitute the secure message contents.

In various embodiments, Comms supports the following data input/output types:

- Filesystem: Unprotected data is read in from transmitter filesystem and written out to receiver filesystem.
- Standard Input/Output: Unprotected data is read in from transmitter terminal and written out to receiver terminal.

In various embodiments, Comms can operate in modes of BOTH, RECEIVE, or TRANSMIT, supporting the following directional capabilities:

- Unidirectional: Data is Packaged on a transmitter and sent to a receiver where it is Unpackaged.
- Bidirectional: A receiver also acts as a transmitter and sends packaged data back to the original transmitter.
- Linked: A receiver also acts as a transmitter and sends data to a receiver other than the original transmitter.

In various embodiments, each of the above communication types can be operated in either DISCRETE or STREAMING communication mode:

- Discrete: A finite packet of data is packaged, transmitted, then unpackaged; both Packager and unPackager processes exit at processing conclusion.
- Streaming: A stream of data packets is packaged, transmitted, then unpackaged continuously as data is read from the stream; both Packager and unPackager remain running until the communication stream is terminated.

In various embodiments, Use Case Examples for Comms can include:

- A sensor device can periodically send collected information back to a controller over unsecured communication channels using Comms. This method will allow the

data to be reconstituted by the receiver but is protected against attackers that may be intercepting data from the intermediate channel.

- Streamed data such as audio can be exchanged between two entities in a secure way. Because separate channels are used for sending and receiving, full duplex communication is also possible.

In some embodiments, Comms 142-a also supports transmitting executable commands and having them executed on a target device. This capability could be used in environments where remote controls must be transmitted to target devices over communication channels that may be insecure. In embodiments, the QuID 124 and QCU 120 are transmitted to a target device as needed (and over separate ratchets) to execute specific task(s) on the target device (e.g., transmit QuID of an executable to a target device, unpackage on the target device, then execute the binary). In embodiments, the QuID 124, QCU 120, and unpackaged data may be removed from volatile memory (ramfs) upon successful completion of task so that dicrypted data is never on the target device outside of the time needed to perform the task. Benefits of this executable embodiment include:

- Dicrypted data is only on the target device for the duration of time needed to execute command(s).
- A user can execute any command on any binary on any target device at any time while the device is running (assuming there is network connectivity to the device).
- Dynamic execution can be combined with any of the other three embodiments to provide additional capability.

In various embodiments, Use Case Examples for Comms can include:

- A sensor node, such as video feed can be sent a remote command to change the parameters of operation (e.g. change direction).
- Securely distribute software updates to a set of target nodes.
- Software/firmware Anti-Tamper: install executable software only when needed to operate the device.
- Remotely control vehicles and other hardware, while keeping the binary and resources needed to perform the remote operations safe in the event of target device theft.

In embodiments, Transit 142 supports different modes for communications (Comms 142-a), data transfer (Transfer 142-b), and remote control of target device(s) (Agent 142-c). All variants implement the same underlying design described with respect to FIG. 12, with slight differences in configuration.

In embodiments, all Transit nodes can operate as a Transmitter 402, a Receiver 404, or a Transceiver 402/404 which allows for concurrent bi-directional transmissions.

In embodiments, Transmitter 402 uses embodiments of the incryption processes for quantum cyber resilient digital asset protection with double-ratchet cryptography to protect data being transmitted. The output of incryption processes for quantum cyber resilient digital asset protection is a QCU 120/QuID 124 pair, which is equivalent to a locked key ring/set of locks. Because both QCU 120/QuID 124 are transmitted together, additional security measures are taken to protect the data in transit.

In embodiments, Receiver 404 uses embodiments of the dicryption processes for quantum cyber resilient digital asset protection with double-ratchet cryptography to protect data being received. Data is read in from either file system, terminal, or external stream service depending on the use case. Input can be read in discrete (once) or streaming mode for any of the three input types.

