

2PC-MPC: ECDSA THRESHOLD SIGNATURE BASED ON A THRESHOLD ADDITIVELY HOMOMORPHIC ENCRYPTION

CROSS-REFERENCE TO RELATED APPLICATIONS

This application claims priority to U.S. Provisional Application No. 63/547,714, filed on 8 November 2023, and claims priority to U.S. Provisional Application No. 63/700,778, filed on 30 September 2024, the entire contents of each of which are incorporated herein by reference.

TECHNICAL FIELD

The present disclosure generally relates to the field of cryptography, and more particularly to public-key encryption schemes.

BACKGROUND

A homomorphic encryption scheme enables mathematical operations or computations to be performed on encrypted data without requiring its decryption. Therefore, the data remains confidential while being processed, even in an untrusted environment. As a result, useful operations may be performed on the encrypted data by a third party without requiring the third party to properly secure the data, as the original data cannot be identified without the decryption key. For example, a person or organization may apply standard encryption techniques to secure sensitive data on cloud platforms, but processing or validating the encrypted data in the cloud would require its decryption, which raises privacy issues and security vulnerabilities and may entail additional costs and resources. By utilizing homomorphic encryption, private data may be shared and evaluated securely in commercial cloud environments without endangering privacy. The cloud service provider only has access to the encrypted data (ciphertext) and can perform computations on the data without decryption. The results of the encrypted processing may be provided to the owner of the private data who can then decrypt the data (into plaintext) via the decryption key. In addition to cloud services, homomorphic encryption can be used by entities in a wide variety of businesses and industries, such as financial services, healthcare, retail, information technology and artificial intelligence systems, to allow access to encrypted data while protecting client or patient privacy.

Homomorphic encryption schemes may be classified into several categories, including partially homomorphic encryption, somewhat homomorphic encryption, and fully homomorphic encryption. A partially homomorphic encryption allows only a single operation to be performed on the ciphertext, such as addition or multiplication. In contrast, fully homomorphic encryption (FHE) supports multiple operations indefinitely, such as both addition and multiplication, enabling a wider range of arbitrary computations to be performed on the ciphertext. However, FHE is characterized by certain limitations, particularly a very large computational overhead and high inefficiency due to protracted time for performing computations. Many FHE schemes are based on lattice mathematics, such as derived from the ring learning with errors (RLWE) problem, and are

generally considered secure from breaches by quantum computing (i.e., post-quantum cryptography).

An Elliptic curve digital signature algorithm (ECDSA) is among the NIST (National Institute of Standards and Technology) standards for digital signature. Much effort has been put into thresholdizing ECDSA in past years, and threshold ECDSA schemes have been widely deployed, mainly due to its widespread use as an authentication scheme for decentralized blockchain instruments, such as cryptocurrencies (e.g., Bitcoin, Ethereum), in which the signature constitutes a proof of ownership of the funds.

Threshold signature based solutions in the cryptocurrency industry flourish and focus on two main types of use cases: “distributed custody” (e.g., Fireblocks, Zengo, and Qredo) and “decentralized bridges” (e.g., Thorchain, Keep Network, and the Internet Computer). Distributed custody leverages threshold signature protocols to mitigate the self-custodial risks of lost and theft, by distributing the secret signing key of the client, who is the legal owner of the funds, between multiple signers. The signing key is secret shared in a way that requires the active consent of both the client and the distributed custodian to produce a signature. A distributed bridge consists of a network of nodes who distributively manage multiple signing keys for different wallets, each holding a pool of funds on different blockchains used to form cross-chain liquidity pools. This enables a single entity (the network) to permissionlessly own, manage and transfer funds (and information) across different blockchains. When a client owning wallets on two different blockchains wishes to transfer funds between them, it can send funds to one end of the bridge’s liquidity pool and request a withdrawal of an equivalent amount to the client’s wallet on the other end. The bridge network then creates and distributedly signs a corresponding transaction that can then be broadcasted by the client to the other blockchain, to complete the exchange.

Due to inherent limitations of existing proposals for threshold signing protocols, all real-world networks are deployed with a rather small number of nodes (for example, on the order of about 10-20). This is not in par with the vision of decentralization since when the number of nodes is small it is difficult to argue that these nodes are not colluding.

To date, practical threshold ECDSA schemes are generally designed to work over unicast channels, assuming secure point-to-point (P2P) channels between every pair of parties. While secure point-to-point channels can be constructed in theory (under an appropriate PKI setup), in practice they inherently incur a high network load (i.e., maintaining a high number of sessions concurrently) and a message complexity of $\Omega(n^2)$, as each party computes and sends a message in private to every other party. For a protocol to take advantage of a broadcast channel it needs to avoid private messages over secure channels (i.e., P_i sending a message to be seen only by P_j), rather, all messages must be public.

Generic MPC protocols in the broadcast channel model are ubiquitous, and oftentimes enjoy extra desirable properties, such as “identifiable abort” and “public verifiability”. The broadcast channel itself may be either implemented in-house by the protocol’s participants or rely on an external implementation. In recent years the use of a blockchain as a broadcast channel has surged, as it offers immutability of messages (which enables easy recovery from failures) as well as incentive mechanisms for faithful participation. However, multiparty threshold ECDSA protocols are rarely designed to operate in a broadcast communication model. One exception is described in

Gennaro et al. (Rosario Gennaro, Steven Goldfeder, and Arvind Narayanan. Threshold-optimal DSA/ECDSA signatures and an application to bitcoin wallet security. In *International Conference on Applied Cryptography and Network Security*, pages 156-174. Springer, 2016.) that relies on a trusted setup, which hinders its applicability to the aforementioned use cases, which were inceptioned in order to avoid the trusted party in the first place. Emulating the setup phase of Gennaro et al. in a distributed setting may be possible in theory but is considered infeasible since it requires a distributed generation of safe primes from which the parameters to a zero-knowledge proof system are derived.

SUMMARY

In accordance with an aspect of the present disclosure, there is provided a method for generating at least one digital signature, including following one or more secure protocols based on a threshold additively homomorphic encryption (TAHE) over a broadcast channel, where a threshold of parties of a decentralized network (DN) produces the digital signature by following a Universally Composable (UC)-secure protocol configured to output at least one of an Elliptic curve digital signature algorithm (ECDSA) signature or a Schnorr signature. At least one client and a threshold of parties of the decentralized network may cooperate to generate the digital signature. The threshold of parties of the decentralized network may cooperate to generate the digital signature without a client. The method may include generating a distributed key using a distributed key generation (DKG) protocol of the one or more secure protocols. The DKG protocol may be a DKG protocol over a synchronous communication channel (“DKG Protocol Sync”), the DKG Protocol Sync being parameterized with a group description $(\mathbb{G}, G, q)$ and interacting with $n + 1$ parties including a client A , and DN parties $B = \{B_i\}_{i \in [n]}$, where all parties hold a public key (pk) as input and agree on a fresh session-identifier (sid) that uniquely identifies a session. The DKG protocol may be a DKG protocol over an asynchronous communication channel (“DKG Protocol Async”), the DKG Protocol Async being parameterized with a group description $(\mathbb{G}, G, q)$, an access structure Γ_B , a unique session identifier sid , and an encryption key to a TAHE scheme pk , where the DKG Protocol Async interacts with a centralized party A_{pidA} and a set of parties $B = \{B_i\}_{i \in [n]}$, where the set of parties B can only send messages to parties in A through a functionality $\mathcal{F}_{global-broadcast}$. The method may further include distributing, by a client that holds a private signing key, the private signing key between the client and the decentralized network. The method may further include pre-signing using a presign protocol of the one or more secure protocols. The presign protocol may be a presign protocol using an ECDSA scheme over a synchronous communication channel (“Presign Protocol ECDSA/Sync”), the Presign Protocol ECDSA/Sync interacting with $n + 1$ parties including a client A , and DN parties $B = \{B_i\}_{i \in [n]}$, where all parties hold a public key (pk) and an encryption of a network share under TAHE (ct_{key}) as input, where all parties verify that a session-identifier sid has never been used before. The presign protocol may be a presign protocol using a Schnorr scheme over a synchronous communication channel (“Presign Protocol Schnorr/Sync”), the Presign Protocol Schnorr/Sync interacting with $n + 1$ parties including a client A , and DN parties $B = \{B_i\}_{i \in [n]}$, where all parties have an encryption of a network share under TAHE (ct_{key}) as input, where all parties verify that a session-identifier (sid) has never been used before. The presign protocol may be a presign protocol using an ECDSA scheme over an asynchronous communication channel (“Presign Protocol ECDSA/Async”), the Presign Protocol ECDSA/Async parameterized with: a group description $(\mathbb{G}, G, q)$, an access structure Γ_B , a unique session identifier sid , an encryption key to a TAHE scheme pk , an ECDSA verification key X , a centralized party share of the verification key X_A , and an encryption of a share of a distributed party of a signing key ct_{key} , the Presign Protocol ECDSA/Async interacting with a centralized party A_{pidA} and a set of parties $B = \{B_i\}_{i \in [n]}$, where the set of parties B can only send messages to parties in A through a functionality $\mathcal{F}_{global-broadcast}$. The presign protocol may be a presign protocol using a Schnorr scheme over an asynchronous communication channel (“Presign Protocol Schnorr/Async”), the Presign Protocol Schnorr/Async parameterized with: a

group description $(\mathbb{G}, G, q)$, an access structure Γ_B , a unique session identifier sid , and an encryption key to a TAHE scheme pk , the Presign Protocol Schnorr/Async interacting with a centralized party A_{pidA} and a set of parties $B = \{B_i\}_{i \in [n]}$, where the set of parties B can only send messages to parties in A through a functionality $\mathcal{F}_{global-broadcast}$. The method may further include signing a message using a sign protocol of the one or more secure protocols. The sign protocol may be a sign protocol using an ECDSA scheme over a synchronous communication channel (“Sign Protocol ECDSA/Sync”), the Sign Protocol ECDSA/Sync interacting with $n + 1$ parties including a client A , and DN parties $B = \{B_i\}_{i \in [n]}$, where all parties hold a public key (pk) , network public nonce shares (R_B^0, R_B^1) , and a commitment on A ’s share of the nonce for signature (K_A) , as input, where all parties verify that a session-identifier (sid) has never been used before. The sign protocol may be a sign protocol using a Schnorr scheme over a synchronous communication channel (“Sign Protocol Schnorr/Sync”), the Sign Protocol Schnorr/Sync interacting with $n + 1$ parties including a client A , and DN parties $B = \{B_i\}_{i \in [n]}$, where all parties have a public key (pk) , network public nonce shares (R_B^0, R_B^1) , a commitment on A ’s share of the nonce for signature (K_A) , an encryption of a network share (ct_{key}) , and encryptions of nonces k_0, k_1 (ct_{k_0}, ct_{k_1}) as input, where all parties verify that a session-identifier (sid) has never been used before. The sign protocol may be sign protocol using an ECDSA scheme over an asynchronous communication channel (“Sign Protocol ECDSA/Async”), the Sign Protocol ECDSA/Async parameterized with a group description $(\mathbb{G}, G, q)$, a public generator (H) used for Pedersen commitments, an access structure Γ_B , a unique session identifier (sid), an encryption key to a TAHE scheme pk , an ECDSA verification key X , a centralized party share of the verification key X_A , an encryption of a share of a distributed party of a signing key ct_{key} , an output of a uniquely determined previous presign protocol $presX, sid = (ct_\gamma, ct_{\gamma \cdot key}, ct_{\gamma \cdot k_0}, ct_{\gamma \cdot k_1}, R_{B,0}, R_{B,1})$, and a message to be signed (msg), the Sign Protocol ECDSA/Async interacting with a centralized party A_{pidA} and a set of parties $B = \{B_i\}_{i \in [n]}$, where the set of parties B can only send messages to parties in A through a functionality $\mathcal{F}_{global-broadcast}$, where the parties output a valid ECDSA signature $\sigma = (r, s)$ with verification key X . The sign protocol may be a sign protocol using a Schnorr scheme over an asynchronous communication channel (“Sign Protocol Schnorr/Async”), the Sign Protocol Schnorr/Async parameterized with: a group description $(\mathbb{G}, G, q)$, an access structure Γ_B , a unique session identifier (sid), an encryption key to a TAHE scheme pk , a public verification key X , an encryption of the share of the distributed party of the private signing key ct_{key} , an output of a unique presign $pres_X = (K_{B,0}, K_{B,1})$, and a message to be signed (msg), the Sign Protocol Schnorr/Async interacting with a centralized party A_{pidA} and a set of parties $B = \{B_i\}_{i \in [n]}$, where the set of parties B can only send messages to parties in A through a functionality $\mathcal{F}_{global-broadcast}$, where each party B_i outputs a valid Schnorr signature (z, e) to the verification key X . Each DN party may have a respective voting power, and the method may further include changing the voting power of at least one DN party using a network reconfiguration routine. The network reconfiguration routine may include a reconfiguration protocol with a varying threshold (“Reconfiguration Protocol Varying Threshold”), the Reconfiguration Protocol Varying Threshold parameterized with a TAHE encryption scheme, a public verifiable secret sharing scheme ($PVSS$), a computational security parameter κ , a statistical security parameter σ , an initial threshold t and a final threshold t' , the Reconfiguration Protocol Varying Threshold interacting with a transferring

quorum $B = \{B_i\}_{i \in [n]}$ and a receiving quorum $B' = \{B'_{i'}\}_{i' \in [n']}$, where the transferring quorum holds a t -out-of- n secret sharing $[sk]_i$ on a secret key of a TAHE (sk), where the transferring quorum and the receiving quorum hold a public key (pk) to the encryption scheme, verification keys of the transferring quorum $\{vk_i\}_{i \in [n]}$, and an encryption of the secret key $ct_{sk} = Enc_{pk}^b(sk; \eta)$, where the Reconfiguration Protocol Varying Threshold outputs a fresh t' -out-of- n' secret shares $[sk]_{i'}$ on the secret key (sk) held by B' and public verification keys $\{vk'_{i'}\}_{i' \in [n']}$ corresponding to the secret shares, where when the Reconfiguration Protocol Varying Threshold begins the parties already executed PVSS.Setup and PVSS.Gen successfully, and holds public keys for the new parties $\{pk_{i'}\}_{i' \in [n']}$. The network reconfiguration routine may include a reconfiguration protocol with a constant threshold ("Reconfiguration Protocol Constant Threshold"), the Reconfiguration Protocol Constant Threshold parameterized with a TAHE encryption scheme, a public verifiable secret sharing scheme PVSS, a computational security parameter κ , and a threshold t , the Reconfiguration Protocol Constant Threshold interacting with a transferring quorum $B = \{B_i\}_{i \in [n]}$ and a receiving quorum $B' = \{B'_{i'}\}_{i' \in [n']}$, where the transferring quorum holds a t -out-of- n secret sharing $[sk]_i$ on a secret key of a TAHE sk , where the transferring quorum and the receiving quorum hold a public key (pk) to the encryption scheme, and verification keys of the transferring quorum $\{vk_i\}_{i \in [n]}$, where the Reconfiguration Protocol Constant Threshold outputs a fresh t -out-of- n secret shares $[sk]_{i'}$ on the secret key sk held by B' and public verification keys $\{vk'_{i'}\}_{i' \in [n']}$ corresponding to the secret shares, where when the Reconfiguration Protocol Constant Threshold begins the parties already executed PVSS.Setup and PVSS.Gen successfully, and holds appropriate public keys for the new parties $\{pk_{i'}\}_{i' \in [n']}$. A plurality of clients and the threshold of parties of the decentralized network may cooperate to generate the digital signature, and the method may further include transferring ownership of a resource from a sender client of the plurality of clients to a receiver client of the plurality of clients using an ownership transfer protocol of the one or more secure protocols. The ownership transfer protocol may be parameterized with a group description $(\mathbb{G}, G, q)$, and interacts with parties A_{old} , A_{new} and $\{B_i\}_{i \in S}$, where A_{old} transfers its secret key to A_{new} and $\{B_i\}_{i \in S}$ approves the transfer, where all parties hold: a public key (pk) to the encryption scheme; a public key of A_{new} ($pk_{A_{new}}$); a public share of A_{old} of a signing key ($X_{A_{old}}$); a public verification key (X) as input, and where the parties agree on a fresh session identifier (sid). At least one secure protocol of the one or more secure protocols may include an identifiable abort. At least one secure protocol of the one or more secure protocols may be publicly verifiable. At least one secure protocol of the one or more secure protocols may be performed over an asynchronous communication channel and comprises guaranteed output delivery. The method may further include performing a generic transformation to securely transform a plaintext space of the TAHE, to align the plaintext space with an ECDSA/Schnorr group of order q . The method may further include a secure function evaluation for the TAHE, allowing a user to evaluate a private affine function f homomorphically on a tuple of ciphertexts $ct_i = Enc(w_i, r_i)$ without revealing information on function f except for $f(x_1, \dots, x_n)$, even given a secret key of the TAHE and randomizers r_i and plaintexts x_i for the encryptions ct_i . The TAHE may be tied with elliptic curve group elements, such that corresponding witnesses thereof satisfy linear relations and have range constraints, using at least one generic zero-knowledge (zk) range proof. The parties of the decentralized network may hold a single secret

share, for an underlying TAHE scheme, and a network share of each secret signing key may be stored publicly encrypted under a corresponding TAHE public key. The presign protocol may be configured to generate at least one presign offline in a common pool of presigns shared with all clients in the network, such that upon signing the clients could use the common pool of presigns to sign transactions. The method may include performing at least one performance boosting procedure include one or more of: (i) batching zero knowledge proofs; (ii) redundant range proofs when using Class-Groups; (iii) amortized decryption; and (iv) synchronous decryption. The decentralized network may be selected from one or more of: a blockchain system; a directed acyclic graphs DAG-based consensus platform; and a blockchain-DAG hybrid. The digital signature may be used for at least one application selected from one or more of: digital wallets; future transactions; bridging blockchains; wallet transfer; and decentralized autonomous organization (DAO). A public key of a digital wallet generated on a first blockchain may be configured to be used as a digital wallet on a second blockchain. A policy may be specified upon generation of the digital wallet on the first blockchain, and the first blockchain may enforce the policy when using the digital wallet for the second blockchain participating in signing requests only on transactions valid with the specified policy. A future transaction may include a signed transaction of a client configured to be executed when a predetermined condition is met, where the blockchain is configured to delay execution of the transaction. A decentralized autonomous organization (DAO) may include at least one child key derived from a parent key, where the child key includes fewer permissions than the parent key.

