SYSTEMS AND METHODS FOR BLOCKCHAIN-ENABLED END-TO-END ENCRYPTION

CROSS-REFERENCE TO RELATED APPLICATIONS

[0001] This application claims the benefit or priority to U.S. Patent Application No. 18/610,733, filed March 20, 2024, the disclosure of which is incorporated herein by reference in its entirety.

BACKGROUND

[0002] The present invention relates generally to the field of cryptographic key management within distributed networks.

SUMMARY OF THE INVENTION

[0003] One embodiment of the invention relates to a system for managing cryptographic keys in a distributed network. The system may comprise a plurality of nodes configured to participate in a blockchain network. The system may comprise a key generation module, operable on at least one of the nodes, to distribute generation of cryptographic keys using a threshold scheme, wherein the cryptographic keys can include a public key and a private key share for each participating node. A verification module, operable on at least one of the nodes, can be configured to validate an integrity and authenticity of the cryptographic key shares exchanged among the nodes using the blockchain network. The system can include a key reconstruction module configured to enable the reconstruction of a private key from a subset of the private key shares when a predetermined threshold number of shares is reached, wherein the private key can be used for decrypting encrypted messages or data transmitted within the network. A secure communication module can be configured to establish secure communication channels between nodes using the cryptographic keys generated by the key generation module. The system can include a storage module configured to store the cryptographic key shares and the public key within the blockchain network.

[0004] Another embodiment is a system for facilitating secure communication between users within a blockchain network, comprising a Multi-Signed Certificate Provider

(MSCP) configured to generate and furnish multi-signed certificates for users' public keys, including an identity key, a one-time key, and a fingerprint key, and to register said certificates on the blockchain network; a blockchain network configured to store the multi-signed certificates, which may be useful for ensuring the integrity and authenticity of these certificates for verification purposes; a smart contract deployed on the blockchain network, configured to manage the registration, retrieval, and verification of multi-signed certificates for establishing secure communication channels between users; an instant messaging server associated with the MSCP, configured to facilitate message delivery by notifying users of incoming messages and ensuring prompt relay of messages; and a user device equipped with a cryptographic module, configured to generate public and private keys, encrypt and decrypt messages using the secure communication channel established based on verified multi-signed certificates from the blockchain network.

BRIEF DESCRIPTION OF THE DRAWINGS

[0005] FIG. 1 illustrates an example topology model, in accordance with some embodiments of the present disclosure;

[0006] FIG. 2 illustrates an example of multi-signed certificate generation and blockchain registration, in accordance with some embodiments of the present disclosure;

[0007] FIG. 3 depicts a sequence diagram illustrating the backup and restoration process in the protocol, in accordance with some embodiments of the present disclosure;

[0008] FIG. 4 depicts crypto-module architecture, in accordance with some embodiments of the present disclosure.

[0009] FIG. 5 illustrates a flow diagram of an example method for establishing secure decentralized end-to-end encryption in a distributed network, in accordance with an embodiment of the present disclosure.

DETAILED DESCRIPTION

[0010] In the digital age, End-to-End Encryption (E2EE) has emerged as a fundamental technology for securing the confidentiality and privacy of digital communications. E2EE ensures that the content of messages can be accessed by the intended recipients, mitigating the risks associated with eavesdropping, data breaches, and unauthorized surveillance. With the proliferation of digital communication platforms and the exchange of sensitive information across the globe, the need for robust security mechanisms to protect data integrity, confidentiality, and authenticity has never been more pressing. The adoption of E2EE by major messaging applications like Facebook, WhatsApp, and Signal, combined with concerns over privacy in the wake of revelations about mass surveillance, has underscored the vital role of E2EE in safeguarding individual and collective digital interactions. This technology represents a critical shift towards user empowerment and control over their personal data, making it increasingly important in today's modern communication systems.

[0011] End-to-End Encryption (E2EE) protocols encompass several core phases to ensure comprehensive security in digital communication.

[0012] **Message Authentication and Integrity**: Beyond encryption, E2EE protocols integrate mechanisms to authenticate and maintain the integrity of messages. This safeguards against tampering during transit and confirms the message's origin. Methods like digital signatures or message authentication codes (MACs) serve this purpose.

[0013] **Key Exchange**: The initial phase centers on secure key exchange methods, such as Diffie-Hellman or Public Key Infrastructure (PKI), to facilitate message encryption and decryption while safeguarding key confidentiality.

[0014] **Encryption and Decryption**: In the subsequent phase, messages or data can be encrypted using the recipient's public key for secure transmission, rendering content unreadable to unauthorized parties. The recipient uses their private key for decryption, ensuring message confidentiality.

[0015] **Backward/forward Encryption Mechanism**: In this context, "forward encryption" ensures that the intended recipient can decrypt and read the messages, while "backward encryption" allows the sender to decrypt and review their previously sent messages. These mechanisms combine security and convenience within E2EE systems.

[0016] **Secure Backup**: This phase ensures the secure backup and restoration of user data, including encryption keys and sensitive content. It entails encrypting and securely storing these backups to prevent unauthorized access or data loss, enabling users to recover their data in case of device loss or failure.

[0017] End-to-End Encryption (E2EE) can rely on a range of sophisticated techniques to ensure the security and privacy of digital communications. Three commonly used methods that have gained prominence in E2EE protocols can be the Diffie-Hellman key exchange, the Double Ratchet algorithm, and the X3DH (Extended Triple Diffie-Hellman) protocol. Diffie-Hellman serves as the cornerstone of secure key exchange, enabling parties to establish a shared secret over public channels while keeping it confidential from eavesdroppers. The Double Ratchet algorithm can be a critical component of secure messaging applications like Signal. It provides forward secrecy and cryptographic ratcheting, ensuring that even if an encryption key can be compromised, past and future messages remain secure. Lastly, the X3DH protocol combines elements of Diffie-Hellman with a pre-key distribution system, simplifying the process of securely exchanging keys and ensuring authenticity in the process. These techniques collectively provide a robust foundation for E2EE systems, delivering the essential attributes of confidentiality, integrity, and future secrecy, all of which can be crucial for secure digital communication in today's interconnected world.

[0018] The centralized nature of protocols introduces security vulnerabilities that merit consideration. One major concern can be the reliance on a central authority or key server, which serves as a single point of failure. Malicious actors targeting this central entity can compromise the entire system's security. Additionally, identity verification, a cornerstone of secure communication, becomes challenging in the era of increasing decentralization and pseudonymous online interactions. Attacks can include identity

spoofing, denial of service, impersonation and man-in-the-middle attacks, potentially undermining the authenticity of communication channels.

[0019] As a result, the need for a trusted and decentralized Public Key Infrastructure (PKI) has emerged as an essential component of contemporary communication security. In a decentralized PKI, the traditional reliance on central certificate authorities can be minimized, reducing the risk associated with a single point of failure. A decentralized end-to-end encryption protocol can leverage the power of blockchain for a Multi-Signed Certificate-To-Identity (MSC-To-IT) system. This distributed approach allows end users to have greater control over their digital identities and cryptographic keys. Employing blockchain technology can enable the creation of tamper-resistant and auditable records for identity verification events. This can fortify the security of the identity verification process and instill transparency and accountability. Such a system enables end users to independently verify each other's identities, mitigating the risks of impersonation and man-in-the-middle attacks.

[0020] A decentralized end-to-end encryption protocol can be based on blockchain technology. The protocol can rely on cutting-edge technology such Ethereum Blockchain and smart contracts to enable decentralized identity storage, together with identity verification and client authentication techniques.

[0021] FIG. 1 illustrates an example topology model. The proposed decentralized end-to-end encryption protocol can consist of four key components depicted in FIG. 1.

[0022] **Users**: The users can be active participants who utilize the decentralized end-to-end encryption protocol for secure communication. Users initiate the communication process and can be responsible for encrypting and decrypting messages.

[0023] **MSCP (Multi-Signed Certificate Provider)**: The MSCP entity acts as the certificate provider, responsible for generating and furnishing the multi-signed certificates. These certificates can be crucial for ensuring the authenticity and security of communication between users.

[0024] **Instant Messaging Server**: plays a vital role in the decentralized end-to-end encryption protocol. Its primary function can be to facilitate message delivery by notifying

the intended recipient when a new message arrives. The server ensures that messages can be promptly relayed to users. Each instant messaging server can be relied to an MSCP.

[0025] **Blockchain Network**: Blockchain technology can be a central component of the decentralized end-to-end encryption protocol. It serves as the secure and tamper-resistant storage medium for multi-signed certificates. By storing certificates on the blockchain, the protocol ensures the integrity and authenticity of these certificates, making them accessible for verification when needed.

[0026] The architecture functions as a secure and decentralized ecosystem where users can communicate with one another while relying on multi-signed certificates stored in the blockchain. The MSCP entity acts as a trusted certificate provider, and the Instant Messaging Server ensures efficient message delivery. This architecture ensures the security and authenticity of communication while leveraging the strengths of blockchain technology.

[0027] The decentralized end-to-end encryption protocol can encompass seven primary mechanisms organized into three phases: Public key distribution, Encryption & decryption, and Identity verification and mutual authentication.

[0028] ***Public Key Distribution***

[0029] To initiate the process, individuals need to register with the MSCP. A new user can establish an account by associating a digital certificate with it and utilizing the unique identity of the participant, as illustrated in FIG. 2.

[0030] Multi-signed certificate generation and blockchain registration

[0031] In order to provide a secure E2EE, the decentralized end-to-end encryption protocol can combine a number of cryptographic techniques, to create a unique AES secret key for every conversation. Categories of public keys can include

[0032] **Identity Key (refer to as I^i)**: A Curve25519 key pair generated for long-term use.

[0033] **One-Time Keys (refer to as E^i)**: A queue of Curve25519 key pairs for one-time usage.

[0034] **Fingerprint Key (refer to as F^i)**: A Ed25519 key pair used for signing the two other keys.