As shown in FIG. 12 for Transmitter 402, Read 410 accesses a set of digital data assets 102 and Package 412 is to produce a QCU 120 and incrypt the QuID 124. At the end of packaging, a QSS 118 is “sprinkled” throughout the QuID 124 to corrupt it and prevent unpackaging without knowledge of the value of the QSS 124 and how to find and extract it from the QuID 124. In some embodiments, both QCU 120 and QuID 124 are independently encrypted using AES256. In various embodiments, Merge 414 merges QCU 120 and QuID 124 together using a blended strings functionality to produce a single data object. In embodiments, the single data object may be encrypted a final time. In embodiments, Transmit 416 transmits the encrypted merge result via double-ratchet TCP communications with multi socket capability to allow for adjustable bandwidth. As long as all nodes in a group have the same QSS, a single transmitter can transmit to 1-n receivers.

As shown in FIG. 12 for Receiver 404, when data is received, it is unmerged, unpackaged, then written to a file system, printed or displayed on an output device, and/or forwarded to an external stream service. At Read 420, data is read in from the TCP multi-socket connection to the

transmitter. At Unmerge 422, data is unmerged using a blended strings capability to separate the QCU 120 from the QuID 124. The QCU 120 and QuID 124 are independently decrypted and stored in volatile program memory. At unPackage, 424 the QSS 118 is extracted from the QuID 124 and checked against the injected QSS 118 used for the Transmitter 402. The QCU 120 is used to decrypt the QuID 124 unless a QSS mismatch is detected. At Write 426, the unpackaged sensitive digital assets 102 can be written to a file system, printed to or displayed on a terminal or display, or forwarded to the external stream service depending on use case. If the receiver is 404 operating in an Agent mode, the received data may be one or both of a terminal command and/or a binary to execute on/by the receiver system.

In embodiments, a transmitter node protects data then sends it to a receiver where the protected data is converted back into original format. Transit leverages EnQuanta hybrid cryptographic core data protection capabilities to prevent an attacker from gaining any useful information if data-in-transit communication messages are intercepted and analyzed. Protected data if intercepted will appear to be noise, and even if collected would be impervious to brute force attacks. This capability is suitable for environments where sensitive data must be transmitted over communication channels that may be insecure.

Variants

Peer-to-Peer

Data is transmitted via TCP either directly from a transmitter to one or more receivers, or via intermediate linking nodes. This variant is simpler than a centralized alternative, as it does not require a central server to facilitate message traffic.

Centralized

Data is transmitted via TCP to a central management server (either directly or via intermediate linking nodes), where the destination target node or group of nodes is determined and forwarded to by the central server.

In embodiments, Transmit includes a Packager binary, unPackager binary, and generated QuID and QCU. To achieve secure transmissions using EnQuanta hybrid cryptographic protection, Packager is run on a transmitter node to protect a communication message, then the generated QuID bundle (in encrypted format) and QCU (in encrypted format) are randomly merged utilizing the blended strings functionality and sent to a receiver node via double ratchet TCP where unPackager is run to restore the secure data.

IO Modes

Transit supports the following data input/output types:

1. Filesystem: Unprotected data is read in from transmitter filesystem, and written out to receiver filesystem
2. Standard Input/Output: Unprotected data is read in from transmitter terminal and written out to receiver terminal
3. External: Unprotected data is streamed into the transmitter and out of the receiver via basic TCP socket communication (i.e. External Stream Service)

Direction Modes

Transit can operate in directional modes of BOTH, RECEIVE, or TRANSMIT, supporting the following directional capabilities:

1. Unidirectional: Data is Packaged on a transmitter and sent to a receiver where it is Unpackaged
2. Bidirectional: A receiver also acts as a transmitter and sends packaged data back to the original transmitter
3. Linked: A receiver also acts as a transmitter, and sends data to a receiver other than the original transmitter.

Communication Modes

Each of the above communication types can be operated in either DISCRETE or STREAMING communication mode:

1. Discrete: A finite packet of data is packaged, transmitted, then unpackaged; both Packager and unPackager processes exit at processing conclusion
2. Streaming: A stream of data packets is packaged, transmitted, then unpackaged continuously as data is read from the stream; both Packager and unPackager remain running until the communication stream is terminated
 - Performs an additional step after unpackaging to execute any scripts or commands that were provided. These actions may be dependent on data assets that were included in the same transmission.