In accordance with an aspect of the present disclosure, there is provided a system for generating a digital signature, including a decentralized network (DN) including a plurality of parties, configured to follow one or more secure protocols based on a threshold additively homomorphic encryption (TAHE) over a broadcast channel, where a threshold of the plurality of parties of the DN produces the digital signature by following a Universally Composable (UC)-secure protocol configured to output at least one of an Elliptic curve digital signature algorithm (ECDSA) signature or a Schnorr signature. At least one client and a threshold of parties of the decentralized network may cooperate to generate the digital signature. The threshold of parties of the decentralized network may cooperate to generate the digital signature without a client. The system may include a network distributed key generation (DKG) module, configured to generate a distributed key using a distributed key generation (DKG) protocol of the one or more secure protocols. The DKG protocol may be a DKG protocol over a synchronous communication channel (“DKG Protocol Sync”), the DKG Protocol Sync being parameterized with a group description $(\mathbb{G}, G, q)$ and interacting with $n + 1$ parties including a client A , and DN parties $B = \{B_i\}_{i \in [n]}$, where all parties hold a public key (pk) as input and agree on a fresh session-identifier (sid) that uniquely identifies a session. The DKG protocol may be a DKG protocol over an asynchronous communication channel (“DKG Protocol Async”), the DKG Protocol Async being parameterized with a group description $(\mathbb{G}, G, q)$, an access structure Γ_B , a unique session identifier sid , and an encryption key to a TAHE scheme pk , where the DKG Protocol Async interacts with a centralized party A_{pidA} and a set of parties $B = \{B_i\}_{i \in [n]}$, where the set of parties B can only send messages to parties in A through a functionality $\mathcal{F}_{global-broadcast}$. The system may be configured to distribute, by a client that holds a private signing key, the private signing key between the client and the decentralized network. The system may include a network ECDSA/Schnorr presign module, configured for

pre-signing using a presign protocol of the one or more secure protocols. The presign protocol may be a presign protocol using an ECDSA scheme over a synchronous communication channel (“Presign Protocol ECDSA/Sync”), the Presign Protocol ECDSA/Sync interacting with $n + 1$ parties including a client A , and DN parties $B = \{B_i\}_{i \in [n]}$, where all parties hold a public key (pk) and an encryption of a network share under TAHE (ct_{key}) as input, where all parties verify that a session-identifier sid has never been used before. The presign protocol may be a presign protocol using a Schnorr scheme over a synchronous communication channel (“Presign Protocol Schnorr/Sync”), the Presign Protocol Schnorr/Sync interacting with $n + 1$ parties including a client A , and DN parties $B = \{B_i\}_{i \in [n]}$, where all parties have an encryption of a network share under TAHE (ct_{key}) as input, where all parties verify that a session-identifier (sid) has never been used before. The presign protocol may be a presign protocol using an ECDSA scheme over an asynchronous communication channel (“Presign Protocol ECDSA/Async”), the Presign Protocol ECDSA/Async parameterized with: a group description $(\mathbb{G}, G, q)$, an access structure Γ_B , a unique session identifier sid , an encryption key to a TAHE scheme pk , an ECDSA verification key X , a centralized party share of the verification key X_A , and an encryption of a share of a distributed party of a signing key ct_{key} , the Presign Protocol ECDSA/Async interacting with a centralized party A_{pidA} and a set of parties $B = \{B_i\}_{i \in [n]}$, where the set of parties B can only send messages to parties in A through a functionality $\mathcal{F}_{global-broadcast}$. The presign protocol may be a presign protocol using a Schnorr scheme over an asynchronous communication channel (“Presign Protocol Schnorr/Async”), the Presign Protocol Schnorr/Async parameterized with: a group description $(\mathbb{G}, G, q)$, an access structure Γ_B , a unique session identifier sid , and an encryption key to a TAHE scheme pk , the Presign Protocol Schnorr/Async interacting with a centralized party A_{pidA} and a set of parties $B = \{B_i\}_{i \in [n]}$, where the set of parties B can only send messages to parties in A through a functionality $\mathcal{F}_{global-broadcast}$. The system may include a network ECDSA/Schnorr sign module, configured for signing a message using a sign protocol of the one or more secure protocols. The sign protocol may be a sign protocol using an ECDSA scheme over a synchronous communication channel (“Sign Protocol ECDSA/Sync”), the Sign Protocol ECDSA/Sync interacting with $n + 1$ parties including a client A , and DN parties $B = \{B_i\}_{i \in [n]}$, where all parties hold a public key (pk), network public nonce shares (R_B^0, R_B^1) , and a commitment on A ’s share of the nonce for signature (K_A), as input, where all parties verify that a session-identifier (sid) has never been used before. The sign protocol may be a sign protocol using a Schnorr scheme over a synchronous communication channel (“Sign Protocol Schnorr/Sync”), the Sign Protocol Schnorr/Sync interacting with $n + 1$ parties including a client A , and DN parties $B = \{B_i\}_{i \in [n]}$, where all parties have a public key (pk), network public nonce shares (R_B^0, R_B^1) , a commitment on A ’s share of the nonce for signature (K_A), an encryption of a network share (ct_{key}), and encryptions of nonces k_0, k_1 (ct_{k_0}, ct_{k_1}) as input, where all parties verify that a session-identifier (sid) has never been used before. The sign protocol may be sign protocol using an ECDSA scheme over an asynchronous communication channel (“Sign Protocol ECDSA/Async”), the Sign Protocol ECDSA/Async parameterized with a group description $(\mathbb{G}, G, q)$, a public generator (H) used for Pedersen commitments, an access structure Γ_B , a unique session identifier (sid), an encryption key to a TAHE scheme pk , an ECDSA verification key X , a centralized party share of the verification key X_A , an encryption of a share of a distributed party of a signing key ct_{key} , an output of a uniquely

determined previous presign protocol $presX, sid = (ct_\gamma, ct_{\gamma \cdot key}, ct_{\gamma \cdot k_0}, ct_{\gamma \cdot k_1}, R_{B,0}, R_{B,1})$, and a message to be signed (msg), the Sign Protocol ECDSA/Async interacting with a centralized party A_{pidA} and a set of parties $B = \{B_i\}_{i \in [n]}$, where the set of parties B can only send messages to parties in A through a functionality $\mathcal{F}_{global-broadcast}$, where the parties output a valid ECDSA signature $\sigma = (r, s)$ with verification key X . The sign protocol may be a sign protocol using a Schnorr scheme over an asynchronous communication channel (“Sign Protocol Schnorr/Async”), the Sign Protocol Schnorr/Async parameterized with: a group description $(\mathbb{G}, G, q)$, an access structure Γ_B , a unique session identifier (sid), an encryption key to a TAHE scheme pk , a public verification key X , an encryption of the share of the distributed party of the private signing key ct_{key} , an output of a unique presign $pres_X = (K_{B,0}, K_{B,1})$, and a message to be signed (msg), the Sign Protocol Schnorr/Async interacting with a centralized party A_{pidA} and a set of parties $B = \{B_i\}_{i \in [n]}$, where the set of parties B can only send messages to parties in A through a functionality $\mathcal{F}_{global-broadcast}$, where each party B_i outputs a valid Schnorr signature (z, e) to the verification key X . Each DN party may have a respective voting power, and the system may include a network reconfiguration module, configured for changing the voting power of at least one DN party using a network reconfiguration routine. The network reconfiguration routine may include a reconfiguration protocol with a varying threshold (“Reconfiguration Protocol Varying Threshold”), the Reconfiguration Protocol Varying Threshold parameterized with a TAHE encryption scheme, a public verifiable secret sharing scheme (PVSS), a computational security parameter κ , a statistical security parameter σ , an initial threshold t and a final threshold t' , the Reconfiguration Protocol Varying Threshold interacting with a transferring quorum $B = \{B_i\}_{i \in [n]}$ and a receiving quorum $B' = \{B'_i\}_{i \in [n']}$, where the transferring quorum holds a t -out-of- n secret sharing $[sk]_i$ on a secret key of a TAHE (sk), where the transferring quorum and the receiving quorum hold a public key (pk) to the encryption scheme, verification keys of the transferring quorum $\{vk_i\}_{i \in [n]}$, and an encryption of the secret key $ct_{sk} = \text{Enc}_{pk}^b(sk; \eta)$, where the Reconfiguration Protocol Varying Threshold outputs a fresh t' -out-of- n' secret shares $[sk]_{i'}$ on the secret key (sk) held by B' and public verification keys $\{vk'_{i'}\}_{i' \in [n']}$ corresponding to the secret shares, where when the Reconfiguration Protocol Varying Threshold begins the parties already executed PVSS.Setup and PVSS.Gen successfully, and holds public keys for the new parties $\{pk_{i'}\}_{i' \in [n']}$. The network reconfiguration routine may include a reconfiguration protocol with a constant threshold (“Reconfiguration Protocol Constant Threshold”), the Reconfiguration Protocol Constant Threshold parameterized with a TAHE encryption scheme, a public verifiable secret sharing scheme PVSS, a computational security parameter κ , and a threshold t , the Reconfiguration Protocol Constant Threshold interacting with a transferring quorum $B = \{B_i\}_{i \in [n]}$ and a receiving quorum $B' = \{B'_i\}_{i \in [n']}$, where the transferring quorum holds a t -out-of- n secret sharing $[sk]_i$ on a secret key of a TAHE sk , where the transferring quorum and the receiving quorum hold a public key (pk) to the encryption scheme, and verification keys of the transferring quorum $\{vk_i\}_{i \in [n]}$, where the Reconfiguration Protocol Constant Threshold outputs a fresh t -out-of- n secret shares $[sk]_{i'}$ on the secret key sk held by B' and public verification keys $\{vk'_{i'}\}_{i' \in [n']}$ corresponding to the secret shares, where when the Reconfiguration Protocol Constant Threshold begins the parties already executed PVSS.Setup and PVSS.Gen successfully, and holds appropriate

public keys for the new parties $\{pk_i\}_{i \in [n]}$. A plurality of clients and the threshold of parties of the decentralized network may cooperate to generate the digital signature, and the system may include an ownership transfer verification module, configured for transferring ownership of a resource from a sender client of the plurality of clients to a receiver client of the plurality of clients using an ownership transfer protocol of the one or more secure protocols. The ownership transfer protocol may be parameterized with a group description $(\mathbb{G}, G, q)$, and interacts with parties A_{old} , A_{new} and $\{B_i\}_{i \in \mathcal{S}}$, where A_{old} transfers its secret key to A_{new} and $\{B_i\}_{i \in \mathcal{S}}$ approves the transfer, where all parties hold: a public key (pk) to the encryption scheme; a public key of A_{new} ($pk_{A_{new}}$); a public share of A_{old} of a signing key ($X_{A_{old}}$); a public verification key (X) as input, and where the parties agree on a fresh session identifier (sid). At least one secure protocol of the one or more secure protocols may include an identifiable abort. At least one secure protocol of the one or more secure protocols may be publicly verifiable. At least one secure protocol of the one or more secure protocols may be performed over an asynchronous communication channel and comprises guaranteed output delivery. The system may be configured for performing a generic transformation to securely transform a plaintext space of the TAHE, to align the plaintext space with an ECDSA/Schnorr group of order q . The system may be configured for performing a secure function evaluation for the TAHE, allowing a user to evaluate a private affine function f homomorphically on a tuple of ciphertexts $ct_i = Enc(w_i, r_i)$ without revealing information on function f except for $f(x_1, \dots, x_n)$, even given a secret key of the TAHE and randomizers r_i and plaintexts x_i for the encryptions ct_i . The TAHE may be tied with elliptic curve group elements, such that corresponding witnesses thereof satisfy linear relations and have range constraints, using at least one generic zero-knowledge (zk) range proof. The parties of the decentralized network may hold a single secret share, for an underlying TAHE scheme, and a network share of each secret signing key may be stored publicly encrypted under a corresponding TAHE public key. The presign protocol may be configured to generate at least one presign offline in a common pool of presigns shared with all clients in the network, such that upon signing the clients could use the common pool of presigns to sign transactions. The system may be configured for performing at least one performance boosting procedure include one or more of: (i) batching zero knowledge proofs; (ii) redundant range proofs when using Class-Groups; (iii) amortized decryption; and (iv) synchronous decryption. The decentralized network may be selected from one or more of: a blockchain system; a directed acyclic graphs DAG-based consensus platform; and a blockchain-DAG hybrid. The digital signature may be used for at least one application selected from one or more of: digital wallets; future transactions; bridging blockchains; wallet transfer; and decentralized autonomous organization (DAO). A public key of a digital wallet generated on a first blockchain may be configured to be used as a digital wallet on a second blockchain. A policy may be specified upon generation of the digital wallet on the first blockchain, and the first blockchain may enforce the policy when using the digital wallet for the second blockchain participating in signing requests only on transactions valid with the specified policy. A future transaction may include a signed transaction of a client configured to be executed when a predetermined condition is met, where the blockchain is configured to delay execution of the transaction. A decentralized autonomous organization (DAO) may include at least one child key derived from a parent key, where the child key includes fewer permissions than the parent key.

BRIEF DESCRIPTION OF THE DRAWINGS

The present disclosure will be understood and appreciated more fully from the following detailed description taken in conjunction with the drawings in which:

Figure 1 is a schematic illustration of a system for generating a digital signature, constructed and operative in accordance with an embodiment of the present disclosure;

Figure 2 is a schematic illustration of a decentralized network party of the system of Figure 1, constructed and operative in accordance with an embodiment of the present disclosure;

Figure 3 is a schematic illustration of a client of the system of Figure 1, operative in accordance with an embodiment of the present disclosure;

Figure 4 is a schematic illustration of a threshold additively homomorphic encryption (TAHE) distributed key generation (DKG) process, operative in accordance with an embodiment of the present disclosure;

Figure 5 is a schematic illustration of exemplary secret key shares of a decentralized network party with a given voting power, operative in accordance with an embodiment of the present disclosure;

Figure 6 is a schematic illustration of a network reconfiguration process, operative in accordance with an embodiment of the present disclosure;

Figure 7 is a schematic illustration of an ECDSA/Schnorr distributed key generation DKG process, operative in accordance with an embodiment of the present disclosure;

Figure 8A is a schematic illustration of an ECDSA/Schnorr asynchronous presign process, operative in accordance with an embodiment of the present disclosure;

Figure 8B is a schematic illustration of a Schnorr presign process, operative in accordance with an embodiment of the present disclosure;

Figure 9 is a schematic illustration of an ECDSA/Schnorr signing process, operative in accordance with an embodiment of the present disclosure;

Figure 10 is a schematic illustration of an ownership transfer process, operative in accordance with an embodiment of the present disclosure;

Figure. 11 illustrates a diagrammatic representation of a machine in the example form of a computing device within which a set of instructions, for causing the machine to perform any one or more of the methodologies discussed herein, may be executed; and

Figure 12 is a flow diagram of a general method for generating a digital signature, operative in accordance with an embodiment of the present disclosure.

DETAILED DESCRIPTION

The disclosed embodiments relate to universal composability (UC) secure threshold ECDSA protocols that may overcome the disadvantages of the prior art by reducing message complexity from $\mathcal{O}(n^2)$ to $\mathcal{O}(n)$ and reducing computational complexity from $\mathcal{O}(n)$ to practically $\mathcal{O}(1)$, while maintaining the important identifiable abort (if protocol fails can identify a malicious party). To this end, the disclosed embodiments utilize a threshold additively homomorphic encryption, as well as novel zero-knowledge proofs, allowing parties for the first time to communicate publicly over a broadcast channel, which is important for use cases such as permissionless bridges and decentralized custody in which P2P channels between every pair of parties cannot be assumed.

Protocols of the present disclosure may employ novel batching and amortization techniques, which are of independent interest, to further reduce the computation and communication complexity. This may be done by leveraging the fact that all messages are public in two novel ways. First, the cost of threshold decryption is amortized to $\mathcal{O}(1)$ exponentiations per party per ciphertext, instead of the naive $\Omega(n)$. It may be shown that combining decryption shares can be done by a single designated party only, who can inform the others about the result in a verifiable manner. Second, the present disclosure provides a batchable and fully aggregateable zero-knowledge proof system that allows verification of a single proof rather than verifying multiple proofs from each prover.

Aggregating proofs comes at a cost of increasing the round complexity, which leads to offering two variants of the protocol: one is on par with 3-round variants known in the art, that scales linearly with the number of parties, while a second one is practically independent in the number of parties but incurs 7 rounds.

The present disclosure introduces the notion of a “2PC-MPC protocol”, in which a user participates in a two-party (2PC) protocol with a distributed party that emulates the second party in multiparty computation (MPC). This notion assures that both the user and a threshold of the distributed party are required to participate in signing (as in hierarchical threshold access structures) whilst abstracting away the internal structure of the distributed party. In particular, the communication and computation complexity of the user remain constant, which allows for distributed custody use-cases to employ decentralization for the first time, as recent growing interest in the industry demands.

Protocols that use a broadcast channel naturally reduce the message complexity from $\Omega(n^2)$ to $\mathcal{O}(n)$, yet each party intuitively reads and processes a message from every other party, and so the computational overhead remains $\Omega(n)$.

The disclosed embodiments may leverage the fact that all messages are public in order to demonstrate two ways to further reduce the computational complexity of processing these messages to practically $\mathcal{O}(1)$. First, in the threshold decryption sub-protocol the parties broadcast their decryption shares, which are then combined to obtain a final plaintext. Combining the decryption shares may incur $\Omega(n)$ exponentiations per party per ciphertext. However, this can be done by a single designated party only, who can inform the other parties about the result in a verifiable manner. When many executions of threshold decryption are run in parallel, picking a different designated party for each ciphertext can reduce the amortized complexity to $\mathcal{O}(1)$.

exponentiations per party per ciphertext. Secondly, to ensure correctness of the execution, each operation by the parties may be accompanied by a zero-knowledge proof. Typically, the parties separately verify a proof broadcasted by any other party before proceeding to the next protocol step. Considering that zero-knowledge proof verification is one of the computationally heaviest operations in many threshold ECDSA protocols, the present disclosure relates to novel zero-knowledge proof systems that may be considered “fully aggregateable”. Namely, one can first aggregate the n proofs into a single aggregated proof, and then verify only one aggregated proof. While verification of the aggregated proof remains a heavyweight operation, aggregating is orders of magnitude cheaper. Thus, the overall computational complexity consists of a simple aggregation of n proofs (n multiplications in a group) followed by only a single proof verification ($\mathcal{O}(1)$ exponentiations) and so even considering thousands of parties, the computational complexity may be reduced to practically $\mathcal{O}(1)$. Such amortization and aggregation mechanisms can thus be taken to the full extent in the context of threshold ECDSA.

Considering concrete efficiency (i.e., wall-clock time to produce a signature), the “DKLS protocol” described in Doerner et al. (Jack Doerner, Yashvanth Kondi, Eysa Lee, and Abhi Shelat. Threshold ECDSA in three rounds. IACR Cryptol. ePrint Arch., page 765, 2023) demonstrates that 250 parties can produce a signature in a few seconds. That protocol, however, relies on a P2P network which requires a large bandwidth. In addition, the DKLS protocol lacks identifiable abort, which hinders it from being used for permissionless use-cases such as the aforementioned bridge, as it would be quite easy for every single party to amount a denial-of-service attack. In that context, one exemplary protocol that has identifiable abort is disclosed in Canetti et al. (Ran Canetti, Rosario Gennaro, Steven Goldfeder, Nikolaos Makriyannis, and Udi Peled. UC non-interactive, proactive, threshold ECDSA with identifiable aborts. In Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security, pages 1769-1787, 2020). Canetti et al. offers two variants to achieve identifiable abort: the first requires each party to verify $\mathcal{O}(n^2)$ zero-knowledge proofs, which is not scalable, whereas the second requires $\mathcal{O}(n)$ verifications, at the cost of three additional communication rounds. Both versions of the protocol are not scalable with the number of parties and become impractical at around 10-20 parties.

The above-mentioned improvements in communication and computation may enable distributed signing solutions like bridges, where the signing key owner is the network itself, to scale to an extremely large number of participants. In contrast, when the owner of the signing key is an external client, as in the distributed custody use case, the signing protocol must be held between the custodial and the client, which motivated designing practical two-party signature protocols, such as described in Lindell (Yehuda Lindell. Fast secure two-party ECDSA signing. In Annual International Cryptology Conference, pages 613-644. Springer, 2017). In such protocols, the custodian acts as a service provider who cannot sign without participation of the client. Such custodian solutions may increase the trust in the service provider but still suffer from centralization. For instance, the service provider may completely deny service from a client or censor specific transactions. In addition, it leaves open the following attack vector. Consider a client who is an organization with multiple authorized signers. Deploying a two-party signing protocol in that setting means duplicating the client share of the signing key among all authorized signers, while the service provider possesses the other share. Thus, it is sufficient for the service provider to collude with only a single member of the organization in order to steal all the client funds.