[0035] Firstly, the user ($\boldsymbol{A}$) generates two pairs of public and private keys: identity key (I^A_{pub}, I^A_{pri}), fingerprint key (F^A_{pub}, F^A_{pri}), and a pair of one-time keys (E^A_{pub}, E^A_{pri}). Subsequently, ($\boldsymbol{A}$) sends a registration request to MSCP, including the user ID $I\,D^A$, the public components of these keys, the timestamp $\boldsymbol{T}$, the keys' validity $\boldsymbol{t}$, the signature of each key using the fingerprint key, and the signature of the entire request. The request can be transmitted via HTTPS.

[0036] Upon receiving the request, the MSCP entity validates the signature, then generates certificates for the identity keys and fingerprint (C^A_{IF}) key, along with a certificate for each one-time key (C^A_n). In fact, (C^A_{IF}) contains the ID of $\boldsymbol{A}$, the timestamp, the key's validity, the two keys and their signature, and the decentralized end-to-end encryption protocol multi-signature. Moreover, (C^A_n) contains the ID of $\boldsymbol{A}$, the timestamp, the key's validity, the key and its signature, and the decentralized end-to-end encryption protocol multi-signature.

[0037] To multi-sign the certificates, MSCP uses the private keys of 2/3 of nodes. The node invokes smart contract functions ($AddIdentityCert$ and $AddOneTimeCert$) for adding the certificates to the blockchain. Then the MSCP generates a response that includes those certificates and its own certificate, signs the request, and sends it to user $\boldsymbol{A}$ through *https*. Finally, the user validates the signature and stores his certificates. Now, user $\boldsymbol{A}$ has access to the system.

[0038] *Certificate Fetching*

[0039] When A intends to send a message to $\boldsymbol{B}$, he invokes the smart contract functions ($GetIdentityCert$ and $GetOneTimeCert$) by providing $\boldsymbol{B's}$ ID.

[0040] The smart contract responds by furnishing $\boldsymbol{A}$ with $\boldsymbol{B's}$ identity certificate (C_{IF}^{B}) and one of the one-time key certificates (C_{n}^{B}).

[0041] When receiving these certificates, $\boldsymbol{A}$ validates them and extracts the identity key (I_{pub}^{B}) and the one-time key (E_{pub}^{B}). These keys can be then utilized to generate a shared master

[0042] *Public Key Verification*

[0043] To enhance the security of the protocol, we propose an Identity Verification and Mutual Authentication (DIVMA) mechanism, enabling two users ($\boldsymbol{A}$ and $\boldsymbol{B}$) to authenticate each other's cryptographic identity. This mechanism initiates when $\boldsymbol{A}$ acquires $\boldsymbol{B}$'s certificate ($\boldsymbol{C_B}$) from the Smart Contract on the Blockchain ($\boldsymbol{Bn}$). Using the public key ($\boldsymbol{K}_{\text{pub}}^{B}$) recovered from the certificate ($\boldsymbol{C_B}$) and assumed to be that of $\boldsymbol{B}$, $\boldsymbol{A}$ executes the DIVMA-based mechanism to ensure the authenticity of this identity.

$\boldsymbol{A}$ sends a read request to the smart contract with $\boldsymbol{B}$'s ID ($\boldsymbol{I\,D^{B}}$) to check if $\boldsymbol{B}$ can already be a user.

The smart contract verifies if (C_{IF}^{B}) exists on the Blockchain:

It performs a lookup of (C_{IF}^{B}).

It validates the request by returning the multi-signed certificate of Bob (C_{IF}^{B}).

$\boldsymbol{A}$ verifies the certificate and retrieves $\boldsymbol{B}$'s public key (E_{pub}^{B}) from it (assumed to be that of $\boldsymbol{B}$).

$\boldsymbol{A}$ sends a transaction:

$$T_{A}^{I} = \left[I\,D^{A}, I\,D^{B}, T, \text{Encrypt}_{\text{pub}}^{B}\left(I\,D^{A}, ID^{B}, T, E_{pub^{\wedge}}\right)\right]$$

Next, $\boldsymbol{B}$ checks the transaction with his private key (E_{priv}^{B}) and sends to $\boldsymbol{A}$'s address:

$$T_{B}^{2} = \left[I\,D^{B}, I\,D^{A}, T+1, \text{Encrypt}_{\text{pub}}^{A}\left(I\,D^{B}, I\,D^{A}, T+1, E_{\text{pub}^{B}}\right)\right]$$

The same verification can be carried out after $\boldsymbol{A}$ receives the transaction ($\boldsymbol{T}$), and mutual authentication can be created between the two users.

[0044] Sending a Secure Message

The shared master secret key can be created by $\boldsymbol{A's}$ messaging app using his private keys and $\boldsymbol{B's}$ public keys:

$$S = \mathrm{ECDH}\left(I_{pri}^{A}, E_{pub}^{B}\right) \| \mathrm{ECDH}\left(E_{pri}^{A}, I_{pub}^{B}\right) \| \mathrm{ECDH}\left(E_{pri}^{A}, E_{pub}^{B}\right)$$

[0045] A hash-based key derivation function (HKDF) can be then used to construct a chain key:

$$R_0 \| R_0 = \mathrm{HKDF}(0, \mathrm{S}, \text{"OLM ROOT"}, 64)$$

from which a message key can be derived:

$$M_0 = \mathrm{HMAC}(C_0, \text{"x01"})$$

[0046] User $\boldsymbol{A}$ encrypts the first message using the dynamic message key (M_0), leading to an update of the chain key (C_i). Following this, $\boldsymbol{A}$ generates a new chain key (C_{i+1}) and encrypts the subsequent message using the freshly created message key (C_{i+1}).

[0047] The system design integrates protection against forward disclosure using the ratchet forward mechanism, which modifies the chain key (C_i) for every incoming message. This ensures that even if a session key can be compromised, the remaining data in the system can remain secure. The compromise can be confined to the information secured by the stolen key, typically affecting only one message. Forward secrecy ensures that if the current key can be discovered, the attacker cannot access messages from prior or subsequent sessions.

[0048] *Receiving a Secure Message*

[0049] Once $\boldsymbol{B}$ receives the encrypted message, it proceeds by decrypting it. Hence, to decrypt this message, $\boldsymbol{B}$ invokes the smart contract functions ($GetIdentityCert$ and) by providing $\boldsymbol{A's}$ ID. The smart contract returns (C_{IF}^{A}) and (C_{n}^{A}). $\boldsymbol{B}$ receives these certificates,

validates them, and extracts the identity key (I_{pub}^{B}) and the one-time key (E_{pub}^{B}). Now, $\boldsymbol{B}$ generates the same shared master secret key $\boldsymbol{S}$ as $\boldsymbol{A}$, leading to the same $\boldsymbol{M_0}$. Subsequently, he decrypts the message.

[0050] *Secure Backup*

[0051] The decentralized end-to-end encryption protocol can introduce a unique approach where users generate their backup decryption key (K) through a familiar secret, such as a password. The backup data can be encrypted using the M_i keys and stored on the user's instant messaging (IM) server, with M_i of those messages additionally encrypted using the user-generated key (K).

[0052] To retrieve their backups, users can access the IM server and decrypt the data using the K they established with a known secret, especially when transitioning between devices. FIG. 3 depicts the sequence diagram illustrating the backup and restoration process in the protocol.

[0053] A crypto-module can effectively encapsulate the intricacies of the E2EE protocol, offering a high level of abstraction that liberates developers to concentrate on their core application logic. By doing so, they can proceed with the confidence that the security and privacy standards provided by the E2EE solution can be robust and reliable. This strategic approach not only enriches the accessibility and usability of the protocol but also acts as a catalyst for its widespread adoption across a diverse array of messaging platforms.

[0054] The benefits of this user-friendly toolkit extend beyond convenience; they also serve as a catalyst for fostering more robust privacy protection within the digital communication landscape. As developers and organizations adopt the decentralized end-to-end encryption protocol more readily, users across the digital ecosystem can enjoy enhanced privacy and security while engaging in their online interactions. Consequently, the messaging landscape stands to become a safer and more secure space for individuals and businesses alike.

[0055] The crypto-module, whose architecture can be presented in FIG. 4, has a single object to interact with. The object can expose methods including:

[0056] Key Pairs Generation: The module generates cryptographic key pairs, consisting of a public key and a private key. The public key can be what gets certified and stored in the smart contract.

[0057] Certificate Generation: The module sends a request to the MSCP in order to create digital certificates for the generated public keys.

[0058] Certificate Retrieval by User: The crypto module includes a function that allows users to retrieve certificates stored in the smart contract.

[0059] Certificate Verification: This function allows users or applications to verify the authenticity and integrity of a certificate using the public key associated with it. The verification process typically involves checking the certificate's digital signature (if applicable) and confirming that the certificate has not been revoked.

[0060] Encryption: This function allows users to encrypt data using the recipient's certificate, ensuring that only the recipient with the corresponding can decrypt and access the information.

[0061] Decrypt: This function allows users to decrypt data that has been previously encrypted with their certificates. It transforms the ciphertext back into its original, human-readable format.

[0062] The blockchain can be employed to validate the exchanged key shares, adding an extra layer of security and transparency to the process. The verification mechanism for the secret key after its reconstruction ensures that the system maintains data integrity and that the key retrieval process can be reliable.

[0063] Furthermore, the system's use of bloom filters to select a subset of players for key generation can be a clever approach that optimizes the process and reduces overhead. Notably, the exact subgroup responsible for any particular key remains concealed, enhancing the protocol's security. Unlike alternative distributed key generation technologies such as

distributed VSS (verifiable secret sharing), which necessitate the active participation and share exchange among all players.

[0064] The Shamir Secret Sharing scheme can be presented below.

[0065] Consider a set of $N > 1$ players $P = \{Pi\}_{i\in}\{1, \ldots, N\}$. Assume that the player P_j holds a secret and wants to distributively store it in form of shares among all the other parties so that none of them can have access to the secret unless a threshold $0 \leq t \leq N$ of parties agree to restore it.

[0066] In the beginning, all the players agree on a field $\mathbb{F}$, for example, a Galois Field $\mathbb{Z}/p\mathbb{Z}$ with a generator g such that p a large prime. The player then transforms, via a one to one mapping, the secret to an element $s \in \mathbb{F}$, and generates the coefficients of a polynomial $f \in \mathbb{F}[X]$ of degree t such that $f(0) = s$, and then computes the vector $(s_i)_{i\in\{1,\ldots,N\}}$, such that $s_i = f(i)$ for any $i \in \{1, \ldots, N\}$.