Blended QuID-QCU

Transit environments (i.e. environments where data must be protected while in transit as opposed to being protected only at cold rest) require both the QuID (ciphertext) and the QCU

(cryptex) to be transmitted to the receiver. Therefore, care must be taken to avoid transmitting the QCU in an easily identifiable and/or recoverable manner. To obfuscate the QCU, a blended string capability can be used to merge the encrypted QCU into the bundled and encrypted QuID before sending the result to the receiver.

Use Cases

- A sensor device can periodically send collected information back to a controller over unsecured communication channels using Comms. This method will allow the data to be reconstituted by the receiver but is protected against attackers that may be intercepting data from the intermediate channel.
- Streamed data such as audio can be exchanged between two entities in a secure way

Peer-To-Peer

- Walkie talkies

Centralized

- Data/messaging application similar to Signal

Comms

- communication of relatively small data packets. Examples:
 - Images
 - Audio
 - Emails
 - Text messages

Transfer

- Securely transfer very large datasets. Examples:
 - Transfer database dump file
 - Send image data from a remotely controlled vehicle to a base station upon returning to origin

Agent

- Send commands to execute on a target device. Examples:
 - A sensor node, such as video feed can be sent a remote command to change the parameters of operation (e.g. change direction)
 - Securely distribute software updates to a set of target nodes

- Software/firmware Anti-Tamper: install executable software only when needed to operate the device
- Remotely control vehicles and other hardware, while keeping the binary and resources needed to perform the remote operations safe in the event of target device theft.

Storage

FIG. 13 is a schematic diagram of an embodiment of a Storage configuration 144 embodiment of the digital asset protection system 100 to provide quantum cyber resilience of dynamic digital assets in non-volatile memory in a presumably secure facility where the digital assets are frequently or continually updated or changes (e.g., networked server storage or RAID servers).

In some embodiments, the Storage product configuration 144 uses a RAID 10 group of nodes to protect data at cold rest. The core operations provided by secure storage are distributing data to the network, retrieving data back from the network, and purging data from the network. In the Storage embodiment, data objects are encrypted and broken into pieces that are distributed across nodes in the network. This approach provides physical separation between the QEs and associated QCU. The packaging and unpacking operations are performed by a primary controller node that is connected to all storage nodes in the Network. Each storage node runs a server component that handles requests from the controller node. Storage 144 also provides data redundancy by organizing storage nodes into a set of RAID 10 stripes with multiple mirror nodes per stripe. In some embodiments, an additional service is deployed in front of the Storage network so that clients can manage their data by interfacing directly with a RESTful API.

In embodiments, Storage 144 supports Distribute, Retrieve, and Purge operations. A separate process flow for each of these operations is shown in the referenced figures.

Distribute

FIG. 14A is a process flow diagram of a Distribute process of an embodiment of the Storage configuration embodiment shown in FIG. 13. The Distribute operation allows clients to store data objects into the secure network. The flow of this operation through the components of the system is described below.

Client

Initially, a client begins with a data object they would like to save in the Storage network. To accomplish this, a RESTful API request is made to the Network API component. The user facing client endpoint is called /package/ and it supports both GET and POST requests. The binary content of the data object is passed as the body of the API request. After making the API request, the client is responsible for reading the generated GUID from the response and storing it for later use in unpackaging and purge requests.

Network API Service

After receiving the request to package a new data object, API service generates a new globally unique identifier (GUID) value for the data. This string will be used to identify and track the data object in its incrypted form. Next, the API writes the data to a GUID-specific location so that it can be processed by the Storage controller. In most cases, the Network API Service will be executed on the same server as the Storage controller, but it is not required. To complete the request, the Network API service updates a GUID-specific configuration file and uses it to invoke a new Storage controller process. The value of the GUID is passed to the process as a command line argument. The Network API waits for this process to complete and reads the exit code from the process. Lastly, a response is sent to the client that indicates the status of the operation (success/fail), based on the exit code.