These issues motivate the need for a decentralized service provider, meaning that the service provider share of the signing key is now shared among a network of n participants that emulate the operation of the provider. This, however, requires a distributed signing protocol for an access structure that is different than the usual threshold access structure known in the art. In the new access structure, half of the signing key is in possession of the client whereas the network, collectively, possesses the other half. To produce a signature, the client along with at least $t + 1$ honest parties must collaborate, keeping the client safe even in the extreme case that all network participants are compromised. In terms of security, a protocol must be proven secure under “universal composition (UC)” in order to be able to run it in parallel with many clients. Bourse et al. (Florian Bourse, Rafael Del Pino, Michele Minelli, and Hoeteck Wee. FHE Circuit Privacy Almost for Free. In *Advances in Cryptology – CRYPTO 2016 - 36th Annual International Cryptology Conference*, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part II, volume 9815 of *Lecture Notes in Computer Science*, pages 62-89. Springer, 2016) addresses a similar access structure in the UC framework, however, since the parties must separately verify zero-knowledge proofs from each other party it cannot scale well for large networks.

Addressing that special access structure can take different approaches. For instance, one may model the client as n virtual parties, so its power is equal to that of the service provider, as described for example in Zyskind et al. (Guy Zyskind, Avishay Yanai, and Alex “Sandy” Pentland. Unstoppable wallets: Chain-assisted threshold ECDSA and its applications. *IACR Cryptol. ePrint Arch.*, page 832, 2023). This approach incurs a high computation and communication complexity on the client, that is dependent on the network size. An alternative would be to use an hierarchical threshold secret sharing scheme with two levels of hierarchy - the client is at the top and the network participants are at the bottom such that reconstruction is possible only by a collaboration between the client and at least $t + 1$ network participants.

The disclosed embodiments relate to Universally Composable (UC)-secure large-scale threshold ECDSA protocols and introduce the notion of 2PC-MPC. Specifically, the client and each network participant possess only one share. However, in contrast to a hierarchical secret sharing scheme, the client is agnostic to the fact that it interacts with a distributed party. Instead, the client follows the protocol as if the client is interacting with a single party only (hence “2PC”), which is emulated by the network via a multiparty computation (MPC) protocol.

It is appreciated that the disclosed embodiments may overcome disadvantages of the prior art in several ways. The disclosed protocols are natively designed to use only a broadcast channel (for sending messages to multiple parties), which makes them inherently applicable to blockchain networks where direct communication channels between pairs of parties are not available. The disclosed protocols may be accompanied with a security proof via a UC simulation and hence match real world needs where a distributed service has to handle many clients concurrently.

The disclosed protocols may differ from existing protocols in at least two aspects. The first differing aspect is access structure. The disclosed protocols may address two access structures together. The first access structure is a two-level hierarchy, and captures the setting of $1 + n$ parties. The first party is alone at the top level and the other n parties are at the second level, such that a qualified set consists of the party at the first level together with any $t + 1$ parties from the second level. Thus, this structure can be referred to as “one plus t out of n ”. When parties have voting

powers, then $t+1$ of the voting powers (rather than of the parties) are required. The second access structure is the usual t out of n ('threshold') access structure. Our protocol for the latter is a byproduct of the former, such that taking the protocol for the first structure and ignoring the first party results in a usual (t, n) signature protocol. That is, a protocol may be designed without a client. The present disclosure is primarily focused on the first, hierarchical, access structure.

The second differing aspect is round complexity vs computational complexity. Each party needs to complete a communication round before continuing to the next round. The disclosed protocols may be optimized to minimize either a number of rounds or computational complexity. When optimized to the number of rounds, a three (3) rounds protocol is obtained, where presignature takes two rounds and the signature takes one round, matching that of existing protocols such as Doerner et al. The disclosed protocols build on a broadcast channel rather than P2P channels, which is leveraged to achieve identifiable abort. However, the number of expensive operations (exponentiations) to be done by each party is linear with the number of parties. The number of expensive operations can be reduced to practically constant when optimizing the protocol for computational complexity, thereby scaling to even thousands of participants becomes practical. This, however, is at the cost of increasing the number of communication rounds to seven (7), six of them for presignature.

Achieving $\mathcal{O}(1)$ exponentiations per party is challenging and may be reflected in two contexts in the disclosed protocols: "threshold decryption" and "zero-knowledge proof verification". Optimizing threshold decryption may be performed via an amortization technique that is applicable for batches of ciphertexts, even from different contexts (i.e., different signing sessions and different clients). This means that the parties can process a batch of n ciphertexts at the cost of 1, namely, each party only performs $\mathcal{O}(1)$ exponentiations. Optimizing zero-knowledge verification may be done via novel aggregateable zero-knowledge proof systems designed for NP relations used in the disclosed protocols. These two optimizations, together with a careful protocol crafting, lead to a situation where the computation of each party is practically independent of the number of parties.

Addressing the aforementioned hierarchical access structure, the present disclosure introduces the concept of a "2PC-MPC threshold" ECDSA/Schnorr. In such a protocol, the user interacts with a network while being unaware of the topology, or even of the precise size of the network. From the user point of view it is interacting with a single party and hence its computation and communication overhead remains low. The network almost perfectly emulates the second party in a 2-out-of-2 ECDSA/Schnorr protocol (i.e., the user is the first party), where messages from the network participants are aggregated into a single message which is effectively the only one received and processed by the user. This feature is important in case the network operates as a blockchain, in which case a user cannot be expected to directly interact with the blockchain nodes, but rather the user broadcasts its messages to be processed by the blockchain and is notified whenever the blockchain has a response. Such a communication model may reduce the attack surface of the system in a real deployment, since the user is unaware of the identities of the participant endpoints over the network. Such a concept may pave the path to a topology hiding protocol, in which the network properties (e.g., size and topology) are hidden from the user.

A 2PC-MPC notion may include an access structure of $t + 1$ -out-of- n , where a client is not aware of an internal structure of the distributed system communicated thereto, and from a

perspective of the client, experiences a 2PC protocol. In particular, the communication complexity of the user is independent of n . In addition, for Class-Group based TAHE, where range proofs are redundant, the computation time is independent of n .

Aspects of the present disclosure may be used for a blockchain-based massively decentralized cryptosystem for securely maintaining digital wallets for any other blockchain. The blockchain may be permissionless, and before the beginning of every epoch, reconfiguration may allow for validators to join or leave the network. The blockchain may be based on a Proof of Stake (PoS), where each validator has a voting power proportional to its stake. In turn, at every reconfiguration validators may also withdraw part of their stake, or increase their stake, which could change their voting power accordingly. The term “blockchain” as used herein may also encompass variations, such as directed acyclic graphs (DAG)-based consensus platforms or blockchain-DAG hybrids. DAG-based designs may allow validators to propose blocks in parallel. This may improve “system latency”, i.e., the time to create a digital wallet or sign a transaction, as well as the “throughput”, i.e., the number of transactions or wallets produced per unit time. In every signing request, aggregation of the decryption shares may be considered a bottleneck, such that each validator can process decryption shares of a different subset of signing requests, amortizing this cost. Clients may communicate with the blockchain by sending transactions and reading blocks. MPC party emulation subprotocols may be implemented over a consensus channel used for producing blocks. A block proposer validator may be responsible for aggregating the last MPC round, computing the message to send to the client, and including it in its proposed block. Such a communication channel may be asynchronous or partially synchronous.

Upon creation, wallets may specify policies, either by using smart contracts implemented on-chain, or using light-clients in order to validate policies posted on another blockchain. Before validators participate in signing a transaction, they validate that it matches the policy of the corresponding wallet. Policies may specify, for example, a “whitelist” or a “blacklist” of accounts, a “transaction limit”, a limit on the number of transactions per epoch, or anything else that can be specified in a smart contract. Compromising a client may not allow an adversary to bypass the policy of its wallet. Compromising the blockchain may only be exploited to bypass policies for wallets for which the client share was also compromised. This may ensure that security can only be enhanced by sharing the key with the blockchain.

The present disclosure may be used in various applications.

One example is digital wallets for any blockchain. The address of a wallet is typically a hash of its public key. Therefore, public keys generated on a blockchain according to disclosed embodiments can be used as accounts on other blockchains (e.g., BitCoin or Ethereum®). A client can make a transaction by initiating a signing protocol with a blockchain according to disclosed embodiment. With this motivation in mind, protocols of the disclosed embodiments support both ECDSA (Elliptic Curve Digital Signature Algorithm) and EdDSA (Edwards-curve Digital Signature Algorithm) signatures, which may support a substantial majority of current blockchains. In one example, upon creation of a digital wallet on a first blockchain, a policy may be specified. The first blockchain may enforce the specified policy when using the digital wallet for the second blockchain participating in signing requests only on transactions that are valid with the specified policy.

Another example is future transactions. A future transaction is a signed transaction published by some client, which is only executed when and if a certain predetermined condition is met. For instance, client A would send x_A bitcoin to fund F when and if bitcoin (BTC) is worth $\leq t_{BTC}$ dollars. In aspects of the present disclosure, the user signing round sends an encryption of the signature, and therefore the blockchain has the capability of delaying transaction execution. For example, validators would send their decryption shares only when and if the condition is met.

A further example is bridging blockchains. An appealing private case of future transactions are bridges. Clients can transfer, for instance, BTC in exchange for Ether. Client A would send a future transaction transferring x_A BTC to liquidity pool P , and condition it on receiving at least y_P Ether from the liquidity pool. In this manner, blockchains can be bridged without relying on implementation of smart contracts or light-clients on either of them. This may allow for bridging BTC with other blockchains.

An additional example is recovery. Clients may produce a future transaction that transfers all of their funds and/or wallet to some other entity. This can be used for recovery of the wallet and its funds in case the client share of the key is lost or compromised. The condition for initiating this recovery transaction can be signing a transaction with another wallet whose client share is more protected, or a combination of signatures from a subset of whitelist wallets with a certain threshold, such as counting on “friends”.

Yet another example is wallet transfer. Wallets may have intrinsic value, similar to non-fungible tokens (NFTs). Therefore, transferring ownership of a wallet can be of independent interest, and is not necessarily equivalent to transferring all of its funds. In aspects of the present disclosure, this may be achieved by transferring the client share, in a publicly verifiable manner. The blockchain can then associate the wallet with the receiver client.

Yet a further example is Decentralized Autonomous Organization (DAO). The “BIP32” standard enables deriving child keys from master keys, as in $X_{child} = a \cdot X_{parent} + b \cdot G$, for public values of a and b . Child keys may have less permissions than their parent key. For instance, the master key may be capable of updating the wallet policy, its child could transfer funds to any address up to a certain amount, and each of its children can transfer funds to certain whitelists of clients. It may be shown that for EdDSA (and Schnorr), protocols of the disclosed embodiments are compatible with the BIP32 standard, even with presigns. Specifically, by choosing tweaks (a, b) adaptively after observing the presigns, cannot be exploited by an adversary in order to forge signatures.

The disclosed embodiments may overcome disadvantages of the prior art by providing a 2PC-MPC protocol for a massively decentralized permissionless Proof-of-Stake (PoS) blockchain, servicing multiple clients concurrently to securely maintain and manage digital wallets for any blockchain. The number of secret shares a validator stores is proportional to its voting power, and is independent of the number of clients in the system. The disclosed embodiments include two reconfiguration protocols for redistribution of such shares. The network party may work over an asynchronous reliable broadcast channel, where the (valid) subset of participating parties in each MPC round is arbitrary and post-determined. This is on par with the asynchronous reliable broadcast channel used in blockchains, where in each round the subset of parties can arbitrarily

change and is not known in advance. Protocols of the disclosed embodiments may support both ECDSA and EdDSA (Schnorr).

In terms of round complexity, the first (commitment) round of the client may be removed at the distributed key generation (DKG) phase, as well as at the “presign” phase of the signing protocol. As a result, both the DKG phase and sign phase may be a one-round trip between the blockchain and the client. The sign phase can be broken into a presign phase that can be executed before the message to be signed is decided, and an online-signing phase. When folded back to the standard threshold access structure (removing the client), DKG may be one-round for Schnorr and two-rounds for ECDSA, and sign may be a single round.

In addition, the disclosed embodiments may alleviate the need for zero-knowledge proofs toward the client. In existing protocols, the client verifies proofs from the blockchain, which therefore must be aggregated, to form proofs over aggregated statements. The intuition is that the client behaves as a signing oracle with its secret key share, and therefore even when operating on maliciously generated messages from the blockchain, the transcript cannot be exploited to forge signatures. Although this may lead to vulnerabilities, the disclosed embodiments may provide for optimization and UC-simulation, which may dramatically reduce the round complexity compared to existing protocols, such as those in which proof aggregation requires three rounds.

Protocols of the disclosed embodiments may be proven secure in a UC-framework, and realize the standard threshold signing functionality. The protocols may be simulated with black-box access to carefully chosen signing oracles. The signing oracles may be less strict than expected, such that the simulation of disclosed asynchronous protocols may be intuitive. On the other hand, the unforgeability with respect to the signing oracles may be reduced to well-known hardness assumptions. Specifically, forging the slightly-enhanced Schnorr signing oracle may reduce to breaking Algebraic One-More Discrete Log (AOMDL), and forging the slightly-enhanced ECDSA signing oracle may be proven secure in the Elliptic Curve Generic Group Model (EC-GGM). Notably, disclosed protocols may circumvent the need to assume Enhanced-ECDSA unforgeability, which is proven in EC-GGM up to $q^{1/3}$, which may not suffice when working over a 256-bit elliptic curves. Instead, the disclosed EC-GGM reduction may suggest security up to $q^{1/2}$, which is on-par with security of standard assumptions over elliptic curves (e.g., discrete logs).

For Schnorr signatures, the presign is client independent. In other words, the presign message does not depend on the blockchain secret key share of some wallet. The present disclosure may prove that presigns can be generated off-line in a common pool shared with all clients in the network. Upon signing, clients could use the same pool of presigns to sign transactions. This may save the cost of generating presigns for wallets that are abandoned, and allows the system to prepare for critical points from being overloaded. The slightly-enhanced Schorr signing oracle of the disclosed embodiments may allow the adversary to ask for an arbitrary number of keys and presigns, and adaptively choose the key-presign-message binding in parallel for all signing requests to follow. The disclosed protocols may support the BIP32 standard and allow adding tweaks in the signing phase for the public key. Despite all the signing freedom given to the adversary, breaking the signing oracle may be reduced to AOMDL.

It is appreciated that the term “Schnorr” or references to a “Schnorr signature” or “Schnorr algorithm” herein may also encompass variations to Schnorr signature schemes, including but not limited to an Edwards-curve Digital Signature Algorithm (EdDSA) digital signature scheme.

The phrase “a threshold of parties” or variations thereof as used herein may further encompass instances of “a threshold of voting power” of parties.

Unless otherwise defined, all terms (including technical and scientific terms) used herein have the same meaning as commonly understood by one of ordinary skill in the art to which this invention belongs. It will be further understood that terms, such as those defined in commonly used dictionaries, should be interpreted as having a meaning that is consistent with their meaning in the context of the specification and claims and should not be interpreted in an idealized or overly formal sense unless expressly so defined herein. Well-known functions or constructions may not be described in detail for brevity and/or clarity.

It will be understood that, although the terms first, second, etc., may be used herein to describe various elements, components, regions, layers and/or sections, these elements, components, regions, layers and/or sections should not be limited by these terms. Rather, these terms are only used to distinguish one element, component, region, layer and/or section, from another element, component, region, layer and/or section.

It will be understood that when an element is referred to as being “on”, “attached” to, “operatively coupled” to, “operatively linked” to, “operatively engaged” with, “connected” to, “coupled” with, “contacting”, “added to, another element, it can be directly on, attached to, connected to, operatively coupled to, operatively engaged with, coupled with, added to, and/or contacting the other element or intervening elements can also be present. In contrast, when an element is referred to as being “directly contacting” another element or “directly added” to another element, there are no intervening elements and/or steps present.

Whenever the term “about” or “approximately” is used, it is meant to refer to a measurable value such as an amount, a temporal duration, and the like, and is meant to encompass variations (e.g., $\pm 20\%$, $\pm 10\%$, $\pm 5\%$, $\pm 1\%$, $\pm 0.1\%$) from the specified value, as such variations are appropriate to perform the disclosed methods.

Certain features of the invention, which are, for clarity, described in the context of separate embodiments, may also be provided in combination in a single embodiment. Conversely, various features of the invention, which are, for brevity, described in the context of a single embodiment, may also be provided separately or in any suitable sub-combination or as suitable in any other described embodiment of the invention. Certain features described in the context of various embodiments are not to be considered essential features of those embodiments, unless the embodiment is inoperative without those elements.

Whenever terms “plurality” and “a plurality” are used it is meant to include, for example, “multiple” or “two or more”. The terms “plurality” or “a plurality” may be used throughout the specification to describe two or more components, devices, elements, units, parameters, or the like. The term set when used herein may include one or more items. Unless explicitly stated, the method embodiments described herein are not constrained to a particular order or sequence. Additionally,

some of the described method embodiments or elements thereof can occur or be performed simultaneously, at the same point in time, or concurrently.

Throughout, this disclosure mentions “disclosed embodiments”, “disclosed systems” and “disclosed methods”, which refer to examples of inventive ideas, concepts, and/or manifestations described herein. The fact that some disclosed embodiments are described as exhibiting a feature or characteristic does not mean that other disclosed embodiments necessarily share that feature or characteristic.

This disclosure employs open-ended permissive language, indicating for example, that some embodiments “may” employ, involve, or include specific features. The use of the term “may” and other open-ended terminology is intended to indicate that although not every embodiment may employ the specific disclosed feature, at least one embodiment employs the specific disclosed feature.

The term “user” is used herein to refer to any individual person or group of persons using or operating a method or system of a disclosed embodiment.

Notation

Throughout the present disclosure, group elements and sets are denoted by uppercase (large) letters while field elements are typically denoted by lowercase (small) letters. Algorithms are denoted by calligraphic uppercase letters. Message identifiers may be denoted in a sans-serif font (identifier). Subscripts are usually reserved for descriptors, and superscripts are used for indexing. In cases where only a descriptor or only an index are needed it will be subscripted. For a finite set A , the notation: $x \leftarrow A$ implies that x is sampled uniformly at random from A . The notation $x \leftarrow A$ may also indicate the output of an algorithm. The notation $\binom{[n]}{t}_B$ denotes all the subsets of B of size t .

The disclosure relates to the ideal functionalities $\mathcal{F}_{broadcast}$, $\mathcal{F}_{global-broadcast}^{\Gamma_B}$ representing two modes of communication used in the disclosed protocols. One mode is an internal communication happening inside the distributed entity B and global communication in which B is presented as a single entity communicating with another of parties A .

Definition: “*Internal Broadcast Functionality $\mathcal{F}_{broadcast}$* ”.

The functionality interacts with a set of parties $B = \{B_i\}_{i \in [n]}$ and an adversary $\mathcal{A}$.

- Upon receiving $(broadcast, sid, i, msg)$ send $(broadcast, sid, i, msg)$ to $\mathcal{A}$ and record.
- Upon receiving $(req-output, sid, i, j)$ from $\mathcal{A}$ check if $(broadcast, sid, i, msg)$ was recorded, if so send $(broadcast, sid, i, msg)$ to B_j .

Definition: “*Global Broadcast Functionality $\mathcal{F}_{global-broadcast}^{\Gamma_B}$* ”.

The functionality interacts with two sets of parties $A = \{A_{pid_A}\}$, $B = \{B_i\}_{i \in [n]}$ and an adversary $\mathcal{A}$.