[0067] Afterwards, a collaboration between $t + 1$ players at least can be required to recover the shared secret, which can be obtained through the Lagrangian interpolation. In other words, given any vector of shares $(s_i)_{i\in I}$, such that $I \subset \{1, \ldots, N\}$, and $|I| = t + 1, s$ can be obtained through:

$$s = f(0) = \sum_{i\in I} f(i) \prod_{j\in I-\{i\}} \frac{0-j}{i-j} = \sum_{i\in I} s_i \prod_{j\in I-\{i\}} \frac{j}{j-i}. \quad (1)$$

[0068] The Distributed Feldman Verifiable Secret Sharing (DF-VSS) can be presented below.

[0069] Assume that the players in P want to establish a pair of cryptographic keys: (1) a secret key $s \in \mathbb{Z}/p\mathbb{Z}$, that can be secretly shared between the players, which none of them can have access to unless a threshold of t players collaborate to restore it, and (2) a public key h such that $h = g^s \bmod p$ which can be safely exposed to the public.

[0070] First, all the players agree on the same field, i.e. the prime p and the generator g. Then each player P_i secretly generates $t + 1$ coefficients $a_{i,0}, a_{i,1}, \ldots, a_{i,t}$ following a

uniform distribution over $\mathbb{Z}/p\mathbb{Z}[X]$ using a cryptographic random number generator, and thus construct the following polynomial with degree t:

$$fi = \sum_{k=0}^{t} a_{i,k} X^k \in \mathbb{Z}/p\mathbb{Z}[X], \quad (2)$$

[0071] and publicly broadcasts $A_{i,k} = g^{a_{i,k}} \bmod p$ for $k \in \{1, \ldots, t\}$ to the players. Getting access to the secretly generated coefficients requires solving a discrete logarithm problem.

[0072] P_i prepares the shares $s_{i,j} = f_i(j) \bmod p$ for all $j \in 1, \ldots, N$, and sends the share $s_{i,j}$ to the player P_j through a secure communication channel.

[0073] The player P_j can verify the share received from any other player P_i by checking if:

$$g^{s_{i,j}} = \prod_{k=0}^{t} (A_{i,k})^{j^k} \, mod \, p. \quad (3)$$

[0074] If the condition can be not satisfied, P_j broadcasts a complaint against P_i. If more than t players broadcast a complaint against a player P_i, it can be banned.

[0075] Let $Q \subset \{1, \ldots, N\}$ be the subset of qualified players indices. The public value can be computed by $h = \prod_{i \in Q} A_{i,0} \bmod p$.

[0076] For each $i \in Q$, the player P_i computes the public verification values $A_k = \prod_{i \in Q} A_{i,k} \bmod p$ for all $k \in \{1, \ldots, t\}$, and his local share $s_i = \sum_{i \in Q} s_{j,i} \bmod p$. The secret key can be $s = \sum_{i \in Q} a_{i,0} \bmod p$, and it remains unknown to all the players from P.

[0077] The secret s can be the first coefficient of the polynomial ψ equal to the sum of the qualified players polynomials in the field $\mathbb{Z}/p\mathbb{Z}[X]$, i.e.

$$s = \psi(0) = \sum_{i \in Q} f_i(0) \; mod \; p. \tag{4}$$

[0078] Also, for any $i \in Q$:

$$\begin{aligned} s_i &= \sum_{j \in Q} f_i(j) \; mod \; p \\ &= \psi(j) \; mod \; p. \end{aligned} \tag{5}$$

[0079] The construction of s requires the coordination of at least $t + 1$ players form Q. For any subset $Q' \subset Q \; of \; t + 1$ elements, s can be obtained by

$$\begin{aligned} s = \psi(0) &= \sum_{i \in Q'} \psi(i) \prod_{j \in Q' - \{i\}} \frac{0 - j}{i - j} \; mod \; p \\ &= \sum_{i \in Q'} s_i \prod_{j \in Q' - \{i\}} \frac{j}{j - i} \; mod \; p. \end{aligned} \tag{6}$$

[0080] The Diffie-Hellman Key Exchange can enable two parties to securely establish a shared secret key over an open communication channel, even in the presence of potential eavesdroppers.

[0081] The protocol similarly relies on the mathematical concept of modular exponentiation in a finite field, e.g. $\mathbb{Z}/p\mathbb{Z}$, and its security relies on the difficulty of the discrete logarithm problem.

[0082] Assume that two players P_i and P_j from P want to establish a secret communication channel by exchanging a secret key s. P_i and P_j respectively generate two secrets a and b from the field. P_i computes $A = g^a \bmod p$ and publicly sends it to P_j, similarly, P_j computes $A = g^b \bmod p$ and sends it to P_i.

[0083] Afterwards, both players independently compute the shared secret key s: P_i computes $s = B^a \bmod p$, and P_j computes $s = A^a \bmod p$.

[0084] The remarkable property of the Diffie-Hellman key exchange can be that both parties compute the same shared secret key s without directly exchanging it: notice that $A^b \bmod p = g^{ab} \bmod p = B^a$. s can then be used for secure communication, such as encrypting and decrypting messages using symmetric-key cryptography.

[0085] A Blockchain can be a distributed ledger across a peer-to-peer connected network of nodes identified by their public-private pair keys $\mathcal{N} = \{(sk_1, pk_1), (sk_2, pk_2), \ldots\}$. It can be seen as a sequence of blocks $(b_1, b_2, \ldots)$, where each block b consists of: (1) a set of transactions t_i, (2) a reference $h(b_{i-1})$ that points to the previous block by a hash function h to maintain the chain order and (3) a cryptographic nonce determined through a consensus mechanism.

[0086] The Blockchain satisfies the following properties. (1) **Immutability**: For each block b_i, its cryptographic hash $h(b_i)$ remains constant once computed, preventing retroactive modifications. (2) **Consensus**: There exists a consensus mechanism $\mathcal{C}$ that ensures agreement among nodes $\mathcal{N}$ on the validity and order of transactions in t_i. (3) **Verification**: For each transaction in t_i send by a node (sk, pk), there exists a corresponding digital signature produced by sk, and verifiable with pk.

[0087] A Bloom filter can be a space-efficient probabilistic data structure particularly useful for membership testing, allowing a set of elements to be efficiently queried to determine if a specific element can be a member of the set.

[0088] Let Ω be a finite/countably finite set, an integer $m > 0$, and a set of $K > 0$ hash functions:

$$\mathcal{H} = \{h_i : \Omega \to \{1, \ldots, m\}\}_{i \in \{1,\ldots,K\}}. \tag{7}$$

[0089] A Bloom filter can be defined with size $m > 0$ and set of hash functions $\mathcal{H}$ as the map:

$$\begin{aligned} B_{\mathcal{H}} &: \mathcal{P}(\Omega) \to \{0,1\}^m \\ A &\mapsto B_{\mathcal{H}}(A) = ([B_{\mathcal{H}}(A)]_i)_{i \in \{1,\ldots,m\}} \end{aligned} \tag{8}$$

[0090] where $\mathcal{P}(\Omega)$ can be the power set of Ω, and $[B_{\mathcal{H}}(A)]_i$ can be the value of the bit at the i-th index in the array $B_{\mathcal{H}}(A)$ for all $A \in \mathcal{P}(\Omega)$, that is:

$$[B_{\mathcal{H}}(A)]_i = \begin{Bmatrix} max\{\delta_{i,h(w)}\} : \omega \in A, h \in \mathcal{H} \\ 0 \end{Bmatrix} \begin{matrix} if\ A \neq \emptyset \\ otherwise \end{matrix} \quad (9)$$

[0091] for all $i \in \{1, \ldots, m\}$.

[0092] **Membership:** Let $\omega \in \Omega, A \subset \Omega$ with $A \neq \emptyset$ and $B_{\mathcal{H}}$ a Bloom filter with size m and set of hash function $\mathcal{H}$.

[0093] To test the membership of ω to A, the images of ω can be computed by the hash functions in $\mathcal{H}$ and test whether the bits at the indices corresponding to these images can be all 1.

[0094] However, *false positive* matches can occur, i.e. the test may be positive for some elements of Ω on $B(A)$ even if $\omega \notin A$, but *false negative* cannot, i.e. if the membership of an element $\omega \in \Omega$ can be negative on $B(A)$, then $\omega \notin A$.

[0095] These two points can be summarized on the following theorem:

[0096] **Lemma 1.** *For all* $\omega \in \Omega, \omega \in A \Longrightarrow [B_{\mathcal{H}}(\mathrm{A})]_{h(w)} = 1, \forall h \in \mathcal{H}$.

[0097] *Proof.* If $\omega \in A$ and A can be not empty, then for all $h \in \mathcal{H}$

$$\begin{aligned} [B_{\mathcal{H}}(A)]_{h(w)} &= max\left\{\delta_{h(\omega),h'(\omega')} : \omega' \in A, h' \in \mathcal{H}\right\} \\ &= max\left(\left\{\delta_{h(w),h(\omega')}\right\} \cup \left\{\delta_{h(\omega),h'(\omega')} : \omega' \in A - \{\omega\}, h' \in \mathcal{H} - \{h\}\right\}\right) \\ &= max\left(\{1\} \cup \left\{\delta_{h(\omega),h'(\omega')} : \omega' \in A - \{\omega\}, h' \in \mathcal{H} - \{h\}\right\}\right) \\ &= 1 \end{aligned}$$

[0098] since the values of the Kronecker deltas in the second set can be either 0 or 1.

[0099] **Bloom filter of a union:** A Bloom filter $B_{\mathcal{H}}$ can be constructed with set of hash functions $\mathcal{H}$, of the union of two subsets A and B of Ω by performing the logical OR operation on the bits of the filters of the two sets.

[0100] **Lemma 2.** $\forall A, B \in \mathcal{P}(\Omega), \forall i \in \{1, \ldots, m\}, [B_{\mathcal{H}}(A \cup B)]_i = [B_{\mathcal{H}}(A)]_i \vee [BH(B)]i.$

[0101] *Proof.* Let $A, B \in \mathcal{P}(\Omega)$,

[0102] If both sets can be empty, the result can be trivial.