Storage - Primary

When the Storage controller application is executed, it is initialized from the GUID command line argument as well as a configuration file with additional configuration information for the Storage system. The first step is to read the data object from the GUID-specific directory and package it into an incrypted form. This step uses a QRNG and QHCs to protect the data and results in a set of QEs 122, as well as a QCU 120. After packaging, the QEs 122 are distributed to the Storage network. An initial count operation is performed to pre-compute the number of QEs 122 that will go to each stripe of the RAID network. After each storage node knows how many QEs 122 are expected, they are distributed. A copy of the files designated for each stripe is sent to each mirror node in the stripe. After the files have been distributed, an acknowledgment is sent that verifies the operation succeeded. Once the QEs 122 have been distributed, the encrypted QCU 120 is divided into randomly sized sections and distributed to the Storage network. Again, the

pieces of the QCU 120 will be sent to different stripes in a round robin fashion, and all mirror nodes will receive the pieces intended for the stripe.

After the distribution process completes, the controller node writes the Quantum RAID Map (QRM) to a GUID-specific location on disk. This file stores information about which Storage nodes belong to which stripes along with the ordering of QCU pieces that allow it to be reconstructed for subsequent operations. Assuming that the received file counts for QEs and QCU sections match the expected values and the QRM was successfully created, the process returns with an exit code of 0. Otherwise, a non-zero exit code is returned.

Storage - RAID Node

In embodiments, each node in the Storage network is running a RAID controller binary (qsrc) started before the system is operational. These binaries act as the server component for communication with the controller node. Therefore, they remain running at all times to listen for connection requests and service those requests when appropriate. For the Distribute operation, the storage nodes will be initialized with the number of files being sent. Then, as the controller node sends QEs and QCU pieces, the count of received files is tracked. The content for each received file is written into a GUID-specific location inside of the RAID Cold Storage (RCS) directory. After the file transmission has completed, the RAID controller responds with a message that indicates whether the distribution has succeeded.

Retrieve

FIG. 14B is a process flow diagram of a Retrieve process of an embodiment of the Storage configuration embodiment shown in FIG. 13. The Retrieve operation is used by clients to get previously stored data objects out of the Storage network. The steps of the operation are below.

Client

To recover a previously stored data object, the user sends a RESTful API request to the Network API component that triggers the Retrieve process flow. The signature for the client endpoint is /unpackage/<GUID>. Both GET and POST requests are supported. As noted, the GUID extracted from a previous /package request must be provided. The response from the Network API Service will contain the binary content of the data object in the HTTP response body.

Network API Service

Upon receiving the /unPackager request, the Network API service extracts the GUID and uses it to create a GUID-specific configuration file for the Storage controller process. Like the Distribute operation, the API service invokes the process, passes the GUID as a command-line option and waits for it to complete. Before sending the client response, the exit code is checked to determine if the Retrieve operation was successful.

Storage - Primary Node

After the Storage controller process is started, an initialization step takes place based on the provided configuration file and GUID value read from the command-line. In order to complete the Retrieve operation, the QRM file for the specified GUID is loaded. This provides the necessary information about how the nodes in the Storage network are organized into stripe groups. Next, the QEs 122 and QCU 120 pieces are requested from the Storage network. During this process, the files are requested from the first available mirror in each stripe group. Requests to additional mirror nodes are not required unless this first request fails. These Retrieve requests are made over a TCP connection to the RAID controllers running on the storage nodes. After the QEs are downloaded, they are saved into memory for subsequent processing. An additional step is required to construct the full QCU from the downloaded pieces, based on the ordering stored in the QRM.

After successfully retrieving the QEs 122 and QCU 120 for the specified GUID, they will be stored in memory. At this point, the controller node can unpackage the original data object to be returned to the user. This decryption step uses the details stored in the QCU 120 to initialize the correct set of QHCs 108 into a cipher stack that is executed in reverse order and uses the encrypted QEs 122 as the input. The result of this operation is that the original data object is recovered and written to a GUID-specific directory. After the data object has been unpackaged and written, the controller node process will exit, again using the return code to indicate the success or failure of the operation.

Storage - RAID Node

The RAID controller binaries are used to Retrieve the QEs 122 and QCU 120 pieces out of the GUID-specific directory inside the RCS area on the storage nodes. Due to the structure of the RCS, the controller node can simply pass the GUID to request all files for a Retrieve operation. The RAID controller can use this GUID to begin transmitting all files for the GUID back to the controller node without needing a separate request for each individual file. As noted before, this

step is executed on the first available mirror node for each stripe group. Dedicated message types are used to communicate success and error conditions back to the controller node.