- Upon receiving $(global-broadcast, sid, i, msg)$ send $(global-broadcast, sid, i, msg)$ to $\mathcal{A}$ and record.
- Upon receiving $(req-output, sid)$ from $\mathcal{A}$ check if $(broadcast, sid, i, msg)$ was recorded for some $i \in B$, if so send $(global-broadcast, sid, i, msg)$ to all parties in A .

$\mathcal{F}_{global-broadcast}^{\Gamma_B}$ seems almost identical to $\mathcal{F}_{broadcast}$, however the receiver does not obtain information on the sender among the parties in B and thus sees them as a single entity.

Another difficulty which arises in an asynchronous model is agreeing upon a subset that has sent valid messages and that has been received by enough parties to continue the execution. Furthermore, an adversary may have some effect on this chosen subset. This is modelled by the ideal functionality $\mathcal{F}_{ACS}$, defined as follows:

Definition: “Asynchronous Common Subset Functionality $\mathcal{F}_{ACS}^{S_B}$ ”.

The functionality interacts with a set of parties $B = \{B_i\}_{i \in [n]}$ and an adversary $\mathcal{A}$ controlling a subset $U_B \subseteq B$ of the parties. It is parameterized by a set of legal subsets $\mathcal{S}_B \subset 2^B$.

- Upon receiving $(validate, sid, i, j)$ from B_i , record and send $(validate, sid, i, j)$ to $\mathcal{A}$.
- Upon receiving $(req-output, sid, i, S_B)$ from $\mathcal{A}$ the functionality does as follows:
 - Check if it has recorded $(authorized-subset, sid, S'_B)$ for $S'_B \neq S_B$. If so, ignore the message
 - Check that $\forall B_j \in S_B$ it has a record $(validate, sid, i, j)$ and that $S_B \in \mathcal{S}_B$. If record $(authorized-subset, sid, S_B)$ and outputs $(authorized-subset, sid, S_B)$ to B_i . Else it ignores the message.

Definition: “Aggregation Protocol $\Pi_{agg}^{\Gamma_B, L}$ ”.

The protocol interacts with a set of parties $B = \{B_i\}_{i \in [n]}$. It is parameterized by an access structure $\Gamma_B \subset 2^B$ along with a Maurer language L . Each party B_i holds a tuple $(Y_i; w_i) \in L$.

- Each B_i does as follows:
 1. Run a ZK protocol $\Pi_{zk}^L(Y_i; w_i)$ generating a proof π_i . Send $(broadcast, sid, i, (proof, Y_i, \pi_i))$ to $\mathcal{F}_{broadcast}$.
 2. Upon receiving $(broadcast, sid, i, (proof, Y_j, \pi_j))$ verify the proof π_j . If the proof verifies then send $(validate, sid, i, j)$ to $\mathcal{F}_{ACS}$, else consider B_j as malicious.
 3. Receive $(authorized-subset, sid, S_B)$ from $\mathcal{F}_{ACS}$ output $\sum_{j \in S_B} Y_j$.

A Shamir secret sharing scheme presented over a finite field is known in the art. But in some use cases, such as when working over a group of unknown order, it is necessary for the sharing to be correct over the integers.

Definition: “Shamir Secret Sharing Over the Integers”.

Let $s \in \mathbb{Z} \cap [-b, b]$ be a secret. Define $\Delta_n = n!$ and define some bound $I(\sigma, n, b)$ on the absolute value of the coefficients of the polynomial. The algorithm $\text{Share}_{t,n}(s)$ picks $a_1, \dots, a_t \leftarrow [-I(\sigma, n, b), I(\sigma, n, b)]$ and outputs $([s]_1, \dots, [s]_n)$ where $[s]_j = p(j)$ and $p(x) = \Delta_n \cdot s + \sum_{i=1}^t a_i x^i$. Reconstruction works as follows: Given a set $S \in \binom{[n]}{t+1}$, then $p(0) = \sum_{j \in S} \lambda_{0,j}^S [s]_j = \Delta_n s$, where $\lambda_{0,j}^S$ is the Lagrange coefficient corresponding to the interpolation of the point i with the point j using the subset S . However, since the Lagrange coefficients might not be integers, they are first multiplied by Δ_n to obtain:

$$\frac{\sum_{j \in S} \Delta_n \lambda_{0,j}^S [s]_j}{\Delta_n^2} = s$$

For a t -out-of- n , the upper bound on the absolute value of the share on s is denoted by $D(\sigma, n, t, b)$.

Definition: “Additively Homomorphic Encryption (AHE)”.

An additively homomorphic encryption scheme is associated with an ensemble $\{\mathcal{K}_\kappa\}_\kappa$ and consists of four polynomial time algorithms $\text{AHE} = (\text{Gen}, \text{Enc}, \text{Dec}, \text{Add})$, specified as follows:

- $\text{Gen}(1^\kappa, \text{aux}) \rightarrow (\text{pk}; \text{sk})$ A probabilistic algorithm that is given a security parameter 1^κ and possibly some auxiliary information aux and samples a key pair $(\text{pk}; \text{sk})$ from $\mathcal{K}_\kappa$. The following assumes that the resulting pk contains the description of the security parameter 1^κ , the auxiliary information aux , as well as the plaintext, randomness and ciphertext spaces $\mathcal{P}_{\text{pk}}, \mathcal{R}_{\text{pk}}, \mathcal{C}_{\text{pk}}$, respectively, where $\mathcal{P}_{\text{pk}}$ is a $\mathbb{Z}$ -module. In addition, for the present disclosure an AHE scheme is associated with a language $L_{\text{GenAHE}}[\kappa] = \{\text{pk}; \text{aux} \mid (\text{pk}; \text{sk}) = \text{Gen}(1^\kappa, \text{aux})\}$ that expresses the fact that pk was generated correctly.
- $\text{Enc}(\text{pk}, \text{pt}; \eta_{\text{enc}}) \rightarrow \text{ct}$. A deterministic algorithm that on input a public key pk , a plaintext $\text{pt} \in \mathcal{P}_{\text{pk}}$, and a randomness $\eta_{\text{enc}} \in \mathcal{R}_{\text{pk}}$, outputs a ciphertext $\text{ct} \in \mathcal{C}_{\text{pk}}$. $\text{Enc}(\text{pk}, \text{pt})$ is defined as a probabilistic algorithm that first universally samples $\eta_{\text{enc}} \in \mathcal{R}_{\text{pk}}$ and then outputs $\text{ct}' = \text{Enc}(\text{pk}, \text{pt}; \eta_{\text{enc}}) \in \mathcal{C}_{\text{pk}}$.
- $\text{Dec}(\text{sk}, \text{ct}) \rightarrow \text{pt}$. A deterministic algorithm that on input a secret key sk and a ciphertext $\text{ct} \in \mathcal{C}_{\text{pk}}$, outputs a plaintext $\text{pt} \in \mathcal{P}_{\text{pk}}$.
- $\text{Add}(\text{pk}, \text{ct}_1, \text{ct}_2) \rightarrow \text{ct}_3$. A deterministic algorithm that on input a public key pk and two ciphertexts $\text{ct}_1, \text{ct}_2 \in \mathcal{C}_{\text{pk}}$, outputs a ciphertext $\text{ct}_3 \in \mathcal{C}_{\text{pk}}$, such that if $\text{ct}_1 = \text{Enc}(\text{pk}, m_1; \eta_1)$ and $\text{ct}_2 = \text{Enc}(\text{pk}, m_2; \eta_2)$ then $\text{ct}_3 = \text{Enc}(\text{pk}, m_1 + m_2; \eta_1 + \eta_2)$, where $m_1 + m_2$ and $\eta_1 + \eta_2$ are over $\mathcal{P}_{\text{pk}}$ and $\mathcal{R}_{\text{pk}}$, respectively. Imposing correctness will ensure that Add is a homomorphic addition. It is noted that efficient homomorphic scalar multiplication is implied by the Add operation (e.g., via double-and-add).

In addition, a homomorphic addition of two ciphertexts ct_1 and ct_2 is denoted by $\text{ct}_1 \oplus \text{ct}_2$, and a multiplication of a ciphertext ct by a scalar α is denoted by $\alpha \odot \text{ct}$.

Every affine function can be efficiently computed homomorphically by an algorithm $\text{Eval}(\text{pk}, f, \text{ct}_1, \dots, \text{ct}_\ell; \eta_{\text{eval}})$. Whenever the randomness in $\text{Eval}(\text{pk}, f, \text{ct}_1, \dots, \text{ct}_\ell)$ is omitted, one

refers to the process of sampling a randomness $\eta_{\text{eval}} \leftarrow \{0,1\}^{\text{poly}(\kappa)}$ and running $\text{Eval}(\text{pk}, f, \text{ct}_1, \dots, \text{ct}_\ell; \eta_{\text{eval}})$. Given a public key pk , an affine function $f(x_1, \dots, x_\ell) = a_0 + \sum_{i=1}^\ell a_i x_i$ (with $a_i \in \mathbb{Z}$ and $\ell, \|a_i\|_2 \in \text{poly}(\kappa)$ for all $0 \leq i \leq \ell$) and ℓ ciphertexts $\text{ct}_1, \dots, \text{ct}_\ell \in \mathcal{R}_{\text{pk}}$, Eval outputs a ciphertext ct .

Definition: “AHE correctness”.

AHE is correct if for every κ , every $t \in \text{poly}(\kappa)$, every ℓ -ary affine function f as above, and every plaintext $\text{pt}_1, \dots, \text{pt}_t \in \mathcal{P}_{\text{pk}}$,

$$\Pr \left[\text{Dec} \left(\text{sk}, \text{Eval}(\text{pk}, f, (\text{ct}_1, \dots, \text{ct}_\ell)) \right) = f(\text{pt}_1, \dots, \text{pt}_\ell) \right] \geq 1 - \text{neg}(\kappa),$$

where $(\text{pk}, \text{sk}) \leftarrow \text{Gen}(1^\kappa)$ and $\text{ct}_i \leftarrow \text{Enc}(\text{pk}, \text{pt}_i)$, and the probability is over the coins used by the algorithms Gen , Enc and Eval .

For correctness, algorithm Eval may simply output $\text{ct} = \text{Enc}(\text{pk}, a_0) \oplus_{i=1}^\ell a_i \odot \text{ct}_i$. However, such as algorithm may not satisfy secure function evaluation.

Definition: “AHE semantic security”.

AHE is semantically secure if for every PPT adversary $\mathcal{A}$ there exists a PPT algorithm $\mathcal{S}$ such that for every pair of PPTs $f; h: \{0,1\}^* \rightarrow \{0,1\}^*$, and every $\text{pt} \in \mathcal{P}_{\text{pk}}$:

$$\left| \Pr \left[\mathcal{A} \left(1^\kappa, \text{Enc}(\text{pk}, \text{pt}), 1^{|\text{pt}|}, h(1^\kappa, \text{pt}) \right) = f(1^\kappa, \text{pt}) \right] - \Pr \left[\mathcal{S} \left(1^\kappa, 1^{|\text{pt}|}, h(1^\kappa, \text{pt}) \right) = f(1^\kappa, \text{pt}) \right] \right| < \text{neg}(\kappa)$$

where $(\text{pk}, \text{sk}) \leftarrow \text{Gen}(1^\kappa)$ and h refers to the auxiliary information function in possession of the adversary. The probability is over the random coins of Gen and Enc .

Definition: “Secure Function Evaluation”.

AHE has secure function evaluation if there exists a PPT algorithm $\mathcal{S}_{\text{Eval}}$ such that for every PPT adversary $\mathcal{A}$, every ℓ -ary affine function f as above, every (pk, sk) and every plaintext $\text{pt}_1, \dots, \text{pt}_\ell \in \mathcal{P}_{\text{pk}}$, and randomness $\eta_1, \dots, \eta_\ell$,

$$\left| \Pr \left[\mathcal{A} \left(1^\kappa, \text{pk}, \text{sk}, \{\text{pt}_i; \eta_i\}_{i=1}^\ell, \text{Eval}(\text{pk}, f, (\text{ct}_1, \dots, \text{ct}_\ell)) \right) = 1 \right] - \Pr \left[\mathcal{A} \left(1^\kappa, \text{pk}, \text{sk}, \{\text{pt}_i; \eta_i\}_{i=1}^\ell, \mathcal{S}_{\text{Eval}}(\text{pk}, f, (\text{pt}_1, \dots, \text{pt}_\ell)) \right) = 1 \right] \right| \leq \text{neg}(\kappa)$$

where $\text{ct}_i \leftarrow \text{Enc}(\text{pk}, \text{pt}_i; \eta_i)$, and the probability is over the coins used by Eval and the random coins of $\mathcal{A}$ and $\mathcal{S}$.

A TAHE scheme is defined be a tuple of algorithms $(\text{Gen}, \text{Enc}, \text{Add}, \text{TDec}, \text{Rec})$. Gen , Enc , Add works similarly to the definition of AHE, while TDec (threshold decryption) is an operation that each party runs locally on the ciphertext and generates a decryption share. Collecting $t + 1$ such decryption shares as inputs to Rec outputs a correct decryption.

Definition: “Threshold additively homomorphic encryption (TAHE)”.

Threshold additively homomorphic encryption is associated with an ensemble $\{K_\kappa\}_\kappa$ and consists of five polynomial time algorithms $\text{TAHE}=(\text{Gen}, \text{Enc}, \text{Add}, \text{TDec}, \text{Rec})$ specified as follows:

- $\text{Gen}(1^\kappa, i, t, n, \text{aux}) \rightarrow (pk, sk_i)$. A probabilistic (possibly interactive) protocol that is given a security parameter 1^κ , the number of parties n , a threshold $1 < t < n$ and possibly some auxiliary information aux and samples a keypair $(pk, \{sk_i\}_{i \in [n]}) \in K_\kappa$. In the following, it is assumed that the resulting pk contains the description of the security parameter 1^κ , the auxiliary information aux , as well as the plaintext, randomness and ciphertext spaces $\mathcal{P}_{pk}, \mathcal{R}_{pk}, \mathcal{C}_{pk}$, respectively, where $\mathcal{P}_{pk}$ is a $\mathbb{Z}$ -module.
- $\text{Enc}(pk, pt; \eta_{enc})$. A deterministic algorithm that on input a public key pk , a plaintext $pt \in \mathcal{P}_{pk}$, and randomness $\eta_{enc} \in \mathcal{R}_{pk}$, outputs a ciphertext $ct \in \mathcal{C}_{pk}$. $\text{Enc}(pk, pt)$ is defined as a probabilistic algorithm that first uniformly samples $\eta_{enc} \leftarrow \mathcal{R}_{pk}$ and then outputs $ct = \text{Enc}(pk, pt; \eta_{enc}) \in \mathcal{C}_{pk}$.
- $\text{TDec}(sk_i, ct) \rightarrow ct_i$. A deterministic algorithm that on input a secret key share sk_i and a ciphertext $ct \in \mathcal{C}_{pk}$ outputs a decryption share $ct_i \in \mathcal{C}_{pk}$.
- $\text{Rec}(S, \{ct_i\}_{i \in S}) \rightarrow pt$. A deterministic algorithm that on input of a set $S \in \binom{[n]}{t+1}$ and a tuple of elements $\{ct_i\}_{i \in S} \in \mathcal{C}_{pk}^S$ outputs a plaintext $pt \in \mathcal{P}_{pk}$.
- $\text{Add}(pk, ct_1, ct_2) \rightarrow ct_3$. A deterministic algorithm that on input a public key pk and two ciphertexts $ct_1, ct_2 \in \mathcal{C}_{pk}$ outputs a ciphertext $ct_3 \in \mathcal{C}_{pk}$.

Definition: “TAHE correctness”.

TAHE is correct if for every κ , every $t \in \text{poly}(\kappa)$, every ℓ -ary affine function f , any subset $S \in \binom{[n]}{t+1}$, and every plaintext $pt_1, \dots, pt_\ell \in \mathcal{P}_{pk}$,

$$\Pr \left[\text{Rec} \left(S, \left\{ \text{TDec} \left([sk]_i, \text{Eval}(pk, f, (ct_1, \dots, ct_\ell)) \right) \right\}_{i \in S} \right) = f(pt_1, \dots, pt_\ell) \right] \geq 1 - \text{neg}(\kappa)$$

where $(pk, \{[sk]_i\}_{i \in [n]}) \leftarrow \text{Gen}(1^\kappa)$ and $ct_i \leftarrow \text{Enc}(pk, pt_i)$, and the probability is over the coins used by the algorithms Gen , Enc and Eval .

A TAHE functionality $\mathcal{F}_{\text{TAHE}}$ is defined below. Let $B = \{B_1, \dots, B_n\}$ be a set of parties and $U \subset B$, where $|U| \leq t < n$, be those parties under the control of the adversary. The TAHE functionality $\mathcal{F}_{\text{TAHE}}$ is parameterized with an AHE scheme $\text{AHE}=(\text{Gen}, \text{Enc}, \text{Dec}, \text{Add})$ with semantic security and secure function evaluation. The parties in B can generate a key and prove its validity to the certifiers. In the context of the disclosed 2PC-MPC protocol, the certifiers are the users (represent by client A) who engage with B . Finally, any set of $t + 1$ honest parties can decrypt a ciphertext using a previously generated key, unless the adversary decides to abort.

Definition: “TAHE functionality $\mathcal{F}_{\text{TAHE}}$ ”

The functionality interacts with $B = (B_1, \dots, B_n)$, an adversary that statically corrupts a subset $\mathcal{U}$ of the parties and a set of certifiers. It is parameterized with $\text{AHE}=(\text{Gen}, \text{Enc}, \text{Dec}, \text{Add})$, κ and $t < n$.

- Upon receiving **(keygen, sid)** from all parties in B , for a fresh sid , run $(pk; sk) \leftarrow \text{AHE.Gen}(1^\kappa)$, send pk to the adversary and receive the adversary response $(\text{continue}, \mathcal{U}')$. If $\text{continue} = 1$ or $\mathcal{U}' \cap \mathcal{U} = \emptyset$ then send **(pubkey, sid, pk)** to all parties in B and store $(\text{sid}, pk; sk)$, otherwise send **(pubkey, sid, $\perp$, $\mathcal{U}' \cap \mathcal{U}$)**.
- Upon receiving **(certify, pk)** from some party, if $(\cdot, pk; \cdot)$ is stored then return 1 to that party, otherwise return 0.
- Upon receiving **(decrypt, pk, ct)** from at least $t + 1$ parties, if $(\text{sid}, pk; sk)$ is stored for some sid and sk , compute $\text{pt} = \text{AHE.Dec}(sk, ct)$, send **(decrypted, pk, ct, pt)** to the adversary and wait to the adversary response $(\text{continue}, \mathcal{U}')$. If $\text{continue} = 1$ or $\mathcal{U}' \cap \mathcal{U} = \emptyset$ then send **(decrypted, pk, ct, pt)** to everyone, otherwise send **(decrypted, pk, ct, $\perp$, $\mathcal{U}' \cap \mathcal{U}$)** to everyone.

Realization of a TAHE functionality $\mathcal{F}_{\text{TAHE}}$ is done using a tuple of protocols $(\text{Gen}, \text{Enc}, \text{Add}, \text{TDec}, \text{Rec})$. The $(\text{Gen}, \text{Enc}, \text{Add})$ protocols are similar to a standard AHE, while TDec is an operation that each party runs locally on a ciphertext and generates a decryption share. Collecting $t + 1$ such decryption shares as inputs to Rec outputs a correct decryption. Since the TDec operation is local, typically one must add a ZK proof that this was done correctly and verify all proofs before applying Rec to them. Verifying these proofs, especially when the number of parties (hence t) is large, can dominate the protocol runtime. In addition, the combination in Rec itself may be very expensive, especially when the plaintext space is of unknown order and so the secret decryption key is shared over the integers (as in a Paillier scheme). One possible optimization is for one party to locally compute Rec and provide a proof of correct recombination. Unfortunately, generation and verification of such proofs may not be efficient in general. But in a case where encryption is of a signature, one may verify the correctness by verifying the signature.