[0103] If only one can be empty, for example B, for all $i \in \{1, \ldots, m\}$:

$$\begin{aligned}[B_{\mathcal{H}}(A \cup B)]_i &= [B_{\mathcal{H}}(A)]_i \vee [B_{\mathcal{H}}(B)]_i \\ &= [B_{\mathcal{H}}(A)]_i \vee 0 \\ &= [B_{\mathcal{H}}(A)]_i\end{aligned}$$

[0104] which holds true since $A \cup B = A$ in this case.

[0105] Otherwise, after writing the rewriting the OR operation $\vee$ of two bits as the the max of their values:

$$\begin{aligned}[B_{\mathcal{H}}(A)]_i \vee [B_{\mathcal{H}}(B)]_i &= max\,[B_{\mathcal{H}}(A)]_i, [B_{\mathcal{H}}(B)]_i \\ &= max\left(max\{\delta_{i,h(\omega)} : \omega \in A, h \in \mathcal{H}\}, max\{\delta_{i,h'\omega'} : \omega' \in B, h' \in \mathcal{H}\}\right) \\ &= max\left(\{\delta_{i,h(\omega)} : \omega \in A, h \in \mathcal{H}\} \cup \{\delta_{i,h'\omega'} : \omega' \in B, h' \in \mathcal{H}\}\right) \\ &= max\,\{\delta_{i,h(\omega)} : \omega \in A \cup B, h \in \mathcal{H}\} \\ &= [B_{\mathcal{H}}(A \cup B)]_i\,,\end{aligned}$$

[0106] which terminates the proof.

[0107] Insertion of Elements to a Bloom Filter: The previous theorem gives a rule for (practical) construction of a Bloom filter of a nonempty set given its elements.

[0108] **Corollary 1.** A Bloom filter of size $m > 0$ and set of hash functions $\mathcal{H}$ of a subset of $n > 0$ elements $A = \{a_j\}_{j\in\{1,\dots,n\}} \subset \Omega$ can be constructed by:

$$[B_{\mathcal{H}}(A)]_i = [B_{\mathcal{H}}(\{a_1\})]_i \vee \dots \vee [B_{\mathcal{H}}(\{a_1\})]_i, \forall i \in \{1,\dots,m\}. \quad (10)$$

[0109] **Bloom Filter of an Intersection.** Let $A, B \subset \Omega, and\ B_{\mathcal{H}}$ a Bloom filter with size $m > 0$ and set of hash functions $\mathcal{H}$. A Bloom filter to test the membership of an element $\omega \in \Omega$ to the intersection $A \cap B$ can be constructed by the logical AND operation on the bits of $B_{\mathcal{H}}(A)$ and $B_{\mathcal{H}}(B)$. Such filter can be denoted as follows: $B_{\mathcal{H}}(A) \wedge B_{\mathcal{H}}(B) = ([B_{\mathcal{H}}(A)]_i \wedge [B_{\mathcal{H}}(B)]_i)_{i\in\{1,\dots,n\}}$.

[0110] **Lemma 3.** $\forall\omega \in \Omega, \forall A, B \subset \Omega, \omega \in A \cap B \Longrightarrow [B_{\mathcal{H}}(A) \wedge B_{\mathcal{H}}(B)]_{h[w]} = \mathbf{1}, \forall h \in \mathcal{H}$.

[0111] Proof. Let $\omega \in \Omega$ and $A, B \subset \Omega$ be:

$$w \in A \cap B \Longrightarrow [B_{\mathcal{H}}(A)]_{h(w)} = 1\ and\ [B_{\mathcal{H}}(B)]_{h(w)} = 1, \forall h \in \mathcal{H}$$
$$\Longrightarrow [B_{\mathcal{H}}(A) \wedge B_{\mathcal{H}}(B)]_{h(w)} = 1, \forall h \in \mathcal{H}$$

[0112] which terminates the proof.

[0113] **Corollary 2.** Let $A_1, A_2, \dots, A_n$ be a family of subsets of Ω, for all $\omega \in \Omega$

$$w \in \bigcap_{i=1}^{n} A_i \Longrightarrow [B_{\mathcal{H}}(A_1) \wedge \dots \wedge B_{\mathcal{H}}(A_n)]_{h(w)} = 1, \forall h \in \mathcal{H}. \quad (11)$$

[0114] **Preimage of a Bloom Filter:** Let Ω be a universal set, $A \subset \Omega$? and $B_{\mathcal{H}}$ a Bloom filter of size $m > 0$ and a nonempty set of hash functions $\mathcal{H}$.

[0115] Let $\widetilde{\mathcal{H}} \subset \mathcal{H}$ and $\widetilde{K} = |\widetilde{\mathcal{H}}|$.

[0116] **Definition 2.** The set can be called:

$$\{\omega \in \Omega : [B_{\mathcal{H}}(A)]_{h(w)} = 1 \forall h \in \widetilde{\mathcal{H}}\} \tag{12}$$

[0117] the preimage of $B_{\mathcal{H}}(A)$ under $\widetilde{\mathcal{H}}$, and it can be denoted as $\langle B_{\mathcal{H}}(A)\rangle_{\widetilde{\mathcal{H}}}$.

[0118] **Lemma 4.** $\forall A \subset \Omega, A \subset \langle B_{\mathcal{H}}(A)\rangle_{\widetilde{\mathcal{H}}}$.

[0119] *Proof.* Easy! By applying the proof of Lemma 1.

[0120] The previous property can be trivial but it clearly illustrates one of the fundamental properties of a Bloom filter, that is, the set of the elements with the positive membership test , given the array $B_{\mathcal{H}}(A)$, contains A, but it may be larger than it because of the presence of *false positives*, i.e. when $\langle B_{\mathcal{H}}(A)\rangle_{\widetilde{\mathcal{H}}} - \{A\} \neq \emptyset$.

[0121] Another useful property in the formalism of the decentralized end-to-end encryption protocol can be derived from the previous lemma .

[0122] **Corollary 3.** For any family of $n > 0$ subsets $A_1, \ldots, A_n \subset \Omega$:

$$\left|\bigcup_{i=1}^{n} A_i\right| \leq |\langle B_{\mathcal{H}}(A_1) \wedge \ldots \wedge B_{\mathcal{H}}(A_n)\rangle_{\mathcal{H}}| \tag{13}$$

[0123] In a dynamic scenario, end users can initiate a request to create a pair of public and private keys. The private key can serve to establish a secure communication channel between end users. The public key may remain accessible to any node.

[0124] In the dynamic version of the decentralized end-to-end encryption protocol, random keys can be generated during various sessions. These keys can be typically not employed directly for message encryption but rather to update a key derivation function. This function, in turn, generates a distinct key for each message within a specific session, ensuring both forward and backward secrecy. This mechanism can be inspired from the double ratchet algorithm. But in contrast to its traditional version the key derivation function updates through a Diffie-Hellman key exchange that necessitates trust in a server for communication, the decentralized end-to-end encryption protocol can use the secret key it generates. This approach can be notably less susceptible to man-in-the-middle attacks.

[0125] The decentralized end-to-end encryption protocol can consist of a set of active players denoted by $\mathcal{P} = \{P_i\}_{i\in\{1,\dots,N\}}$. It can be assumed that the number of active players remains fixed, although it may vary over time and across different sessions.

[0126] Communication among the players occurs through two distinct channels. (1) A public communication channel: When a player broadcasts a message, it reaches all other players. (2) Secure communication channels: These channels exist between any two players, such as P_i and P_j, and their security level can be determined by the cryptographic primitives upon which they can be built.

[0127] A Blockchain can be constructed across $\mathcal{P}$. It can be assumed that each participant, i.e. player $P_i \in \mathcal{P}$, can initiate a transaction involving a piece of data D, denoted as $T_{P_i}(D)$, through a predefined consensus mechanism. This transaction can be openly recorded on the Blockchain, and any node, not limited to the players, can have the ability to verify it.

[0128] The entities interacting with the decentralized end-to-end encryption protocol (e.g., the players and the end users) reach consensus on a large prime p and a generator g of a Galois field $\mathbb{Z}/p\mathbb{Z}$.

[0129] The players reach consensus on the size m of a Bloom filter and a set of K hash function $\mathcal{H} = \{h_1, h_2, \dots, h_K\}$, s.t. for any $i \in \{1, \dots, K\}$

$$h_i : \mathbb{Z}/p\mathbb{Z} \longrightarrow \{1, \dots, m\} \tag{14}$$

[0130] Each player can be identified by a unique ID, which can be universally known by any external entity, importantly the other players and end users. The identifier of a player P_i can be any number form $\mathbb{Z}/p\mathbb{Z}$ under the constraint that $p > N$. For simplicity, each player $P_i \in \mathcal{P}$ can be identified by its index i in the set.

[0131] Any external user can request and build a secure communication channel with any player from $\mathcal{P}$. There exists another mechanism to assign a distinct and universally

recognized identifier, denoted as $Id_{\mu_0}0$, to each end user μ_0. These identifiers can be known to all end users, allowing them to recognize each other.

[0132] The players agree on a hash function $\sigma^{(.)}(.)$ that map the indices (actual public identifiers) of the players to strictly ordered integers. Updates by a value $x \in \mathbb{Z}/p\mathbb{Z}$, it maps any input $y \in \{1,\ldots,n\}$ to produce an output $\sigma^{(x)}(y)$ in the same set, satisfying the requirements of a cryptographic hash function.

[0133] Consider an end user μ_0 wanting to establish a shared secret key with a set of other end users $\{\mu_1, \mu_2, \ldots, \mu_n\}$ via decentralized end-to-end encryption protocol. μ_0 sends a request to an honest player $P_{req} \in \mathcal{P}$ specifying the other users identifiers $Id_{\mu_1}, Id_{\mu_2}, \ldots, Id_{\mu_n}$, and a treshhold $t > 0$ of the desired least number of players required to reconstruct the key. The value of t can be chosen to balance between the required degree of security and the communication overhead.