Purge

FIG. 14C is a process flow diagram of a Purge process of an embodiment of the Storage configuration embodiment shown in FIG. 13. The Purge operation is used to delete data objects from the Storage network. The steps needed to complete this operation are described below.

Client

Like the Retrieve operation, the client can submit a GET or POST request to the Network API Service using the GUID associated with a data object that has previously been stored in the network. The client endpoint is /purge/<GUID>. After the Purge operation is completed, the GUID will no longer be associated with a valid data object in the Storage network. Therefore, after making the API request and parsing the response to verify that the operation was successful, the client is responsible for updating their records to remove the GUID from their application storage or marking it as unavailable.

Network API Service

When the Network API Service receives a Purge request, the GUID will be passed as a path parameter. This value is extracted and used to initialize a configuration file for invoking the Storage controller binary. As before, the GUID value is passed to this process as a command-line argument. After the process is started, the Network API waits for it to terminate and uses the exit code to determine the correct status code to use when sending the REST response back to the client.

Storage - Primary

After the Storage controller application is started and initialized with the GUID from the command-line, the QRM file is loaded from a GUID-specific storage area. After these steps are completed, a Purge request is made to all RAID controllers in the Storage network. By making this request to all nodes, the data object is permanently removed from the network, including the duplicated copies that were stored across mirror nodes in the same stripe group. After receiving confirmation from all mirror nodes that the requested Purge operation was successful, the controller node will delete the QRM and associated directory for the GUID from its storage before the application process completes and an appropriate exit code is returned.

Storage - RAID Node

When a Purge request is received by a RAID controller, the entire GUID-specific directory within that node's RCS directory is removed. This effectively removes all QEs and QCU pieces from the server. Afterwards, a response message is sent back to the controller node to indicate if the operation was successful.

In embodiments, Storage product uses a RAID 10 group of nodes to protect data at cold rest. Data is protected similar to Vault, then broken into pieces that are distributed across nodes in the network. This approach provides physical separation between protected data and associated keys and also further physically separates protected data pieces so that all pieces of a whole are never located together. Storage also provides RAID 10 benefits of striping and mirroring, in addition to incryption protection of data at rest. The network exposes a RESTful API that is used to distribute data to the network, retrieve data back from the network, or purge data from the network. When an initial distribution request is made to save data in the Storage, a globally unique identifier (GUID) is assigned so that the user can make subsequent requests to retrieve or purge the saved data. This system is ideal for longer-term storage of data or for an environment where ransomware attacks are a possibility. Storage provides data redundancy, so even if an attacker compromises or locks a node, data is still protected and retrievable from one of the mirror nodes. In some embodiments, Storage also can be provided with the ability to continuously monitor the health of all nodes in the network and dynamically repopulate data when required. This allows nodes to be taken offline for maintenance without any impact to how the network functions. An alternative Storage setup involves RAID 0, which is simpler and only requires two nodes, but does not provide fault tolerance or attack resilience.

In embodiments, Storage processes RESTful API HTTP requests to Package (binary data provided on the request), Unpackage, or Purge data within a RAID 10 network.

- For each RESTful API Package request: QuID and QCU are generated (in memory) and distributed to the RAID network for each Package request. A Quantum RAID MAP (QRM) is generated for each request and stored in a persistent memory area.
- For each Unpackage request: QuID and QCU identified by provided GUID are retrieved from the RAID network (into memory), reconstituted into original format (in memory), and returned to User on HTTP response

- For each Purge request: QRM, QuID, and QCU identified by provided GUID are deleted.

Key Features

- Data Protection in cold storage, by maintaining physical separation between QuID and QCU.
- Data Protection in Transit, using the Double Ratchet Algorithm to protect all communications
- Storage does not save protected data to disk or memory on Primary node
- Hardware fault tolerance via data redundancy
- Resilience protection against ransomware

RAID10

A RAID10 design can be employed by the Storage product to provide 1) data redundancy via mirroring, and 2) data separation via striping. This makes it harder for attackers to 1) compromise protected data via denial of service attacks, as doing so would require locking all mirrors within a stripe, 2) steal a full set of protected data, as doing so would require accessing all QEs from at least one mirror of each stripe.