Definition: “Dynamical TAHE”

In the disclosed protocols, the parties need to be able to re-share their secret keys such that the distribution of the secret keys represent their current stake in the blockchain. To this end, the TAHE may also include a reconfiguration protocol: $\text{ReConf}(1^\kappa, 1^\sigma, t, t', n, n', \text{pk}, \text{aux}, [\text{sk}]_i)$, which is a probabilistic (possibly interactive) protocol that is given a security parameter 1^κ , a statistical security parameter 1^σ , the initial threshold t , the final threshold t' , the initial number of parties n , possibly some auxiliary information aux , and the output $(\text{AHE}, [\text{sk}]_i)$ of the protocol. A DTAHE is considered secure if it realizes the ideal functionality defined below:

Definition: “Dynamical Access Structure AHE Version 2: $\mathcal{F}_{\text{DAHE}}$ ”

The functionality interacts with a set of parties $B = (B_1, \dots, B_n)$ and an adversary controlling a subset $U \subseteq B$. It is parameterized with an $\text{AHE}=(\text{Gen}, \text{Enc}, \text{Dec}, \text{Add})$ along with an initial access structure Γ_{keygen} .

1. Key Generation: Upon receiving (keygen,pid,sid) from a subset $S_{\text{keygen}} \in \Gamma_{\text{keygen}}$ from a fresh sid, run $(pk; sk) \leftarrow \text{AHE.Gen}(1^\kappa)$, send pk to the adversary and receive the adversary response (continue, U'). If $U \cap U' = \emptyset$ then send (pubkey,sid,pk) to all parties in B , define $\Gamma = \Gamma_{\text{keygen}}$ and store (access – structure,sid, Γ) to memory, otherwise send (pubkey,sid, $\perp$, $U \cap U'$) to all parties in B .
2. Decryption: Upon receiving a (decrypt,pid,sid,ct) let $S_{\text{decrypt,sid,ct}} = S_{\text{decrypt,sid,ct}} \cup \{\text{pid}\}$ (if no request was recorded yet treat $S_{\text{decrypt,sid,ct}}$ as the empty set) and do the following:
 - (a) Check if (decrypted,sid,ct,pid) in memory else calculate $\text{pt} \leftarrow \text{AHE.Dec}(pk,ct; sk)$ and store (decrypted,sid,ct,pid) to memory.
 - (b) Check if $S_{\text{decrypt,sid,ct}} \cup U \in \Gamma$ for (access – structure,sid, Γ) in memory, if true send (decrypted,sid,ct,pid) to U .
 - (c) Check if $S_{\text{decrypt,sid,ct}} \in \Gamma$ for (access – structure,sid, Γ), if true send (decrypted,pk,ct,pt) to the adversary and wait for the adversary response (continue, U'') if $U \cap U'' = \emptyset$ send (decrypted,sid,ct,pt) to everyone, otherwise send (decrypted,pk, $\perp$, $U \cap U''$).
3. Reconfiguration: Upon receiving (re-configure,pid,sid, Γ') from a subset $S \in \Gamma$, delete (re-configure,sid, Γ) and store (re-configure,sid, Γ') instead. Furthermore, delete any message titled decrypt or decrypted along with the sets $S_{\text{decrypt,sid,ct}}$.

The language associated with relation R is the set $L_R = \{x \mid \exists w : (x, w) \in R\}$. $L[\text{params}] = \{S; s \mid f(\text{params}, S, s)\}$ denotes a language $\{x \mid \exists s : f(\text{params}, S, s) = 1\}$, where params typically refers to global parameters (e.g., a public key of an encryption scheme, or a global verification key), S refers to a set of public values, and s refers to a set of secret values. Note that $\mathcal{F}_{zk}^R$ implicitly requires knowledge of the witness s from the prover P . Note that params may be omitted from a language specification when it is clear from context. The disclosed protocols may include the following languages:

- Knowledge of the discrete log of an elliptic-curve point:

$$L_{\text{DL}}[(\mathbb{G}, G, q)] = \{(P; x) \mid P = x \cdot G\}.$$

- Knowledge of decommitment:

$$L_{\text{DCom}}[\text{pp}] = \{C; m, \rho \mid C = \text{Com}_{\text{pp}}(m; \rho) \wedge m \in \mathcal{M}_{\text{pp}} \wedge \rho \in \mathcal{R}_{\text{pp}} \wedge C \in \mathcal{C}_{\text{pp}}\}.$$

- Knowledge of plaintext:

$$L_{\text{pt}}[\text{AHE}, pk] = \{\text{ct}; m, \eta \mid \text{ct} = \text{AHE.Enc}(pk, m, \eta) \wedge m \in \mathcal{P}_{\text{pk}} \wedge \eta \in \mathcal{R}_{\text{pk}} \wedge C \in \mathcal{C}_{\text{pk}}\}.$$

If ct is a vector then interpret the language as proving knowledge of plaintext for each element in the vector.

- Commitment of discrete log:

$$L_{\text{DComDL}}[\text{pp}, (\mathbb{G}, G, q)] = \{C, Y; m, \rho \mid C = \text{Com}_{\text{pp}}(m; \rho) \wedge Y = \sum_{\ell} m_{\ell} \cdot G_{\ell}\}.$$

- Vector Commitment of discrete log:

$$L_{\text{VecDComDL}}[\text{pp}, (\mathbb{G}, G, q)] = \{C, Y; m, \rho \mid \forall \ell \ C_\ell = \text{Com}_{\text{pp}}(m_\ell; \rho_\ell) \wedge Y = \sum_\ell m_\ell \cdot G_\ell\}.$$

- Equality between two commitments with different public parameters:

$$L_{\text{DComEQ}}[\text{pp}_1, \text{pp}_2] = \{C_1, C_2; m, \rho_1, \rho_2 \mid C_1 = \text{Com}_{\text{pp}_1}(m; \rho_1) \wedge C_2 = \text{Com}_{\text{pp}_2}(m; \rho_2)\}.$$

- Committed affine evaluation:

$$L_{\text{DComEval}}[\text{pp}, pk, ct, q, \ell, s, \kappa] = \{ct^{\text{Eval}}, C; a; \rho, \omega, \eta \mid ct^{\text{Eval}} = \text{AHE.Eval}(pk, f_a(x), ct; \eta) \wedge C = \text{Com}_{\text{pp}}(a; \rho) \wedge a_\ell \in [0, q) \wedge \rho \in \mathcal{R}_{\text{pp}} \wedge \eta \in \mathcal{R}_{\text{pk}}\}.$$

- AHE encryption of a discrete log:

$$L_{\text{EncDL}}[pk, (\mathbb{G}, G, q)] = \{(ct, X; x, \eta) \mid ct = \text{AHE.Enc}(pk, x; \eta) \wedge X = x \cdot G \wedge x \in [0, q)\}.$$

- AHE scaling of a discrete log:

$$L_{\text{ScaleDL}}[pk, (\mathbb{G}, G, q), ct_0] = \{(ct_1, X; x, \eta) \mid ct = \text{AHE.Scale}(pk, ct_0, x; \eta) \wedge X = x \cdot G \wedge x \in [0, q)\}.$$

- AHE analog of a DH tuple, namely ciphertexts of values

$$L_{\text{EncDH}}[pk, ct_x] = \{(ct_y, ct_z; y, \eta_y, \eta_z) \mid ct_y = \text{AHE.Enc}(pk, y; \eta_y) \wedge ct_z = \text{AHE.Eval}(pk, f, ct_x; 0; \eta_z) \text{ s.t. } f(x) = y \cdot x \bmod q \wedge y \in [0, q)\}.$$

- Correct AHE key generation:

$$L_{\text{GenAHE}}[\kappa] = \{pk; \text{aux} \mid (pk; sk) = \text{AHE.Gen}(1^\kappa, \text{aux})\}.$$

All languages can be extended to include range claims, which may be done by adding range to the name of the language and including a vector representing the bounds on the witnesses.

A Publicly Verifiable Secret Sharing (PVSS) allows a dealer to share a secret among a set of recipients for some monotone access structure, while any recipient can verify the correctness of the shares of every other recipient without further communication. Typically, this is achieved by the dealer generating encryptions under public keys of each recipient while proving using a Non-Interactive Zero-Knowledge proof that the sharing was done correctly. A main advantage of the aforementioned approach is that it prevents the need for a complicated and communication complex accountability mechanism in case a recipient claim they received a wrong share.

Definition: “Publicly Verifiable Secret Sharing (PVSS)”

A PVSS scheme consists of the following algorithms:

- Setup:
 - $\text{pp} \leftarrow \text{Setup}(1^\kappa, \text{ip})$ outputs public parameters pp . The initial parameters ip contain information about the number of parties, privacy and Reconstruction thresholds and spaces of secrets and shares. The public parameters include a description of the key space $\mathcal{K}_\kappa$, which should be thought of as a Cartesian product over the public key share and the private key share along with some relation they satisfy.

- $(pk_i, \pi_{pk}^i, sk_i) \leftarrow \text{Gen}(pp, i)$ where $(pk; sk) \in \mathcal{K}_\kappa$ and π_{pk} is a proof meant to assert that pk is a valid public key.
- $0/1 \leftarrow \text{Verify}(pp, j, pk_j, \pi_{pk}^j)$, a verification algorithm for the proof π_{pk}^j .
- Distribution: $(\{ct_i\}_{i \in [n]}, \pi_{share}) \leftarrow \text{Dist}(pp, \{pk\}_{i \in [n]}, s)$, where $s \in \mathcal{S}$ is a valid secret, outputs “encrypted shares” ct_i , and a proof that they are correct π_{share} .
- Distribution Verification: $0/1 \leftarrow \text{VerifySharing}(pp, \{pk_i, ct_i\}_{i \in [n]}, \pi_{share})$, a verification algorithm for π_{share} .
- Reconstruction:
 - $([s]_i, \pi_{dec}^i) \leftarrow \text{DecShare}(pp, pk_i, sk_i, ct_i)$ outputs a decrypted share $[s]_i$ and a proof π_{dec}^i of correct decryption.
 - $(s') \leftarrow \text{Rec}(pp, \{[s]_i\}_{i \in \mathcal{S}})$ either returns abort if $S \notin \Gamma$ or an element of the secret space $s' \in \mathcal{S}$ if $S \in \Gamma$.
- Reconstruction Verification: $0/1 \leftarrow \text{VerifyDec}(pp, pk_i, ct_i, [s]_i, \pi_{dec}^i)$, a verification algorithm for π_{dec}^i .

Reference is made to Figure 1, which is a schematic illustration of a system 100 for generating a digital signature, constructed and operative in accordance with an embodiment of the present disclosure. System 100 includes a decentralized network (DN) 105 and a plurality of clients 210 ($A_1, \dots, A_n$). Decentralized network 105 includes a plurality of parties 110 ($B_1, \dots, B_n$). Each party (B_i) has a respective voting power (VP_i). Parties 110 are communicatively coupled with one another, such that each party 110 may receive and transmit messages or information to another party 110. Each client 210 may receive and transmit messages to or from at least one other party. In the example configuration depicted in Fig. 1, client 1 is communicatively coupled with party 1; each of client 2 and client 3 is communicatively coupled with party 3, and client 4 is communicatively coupled with each of party 2 and party 4.

Reference is made to Figures 2 and 3. Figure 2 is a schematic illustration of a decentralized network party 110 of system 100, constructed and operative in accordance with an embodiment of the present disclosure. Figure 3 is a schematic illustration of a client 210 of system 100, constructed and operative in accordance with an embodiment of the present disclosure. Each of decentralized network (DN) party 110 and client 210 may be implemented in several ways, such as for example as an integrated circuit, and/or an embedded software, application, module, process or subsystem of a computer, and may be used as part of one or more tools or activities over the Internet or an intranet. In one example, party 110 and client 210 are respectively embodied by an integrated circuit (IC) or system on chip (SOC) 120, 220 of a computing device, such as a smartphone, a laptop computer, a mobile computer, a tablet computer, a wearable computer, or any type of electronic device with computing and processing capabilities. It should be noted, however, that the present invention is not limited to the described embodiments and can be used in any system where digital signature generation may be implemented.

SOC 120 of party 110 includes on-chip components, including a processor 122, a memory 124, a communication interface 126, and a plurality of DN party modules 130, which are interconnected or communicatively coupled via a bus 128. Similarly, SOC 220 of client 210 includes on-chip components, including a processor 222, a memory 224, a communication interface 226, and a plurality of client modules 230, which are interconnected or communicatively coupled via a bus 228. Processor 122, 222 performs data processing and may receive instructions or information from other components of SOC 120, 220. Memory 124, 224 is configured for data storage. Communication interface 126, 226 may transmit information to and/or receive information from an external source, such as one or more data networks, using any suitable communication channel or network model and data transmission protocol. Information may be conveyed between the components of respective SOC 120, 220 over bus 128, 228.

DN party modules 130 includes a network distributed key generation (DKG) module 140, a network ECDSA/Schnorr presign module 150, a network Schnorr presign module 155, a network ECDSA sign module 160, a network Schnorr sign module 165, a network dynamic threshold additively homomorphic encryption (DTAHE) module 170, and an ownership transfer verification module 180. Network DTAHE module 170 includes a network reconfiguration (reconfig) module 175. Client modules 230 includes a client distributed key generation (DKG) module 240, a client ECDSA/Schnorr presign module 250, a client Schnorr presign module 255, a client ECDSA sign module 260, a client Schnorr sign module 265, a client DTAHE module 270, and a client ownership transfer module 280. DN party modules 130 and client modules 230 are configured to execute cryptographic operations or protocols, such as by applying one or more mathematical operations, equations and/or processes on variables or input data, which may result in corresponding output data. Such cryptographic operations may include processes relating to digital signature generation, such as distributed key generation, pre-signature, signature, encryption, and decryption processes. Processing associated with DN party modules 130 may be implemented at least in part using network processor 122, while associated with client modules 230 may be implemented at least in part using client processor 222. It is noted that the functionality of DN party 110 may be implemented as a stand-alone network party module 130 but is described in Fig. 2 as part of an SOC 120. Similarly, the functionality of client 210 may be implemented as a stand-alone client module 230 but is described in Fig. 3 as part of an SOC 220. In embodiments, the functionality of one or more DN party modules 130 may be implemented in hardware, software, firmware, or a combination thereof.

DN party modules 130 and client modules 240 may be communicatively coupled through at least one communication network, and may be executed by one or more devices or components, each of which may reside at a respective location. Accordingly, information may be conveyed between the respective modules over the communication network as well as to/from other networks communicatively coupled thereto, via any suitable data communication channel or network, using any suitable channel or network model and any suitable data transmission protocol, such as via a secured communication protocol. For example, a client key generation may be implemented by a client DKG module 240 executing in a cloud server using a cloud computing platform.

The components and devices of system 100 may be based in hardware, software, or combinations thereof. It is appreciated that the functionality associated with each of the devices or

components of system 100 may be distributed among multiple devices or components, which may reside at a single location or at multiple locations. For example, the functionality associated with processor 122, 222 may be distributed between a single processing unit or multiple processing units, including different types of processing units. Furthermore, at least part of the functionality associated with party 110 and/or client 210 may reside externally to SOC 120 or SOC 220. For example, the functionality of at least one of DN party modules 130 may operate at least partially on SOC 120 (e.g., using processor 122) and at least partially on a server or remote computer system accessible over a communications network, such as a cloud computing server. System 100 may optionally include and/or be associated with additional components or modules not shown in Figures 1, 2, 3 for enabling the implementation of the disclosed subject matter.

System 100 is generally configured for securely generating a digital signature. A digital signature scheme includes a key generation routine, a signing routine, and a signature verification routine. In turn, system 100 implements a secure distributed key generation protocol, a secure distributed signing protocol, and an optional secure distributed pre-signing protocol. A key generation routine outputs a private key and a corresponding public key, where the private key is selected from a set of possible private keys. System 100 implements key generation securely by implementing a distributed key generation (DKG) protocol, in which multiple parties contribute to the generation of a shared secret key. A DKG protocol does not rely on a trusted third party, but rather a threshold of honest parties determines whether a public and private key set can be computed successfully.

In accordance with a 2PC-MPC protocol of the disclosed embodiments, there are n independent parties $B_1, \dots, B_n$ that together emulate a single party B interacting with a client A . When party B is distributed, all secret values of B are distributed as well. In particular, the AHE private key sk should also be shared amongst the parties $B_1, \dots, B_n$. This leads to the definition of a threshold parameter $t \leq n$, and the assumption that an adversary may corrupt at most t parties out of $B_1, \dots, B_n$. A UC-secure protocol for threshold AHE key generation and threshold decryption may be used as an instantiation.

The protocols of the present disclosure, including DKG, signing, and pre-signing protocols, may be based on threshold additively homomorphic encryption (TAHE). In one example, TAHE may be a dynamic TAHE (DTAHE) to allow for changing voting powers of parties.

Reference is made to Figure 4, which is a schematic illustration of a threshold additively homomorphic encryption (TAHE) distributed key generation (DKG) process, operative in accordance with an embodiment of the present disclosure. A TAHE-DKG routine is implemented by one or more parties 110 of decentralized network 105, for generating keys of a TAHE. Network DKG module 140 of a respective party (B_i) implements a TAHE-DKG protocol to produce a respective secret private key (sk_i) for each party, verification keys ($\vec{vk}$) for all parties, a public key (pk), an optional evaluation key (evk), and an optional encryption of the secret key (ct_{sk}). For example, network DKG module 140 implements $\text{Gen}(1^\kappa, i, t, n, aux) \rightarrow (pk, sk_i)$ described previously (where pk may include $\vec{vk}, evk, ct_{sk}$).

In permissionless blockchains, voting power refers to the weight that a validator has in the consensus process, which is used to validate transactions and add new blocks to the blockchain. In

Proof of Stake (PoS) blockchains, voting power is proportional to the amount of cryptocurrency that a participant has staked or locked up in the network. Validators are chosen to propose and validate new blocks based on the amount of staked coins they hold, meaning the more they stake, the higher the voting power. Reconfiguration may allow validators to join or leave the network, and validators may also withdraw part of their stake or increase their stake, which could change their voting power accordingly.

Reference is made to Figure 5, which is a schematic illustration of exemplary secret key shares of a decentralized network party with a given voting power, operative in accordance with an embodiment of the present disclosure. The number of shares of each party 110 is proportional to its voting power (VP_i). Accordingly, as the voting power of a party increases so does the number of its secret key shares. For example, Party 1 produces a secret key (sk_1) that is proportional to its voting power (VP_1): $sk_1 = \{sk_1^1, \dots, sk_1^{VP_1}\}$.

The voting power of a party may be changed, when the protocols are based on a DTAHE. Reference is made to Figure 6, which is a schematic illustration of a network reconfiguration process, operative in accordance with an embodiment of the present disclosure.

In the example configuration depicted in Fig. 6, the voting power of Party 1 is changed from VP_1 to VP_1' , where $\Delta VP_1 > 0$, such that the Party 1 increases its voting power (i.e., $VP_1' > VP_1$). The voting power of Party 2 is changed from VP_2 to VP_2' , where $\Delta VP_2 < 0$, such that Party 2 decreases its voting power (i.e., $VP_2' < VP_2$). The voting power of Party 3 is changed from VP_3 to VP_3' , where $\Delta VP_3 = -VP_3$, such that Party 3 remains with a voting power of 0 and no longer has any secret shares or voting power. Party 4 starts off with no voting power and no secret key shares (i.e., $VP_4 = 0$), and has its voting power increased to provide it with new secret key shares (i.e., $VP_4' = \Delta VP_4$). Following the reconfiguration, each party obtains a secret key share in accordance with its new voting power: Party 1 generates sk_1' based on VP_1' ; Party 2 generates sk_2' based on VP_2' ; Party 4 generates sk_4' based on VP_4' ; and Party 3 does not generate a secret key share (since $VP_3' = 0$).