[0134] The player creates a Blockchain transaction containing a generated identifier for the communication session τ along with the identifiers of the users that can get the secret key through the decentralized end-to-end encryption protocol (e.g.,$Id_{\mu_0}, Id_{\mu_1}, \ldots, Id_{\mu_n}$) and the threshold τ. Therefore, the player creates $T_{P_{req}}((\tau, t, Id_{\mu_0}, Id_{\mu_1}, \ldots, Id_{\mu_n}))$.

[0135] Therefore the player broadcasts, using the public communication channel, a message requesting the other players the beginning of the key generation process for the session identified by τ. The identity of the legitimate end users can be checked by the Blockchain transaction $T_{P_{req}}((\tau, t, Id_{\mu_0}, Id_{\mu_1}, \ldots, Id_{\mu_n}))$.

[0136] After receiving the key generation request message for a session τ, each player $P_i \in \mathcal{P}$ (including the broadcaster P_{req}) extracts the requesting end users identifiers and the threshold τ. These values can be verified from the transaction $T_{P_{req}}((\tau, t, Id_{\mu_0}, Id_{\mu_1}, Id_{\mu_2}, \ldots))$ and the public key of P_{req}. Afterwards, each player P_i performs the following steps:

- P_i randomly generates, using a cryptographic random numbers generator, through a uniform distribution over the field $\mathbb{Z}/p\mathbb{Z}$, $t+1$ coefficients $a_{i,0}^{(\tau,0)}, a_{i,1}^{(\tau,0)}, \ldots, a_{i,1}^{(\tau,0)}$ and prepares the polynomial:

$$f_i^{(\tau,0)} = \sum_{k=0}^{t} a_{i,k}^{(\tau,0)} X^k \in \mathbb{Z}/p\mathbb{Z}[X] \quad (15)$$

- P_i computes the values $s_{i,1}^{(\tau,0)}, s_{i,2}^{(\tau,0)}, \ldots, s_{i,N}^{(\tau,0)}$, such that for all $j \in \{1, \ldots, N\}$

$$s_{i,j}^{(\tau,0)} = f_i^{(\tau,0)}(j) \; mod \; p. \quad (16)$$

- P_i randomly chooses a subset of players' indices $V_i^{(\tau)} \subset \{1, \ldots, N\}$ such that:

$$|V_i^{(\tau)}| \geq \left\lceil \frac{N+1}{2} \right\rceil. \quad (17)$$

- P_i inserts the values corresponding to the chosen identifiers in a Bloom filter $b_i^{(\tau,0)}$ with size m and set of hash functions $\mathcal{H}$:

$$b_i^{(\tau)} = B_H\left(\left\{s_{i,1}^{(\tau,0)}\right\} j \in V_i^{(\tau,0)}\right). \quad (18)$$

[0137] P_i computes the public values $A_{i,0}^{(\tau,0)}, \ldots, A_{i,t}^{(\tau,0)}$ such that for all $j \in \{1, \ldots, N\}$

$$A_{i,k}^{(\tau,0)} = g^{a_{i,k}^{(\tau,0)}} \; mod \; p. \quad (19)$$

[0138] P_i creates a Blockchain transaction for the Bloom filter $T_{P_i}(b_i^{(\tau,0)})$ and another one for the public values $T_{P_i}(A_{k,1}^{(\tau,0)}, \ldots, A_{i,N}^{(\tau,0)})$.

[0139] P_i secretly sends to each player $P_j \in P - \{P_i\}$, using the secure communication channel, the value $s_{i,j}^{(\tau,0)}$.

[0140] P_i broadcasts using the public communication channel the values $A_{k,1}^{(\tau,0)}, \ldots, A_{i,N}^{(\tau,0)}$ and $b_i^{(\tau,0)}$.

[0141] After completing the previous steps, each player P_i waits to receive the secret shares, the verification values, and the Bloom filters from the other players, and proceeds the following steps.

[0142] Each player $P_j \in P - \{P_i\}, P_i$, verifies the received values $A_{j,k}^{(\tau,0)}$, for all $k = \{1, \ldots, k\}$, through the Blockchain transaction $T_{P_i}(A_{k,1}^{(\tau,0)}, \ldots, A_{i,N}^{(\tau,0)})$, as well as the Bloom filter $b_j^{(\tau,0)} via\ T_j(b_j^{(\tau,0)})$.

[0143] P_i verifies the received shares from each player $P_j \in P - \{P_i\}$ through:

$$g^{s_{i,k}^{(\tau,0)}} mod\ p = \prod_{k=1}^{N} (A_{i,k}^{(\tau,0)})^{j^k} mod\ p. \quad (20)$$

[0144] If the test fails, P_i broadcasts a complaint message against P_j to the other players.

[0145] The players wait for the reception of the complaint messages from the others, once received, each player P_i proceeds the following,

[0146] If P_i received from the other players more than $\frac{N+1}{2}$ complaints against a player P_j, including its own, i.e. the majority complained against it, P_i excludes it and proceeds the following steps.

[0147] It can be assumed that the set of non-disqualified players can be common among all the remaining players, $Q^{(\tau,0)}$ can denote the set of qualified players indices.

[0148] P_i aggregates the Bloom filters of the qualified players by the logical AND operation to construct the participants Bloom filter $b_Q^{(\tau,0)}$ given by

$$b_Q^{(\tau)} = \bigvee_{j \in Q^{(\tau,0)}} b_j^{(\tau)}. \tag{21}$$

[0149] P_i computes the public value $h^{(\tau,0)}$ from the public values already prepared in step 4.3:

$$h^{(\tau,0)} = \prod_{k=1}^{N} A_{0,k}^{(\tau,0)} \ mod\ p. \tag{22}$$

[0150] P_i prepares the one-way function $\sigma^{(h^{(\tau,0)})}$ and computes the indices $\sigma^{(h^{(\tau,0)})}(j)$ for every $j \in Q^{(\tau,0)}$. Then P_i sorts these values into an array that can be denoted by

$$T_{P_i}^{(\tau)} = \left(\varsigma_1^{(\tau)}, \varsigma_2^{(\tau)}, \ldots, \varsigma_{|Q^{(\tau,0)}|}^{(\tau)}\right) \tag{23}$$

[0151] An important property about the previously constructed array can be that it can be common among all the other players. The same function $\sigma^{(h^{(\tau,0)})}$ can be used in the construction of any players, and the set of qualified players $Q^{(\tau,0)}$ can be common as well, consequently:

$$T_{P_i}^{\tau} = T_{P_j}^{\tau} \forall i, j \in Q^{(\tau,0)} \tag{24}$$

[0152] It can be noted that the number of end users specified in the first key generation request can be less than half of the number of players. This assumption can be practically evident. In addition, the order at which these end users was specified in the first key generation request remains preserved and publicly known $\mu_0, \mu_1, \ldots, \mu_n$.

- P_i determines the indices of the players j_1 and j_2 such that:

$$\varsigma_1^{(\tau)} = \sigma^{(h^{(\tau,0)})}(j_1) \text{ and } \varsigma_1^{(\tau)} = \sigma^{(h^{(\tau,0)})}(j_2) \tag{25}$$

[0153] if the membership of both $S_{j_{1,i}}^{(\tau)}$ and $S_{j_{2,i}}^{(\tau)}$ of the Bloom filter $b_Q^{(\tau)}$ using the hash functions in $\mathcal{H}$ can be positive, then P_i *can be* selected to be responsible to send the shares to μ_0.

[0154] In other words, if the players P_{j1} and P_{j2} both voted on P_i, i.e. $i \epsilon V_{j_1}^{(\tau)}$ and $i \epsilon V_{j_2}^{(\tau)}$, then:

$$S_{i,j_1}^{(\tau)} \in \langle b_{j_1}^{(\tau)} \rangle_{\mathrm{H}} \text{ and } S_{i,j_2}^{(\tau)} \in \langle b_{j_2}^{(\tau)} \rangle_{\mathrm{H}} \tag{26}$$

[0155] and therefore by exploiting preimage and union properties of Bloom filters:

$$S_{i,j_1}^{(\tau)}, S_{i,j_2}^{(\tau)} \in \langle b_Q^{(\tau,0)} \rangle_{\mathrm{H}}, \tag{27}$$

[0156] and thus both membership tests of $S_{i,j_1}^{(\tau)}$ and $S_{i,j_2}^{(\tau)}$ to $b_Q^{(\tau,0)}$ can be positive.

[0157] P_i performs the same operation for the other users $\mu_1, \ldots, \mu_n$. More specifically, for each user μk, the player determines the players indices j_{k+1} and j_{k+2}corresponding to $\varsigma_1^{(\tau)}$ and $\varsigma_1^{(\tau)}$ as above, and decides if it can be responsible for sending the local 1 1 shares to the user μ_k by testing the membership of $S_{j_{k+1,i}}^{(\tau)}$ and $S_{j_{k+2,i}}^{(\tau)}$ to $b_Q^{(\tau)}$ using the hash functions in $\mathcal{H}$.