Randomized Mirrors

A stripe is composed of at least two mirror nodes. The set of mirror nodes comprising a stripe is randomly selected from available nodes each time a distribution is performed. Example: Given 12 nodes total, and 2 stripes (each with 6 nodes), there are 924 unique combinations of nodes that can be randomly chosen from for each distribution.

Data Loss Computations

In this section, an analysis of the probability that the RAID network can continue operation without data loss, given loss of a number of randomly selected nodes. Assume an example, with a RAID network that has 12 total nodes, split into 2 stripes, with 6 mirror nodes per stripe.

Divided QCU

The QCU can be split using the Randomized Multi-Section String Split feature, and each section is distributed to a different stripe (and mirrored across all nodes within that stripe).

Encrypted QRM

The QRM is encrypted on disk using an arbitrary obfuscated hard-coded encryption key. It stores the RAID stripe/mirror configuration for each distribution made into Storage. Each distribution (with unique GUID) gets a different randomized configuration of which nodes are in which stripe.

Use Cases

- Organizations can install Storage to a group of nodes located on-premises, in the cloud, or use a hybrid approach. The number of nodes and RAID configuration can be customized for each deployment, depending on the specific storage and redundancy requirements.
- Storage can also be offered as a SAAS solution, through a licensed vendor that manages the nodes used as the RAID nodes for a given customer implementation of Storage.

Sensitive Data Protection

Restoration Area - RAM File System (RAMFS)

Protected data is reconstituted into virtual memory (RAMFS) rather than being written to disk, minimizing the system's attack surface.

- When power is removed (intentionally or otherwise) from the device where sensitive data was reconstituted, that sensitive data will disappear leaving no residue behind for malicious apps to recover
- Monitor for and prevent unauthorized access to the RAMFS area

Cleanup Service

A dedicated cleanup binary can be configured to manually or automatically clean up and unmount the restoration area for situations such as outlet power source switched to battery or a location of a device going outside a specified gps area, altitude, etc.

Storage Device

As an alternative to RAMFS, manual or automatic use of a secure storage device (ex: Key encrypted USB drive) into which to restore protected data can be configured.

Memory-Only Mode

For all but the Vault configuration, the QuID can be managed entirely in memory the same way the QCU is (although QuID in memory is more complex than QCU in memory). Memory-only mode will be relatively simple for handling small amounts of data read from the terminal, or for handling files smaller than the large-data threshold. For large files, and for streaming large datasets, logic must be implemented to allow the large data to not only be processed in chunks (as is currently done for large files), but also transmitted to a receiver (where applicable, ex: Transit, Storage, etc.) in chunks. Processing and transmitting chunks as they become available would be best as it would allow the receiver to start processing data while the original source data is still being streamed in from the external source. The receiver must be able to receive the chunks and collect them back together (similar to existing TCP message packet logic in QSC).

QuID Bundling and Encryption

The QuID is composed of Quantum Elements (QEs). To simplify management of the QEs, and improve QuID security, QEs can optionally be combined into a single file and encrypted as shown, for example, in FIG. 15.

QuID Bundling

- The QuID bundling functionality is used to collect all Quantum Elements into a single file that can be transmitted efficiently.
- The bundling process can combine an arbitrary set of files and directories. The file sizes and relative path locations are stored in a metadata section that is combined with the binary content of the bundled files. The bundle file layout is optimized so that bundle files can be efficiently created and used.
- QuID bundling is analogous to the TAR command but operates without requiring additional dependencies or system calls.

- For maximum security, QuID bundle files are named using the Stealthy Strings capability so that they can be identified in a given subdirectory.

QuID Encryption

- Accomplished with standard NIST AES-GSM-256 or Libsodium XChaCha20-Poly1305
- The QuID bundle encryption key is stored in the QCU

QCU Encryption

- Accomplished with Libsodium XChaCha20-Poly1305 or AES-256-GSM
- The QCU encryption key can be the 32-byte output of a custom hashing/obfuscation function that uses the 64-byte hashed System ID as the input value.