In one example the reconfiguration protocol uses an encryption of the secret key (ct_{sk}) parameter generated by the DTAHE-DKG protocol, and in another example the reconfiguration protocol does not use ct_{sk} . The first example also supports changing the threshold. This can potentially be used to adjust the resolution of the voting power distribution.

The following describes a reconfiguration protocol that uses ct_{sk} .

Definition: “Reconfiguration Protocol with Varying Threshold”

(Re-Configuration with Varying Threshold)

$$\prod_{ReConfigure}^{var-th} : Re-configure(TAHE, PVSS, 1^\kappa, 1^\sigma, t, t', n, n')$$

Reconfiguration Protocol with Varying Threshold is parameterized with a TAHE encryption scheme, a public verifiable secret sharing scheme PVSS, a computational security parameter κ , a statistical security parameter σ , an initial threshold t and a final threshold t' . The protocol interacts with a transferring quorum $B = \{B_i\}_{i \in [n]}$ and a receiving quorum $B' = \{B'_i\}_{i \in [n']}$. The

transferring quorum holds a t -out-of- n secret sharing $[sk]_i$ on the secret key of the TAHE sk . Both quorums hold a public key pk to the encryption scheme, the verification keys of the transferring quorum $\{vk_i\}_{i \in [n]}$ along with an encryption of the secret key $ct_{sk} = \text{Enc}_{pk}^b(sk; \eta)$. The protocol outputs a fresh t' -out-of- n' secret shares $[sk]_{i'}$ on the secret key sk held by B' and public verification keys $\{vk'_{i'}\}_{i' \in [n']}$ corresponding to the shares. It is assumed that when the protocol begins the parties already executed the PVSS.Setup and PVSS.Gen successfully, and as such holds appropriate public keys for the new parties $\{pk_{i'}\}_{i' \in [n']}$.

1. $B_i \in B$ does as follows:

Round 1:

- (a) Sample a mask $r_i \leftarrow [-D(\mathcal{K}_K) \cdot 2^{\sigma \cdot \omega}, D(\mathcal{K}_K) \cdot 2^{\sigma \cdot \omega}]$.
- (b) Calculate $ct_{mask}^i = \text{Enc}_{pk}^{b, 2^\sigma}(r_i; \eta_i)$. In addition generate proof $\pi_{mask}^i \leftarrow \Pi_{zk}^{L_{sk} \text{ masked}[pk, b \cdot 2^\sigma]}(ct_{mask}^i; r_i; \eta_i)$.
- (c) $(\{ct_{share}^{i, j'}\}_{j' \in [n]}, \{C_{i, j'}\}_{j' \in [n]}, \pi_{share}^{i'}) \leftarrow \text{PVSS.Dist}\left(\text{pp}, \{pk_{j'}\}_{j' \in [n]}, \text{Pack}_b([r_i]_{b \cdot 2^\sigma})\right)$.
- (d) Send $\left(\text{broadcast}, \text{sid}, i', (ct_{mask}^i, \pi_{mask}^i, \{ct_{share}^{i, j'}\}_{j' \in [n]}, \{C_{i, j'}\}_{j' \in [n]}, \pi_{share}^{i'})\right)$ to $\mathcal{F}_{broadcast}$.

Broadcast Round:

- (a) Receive $\left(\text{broadcast}, \text{sid}, j, (ct_{mask}^j, \pi_{mask}^j, \{ct_{share}^{j, j'}\}_{j' \in [n]}, \{C_{j, j'}\}_{j' \in [n]}, \pi_{share}^j)\right)$ from $\mathcal{F}_{broadcast}$.
- (b) Verify the proofs. If any of them fail consider j malicious. Else send $(\text{validate}, \text{sid}, i, j)$ to $\mathcal{F}_{ACS}^{\binom{[n]}{t+1} B}$.

Round 3:

- (a) Receive S_B from $\mathcal{F}_{ACS}$ and set $ct_{masked-key} = ct_{sk} \oplus \sum_{j \in S_B} ct_{mask}^j$.
- (b) Set $(ds_i, \pi_{ds}^i) \leftarrow \text{TAHE.TDec}_{pk}(ct_{masked-key}, vk_i; [sk]_i)$ and send $(\text{broadcast}, \text{sid}, i, (ds_i, \pi_{ds}^i))$ to $\mathcal{F}_{broadcast}$.
- (a) Broadcast Round: Upon receiving $(\text{broadcast}, \text{sid}, j, (ds_j, \pi_{ds}^j))$ verify that the proof π_{ds}^j is valid; if it fails consider j malicious.

(b) Collect a subset $S_B^i \in \binom{[n]}{t+1}$ with validated proofs and calculate $sk_{masked} = \text{TAHE.Rec}_{pk}^b(\{ds_j\}_{j \in S_B^i})$.

(c) Send

$$\left(\text{global-broadcast}, \text{sid}, i, \left(sk_{masked}, \left\{ \{ct_{Share}^{j,j'}\}_{j' \in [n']}, \{C_{j,j'}\}_{j' \in [n']}\right\}_{j \in S_B} \right) \right) \text{ to } \mathcal{F}_{\text{global-bro}}.$$

2. $B'_{i'} \in B$ does as follows:

(a) Upon

receiving

$$\left(\text{global-broadcast}, \text{sid}, \left(sk_{masked}, \left\{ \{ct_{Share}^{j,j'}\}_{j' \in [n']}, \{C_{j,j'}\}_{j' \in [n']}\right\}_{j \in S_B} \right) \right) \text{ from } \mathcal{F}_{\text{global-broadca}}, \text{ set } ([r_j]_{i'}) \leftarrow \text{PVSS.DecShare}(\text{pp}, \text{pk}_{i'}, sk_{i'}, ct_{Share}^{j,i'}).$$

(b) Calculate $[sk]_{i'} = \Delta sk_{masked} - \sum_{j \in S_B} [r_j]_{i'}$.

(c) Calculate $vk'_{j'} = \text{Com}_{pp}(\Delta sk_{masked}) - \sum_{j \in S_B} C_{j',j}$.

(d) Record $([sk]_{i'}, \{vk_{j'}\}_{j' \in [n']})$.

Definition: “Reconfiguration Protocol with Constant Threshold”

(Re-Configuration with Constant Threhsold

$$\prod_{\text{reConfigure}}^{\text{const-th}} : \text{Re-configure}(\text{TAHE}, \text{PVSS}, 1^\kappa, t, n))$$

Reconfiguration Protocol with Constant Threshold is parameterized with a TAHE encryption scheme, a public verifiable secret sharing scheme PVSS, a computational security parameter κ , and a threshold t . The protocol interacts with a transferring quorum $B = \{B_i\}_{i \in [n]}$ and a receiving quorum $B' = \{B'_{i'}\}_{i' \in [n']}$. The transferring quorum holds a t -out-of- n secret sharing $[sk]_i$ on the secret key of the TAHE sk . Both quorums hold a public key to the encryption scheme pk and the verification keys of the transferring quorum $\{vk_i\}_{i \in [n]}$. The protocol outputs a fresh t -out-of- n secret shares $[sk]_{i'}$ on the secret key sk held by B' and public verification keys $\{vk'_{i'}\}_{i' \in [n']}$ corresponding to the shares. It is assumed that when the protocol begins the parties already executed the PVSS.Setup and PVSS.Gen successfully, and as such holds appropriate public keys for the new parties $\{pk_{i'}\}_{i' \in [n']}$.

1. $B_i \in B$ does as follows:

Round 1:

(a) Samples $r_i \leftarrow \mathcal{K}_\kappa$.

(b) $(\{ct_{Share,old}^{i,j}\}_{j \in [n]}, \{C_{i,j}\}_{j \in [n]}, \pi_{Share,old}^i) \leftarrow \text{PVSS.Dist}(\text{pp}, \{pk_j\}_{j \in [n]}, r_i)$.

- (c) $\left(\{ct_{Share,new}^{i,j'}\}_{j' \in [n]}, \{C_{i,j'}\}_{j' \in [n]}, \pi_{Share,new}^i \right) \leftarrow \text{PVSS.Dist}\left(\text{pp}, \{\text{pk}_{j'}\}_{j' \in [n]}; r_i\right).$
- (d) Send $\left(\text{broadcast}, \text{sid}, i, \left(\left\{ \begin{array}{l} ct_{Share,old}^{i,j} \\ ct_{Share,new}^{i,j'} \end{array} \right\}_{j \in [n]}, \left\{ C_{i,j} \right\}_{j \in [n]}, \pi_{Share,old}^i, \pi_{Share,new}^i \right) \right)$ to $\mathcal{F}_{broadcast}.$

Round 2:

- (a) Upon receiving $\left(\text{broadcast}, \text{sid}, j, \left(\left\{ \begin{array}{l} ct_{Share,old}^{j,j^*} \\ ct_{Share,new}^{j,j'} \end{array} \right\}_{j^* \in [n]}, \left\{ C_{j,j^*} \right\}_{j^* \in [n]}, \pi_{Share,old}^j, \pi_{Share,new}^j \right) \right)$ from $\mathcal{F}_{broadcast}.$
- (b) Verify the proofs and check the parameters of $\pi_{Share,old}^j$ and $\pi_{Share,new}^j$ contain the same public commitment to the point at zero. If any of the checks fail consider j malicious, else send (validate, sid, pid) to $\mathcal{F}_{ACS}^{([n] \atop t+1)B}.$

Round 3:

- (a) Receive (authorized-subset, sid, S_B). For every $j \in S_B$ set $r_{j,i} = \text{PVSS.DecShare}(\text{pp}, \text{pk}_i, \text{sk}_i, ct_{Share,old}^{j,i}).$
- (b) Set $[sk]_i = [sk]_i + \sum_{j \in S_B} r_{j,i}.$
- (c) $\left(\{ct_{Share}^{i,j'}\}_{j' \in [n]}, \{C'_{i,j'}\}_{j' \in [n]}, \pi_{Share}^i \right) \leftarrow \text{PVSS.Dist}\left(\text{pp}, \{\text{pk}_{j'}\}_{j' \in [n]}; [sk]_i\right).$
- (d) Send $\left(\text{broadcast}, \text{sid}, i, \left(\{ct_{Share}^{i,j'}\}_{j' \in [n]}, \{C'_{i,j'}\}_{j' \in [n]}, \pi_{Share}^i \right) \right)$ to $\mathcal{F}_{broadcast}.$

Round 4:

- (a) Receive $\left(\text{broadcast}, \text{sid}, j, \left(\{ct_{Share}^{j,j'}\}_{j' \in [n]}, \{C'_{j,j'}\}_{j' \in [n]}, \pi_{Share}^j \right) \right).$
- (b) Verify the proofs π_{Share}^j and check that the commitment in the public parameters of π_{Share}^j is equal to $\text{vk}_j + \sum_{j^* \in S_B} C'_{j^*,j}.$ If any of the checks fail consider j malicious, else send (validate, sid, i) to $\mathcal{F}_{ACS}^{([n] \atop t+1)B}.$

Broadcast Round:

- (a) Receive (authorized-subset, sid, $\bar{S}_B$) from $\mathcal{F}_{ACS}.$ $\models$

$$(b) \text{ Send } \left(\text{global-broadcast}, \text{sid}, i, \left(\left\{ \left\{ ct_{Share}^{j,j'}, C'_{j,j'} \right\}_{j' \in [n']} \right\}_{j \in \bar{S}_B}, \left\{ \left\{ ct_{Share,new}^{j^*,j'}, C'_{j^*,j'} \right\}_{j' \in [n]} \right\}_{j \in S_B} \right) \right) \text{ to } \mathcal{F}_{\text{global-broadcast}}.$$

2. $B_i \in B$ does as follows:

$$(a) \text{ Receive } \left(\text{global-broadcast}, \text{sid}, i, \left(\left\{ \left\{ ct_{Share}^{j,j'}, C'_{j,j'} \right\}_{j' \in [n']} \right\}_{j \in \bar{S}_B}, \left\{ \left\{ ct_{Share,new}^{j^*,j'}, C'_{j^*,j'} \right\}_{j' \in [n]} \right\}_{j \in S_B} \right) \right)$$

(b) For every $j^* \in S_B$, set $r'_{j^*,i'} = \text{PVSS.DecShare}(\text{pp}, \text{pk}_{i'}, \text{sk}_{i'}, ct_{Share,new}^{j^*,i'})$.

(c) For every $j \in \bar{S}_B$, set $\bar{r}_{j,i'} = \text{PVSS.DecShare}(\text{pp}, \text{pk}_{i'}, \text{sk}_{i'}, ct_{Share}^{j,i'})$.

(d) Set $[sk]_{i'}' = \frac{1}{\Delta^3} \sum_{j^* \in S_B} \sum_{j \in \bar{S}_B} \Delta \lambda_{i',j^*}^{S_B} \Delta \lambda_{0,j}^{S_B} \bar{r}_{j,j^*} - \sum_{j^* \in S_B} r'_{j^*,i'}$.

(e) Set $vk'_{j'} = \sum_{j^* \in S_B} \sum_{j \in \bar{S}_B} \Delta \lambda_{j',j^*}^{S_B} \Delta \lambda_{0,j}^{S_B} C'_{j,j^*} - \sum_{j^* \in S_B} C_{j^*,j'}$.

(f) Record $([sk]_{i'}', \{vk_{j'}\}_{j' \in [n]})$.

The DKG protocol may apply for both an Elliptic Curve Digital Signature Algorithm (ECDSA) and a Schnorr algorithm. Reference is made to Figure 7, which is a schematic illustration of an ECDSA/Schnorr distributed key generation (DKG) process, operative in accordance with an embodiment of the present disclosure. A DKG operation is implemented by a network DKG module 140 of one or more parties 110 of decentralized network 105, and by a client DKG module 240 of one or more clients 210. Client 210 receives as input a party identifier (pid), a policy ($policy$), and a private key share (x_A), and outputs a public key (X). Parties 110 and client 210 may access information from a shared database 190, which may include: an encryption of network share under the TAHE (ct_{key}), a public key share (X_A), a public key (X), a party identifier (pid), a policy ($policy$), and a certificate. The certificate ensures the client that a subset of validators with enough voting power (at least $t+1$ VP) agree on this information (i.e., that the output was generated by a valid group of validators).

In one example, the ECDSA/Schnorr DKG operation is performed on a synchronous communication channel.

Definition: “DKG Protocol on Synchronous Communication Channel (DKG Protocol Sync)”

DKG Protocol Sync: ($\text{Key-Generation } \Pi_{\text{keygen}}: \text{KeyGen}(\mathbb{G}, G, q)$)

DKG Protocol - Sync is parameterized with the group description $(\mathbb{G}, G, q)$. The protocol interacts with $n + 1$ parties $A, B_1, \dots, B_n$, where all parties hold pk as input. The parties agree on a fresh session identifier sid and do as follows:

1. A 's first message:
 - (a) A samples a random $x_A \leftarrow \mathbb{Z}_q$ and computes $X_A = x_A \cdot G$.
 - (b) A sends $(com-prove, pid_A, X_A; x_A)$ to $\mathcal{F}_{com-zk}^{L_{DL}}$.
2. B_i 's message:
 - (a) Each B_i receives $(receipt, pid_A)$ from $\mathcal{F}_{com-zk}^{L_{DL}}$.
 - (b) Each B_i samples a random $x_i \leftarrow \mathbb{Z}_q$ and computes $X_i = x_i \cdot G$.
 - (c) Only on first execution: Each B_i sends $(keygen, sid)$ to $\mathcal{F}_{TAHE}$ and waits for the functionality's response. If the functionality responds with $(pubkey, sid, \perp, U' \cap U)$ then output $U' \cap U$ and abort; otherwise, if the functionality responds with $(pubkey, sid, pk)$ then continue.
 - (d) Each B_i computes $ct_i = AHE.Enc(pk, x_i; \rho_i)$ for a randomly chosen ρ_i .
 - (e) Each B_i sends $(prove, sid, pid_i, X_i, ct_i; x_i; \rho_i)$ to $\mathcal{F}_{agg-zk}^{L_{EncDL}[pk, (\mathbb{G}, G, q)]}$ and $(prove, sid, X_i; x_i)$ to $\mathcal{F}_{agg}^{L_{DL}[(\mathbb{G}, G, q)]}$.
3. B_i 's verification:
 - (a) Each B_i receives $(proof, sid, X_B, ct_{key})$ from $\mathcal{F}_{agg-zk}^{L_{EncDL}}$, if not, it receives and outputs the set of corrupted parties and aborts.
4. A 's second message:
 - (a) A receives $(broadcast, sid, X_B)$ from $\mathcal{F}_{agg-zk}^{L_{DL}}$.
 - (b) A sends $(decom-proof, pid_A)$ to $\mathcal{F}_{com-zk}^{L_{DL}}$.
5. B_i 's verification:
 - (a) Each B_i receives $(decom-proof, pid_A, X_A)$ from $\mathcal{F}_{com-zk}^{L_{DL}}$, if not, it aborts.
6. Output:
 - A outputs $X = X_A + X_B$ and records $(keygen, X_B, X; x_A)$
 - Each B_i outputs $X = X_A + X_B$ and records $(keygen, X_A, X, ct_{key}, pk)$.

In a further example, the ECDSA/Schnorr DKG operation is performed on an asynchronous communication channel.

Definition: “DKG Protocol on Asynchronous Communication Channel (DKG Protocol Async)”

DKG Protocol Async: $(Key-Generation \Pi_{keygen}: KeyGen(\mathbb{G}, G, q, \Gamma_B, sid, pk))$

DKG Protocol Async is parameterized with the group description $(\mathbb{G}, G, q)$ along with an access structure Γ_B , a unique session identifier sid , and an encryption key to a TAHE scheme pk . The protocol interacts with a centralized party A_{pidA} and a set of parties $B = \{B_i\}_{i \in [n]}$. The set of

parties B can only send messages to parties in A through the functionality $\mathcal{F}_{global-broadcast}$. The parties output the public verification key X and record the following:

- X_A : the centralized party's share of a public verification key for an ECDSA/Schnorr signature (implicitly gives the distributed party share);
- the distributed party B also records ct_{key} an encryption of the secret share held by the distributed party B ;
- the centralized party A_{pidA} also records x_A , its secret share of an ECDSA/Schnorr signature key.

The parties do as follows:

1. Each B_i does as follows:

Round 1

- (a) Sample $x_i \leftarrow \mathbb{Z}_q$ and $\eta_i \leftarrow \mathcal{R}_{pk}$.
- (b) Compute $X_i = x_i \cdot G$, and $ct_i = \text{AHE.Enc}(pk, x_i; \eta_i)$.
- (c) Run $\Pi_{agg}^{\Gamma_B, L_{EncDL}[pk, (\mathbb{G}, G, q)]}$ on inputs (X_i, x_i, η_i) receiving (X_B, ct_{key}) .

Broadcast Round:

- (a) Send $(\text{global-broadcast}, \text{sid}, i, (pid_A, X_B))$ to $\mathcal{F}_{global-broadcast}^{\Gamma_B}$.
2. A_{pidA} does as follows:
 - (a) Receive $(\text{global-broadcast}, \text{sid}, (pid_A, X_B))$ from $\mathcal{F}_{global-broadcast}^{\Gamma_B}$.
 - (b) Verify that $X_B \in \mathbb{G}$, else consider B malicious and abort.
 - (c) Sample $x_A \leftarrow \mathbb{Z}_q$ and set $X_A = x_A \cdot G$.
 - (d) Run $\Pi_{zk-uc}^{L_{DL}[(\mathbb{G}, G, q)]}(X_A; x_A)$ generating a proof π_{DL} .
 - (e) Send $(\text{broadcast}, \text{sid}, pid_A, X_A, \pi_{DL})$ to $\mathcal{F}_{broadcast}$.