[0158] As a result, P_i determines the subset of users for whom it can be responsible to send the secret key shares for the communication channel identified by τ. The set of their indices can be denoted by

$$U_{P_i}^{(\tau)} = \left\{ k \in \{0, \ldots, n\} : S_{i,j_1}^{(\tau)}, S_{i,j_2}^{(\tau)} \in \langle b_Q^{(\tau,0)} \rangle_H \, s.t. \begin{cases} \sigma^{(h^{(\tau,0)})}(J_1) = \varsigma_{k+1}^{(\tau)} \\ \sigma^{(h^{(\tau,0)})}(J_{1S}) = \varsigma_{k+2}^{(\tau)} \end{cases} \right\} \tag{28}$$

[0159] $P_{\mu_k}^{\tau}$ the subset of players responsible for sending the secret shares to the user μ_k can be denoted by $k = 0, \ldots, n$

$$P_{\mu_k}^{(\tau)} = \left\{ i \in Q^{(\tau,0)} : S_{,j_{1,i}}^{(\tau)}, S_{j_{2,o}}^{(\tau)} \in \langle b_Q^{(\tau,0)} \rangle_H \, s.t. \begin{Bmatrix} \sigma^{(h^{(\tau,0)})}(j_1) = \varsigma_{k+1}^{(\tau)} \\ \sigma^{(h^{(\tau,0)})}(j_2) = \varsigma_{k+2}^{(\tau)} \end{Bmatrix} \right. \tag{29}$$

[0160] **Optimal Number of Votes:** The number of selected players to be responsible for the construction of the secret key can exceed the threshold $t + 1$. Hence, for all $k \in \{0, \ldots, n\}$:

$$|P_{\mu_k}^{\tau}| \geq t + 1 \tag{30}$$

[0161] **Lemma 5**. *The optimal number of votes that the players can make, so that the number of players that can send the shares to each user exceeds the threshold t, can be:*

$$\left\lfloor \frac{|Q^{(\tau,0)}| + t + 2}{2} \right\rfloor$$

[0162] *Proof*:

$$v_k = \left\{ i \in Q^{(\tau,0)} : S_{,j_{1,i}}^{(\tau)} \in \langle b_Q^{(\tau,0)} \rangle_H \, s.t. \left\{ \sigma^{(h^{(\tau,0)})}(j) = \varsigma_{k+1}^{(\tau)} \right\} \right. \tag{31}$$

[0163] It can be observed that:

$$P_k^{\tau} = v_k \cap v_{k+1} \tag{32}$$

[0164] and, by exploiting the preimage property of Bloom filters, that:

$$V_j^{(\tau)} \cap Q^{(\tau,0)} \subset v_k \text{ with } \sigma^{(h^{(\tau,0)})}(j) = \varsigma_{k+1}^{(\tau)} \tag{33}$$

[0165] Thus:

$$V_j^{(\tau)} \cap V_j^{(\tau)} \cap Q^{(\tau,0)} \subset v_k \cap v_{k+1} \text{with} \begin{cases} \sigma^{(h^{(\tau,0)})}(j)=\varsigma_{k+1}^{(\tau)} \\ \sigma^{(h^{(\tau,0)})}(j')=\varsigma_{k+2}^{(\tau)} \end{cases}. \tag{34}$$

[0166] Afterwards, for such numbers j and j′ can be after simple arithmetic:

$$\begin{aligned} |P_k^{\tau}| = |v_k \cap v_{k+1}| &\geq |\left(V_j^{(\tau)} \cap Q^{(\tau,0)}\right) \cap \left(V_{j'}^{(\tau)} \cap Q^{(\tau,0)}\right)| \\ &\geq 2|V_j^{(\tau)}| - |Q^{(\tau,0)}|. \end{aligned} \tag{35}$$

[0167] Because it can be shown that for any two subset A and B of a finite set C, s.t. |A| = |B|, it can be $|A \cap B| \geq 2|A| - |C|$. That can be because $|A \cap B|$ =|A| + |B|−$|A \cup B| = 2|A| - |A \cup B| \geq 2|A| - |C|$, since $A \cup B \subset C$.

[0168] Afterwards

$$2\left|V_j^{(\tau)}\right| - \left|Q^{(\tau,0)}\right| \geq t+1 \Rightarrow \left|V_j^{(\tau)}\right| \geq \frac{\left|Q^{(\tau,0)}\right| + t + 1}{2} \tag{36}$$

[0169] the minimum number of votes that satisfies the inequality can be:

$$\left|V_j^{(\tau)}\right| = \frac{\left|Q^{(\tau,0)}\right| + t + 1}{2} \tag{37}$$

[0170] and this for all $j \in Q^{(\tau,0)}$, concluding the proof.

[0171] **Local Shares Construction:** P_i constructs the local share of the session identified by τ , denoted by $s_i^{(\tau,0)}$, by summing the received shares from the qualified players:

$$s_i^{(\tau,0)} = \sum_{j \in Q^{(\tau,0)}} s_{i,j}^{(\tau,0)} \tag{38}$$

[0172] P_i can send to the end users $\{u_k\}_{k\in U_{p_i}^{(\tau)}}$ the share $s_i^{(\tau,0)}$.

[0173] Each user μ_k with $k = 0,\ldots,n$ can reconstruct the secret key from the shares sent to it by the players in $P_{\mu_k}^{(\tau)}$.

[0174] μ_k receives the shares $\{s_i^{(\tau,0)}\}_{i\in P_{\mu_k}^{\mu_k}}$

[0175] μ_k verifies each share $s_i^{(\tau,0)}$ by checking:

$$g_i^{s^{(\tau,0)}} mod\ p = \prod_{j\in Q^{(\tau,0)}} \prod_{l=0}^{t} (A_{i,j}^{(\tau,0)})^{i^l} \text{mod p} \tag{39}$$

[0176] This requires knowledge of the set of qualified players, which can be obtained either from publicly broadcast complaints during the verification of shares or simply obtained from trusted player(s) to minimize overhead on the end users.

[0177] μ_k constructs the symmetric key of the session τ after selecting a subset $R \subset P^{\tau}$ of $t+1$ elements:

$$s_i^{(\tau,0)} = \sum_{i\in P^{(\tau)}} s_{i,j}^{(\tau,0)} \prod_{j\in P^{(\tau)}-\{i\}} \frac{j}{j-i}\ mod\ p. \tag{40}$$

[0178] Note that $s^{(\tau,0)}$ can also be optionally verified by checking whether $g^{s^{(\tau,0)}}$ mod p can be congruent to $h^{(\tau,0)}$, where $h^{(\tau,0)}$ can be obtained in the same manner as described in 4.5.

[0179] A significant feature of this protocol can be its capability to facilitate key recovery in case users lose their generated key. This feature eliminates the necessity for permanent storage of secret keys on the device, reducing the risk of key loss, especially when considering the physical storage of the device as a potential vulnerability.

[0180] If a user μ_k wants to recover the key $s^{(\tau)}$ it can perform the fallowing steps.

[0181] μ_k requests a neighboring/trusted player to recover the key specifying its identity Id_{μ_k} and the identifier of the key (τ).

[0182] The player broadcasts the request to the other players.

[0183] Each player P_i checks if k belongs to $U_{P_i}^{(\tau)}$, as well as the identity of the requester.

[0184] If P_i can be responsible for μ_k, then it can send to it the share $s^{(\tau)}$ through the secure communication channel.

[0185] μ_k can then receive the shares $\{s_i^{(\tau,0)}\}_{i\in P_{\mu_k}^{(\tau)}}$, verify them, and reconstruct the key;

[0186] $s^{(\tau)}$ as described in the key construction process.

[0187] **Generation of Subsequent Secret Keys:** In certain scenarios, the initial end user, or generally any user μ_k in the group that shares $s^{(\tau,0)}$, may request establishing additional shared secret keys with the same group of users during the ongoing communication session τ. The decentralized end-to-end encryption protocol can introduce an advantageous feature in this context, simplifying and reducing the cost of generating subsequent keys $s^{(\tau,1)}$, $s^{(\tau,2)}$, . . . in comparison to the initial key $s^{(\tau,0)}$.

[0188] Specifically, the generation of the keys $s^{(\tau,1)}$, $s^{(\tau,2)}$, , . . . does not require the selection of a new subset of responsible players for each user. Instead, the previous subset of players $P_{\mu_k}^{(\tau)}$ remains unchanged unless a user explicitly requests the creation of a new key for another session τ'. As a result, this approach minimizes operational overhead, thereby reducing the need for activities such as voting, creation of the Bloom filters, and the corresponding transactions.

[0189] For this purpose, in response to a request for creating a key $s^{(\tau,q)}$, , with $q > 0$, the process goes as follows; let $P^{\tau} = = U_{k=0}^{n} P_{\mu_k}^{(\tau)}$ denote the subset of players that were selected for τ , i.e. during the generation of the first key.

[0190] Each player P_i generates again t coefficients $a_{i,0}^{(\tau,q)},\ldots,a_{i,t}^{(\tau,q)}$ to prepare a new polynomial $f_i^{(\tau,q)}=\sum_{k=0}^{t}a_{i,t}^{(\tau,q)}X^k$to be publicly broadcast with the corresponding transaction $T_{P_i}(\left(A_{i,0}^{(\tau,q)},\ldots,A_{i,t}^{(\tau,q)}\right))$, and finally sends to each player P_j through the secure communication channel the share $s_{i,j}^{(\tau,q)}=f_i^{(\tau,q)}(j)\bmod p$.

[0191] P_i checks the received shares using the public values, determines after the complaining phase the set of qualified players $Q^{(\tau,q)}$ and computes the local share $s_i^{(\tau,q)}=\sum_{j\in Q^{(\tau,q)}}s_{i,j}^{(\tau,q)}$ and secretly sends to all the users in $U_{P_i}^{(\tau,0)}$.

[0192] Each user μ_k receives the shares:

$$\{s_i^{(\tau,q)}\}_{i\in P_{u_k}^{\tau}\cap Q^{(\tau,q)}} \tag{41}$$

[0193] The condition $|P_{\mu_k}^{(\tau)}\cap Q^{(\tau,q)}|$ can be satisfied to proceed the key construction. μ_k can finally construct the secret key $s^{(\tau)}$ based on received shares. The recovery of the key $s^{(\tau,0)}$ can be performed using the same process explained earlier.

[0194] The decentralized end-to-end encryption protocol can have three building blocks: i) the DF-VSS distributed key generation protocol, ii) The distributed ledger (Blockchain), and iii) the use of Bloom filters to select the subsets of players responsible for generating subsequent keys in a session τ and sending the shares of the secret key to the corresponding users. The DF-VSS protocol can ensure that the cryptographic key shares can be verifiable by the nodes without revealing the actual shares to unauthorized parties.

[0195] The security of the used consensus mechanism to create the transactions, the security of solving the discrete logarithm problem, the security of the hash functions used in the insertion to the Bloom filters, i.e. the functions in the set $\mathcal{H}$, and the function $\sigma^{(.)}(.)$ in term of their satisfaction of the requirements of Pseudorandom Function (PRF) can be assumed.

[0196] The protocol offers flexibility to be adapted to various use cases. The decentralized end-to-end encryption protocol can be applied in any context where the generation of a pair of public and private keys can be required, with the following advantages:

[0197] Decentralization: In contrast to systems that rely on a single entity to generate the key pair, the key generation process in the decentralized end-to-end encryption protocol can be done in a distributed manner, the public key can be safely and publicly exposed, but no entity can access to the private key, and a collaboration of at least a threshold $t + 1$ of system entities can be required for its reconstruction.