Security Features

As data is packaged, the keys and other metadata for the entire hybrid cryptography process are collected into the QCU. This is a JSON file that is used to reverse the hybrid cryptography process during unpackaging. To further secure the QCU information, this file may also be AES256 encrypted as a final step.

It would be unlikely that an attacker would obtain a QuID without an associated unPackager binary, and thus, given an associated QCU as well, would have all three puzzle pieces needed to be able to unpackage the QuID. However, given a QCU without an associated QuID and unPackager, an attacker may seek to analyze the structure and contents of the QCU to find and exploit weaknesses (if any). In these embodiments, this is prevented by the fact that the QCU is AES256-encrypted, requiring an attacker to first brute-force the AES encryption before being able to analyze the obfuscated contents of the QCU. In embodiments, the process automatically generates a QCU encryption key that is derived from features inherent to the process, so there is no need to store the key, precluding an attacker from ever being able to gain access to it without 1) access to information in the unPackager binary, and 2) an understanding of the process used to generate the key.

Various embodiments of systems, devices, and methods have been described herein. These embodiments are given only by way of example and are not intended to limit the scope of the

claimed inventions. It should be appreciated, moreover, that the various features of the embodiments that have been described may be combined in various ways to produce numerous additional embodiments. Moreover, while various materials, dimensions, shapes, configurations, and locations, etc. have been described for use with disclosed embodiments, others besides those disclosed may be utilized without exceeding the scope of the claimed inventions.

Persons of ordinary skill in the relevant arts will recognize that the subject matter hereof may be composed of fewer features than illustrated in any individual embodiment described above. The embodiments described herein are not meant to be an exhaustive presentation of the ways in which the various features of the subject matter hereof may be combined. Accordingly, the embodiments are not mutually exclusive combinations of features; rather, the various embodiments can be composed of a combination of different individual features selected from different individual embodiments, as understood by persons of ordinary skill in the art. Moreover, elements described with respect to one embodiment can be implemented in other embodiments even when not described in such embodiments unless otherwise noted.

Although a dependent claim may refer in the claims to a specific combination with one or more other claims, other embodiments can also include a combination of the dependent claim with the subject matter of each other dependent claim or a combination of one or more features with other dependent or independent claims. Such combinations are proposed herein unless it is stated that a specific combination is not intended.

CLAIMS

1. A computer-implemented digital asset protection system configured to be executed by one more computer systems to protect a set of original files of digital assets as incrypted data that is quantum cyber resilient, the digital asset protection system comprising:

a packager system operably configured to use a set of random seeds and a set of hybrid cryptography ciphers that includes at least one standard cipher and a plurality of non-standard ciphers to incrypt the set of original files as incrypted data in a set of ciphertexts with each ciphertext having a corresponding cryptex that includes information used to determine a dynamic cipher stack, a set of cipher keys, and a set of meta data unique to that ciphertext; and

an unpackager system operably configured to dicrypt the incrypted data of each ciphertext in the set of ciphertexts using the corresponding cryptex to restore the set of original files.

2. The computer-implemented digital asset protection system of claim 1, wherein the packager system further includes a protocol of adding a set of decoy files to the set of original files, each decoy file formed of a random amount of random noise, and wherein the cryptex includes an indication for each file of the set of original files of whether the file is a decoy file.

3. The computer-implemented digital asset protection system of claims 1-2, wherein the set of ciphertexts includes one or more quantum incrypted data ciphertexts, and the corresponding cryptex for each ciphertext is a quantum containment unit, the dynamic cipher stack includes at least one standard cipher as a first cipher or a last cipher in the cipher stack and a pseudo randomized number, selection, and order of four or more non-standard ciphers in the cipher stack, the set of random seeds includes one or more random number seeds generated by one of a standard pseudo-random number generator source, a random number generator source, or a quantum random number generator source, and the set of cipher keys includes one or more keys and/or nonces generated for each of the set of cipher keys using the set of random seeds.