3. Output:

Each B_i does as follows:

- (a) Receive $(\text{sid}, pid_A, X_A, \pi_{DL})$.
- (b) Verify π_{DL} , if fails consider A_{pidA} malicious and abort, otherwise continue.
- (c) Set $X = X_A + X_B$.
- (d) Output X and record $(\text{keygen}, pid_A, X, X_A, ct_{key})$.

Party A_{pidA} does as follows

- (a) Set $X = X_A + X_B$.
- (b) Output X and record $(\text{keygen}, pid_A, X, X_A; x_A)$.

The signing process can be broken into a presign phase that can be executed before the message to be signed is decided, and an online-signing phase. Presigns can be used to reduce the latency at the online-signing phase. Specifically, presign may involve performing a preliminary computation before deciding on a message, that produces latency in the online signing phase where the message to be signed is being decided.

The presign protocol may apply for both a ECDSA or a Schnorr algorithm over both a synchronous or an asynchronous network. A presign protocol using ECDSA may be 2 rounds, and a pressing protocol using Schnorr may be one round. Reference is made to Figures 8A and 8B. Figure 8A is a schematic illustration of an ECDSA/Schnorr asynchronous presign process, operative in accordance with an embodiment of the present disclosure. Figure 8B is a schematic illustration of a Schnorr presign process, operative in accordance with an embodiment of the present disclosure.

Referring to Fig.8A, a first presign operation is implemented by an asynchronous network ECDA/Schnorr presign module 150 of one or more parties 110 of decentralized network 105, and by a client ECDA/Schnorr presign module 250 of one or more clients 210. Client 210 receives as input a party identifier (pid). Parties 110 and client 210 may access information from a shared database 190, which may include: two network public nonce shares ($R_{B,0}$, $R_{B,1}$), and a group of ciphertexts (ct_γ), ($ct_{\gamma \cdot key}$), ($ct_{\gamma \cdot k_0}$), ($ct_{\gamma \cdot k_1}$) to be used by the client during the online-signing phase for homomorphically evaluating the signature, a public key (X), and a certificate.

In one example, the presign routine is performed using an ECDSA signature scheme on a synchronous communication channel.

Definition: “Presign Protocol using ECDSA on Synchronous Communication Channel (Presign Protocol ECDSA/Sync)”

Presign Protocol ECDSA/Sync: ($Presigning \Pi_{pres}; Presign(\mathbb{G}, G, q), sid$)

Presign Protocol ECDSA/Sync interacts with $n + 1$ parties $A, B_1, \dots, B_n$, where all have pk and ct_{key} as input. Before proceeding, all parties verify that session identifier sid has never been used before, then do as follows:

1. A 's message:

(a) A samples a random $k_A, \rho_1 \leftarrow \mathbb{Z}_q$ and computes $K_A = Com(k_A; \rho_1)$.

(b) A sends ($prove, sid, pid_A, K_A; k_A, \rho_1$) to $\mathcal{F}_{zk}^{LDCOM}$.

2. B_i 's message:

Round 1

(a) B_i receives ($proof, sid, pid_A, K_A$) from $\mathcal{F}_{zk}^{LDCOM}$, if not, it aborts.

(b) B_i samples $k_i \leftarrow \mathbb{Z}_q^*$ and computes $R_i = k_i \cdot G$. Denote $k_B = \sum_i k_i$.

(c) B_i samples $\gamma_i \in [0, q) \eta_{mask_1}^i, \eta_{mask_2}^i, \eta_{mask_3}^i, \eta_{mask_4}^i \in Rpk$, and computes:

i. $ct_1^i = AHE.Enc(pk, \gamma_i; \eta_{mask_1}^i)$;

- ii. $ct_2^i = AHE.Eval(pk, f_i, ct_{key}; \eta_{mask_2}^i)$, where: $f_i(x) = \gamma_i \cdot x$;
- iii. $ct_3^i = AHE.Enc(pk, k_i; \eta_{mask_3}^i)$.
- (d) B_i sends $(prove, sid \parallel \gamma, pid_i, ct_1^i, ct_2^i; \gamma_i, \eta_{mask_1}^i, \eta_{mask_2}^i)$ to $\mathcal{F}_{agg-zk}^{L_{EncDH}[pk, ct_{key}]}$.
- (e) B_i sends $(prove, sid, pid_i, ct_3^i, k_i, \eta_{mask_3}^i)$ to $\mathcal{F}_{agg-zk}^{L_{EncDL}}$.

Round 2

- (a) B_i receives $(prove, sid \parallel \gamma, pid_i, ct_1, ct_2;)$ from $\mathcal{F}_{agg-zk}^{L_{EncDH}[pk, ct_{key}]}$, and $(prove, sid, R_B, ct_3)$ from $\mathcal{F}_{agg-zk}^{L_{EncDL}}$.
- (b) Otherwise, if B_i receives $(malicious, sid, U')$ from $\mathcal{F}_{agg-zk}^{L_{EncDH}[pk, ct_{key}]}$ and/or $(malicious, sid, U'')$ from $\mathcal{F}_{agg-zk}^{L_{EncDL}}$, it records the malicious parties (M and/or M').
- (c) B_i computes $ct_4^i = AHE.Eval(pk, f_i', ct_1; \eta_{mask_4}^i)$ where: $f_i'(x) = k_i \cdot x$.
- (d) B_i sends $(prove, sid \parallel k, pid_i, ct_3^i, ct_4^i, k_i, \eta_{mask_3}^i, \eta_{mask_4}^i)$ to $\mathcal{F}_{agg-zk}^{L_{EncDH}[pk, ct_1]}$.

Proof verification:

- (a) B_i receives $(proof, sid \parallel k, pid_i, ct_3, ct_4)$ from $\mathcal{F}_{agg-zk}^{L_{EncDH}[pk, ct_1]}$. Otherwise, if B_i receives $(malicious, sid, U'')$ from $\mathcal{F}_{agg-zk}^{L_{EncDH}[pk, ct_1]}$, it records the malicious parties and aborts.

3. A 's verification:

- (a) A receives $(proof, sid \parallel R_B, ct_3)$ from $\mathcal{F}_{agg-zk}^{L_{EncDL}}$, if not, it aborts.
- (b) A receives $(proof, sid \parallel \gamma, ct_1, ct_2)$ from $\mathcal{F}_{agg-zk}^{L_{EncDH}[pk, ct_{key}]}$, if not, it aborts.

4. Output:

- A records $(presign, sid, R_B, ct_1, ct_2; k_A, \rho_1)$, where ct_1 and ct_2 are encryptions of γ and $\gamma \cdot x_B$.
- B_i records $(presign, sid, R_B, K_A, ct_1, ct_2, ct_4)$, where ct_4 encrypts $\gamma \cdot k_B \bmod q$.

In another example, the presign routine is performed using a Schnorr signature scheme on a synchronous communication channel.

Definition: “Presign Protocol using Schnorr on Synchronous Communication Channel (Presign Protocol Schnorr/Sync)”

Presign Protocol Schnorr/Sync: $(Presign \Pi_{pres}: Presign(\mathbb{G}, G, q), sid)$

Presign Protocol Schnorr/Sync interacts with $n + 1$ parties $A, B_1, \dots, B_n$, where all parties hold pk and $B_1, \dots, B_n$ have ct_{key} as input. Before proceeding, all parties verify that session identifier sid has never been used before, then do as follows:

1. A 's message:
 - (a) A samples a random $k_A, \rho \leftarrow \mathbb{Z}_q$ and computes $K_A = Com(k_A; \rho)$.
 - (b) A sends $(prove, sid, pid_A, K_A; k_A, \rho)$ to $\mathcal{F}_{zk}^{LDCOM}$.
2. B_i 's message:
 - (a) B_i receives $(proof, sid, pid_A, K_A)$ from $\mathcal{F}_{zk}^{LDCOM}$, if not, it aborts.
 - (b) B_i samples $k_i^0, k_i^1 \leftarrow \mathbb{Z}_q^*$ and computes $R_i^0 = k_i^0 \cdot G$ and $R_i^1 = k_i^1 \cdot G$. Denote $k_B^0 = \sum_i k_i^0$ and $k_B^1 = \sum_i k_i^1$.
 - (c) B_i samples $\eta_i^0 \in R_{pk}$, and computes $ct_{k_0}^i = AHE.Enc(pk, k_i^0; \eta_i^0)$ and $ct_{k_1}^i = AHE.Enc(pk, k_i^1; \eta_i^1)$.
 - (d) B_i sends $(prove, sid, pid_i, R_i^0, ct_{k_0}^i; k_i^0, \eta_i^0)$ and $(prove, sid, pid_i, R_i^1, ct_{k_1}^i; k_i^1, \eta_i^1)$ to $\mathcal{F}_{agg-zk}^{LEncDL[(\mathbb{G}, G, q)]}$.
 - (e) B_i sends $(prove, sid, pid_i, R_i^0; k_i^0)$ and $(prove, sid, pid_i, R_i^1; k_i^1)$ to $\mathcal{F}_{agg-zk}^{LDL[(\mathbb{G}, G, q)]}$.
3. B_i 's verification:
 - (a) B_i receives $(proof, sid, R_B^0, ct_{k_0})$ and $(proof, sid, pid, R_B^1, ct_{k_1})$ from $\mathcal{F}_{agg}^{LEncDL[(\mathbb{G}, G, q)]}$.
 - (b) Otherwise, if B_i receives $(malicious, sid, U'')$ from $\mathcal{F}_{agg-zk}^{LEncDL[(\mathbb{G}, G, q)]}$, it records malicious parties U and aborts.
4. A 's verification:
 - (c) A receives $(proof, sid, R_B^0)$ and $(proof, sid, R_B^1)$ from $\mathcal{F}_{agg-zk}^{LEncDL[(\mathbb{G}, G, q)]}$, if not, it aborts.
5. Output:
 - A records $(presign, sid, R_B^0, R_B^1, K_A; k_A, \rho)$.
 - B_i records $(presign, sid, R_B^0, R_B^1, K_A, ct_{k_0}, ct_{k_1})$.

In a further example, the presign routine is performed using an ECDSA signature scheme on an asynchronous communication channel.

Definition: "Presign Protocol using ECDSA on Asynchronous Communication Channel (Presign Protocol ECDSA/Async)"

Presign Protocol ECDSA/Async:

$(Presign \Pi_{pres}: Presign(\mathbb{G}, G, q, sid, pk, X, X_A, ct_{key}))$

Presign Protocol ECDSA/Async is parameterized with the group description $(\mathbb{G}, G, q)$ along with an access structure Γ_B , a unique session identifier sid , an encryption key to a TAHE scheme pk , an ECDSA verification key X , the centralized party share of the verification key X_A , and an encryption of the share of the distributed party of the signing key ct_{key} . The protocol interacts with a centralized party A_{pidA} and a set of parties $B = \{B_i\}_{i \in [n]}$. The set of parties B can only send messages to parties in A through the functionality $\mathcal{F}_{global-broadcast}$. Each party B_i outputs:

- ct_γ : an encryption of a randomizer γ ;
- $ct_{\gamma \cdot key}$: an encryption of the randomizer γ multiplied by the signing key share of the blockchain x_B ;
- $ct_{\gamma \cdot k_0}, ct_{\gamma \cdot k_1}$: an encryption of a randomized nonces $k_0 \cdot \gamma$, and $k_1 \cdot \gamma$, respectively;
- $R_{B,0}, R_{B,1}$: points on the curve equal to $k_0 \cdot G$, and $k_1 \cdot G$, respectively.

1. Each B_i does as follows:

Round 1

(a) Sample a random $\gamma_i \leftarrow \mathbb{Z}_q$ and $\eta_1^i, \eta_2^i \leftarrow \mathcal{R}_{pk}$, and compute:

- i. $ct_\gamma^i = \text{AHE.Enc}(pk, \gamma_i, \eta_1^i)$.
- ii. $ct_{\gamma \cdot key}^i = \text{AHE.Scale}(pk, ct_{key}, \gamma_i; \eta_2^i)$.

(b) Run $\Pi_{agg}^{\Gamma_B, L_{EncDH}[pk, ct_{key}]}$ on inputs $(ct_\gamma^i, ct_{\gamma \cdot key}^i; \gamma_i; \eta_1^i, \eta_2^i)$ receiving $(ct_\gamma, ct_{\gamma \cdot key})$.

Round 2

(a) Sample a random $k_0^i, k_1^i \leftarrow \mathbb{Z}_q$ and $\eta_{3,0}^i, \eta_{3,1}^i \leftarrow \mathcal{R}_{pk}$, and compute:

- i. $ct_{\gamma \cdot k_0}^i = \text{AHE.Scale}(pk, ct_\gamma, k_0^i, \eta_{3,0}^i)$.
- ii. $ct_{\gamma \cdot k_1}^i = \text{AHE.Scale}(pk, ct_\gamma, k_1^i, \eta_{3,1}^i)$.
- iii. $R_{B,0}^i = k_0^i \cdot G$.
- iv. $R_{B,1}^i = k_1^i \cdot G$.

(b) Run $\Pi_{agg}^{\Gamma_B, L_{ScaleDL}[pk, (\mathbb{G}, G, q), ct_\gamma]}$ on inputs $(ct_{\gamma \cdot k_0}^i, R_{B,0}^i; k_0^i, \eta_{3,0}^i)$ and $(ct_{\gamma \cdot k_1}^i, R_{B,1}^i; k_1^i, \eta_{3,1}^i)$ receiving $(ct_{\gamma \cdot k_0}, R_{B,0}), (ct_{\gamma \cdot k_1}, R_{B,1})$.

Broadcast Round:

(a) Send $(\text{global-broadcast}, sid, i, (pid_A, ct_\gamma, ct_{\gamma \cdot key}, ct_{\gamma \cdot k_0}, ct_{\gamma \cdot k_1}, R_{B,0}, R_{B,1}))$ to $\mathcal{F}_{global-broadcast}^{\Gamma_B}$.

2. Output:

- A_{pidA} receives $(\text{global-broadcast}, \text{sid}, (\text{pid}_A, \text{ct}_\gamma, \text{ct}_{\gamma \cdot \text{key}}, \text{ct}_{\gamma \cdot k_0}, \text{ct}_{\gamma \cdot k_1}, R_{B,0}, R_{B,1}))$ from $\mathcal{F}_{\text{global-broadcast}}^{\Gamma_B}$.
- A_{pidA} verifies that $R_{B,0}, R_{B,1} \in \mathbb{G} \setminus \{0\}$ and $\text{ct}_\gamma, \text{ct}_{\gamma \cdot \text{key}}, \text{ct}_{\gamma \cdot k_0}, \text{ct}_{\gamma \cdot k_1} \in \mathcal{C}_{pk}$, else it aborts and considers B malicious.
- Each party B_i records $(\text{sid}, X, \text{ct}_\gamma, \text{ct}_{\gamma \cdot \text{key}}, \text{ct}_{\gamma \cdot k_0}, \text{ct}_{\gamma \cdot k_1}, R_{B,0}, R_{B,1})$.

In yet another example, the presign routine is performed using a Schnorr signature scheme on an asynchronous communication channel.

Definition: “Presign Protocol using Schnorr on Asynchronous Communication Channel (Presign Protocol Schnorr/Async)”

Presign Protocol Schnorr/Async: $(\text{Presign } \Pi_{\text{pres}}: \text{SchPresign}(\mathbb{G}, G, q, \Gamma_B, \text{sid}, pk))$

Presign Protocol Schnorr/Async is parameterized with a group description $(\mathbb{G}, G, q)$ along with an access structure Γ_B , a unique session identifier sid , and an encryption key to a TAHE scheme pk . The protocol interacts with a centralized party A_{pidA} and a set of parties $B = \{B_i\}_{i \in [n]}$. The set of parties B can only send messages to parties in A through the functionality $\mathcal{F}_{\text{global-broadcast}}$. Each party B_i outputs:

- $K_{B,0}, K_{B,1}$: points on the curve equal to $k_0 \cdot G$, and $k_1 \cdot G$, respectively.
 - The distributed parties also output $\text{ct}_{k_0}, \text{ct}_{k_1}$: encryptions of k_0, k_1 , respectively
1. Each B_i does as follows:

Round 1

- (a) B_i samples a random $k_0^i, k_1^i \leftarrow \mathbb{Z}_q$ and $\eta_0^i, \eta_1^i \leftarrow \mathcal{R}_{pk}$, and computes:

- i. $K_{B,0}^i = k_0^i \cdot G$
- ii. $K_{B,1}^i = k_1^i \cdot G$
- iii. $\text{ct}_{k_0}^i = \text{AHE.Enc}(pk, k_0^i, \eta_0^i)$
- iv. $\text{ct}_{k_1}^i = \text{AHE.Enc}(pk, k_1^i, \eta_1^i)$

- (b) Run $\Pi_{agg}^{L_{EncDL}[pk, (\mathbb{G}, G, q)]}$ on inputs $(\text{ct}_{k_1}^i, K_{B,0}^i; k_0^i, \eta_0^i)$ and $(\text{ct}_{k_1}^i, K_{B,1}^i; k_1^i, \eta_1^i)$ receiving $(\text{ct}_{k_0}, K_{B,0}), (\text{ct}_{k_1}, K_{B,1})$.

Broadcast Round:

- (a) Send $(\text{global-broadcast}, \text{sid}, i, (\text{pid}_A, K_{B,0}, K_{B,1}))$ to $\mathcal{F}_{\text{global-broadcast}}$.

2. Output:

- A_{pidA} receives $(\text{global-broadcast}, \text{sid}, (pid_A, K_{B,0}, K_{B,1}))$ from $\mathcal{F}_{\text{global-broadcast}}^{\Gamma_B}$.
- A_{pidA} verifies that $K_{B,0}, K_{B,1} \in \mathbb{G} \setminus \{0\}$, else it aborts and considers B malicious.
- Both parties output $K_{B,0}, K_{B,1}$.
- Parties in B also record ct_{k_0}, ct_{k_1} .

An online-signing operation produces a signature given a message, a presign (if needed), and private key shares. The signing protocol may apply for both a ECDSA or a Schnorr algorithm over both a synchronous or an asynchronous network. Reference is made to Figure 9, which is a schematic illustration of an ECDSA/Schnorr signing process, operative in accordance with an embodiment of the present disclosure. A first signing routine is implemented by an asynchronous network ECDA sign module 160 of one or more parties 110 of decentralized network 105, and by a client ECDA sign module 260 of one or more clients 210. A second signing routine is implemented by an asynchronous network Schnorr sign module 165 of one or more parties 110 of decentralized network 105, and by a client Schnorr sign module 265 of one or more clients 210. Client 210 receives as input a party identifier (pid), an optional presign ($presign$), a private key share (x_A), and a message (msg) to be signed. Parties 110 and client 210 may access information from a shared database 190, which may include: a signature and a certificate.

In one example, the signing routine is performed using an ECDSA signature scheme on a synchronous communication channel.