[0198] Complete verifiability: This allows for the detection of any malicious activity during and after the process, ensuring the integrity of the generated keys through the use of Blockchain technology.

[0199] Key storage and distribution: Any public key $h^{(\tau,0)}$ and its corresponding identity to the entities requiring it can be publicly exposed and can be verified by any entity through the Blockchain transactions of the players on the publicly broadcast values from which it can be derived, i.e. the values $A_{i,0}^{(\tau,0)}$ for any $i \in Q^{(\tau,0)}$.

[0200] Easy recovery of private keys: Owners can restore a lost private key by requesting it again.

[0201] The proposed protocol can be used to create secure communication channel between two users Alice and Bob. The following outlines the proposed system design.

[0202] The decentralized end-to-end encryption protocol can function as the key agreement protocol within the messaging system, enabling the creation of shared secret keys between Alice and Bob. These keys can not be employed directly for symmetrically encrypting the exchanged messages but can serve a crucial role in the Double Ratchet algorithm. This usage in the Double Ratchet algorithm provides a two-fold benefit in terms of security: forward secrecy and backward secrecy. The double ratchet algorithm can

provide forward secrecy and cryptographic ratcheting, ensuring the security of message exchanges over time.

[0203] Forward secrecy ensures that past messages remain confidential even if current keys can be compromised. In other words, if an adversary succeeds to eavesdrop $t + 1$ secure communication channels between Alice and the players responsible for it, and thus compromised a certain secret key $s^{(\tau,0)}$, the adversary can not be able, using that key, to decipher the past messages exchanged between Alice and Bob during the session τ. With τ in this context identifies the messaging session between these two users.

[0204] Backward secrecy guarantees that compromising future keys can not compromise the secrecy of past messages.

[0205] It can be the fact that there can be a mechanism to ensure that a user, e.g. Alice, can be the true owner of an identifier Id_{Alice} by which it can be recognized by the players responsible for it, i.e. $P_{\mu}^{(\tau)}$ when they can send to it the shares to construct the future secret keys $s^{(\tau,0)}$, $s^{(\tau,1)}$, . . . through the secure communication channel they can build with her.

[0206] Furthermore, there can be the risk of an entity, like Charlie, impersonating Bob by falsely claiming to be the owner of the identifier Id_{Bob}. Consequently, when Alice requests the generation of keys for communication session τ , she explicitly specifies that the keys, $s^{(\tau,0)}$, $s^{(\tau,1)}$, . . ., can be generated for exchange with a user identified as Id_{Bob}. While the players can verify this through Blockchain transactions, there can be no guarantee that the key shares can be securely sent to Bob by the subset of players responsible for him. In this context, Charlie could potentially execute a man-in-the- middle attack.

[0207] To address the mentioned problems the following system design can be introduced.

[0208] **User registration and identity binding**. The communicating users can first create accounts in the system. During this process, each user μ can generate a pair of private/public cryptographic keys sk_{μ} and pk_{μ}. These keys can typically be used for

establishing secure communication channels between the user and the decentralized end-to-end encryption protocol players. To ensure trust and security, each user's public key sk_{μ} can be associated with a digital certificate. These certificates can be typically issued by trusted Certificate Authorities (CAs) and bind the user's public key to its identifier Id_{μ}, providing authentication and preventing impersonation. This process can be often used in conjunction with the HTTPS protocol, and more specifically, TLS. When users connect to the system using HTTPS/TLS, their certificates can be checked to confirm the authenticity of the public keys, allowing for encrypted and authenticated communication between users and the system.

[0209] Rather than depending on a centralized Certificate Authority (CA), the association between pk_{μ} and Id_{μ} can be established via a smart contract involving the system's participants, and the confirmation of identity can be accomplished by examining the multi-signed transaction linked to user registration.

[0210] **Key agreement**. The key exchange for the subsequent steps can be facilitated using the decentralized end-to-end encryption protocol. When the initial user, like Alice, initiates the conversation, she sends a request to establish a shared symmetric encryption key with Bob. Utilizing symmetric encryption methods such as AES for message encryption can be particularly suitable for instant messaging, as it minimizes the computational overhead on the devices. In fact, the exchanged key can not be employed directly for encryption, it can instead be used within the context of the Double Ratchet algorithm.

[0211] **Messages encryption, sending, and receiving:** This process can be done using the Double Ratchet algorithm. The fundamental concept within this algorithm can be the *KDF chain*, defined as a cryptographic function that takes a secret, a random *KDF key*, and input data to produce an output that appears random, as long as the key remains confidential, satisfying the requirements of a cryptographic Pseudorandom Function.

[0212] In a *Double Ratchet session* involving Alice and Bob, each participant maintains three *KDF keys*: one for a *root chain*, one for a *sending chain*, and another for a *receiving chain*. Alice's sending chain corresponds to Bob's receiving chain, and vice versa. As messages can be exchanged, new decentralized end-to-end encryption protocol secret keys can be shared, and they can be used as inputs to the *root chain*. The output keys generated by

the *root chain* then become fresh *KDF keys* for the *sending* and *receiving chains*. This process can be referred to as the decentralized end-to-end encryption protocol *ratchet*. The *sending* and *receiving chains* progress as messages can be transmitted and received, and the keys they produce can be employed for the encryption and decryption of messages. This process can be known as the *symmetric-key ratchet*.

[0213] **Document Storage:** A user can request the generation of a public/private key in order to encrypt and store a document in form of replicas among different players.

[0214] Most of the conventional cloud storage approaches rely on a central server that generates and/or shares encryption keys and sends them to the user in order to encrypt the document before sending it through a public network. However, nothing can prevent the centralized server to decipher the stored document and eavesdrop the client's data.

[0215] Some proposals, including, have tackled this issue by introducing a password-based technique known as SecretStore. The primary aim of SecretStore can be to prevent administrators from accessing users' stored data. The approach involves client-side encryption of the data before a second layer of encryption can be applied using the server's key, after which the document can be uploaded.

[0216] Client-side encryption in SecretStore can be carried out iteratively using a set of public keys, with the user determining the number of iterations based on their desired level of security. This approach also addresses the challenge of securely storing private keys for future decryption on the user's device. Since users cannot feasibly maintain their private keys indefinitely, as opposed to a server, various factors such as platform switches, device damage, or loss can impact key availability. To mitigate this, the authors propose a regenerative private key method, allowing private keys to be generated or regenerated using a user-provided password. In addition, the authors proposed a mechanism to strengthen the passwords.

[0217] Similarly, OpenEthereum's SecretStore can be an open-source platform for distributed elliptic curve key pair generation through a cryptographic protocol. SecretStore can ensure that the private key remains concealed from each participating party while the

public key can be computed and shared safely. Each party holds a share of the private key, and a subset of at least $t + 1$ parties can collaborate to reconstruct the private key, while a group of fewer than $t + 1$ parties cannot. Additionally, the private key shares can be stored separately by many parties, and the system enforces threshold retrieval based on Blockchain permissions, necessitating the consensus of at least $t + 1$ parties for operations requiring the private key.

[0218] Additionally, SecretStore enables the storage of documents. Through the permissioning contract, a user can grant access to specified individuals for the documents stored on the platform. To achieve this, a pair of keys can be generated, with the public key being used to encrypt the document on the client's side. This encrypted document can then be uploaded, stored, and replicated across the system's nodes. Subsequently, other users can request the corresponding private key for decryption and access to the stored document.

[0219] Similar ideas can be adopted using the decentralized end-to-end encryption protocol. A user μ owning a document D can request generation of a pair of public key $h^{(\tau,0)}$ and secure key $s^{(\tau,0)}$, the identifier can serve here as an identifier for the document D. μ_0 can specify itself as the owner of the key pair in the case of storing individually secret data. The user can request sharing these keys with other users $\mu_1, \ldots, \mu_n$ by specifying their identifiers in the key generation request in the case of storing a shared document among them. The public key $h^{(\tau,0)}$ can be used to encrypt the document for more security by avoiding the local construction of the secret key in the device in spite of the fact that the decentralized end-to-end encryption protocol can be secure the adversary can corrupt all the secure communication channels between the user and the anonymous players responsible for it. Then D can be encrypted by any public-key discrete logarithm based cryptographic primitive like ElGamal, and upload the encrypted document to be distributively replicated among the decentralized end-to-end encryption protocol players. Afterwards, any user $\mu_0, \mu_1, \ldots, \mu_n$ can request getting the encrypted version of the document and the retrieval of the secret key $s^{(\tau,0)}$ to decipher it.

[0220] For future documents, if shared between the same users, a user among them (the owner of the new document) can request directly the construction of a key pair $h^{(\tau,1)}$ and

$s^{(\tau,1)}$, which can be less costly to get to encrypt the new document. In some applications, the utilization of the private key $s^{(\tau,0)}$ for encrypting documents via a symmetric encryption method like AES can be advantageous, particularly on resource-constrained devices, due to its lower computational cost when compared to public-key cryptography. This approach can be well-suited for such devices where efficiency can be a priority.

[0221] FIG. 5 illustrates a flow diagram of an example method for establishing secure decentralized end-to-end encryption in a distributed network. The method 500 may be implemented using any one or more of the components and devices detailed herein in conjunction with FIGs. 1–4. In overview, the method 500 may be performed by a suite of network nodes, user devices, and supporting blockchain infrastructure within a distributed network environment, in some embodiments. The nodes within the network can include user devices and blockchain participants, among others, that collectively contribute to the secure exchange of cryptographic keys and subsequent communication. Additional, fewer, or different operations may be performed in the method 500 depending on the embodiment. At least one aspect of the operations is directed to a system, method, apparatus, or a computer-readable medium.

[0222] At STEP 502, the method can involve generating cryptographic keys where each user within the distributed network independently creates a key pair. This pair consists of a private key for the user's confidential use and a public key for sharing across the network. The generation of keys is a foundational step that underpins the security of the entire encryption process, ensuring that all subsequent operations are grounded on secure and unique cryptographic credentials for each participant.

[0223] At STEP 504, the method can include distributing the public keys using a blockchain network, wherein the public keys are broadcasted or registered on a decentralized ledger. This ensures that the keys are immutably and transparently recorded, providing a reliable reference for verifying user identities and facilitating secure interactions.