4. The computer-implemented digital asset protection system of claims 1-3, wherein the packager system incrypts each ciphertext of incrypted data as a set of pieces each having an obfuscated piece name and the set of pieces arranged in a random element order based on the set of random seeds with the random element order and the obfuscated piece name stored as meta data in the corresponding cryptex, and wherein the unpackager system dicrypts each ciphertext using the corresponding cryptex by accessing the set of pieces based on the obfuscated pieces names and reordering the set of pieces arranged in the random element order to restore an original order to the set of pieces and then using the dynamic cipher stack and set of cipher keys and meta data in the corresponding cryptex to restore the original set of files with an original set of file names for each original file from the set of meta data.

5. The computer-implemented digital asset protection system of claims 4, wherein each piece in the set of pieces is a random size and is not padded if a size of the piece is larger than the threshold size and is padded with a set of random noise generated from the set of random seeds to exceed the threshold size if the size of the piece is smaller than the threshold size.

6. The computer-implemented digital asset protection system of claims 4-5, wherein the packager system further includes a protocol of injecting random noise generated from the set of random seeds into each piece of the set of pieces by determining a set of random locations in the piece based on the set of random seeds and injecting random noise at each random location of the set of random locations, and wherein the cryptex includes information representing the set of random locations.

7. The computer-implemented digital asset protection system of claims 1-6, wherein the unpackager system is operably configured to be executed from a volatile memory and the set of original files is only restored into a volatile memory.

8. The computer-implemented digital asset protection system of claims 1-7, wherein the set of original digital assets includes one or more of data, code, databases, training sets, weighting sets, text, ciphertext that is hardware and/ or software encrypted data, digital content or digitally

personal identifiable information, or any electronic or optical information in the form of a file, a record or an element as one of the set of digital assets that is at rest, in use or in transit.

9. The computer-implemented digital asset protection system of claim 1, wherein the digital asset protection system is an anti-tamper protection system for data at rest and the set of ciphertexts and the corresponding set of cryptexes are configured to be delivered and stored in a secure storage of a primary node of target computing system with the set of ciphertexts being deliverable to and stored in a non-volatile memory of a set of secondary nodes of the target computer and an executable of the unpackager system and the set of cryptexes being deliverable to the and stored in a volatile memory of the set of secondary nodes to restore the original files of the set of digital assets incrypted by the set of ciphertexts in the non-volatile memory to the volatile memory of the set of secondary nodes.

10. The computer-implemented digital asset protection system of claim 1, wherein the digital asset protection system is a ransomware resilient protection system for data at rest in a RAID storage system having a controller node and a set of storage node, and wherein the set of ciphertexts and the corresponding set of cryptexes are configured to be fragmented and delivered to the RAID storage system with the controller node configured to reconstitute the fragmented set of ciphertexts using the corresponding set of cryptexes.

11. The computer-implemented digital asset protection system of claim 1, wherein the digital asset protection system is a streaming communication system for data in transit having a transmitter node configured with a packager system to operate in a low-latency mode that blends and salts the set of ciphertexts and the corresponding set of cryptexes with a symmetric secret into packets of a data stream to be communicated over an unsecure channel to a receiver node which has prior access to the symmetric secret and is configured with an unpackager system to reconstitute the data stream as the set of ciphertexts using the corresponding set of cryptexes from which the data in transit is restored.

12. The computer-implemented digital asset protection system of claim 1, wherein the digital asset protection system is a data transfer system for data in transit having a transmitter node

configured with a packager system configured to operate in a normal mode that blends and salts the set of ciphertexts and the corresponding set of cryptexes with a symmetric secret into packets of a bulk data transfer to be communicated over an unsecure channel to a receiver node which has prior access to the symmetric secret and is configured with an unpackager system to reconstitute the bulk data transfer as the set of ciphertexts using the corresponding set of cryptexes from which the data in transit is restored.

13. The computer-implemented digital asset protection system of claim 1, wherein the digital asset protection system is a data transfer system for data in transit having a transmitter node configured with a packager system configured to operate in a mode that blends and salts the set of ciphertexts and the corresponding set of cryptexes with a symmetric secret into packets of data transfer to be communicated over a TCP double ratchet channel to a receiver node which has prior access to the symmetric secret over the TCP double ratchet channel and is configured with an unpackager system to reconstitute the data packets as the set of ciphertexts using the corresponding set of cryptexes from which the data in transit is restored.