[0224] At STEP 506, the method can involve the verification of the distributed public keys through a blockchain-based verification process. This process may utilize smart contracts or consensus mechanisms to authenticate public keys, ensuring that each key is

legitimately associated with the corresponding user, thereby preventing impersonation and ensuring secure key management.

[0225] At STEP 508, the method can engage a secure key agreement protocol, such as Diffie-Hellman or a similar blockchain-facilitated mechanism, enabling two or more parties to collaboratively generate a shared secret key. This step is critical for ensuring that the secret key is known only to the parties involved in the communication and remains confidential, despite the exchange potentially occurring over an unsecured channel.

[0226] At STEP 510, the method can establish secure channels for communication using the shared secret key agreed upon in the previous step. These channels can be used for the encrypted exchange of messages, safeguarding them against eavesdropping and unauthorized interception, thereby ensuring the privacy and confidentiality of the communication.

[0227] At STEP 512, the method can provide for the encryption and sending of messages, where users can encrypt messages with the shared secret key before transmission. This step can ensure that even if a message is intercepted, without the corresponding secret key, it would remain unintelligible to the interceptor.

[0228] At STEP 514, the method can enable the reception and decryption of encrypted messages. When a message is received, the recipient can use their copy of the shared secret key to decrypt the message, converting it back into its original, readable format. This completes the secure communication loop, allowing for private and secure information exchange within the distributed network.

[0229] While various embodiments of the present solution have been described above, it should be understood that they have been presented by way of example only, and not by way of limitation. Likewise, the various diagrams may depict an example architectural or configuration, which are provided to enable persons of ordinary skill in the art to understand example features and functions of the present solution. Such persons would understand, however, that the solution is not restricted to the illustrated example architectures or configurations, but can be implemented using a variety of alternative architectures and

configurations. Additionally, as would be understood by persons of ordinary skill in the art, one or more features of one embodiment can be combined with one or more features of another embodiment described herein. Thus, the breadth and scope of the present disclosure should not be limited by any of the above-described illustrative embodiments.

[0230] It is also understood that any reference to an element herein using a designation such as "first," "second," and so forth does not generally limit the quantity or order of those elements. Rather, these designations can be used herein as a convenient means of distinguishing between two or more elements or instances of an element. Thus, a reference to first and second elements does not mean that only two elements can be employed, or that the first element must precede the second element in some manner.

[0231] Additionally, a person having ordinary skill in the art would understand that information and signals can be represented using any of a variety of different technologies and techniques. For example, data, instructions, commands, information, signals, bits and symbols, for example, which may be referenced in the above description can be represented by voltages, currents, electromagnetic waves, magnetic fields or particles, optical fields or particles, or any combination thereof.

[0232] A person of ordinary skill in the art would further appreciate that any of the various illustrative logical blocks, modules, processors, means, circuits, methods and functions described in connection with the aspects disclosed herein can be implemented by electronic hardware (e.g., a digital implementation, an analog implementation, or a combination of the two), firmware, various forms of program or design code incorporating instructions (which can be referred to herein, for convenience, as "software" or a "software module"), or any combination of these techniques. To clearly illustrate this interchangeability of hardware, firmware and software, various illustrative components, blocks, modules, circuits, and steps have been described above generally in terms of their functionality. Whether such functionality is implemented as hardware, firmware or software, or a combination of these techniques, depends upon the particular application and design constraints imposed on the overall system. Skilled artisans can implement the described functionality in various ways for each particular

application, but such implementation decisions do not cause a departure from the scope of the present disclosure.

[0233] Furthermore, a person of ordinary skill in the art would understand that various illustrative logical blocks, modules, devices, components and circuits described herein can be implemented within or performed by an integrated circuit (IC) that can include a general purpose processor, a digital signal processor (DSP), an application specific integrated circuit (ASIC), a field programmable gate array (FPGA) or other programmable logic device, or any combination thereof. The logical blocks, modules, and circuits can further include antennas and/or transceivers to communicate with various components within the network or within the device. A general purpose processor can be a microprocessor, but in the alternative, the processor can be any conventional processor, controller, or state machine. A processor can also be implemented as a combination of computing devices, e.g., a combination of a DSP and a microprocessor, a plurality of microprocessors, one or more microprocessors in conjunction with a DSP core, or any other suitable configuration to perform the functions described herein.

[0234] If implemented in software, the functions can be stored as one or more instructions or code on a computer-readable medium. Thus, the steps of a method or algorithm disclosed herein can be implemented as software stored on a computer-readable medium. Computer-readable media includes both computer storage media and communication media including any medium that can be enabled to transfer a computer program or code from one place to another. A storage media can be any available media that can be accessed by a computer. By way of example, and not limitation, such computer-readable media can include RAM, ROM, EEPROM, CD-ROM or other optical disk storage, magnetic disk storage or other magnetic storage devices, or any other medium that can be used to store desired program code in the form of instructions or data structures and that can be accessed by a computer.

[0235] In this document, the term “module” as used herein, refers to software, firmware, hardware, and any combination of these elements for performing the associated functions described herein. Additionally, for purpose of discussion, the various modules are described as discrete modules; however, as would be apparent to one of ordinary skill in the

art, two or more modules may be combined to form a single module that performs the associated functions according to embodiments of the present solution.

[0236] Additionally, memory or other storage, as well as communication components, may be employed in embodiments of the present solution. It will be appreciated that, for clarity purposes, the above description has described embodiments of the present solution with reference to different functional units and processors. However, it will be apparent that any suitable distribution of functionality between different functional units, processing logic elements or domains may be used without detracting from the present solution. For example, functionality illustrated to be performed by separate processing logic elements, or controllers, may be performed by the same processing logic element, or controller. Hence, references to specific functional units are only references to a suitable means for providing the described functionality, rather than indicative of a strict logical or physical structure or organization.

[0237] Various modifications to the embodiments described in this disclosure will be readily apparent to those skilled in the art, and the general principles defined herein can be applied to other embodiments without departing from the scope of this disclosure. Thus, the disclosure is not intended to be limited to the embodiments shown herein, but is to be accorded the widest scope consistent with the novel features and principles disclosed herein, as recited in the claims below.

WHAT IS CLAIMED IS:

1. A system for facilitating secure communication between users within a blockchain network, comprising:

a Multi-Signed Certificate Provider (MSCP) configured to generate and furnish multi-signed certificates for users' public keys, including an identity key, a one-time key, and a fingerprint key, and to register said certificates on the blockchain network;

a blockchain network configured to store the multi-signed certificates;

a smart contract deployed on the blockchain network, configured to manage the registration, retrieval, and verification of multi-signed certificates for establishing secure communication channels between users;

an instant messaging server associated with the MSCP, configured to facilitate message delivery by notifying users of incoming messages and ensuring prompt relay of messages; and

a user device equipped with a cryptographic module, configured to generate public and private keys for the secure communication channel established based on verified multi-signed certificates from the blockchain network.

2. The system of claim 1, wherein the MSCP is further configured to:

validate registration requests from users, including verifying signatures of the keys using the fingerprint key;

multi-sign the certificates using private keys of a predetermined number of nodes within the blockchain network; and

invoke smart contract functions to add the multi-signed certificates to the blockchain network.

3. The system of claim 1, wherein the smart contract is further configured to:

provide functions for adding identity and one-time certificates to the blockchain network;

facilitate the retrieval of multi-signed certificates by users, enabling the secure exchange of messages between them; and

perform identity verification and mutual authentication by verifying the authenticity of certificates and public keys retrieved from the blockchain network.

4. The system of claim 1, wherein the user device's cryptographic module is further configured to:

send registration requests to the MSCP over a secure connection, including user ID, public components of the keys, a timestamp, keys' validity, and signatures;

receive and validate the MSCP's response containing the multi-signed certificates and the MSCP's own certificate; and

generate a shared master secret using the public keys retrieved from the blockchain network for encrypting and decrypting messages.

5. A system for managing cryptographic keys in a distributed network, the system comprising:

a plurality of nodes configured to participate in a blockchain network;

a key generation module, operable on at least one of the nodes, configured to distribute generation of cryptographic keys using a threshold scheme, where the cryptographic keys include a public key and a private key share for each participating node;

a verification module, operable on at least one of the nodes, configured to validate an integrity and authenticity of the cryptographic key shares exchanged among the nodes using the blockchain network;

a key reconstruction module, configured to enable the reconstruction of a private key from a subset of the private key shares when a predetermined threshold number of shares is reached, wherein the private key is used for decrypting encrypted messages or data transmitted within the network;

a secure communication module, configured to establish secure communication channels between nodes using the cryptographic keys generated by the key generation module; and

a storage module, configured to store the cryptographic key shares and the public key within the blockchain network.

6. The system of claim 5, wherein the key generation module is further configured to implement Shamir's Secret Sharing scheme for distributing the private key shares among the nodes.

7. The system of claim 5, wherein the verification module utilizes digital signatures or message authentication codes (MACs) to verify the integrity and origin of the messages transmitted within the network.

8. The system of claim 5, further comprising a secure backup module configured to encrypt and securely store backups of user data and cryptographic keys.

9. The system of claim 5, wherein the key generation module employs a distributed Feldman Verifiable Secret Sharing (DF-VSS) protocol.

10. The system of claim 5, wherein the key reconstruction module utilizes Lagrange interpolation to reconstruct the private key from the subset of private key shares.

11. The system of claim 1, wherein the blockchain network is configured to facilitate the decentralized Public Key Infrastructure (PKI) to minimize reliance on centralized certificate authorities.

12. The system of claim 5, wherein the secure communication module comprises a double ratchet algorithm to provide forward secrecy and cryptographic ratcheting.

13. The system of claim 5, further comprises a bloom filter module configured to select subsets of nodes for participation in key generation and distribution processes without revealing the exact subgroup responsible.

14. The system of claim 5, wherein the storage module employs a blockchain-based ledger for recording and verifying the cryptographic key generation, distribution, and usage events, to provide an auditable trail.

15. The system of claim 5, wherein the system is further configured to support the generation and management of multi-signed certificates for identity verification.