Definition: “Sign Protocol using ECDSA on Synchronous Communication Channel (Sign Protocol ECDSA/Sync)”

Sign Protocol ECDSA/Sync: ($Signing \Pi_{\text{sign}}: \text{Sign}((\mathbb{G}, G, q), \text{sid}, \text{msg})$)

Sign Protocol ECDSA/Sync interacts with $n + 1$ parties $A, B_1, \dots, B_n$, where all have pk, K_A, R_B^0 and R_B^1 as input. $B_1, \dots, B_n$ has in addition ct_{key}, ct_{k_0} , and ct_{k_1} as input. A has in addition k_A, ρ , and msg as input. Before proceeding, all parties verify that session identifier sid has never been used before, retrieve a corresponding presign output if needed, then do as follows:

1. A 's message:

- A computes $R = (k_A)^{-1} \cdot R_B = (k_A^{-1} k_B \cdot G)$ and $r = R_{x\text{-axis}} \bmod q$. Denote $k = k_A^{-1} k_B$.
- A samples $\rho_2 \in \mathcal{R}_{pp}$ and computes $U_A = \text{Com}(k_A \cdot x_A; \rho_2)$.
- A sets $a_1 = r \cdot k_A \cdot x_A + m \cdot k_A$ and $a_2 = r \cdot k_A$, where $m := \mathcal{H}(msg)$.
- A homomorphically evaluates ct_1, ct_2 on its private function $f_A(x_A, x_A) := a_1 x_1 + a_2 x_2$:
 - $ct_A \leftarrow AHE.Eval(pk, f_A, ct_1, ct_2; \eta_{eval})$
- A sends message and the following proofs:

- A sends $(\text{prove}, \text{sid}, \text{pid}_A, K_A, R_B; k_A, \rho_1)$ to $\mathcal{F}_{zk}^{LDComDL[pp, (\mathbb{G}, G, q)]}$.
- A sends $(\text{prove}, \text{sid}, \text{pid}_A, K_A, U_A; k_A, x_A, \rho_1, \rho_2)$ to $\mathcal{F}_{zk}^{LDComRatio[pp, (\mathbb{G}, G, q), X_A]}$.
- A computes $C_1 = (r \odot U_A) \oplus (m \odot K_A)$ and $C_2 = (r \odot K_A)$ and sends $(\text{prove}, \text{sid}, \text{pid}_A, \text{ct}_A, C_1, C_2; a_1, r \cdot \rho_2 + m, r \cdot \rho_1 + m, \eta)$ to $\mathcal{F}_{zk}^{LDComEval[pp, pk, \text{ct}_1, \text{ct}_2]}$.

2. B_i 's verification and output:

(a) B_i receives msg and the following proofs, otherwise it aborts:

- $(\text{proof}, \text{sid} \parallel \text{pid}_A, K_A, R_B)$ from $\mathcal{F}_{zk}^{LDComDL[pp, (\mathbb{G}, G, q)]}$.
- $(\text{proof}, \text{sid} \parallel \text{pid}_A, K_A, U_A)$ from $\mathcal{F}_{zk}^{LDComRatio[pp, (\mathbb{G}, G, q), X_A]}$.
- $(\text{proof}, \text{sid} \parallel \text{pid}_A, \text{ct}_A, C_1, C_2)$ from $\mathcal{F}_{zk}^{LDComEval[pp, pk, \text{ct}_1, \text{ct}_2]}$.

(b) B_i verifies that the values used in the proofs are consistent with values obtained previously by B_i , specifically:

- There are records $(\text{keygen}, X_A, X, \text{ct}_{key}, pk)$ and $(\text{presign}, \text{sid}, R_B, K_A, \text{pt}')$; and
- $C_1 = (r \odot U_A) \oplus (m \odot K_A)$ and $C_2 = (r \odot K_A)$, where $r = R_{x\text{-axis}} \bmod q$ and $m = \mathcal{H}(\text{msg})$.

(c) B_i sends $(\text{decrypt}, pk, \text{ct}_A)$ and $(\text{decrypt}, pk, \text{ct}_4)$ to $\mathcal{F}_{TAHE}$ and waits for its response. Let the responses be $(\text{decrypted}, pk, \text{ct}_A, \text{pt}_A, U_A)$ and $(\text{decrypted}, pk, \text{ct}_4, \text{pt}_4, U_4)$ respectively; if $\text{pt}_A = \perp$ or $\text{pt}_4 = \perp$ then output $U_A \cup U_4$ and abort; otherwise, compute $s' = \text{pt}_4^{-1} \cdot \text{pt}_A \bmod q$ (which is equal to $(\gamma k_B)^{-1} \cdot ((rk_A x_A + mk_A)\gamma + rk_A \gamma x_B) = k^{-1}(rx + m) \bmod q$ as required) and output $s = \min\{s', q - s'\}$ (to ensure uniqueness of the signature).

3. Output:

- B_i outputs $\sigma = (r, s)$.

In another example, the sign routine is performed using a Schnorr signature scheme on a synchronous communication channel.

Definition: “Sign Protocol using Schnorr on Synchronous Communication Channel (Sign Protocol Schnorr/Sync)”

Sign Protocol Schnorr/Sync: $(\text{Signing } \Pi_{\text{sign}}: \text{Sign}((\mathbb{G}, G, q), \text{sid}, \text{msg}))$

Sign Protocol Schnorr/Sync interacts with $n + 1$ parties $A, B_1, \dots, B_n$, where all have pk, K_A, R_B^0 and R_B^1 as input. $B_1, \dots, B_n$ has in addition $\text{ct}_{key}, \text{ct}_{k_0}$, and ct_{k_1} as input. A has in addition k_A ,

ρ , and msg as input. Before proceeding, all parties verify that session identifier sid has never been used before, retrieve a corresponding presign output if needed, then do as follows:

1. A 's message:
 - (a) A computes $R_A = k_A \cdot G$.
 - (b) A sends $(prove, sid, pid_A, K_A, R_A; k_A, \rho)$ to $\mathcal{F}_{zk}^{LDCoMDL[(\mathbb{G}, G, q)]}$.
 - (c) A calls the random oracle $\mathcal{H}(msg)$ and receives m .
 - (d) A calls the random oracle $\mathcal{H}(msg, R_A, R_B^0, R_B^1)$ and receives μ .
 - (e) A computes $R = R_A + R_B^0 + \mu \cdot R_B^1$.
 - (f) A calls the random oracle $\mathcal{H}(R, X, m)$ and receives e .
 - (g) A computes $z = K_A + e \cdot x_Z$.
 - (h) A sends msg and z to $\{B_i\}_{i \leq \eta}$.
2. B_i 's verification and output:
 - (a) B_i receives msg and z from A , and $(proof, sid, pid_A, K_A, R_A)$ from $\mathcal{F}_{zk}^{LDCoMDL[(\mathbb{G}, G, q)]}$. Otherwise, it aborts.
 - (b) B_i calls the random oracle $\mathcal{H}(msg)$ and receives m .
 - (c) B_i calls the random oracle $\mathcal{H}(m, R_A, R_B^0, R_B^1)$ and receives μ .
 - (d) B_i computes $R = R_A + R_B^0 + \mu \cdot R_B^1$.
 - (e) B_i calls the random oracle $\mathcal{H}(R, X, m)$ and receives e .
 - (f) B_i computes $ct_B = z \oplus ct_k \oplus (e \odot ct_{key})$.
 - (g) B_i sends $(decrypt, pk, ct_B)$ to $\mathcal{F}_{TAHE}$ and receives $(decrypted, pk, ct_B, pt_B, \mathcal{U}A)$. Otherwise, it records the malicious parties and aborts.
 - (h) B_i verifies that $pt_B \cdot G = R + e \cdot X$. If not, it aborts.
3. Output:
 - B_i outputs $\sigma = (R, pt_B)$.

In a further example, the sign routine is performed using an ECDSA signature scheme on an asynchronous communication channel.

Definition: “Sign Protocol using ECDSA on Asynchronous Communication Channel (“Sign Protocol ECDSA/Async)”

Sign Protocol ECDSA/Async:

$$(Sign \Pi_{sign}: Sign(\mathbb{G}, G, H, q, sid, \Gamma_B, pk, X, ct_{key}, presX, msg))$$

Sign Protocol ECDSA/Async is parameterized with a group description $(\mathbb{G}, G, q)$ along with another public generator H used for Pedersen commitments, an access structure Γ_B , a unique session identifier sid , an encryption key to a TAHE scheme pk , a public ECDSA verification key X , the centralized party share of the public verification key X_A , an encryption of the share of the distributed party of the private signing key ct_{key} , the output of a uniquely determined previous presign protocol $presX$, $sid = (ct_\gamma, ct_{\gamma \cdot key}, ct_{\gamma \cdot k_0}, ct_{\gamma \cdot k_1}, R_{B,0}, R_{B,1})$, and the message to be signed msg . The protocol interacts with a centralized party A_{pidA} and a set of parties $B = \{B_i\}_{i \in [n]}$. The set of parties B can only send messages to parties in A through the functionality $\mathcal{F}_{global-broadcast}$. The parties output a valid ECDSA signature $\sigma = (r, s)$ with verification key X .

1. A_{pidA} does as follows:

- (a) Calls the random oracle $\mathcal{H}(msg)$ and receives m .
- (b) Samples $k_A, \alpha, \beta \leftarrow \mathbb{Z}_q$ and $\rho_0, \rho_1, \rho_2, \rho_3 \leftarrow \mathcal{R}_{pk}$ and computes:
 - i. $C_k = \text{Pedersen.Com}_{G,H}(k_A; \rho_0)$
 - ii. $C_\alpha = \text{Pedersen.Com}_{G,H}(\alpha; \rho_1)$
 - iii. $C_\beta = \text{Pedersen.Com}_{G,H}(\beta; \rho_2)$
 - iv. $C_{kx} = \text{Pedersen.Com}_{X_A,H}(k_A; \rho_3)$
- (c) Calls the random oracle $\mathcal{H}(msg)$ and receives m .
- (d) Runs the protocols $\prod_{zk-uc}^{LDCOM[C_k,H]}(G; k_A^{-1}, -k_A^{-1}, \rho_0)$, $\prod_{zk-uc}^{LDCOM[C_\alpha,H]}(G; \alpha^{-1}, -\alpha^{-1}, \rho_1)$ and $\prod_{zk-uc}^{LDCOM[C_\beta,H]}(C_\beta; \beta, \rho_2)$, generating proofs π_k and π_α, π_β .
- (e) Calls the random oracle $\mathcal{H}(sid, \mathbb{G}, G, q, H, X, presX_{sid}, C_k, C_{kx}, X_A, C_\alpha, C_\beta, \pi_k, \pi_\alpha, \pi_\beta)$ and receives μ_k . It then computes:
 - i. $ct_\gamma \cdot k = (ct_\gamma \cdot k_0) \oplus (\mu_k \odot ct_\gamma \cdot k_1)$
 - ii. $R'_B = (R_{B,0}) + (\mu_k \cdot R_{B,1})$
- (f) Samples $\eta_0, \eta_1 \leftarrow \mathcal{R}_{pk}$, and computes:
 - i. $ct_{\alpha,\beta} = \text{AHE.Eval}(pk, (ct_\gamma, ct_\gamma \cdot k), (0, \beta, \alpha); \eta_0)$
 - ii. $R_B = (\alpha \cdot R'_B) + (\beta \cdot G)$
 - iii. $R = k_A^{-1} \cdot R_B$ and $r = R_{x-axis}$
 - iv. $\alpha_1 = r \cdot k_A \cdot X_A + m \cdot k_A$ and $\alpha_2 = r \cdot k_A$
 - v. $ct_A = \text{AHE.Eval}(pk, (ct_\gamma, ct_{\gamma \cdot key}), (0, \alpha_1, \alpha_2); \eta_1)$
 - vi. $C_1 = (r \odot C_{kx}) \oplus (m \odot C_k)$ and $C_2 = r \odot C_k$.
- (g) Generate the following proofs:

- i. $\pi_{kx} \leftarrow \prod_{zk}^{L_{DComEq}[(G,H),(X_A,H)]}(C_k, C_{kx}; k_A, \rho_0, \rho_3).$
- ii. $\pi_R \leftarrow \prod_{zk}^{L_{DComDL}[(G,H),(\mathbb{G},G,q)]}(C_k, R_B; k_A, \rho_0).$
- iii. $\pi_{R_B} \leftarrow \prod_{zk}^{L_{VecDComDL}[(G,H),(\mathbb{G},(R'_B,G),q)]}(C_\alpha, C_\beta, R_B; (\alpha, \beta), \rho_1, \rho_2).$
- iv. $\pi_{ct_{\alpha,\beta}} \leftarrow \prod_{zk}^{L_{DComEval}[G,H,pk(ct_Y,ct_{Y,k})(\mathbb{G},G,q)]}(ct \ \alpha, \beta, (C_\beta, C_\alpha); (\beta, \alpha), \rho_0, \rho_1, \eta_0).$
- v. $\pi_{ct_A} \leftarrow \prod_{zk}^{L_{DComEval}[G,H,pk(ct_Y,ct_{Y-key})(\mathbb{G},G,q)]}(ct_A, (C_1, C_2); (\alpha_1, \alpha_2), \rho_3 \cdot r + \rho_0 \cdot m, r \cdot \rho_0, \eta_1).$

(h) Send $R, R_B, C_k, C_\alpha, C_\beta, C_{kx}, ct_A, ct \ \alpha, \beta, \pi_k, \pi_\alpha, \pi_\beta, \pi_{kx}, \pi_R, \pi_{R_B}, \pi_{ct_{\alpha,\beta}}, \pi_{ct_A}.$

2. Each B_i does as follows:

Round 1

- (a) Call the random oracle $\mathcal{H}(msg)$ and receives m .
- (b) Receive $R, R_B, C_k, C_\alpha, C_\beta, C_{kx}, ct_A, ct \ \alpha, \beta, \pi_k, \pi_\alpha, \pi_\beta, \pi_{kx}, \pi_R, \pi_{R_B}, \pi_{ct_{\alpha,\beta}}, \pi_{ct_A}.$
- (c) Call the random oracle $\mathcal{H}(sid, msg, \mathbb{G}, G, qH, X, presX, sid, C_k, C_{kx}, X_A, C_\alpha, C_\beta, \pi_{kx}, \pi_\alpha, \pi_\beta)$ and compute:
 - i. $ct_{Y,k} = (ct_{Y,k_0}) \oplus (\mu_k \odot ct_{Y,k_1})$
 - ii. $R'_B = (R_{B,0}) + (\mu_k \cdot R_{B,1})$
 - iii. $C_1 = (r \odot C_{kx}) \oplus (m \odot C_k)$
 - iv. $C_2 = r \odot C_k.$
- (c) Verify the proofs. If fail abort and consider A_{pidA} malicious.
- (d) Send $(decrypt, pk, ct_A)$ and $(decrypt, pk, ct \ \alpha, \beta)$ to $\mathcal{F}_{TAHE}.$

Broadcast Round:

- (a) Receive $(decrypt, pk, ct_A, pt_A)$ and $(decrypt, pk, ct \ \alpha, \beta, pt_A)$ from $\mathcal{F}_{TAHE}.$
- (b) Compute $s' = pt_A^{-1} \cdot pt_A \bmod q$ and $s = \min\{s', q - s'\}.$
- (c) Send $(global-broadcast, sid, i, (r, s))$ to $\mathcal{F}_{global-broadcast}^{\Gamma_B}.$

3. Output:

- A_{pidA} receives $(global-broadcast, sid, i, (r, s))$ from $\mathcal{F}_{global-broadcast}^{\Gamma_B}.$
- A_{pidA} verifies that (r, s) is a valid ECDSA signature, if the verification fails it aborts and considers B malicious.

- Both A_{pidA} and B output (r, s) .

In yet another example, the sign routine is performed using a Schnorr signature scheme on an asynchronous communication channel.

Definition: “Sign Protocol using Schnorr on Asynchronous Communication Channel (Sign Protocol Schnorr/Async)”

Sign Protocol Schnorr/Async is parameterized with a group description $(\mathbb{G}, G, q)$ along with an access structure Γ_B , a unique session identifier sid , an encryption key to a TAHE scheme pk , a public verification key X , an encryption of the share of the distributed party of the private signing key ct_{key} , the output of a uniquely presign $pres_X = (K_{B,0}, K_{B,1})$, and the message to be signed msg . The protocol interacts with a centralized party A_{pidA} and a set of parties $B = \{B_i\}_{i \in [n]}$. The set of parties B can only send messages to parties in A through the functionality $\mathcal{F}_{global-broadcast}$. The parties output a valid Schnorr signature (z, e) to a verification key X .

1. A_{pidA} does as follows:

- (a) Samples $k_A \leftarrow \mathbb{Z}_q$ and computes $K_A = k_A \cdot G$.
- (b) Calls the random oracle $\mathcal{H}(sid, X, pres_{X,sid}, K_A, msg)$ and receives μ_k .
- (c) Computes $K = (K_{B,0}) + (\mu_k \cdot K_{B,1}) + K_A$.
- (d) Calls the random oracle $\mathcal{H}(K, X, msg)$ and receives e .
- (e) Computes $z_A = k_A + e \cdot x_A$.
- (f) Sends $(broadcast, sid, (z_A, K_A))$ to $\mathcal{F}_{broadcast}$.

2. Each B_i does as follows:

Round 1

- (a) Receive $(broadcast, sid, (z_A, K_A))$ from $\mathcal{F}_{broadcast}$ and verify that $K_A \in \mathbb{G}$ else consider A_{pidA} malicious and abort.
- (b) Call the random oracle $\mathcal{H}(sid, X, pres_{X,sid}, K_A, msg)$ and receives μ_k .
- (c) B_i computes:
 - i. $ct_k = ct_{k_0} \oplus \mu_k \cdot ct_{k_1}$
 - ii. $K = (K_{B,0}) + (\mu_k \cdot K_{B,1}) + K_A$.
- (d) Calls the random oracle $\mathcal{H}(K, X, msg)$ and receives e .
- (e) Check that $z_A G = K_A + e \cdot X_A$. If not, consider A_{pidA} malicious and abort.
- (f) Compute $ct_B = ct_k \oplus (e \cdot ct_{key}) \oplus z_A$.
- (e) Send $(decrypt, pk, ct_B)$ to $\mathcal{F}_{TAHE}$.

Broadcast Round:

(a) Receive (decrypted, pk, ct_B, pt_B) from $\mathcal{F}_{TAHE}$.

(b) Set $\sigma = (\text{pt}_B, e)$.

(c) Send (global-broadcast, sid, i, σ) to $\mathcal{F}_{\text{global-broadcast}}$.

3. Output:

- A_{pid_A} receives (global-broadcast, sid, i, σ) from $\mathcal{F}_{\text{global-broadcast}}$.
- A_{pid_A} verifies that σ is a valid signature, else it aborts and considers B malicious.
- Both parties output σ .

It is noted that any output of a disclosed protocol (e.g., a DKG and/or Presign protocol) of parties in B may be stored in a shared public database. This may allow for a different subset of parties who did not participate in DKG to continue with a presign routine, or a different subset of parties who did not participate in DKG and/or presign to continue with a sign routine.

A user may transfer ownership of a resource, such as a digital wallet, to another user. Reference is made to Figure 10, which is a schematic illustration of an ownership transfer process, operative in accordance with an embodiment of the present disclosure. A sender client 210 seeks to transfer ownership of a resource (e.g., a digital wallet) to a receiver client 210. An ownership transfer routine is implemented by client ownership transfer modules 280 of sender client 210 and receiver client, and the ownership transfer is verified by ownership transfer verification module 180 of one or more parties 110 of decentralized network 105.

Sender client 210 receives as input a sender party identifier (pid_s), a receiver party identifier (pid_r), a receiver public key (X), and a private key share (x_A). Receiver client 210 receives as input a receiver secret decryption key share (sk_r), and outputs a private key share (x_A). Parties 110 and client 210 may access information from a shared database 190 for a share ($Enc_{pid_r}(x_A)$). Sender client 210 encrypts its private key share (x_A) under the receiver public key (X), along with a zero-knowledge proof of correctness. Network 105 verifies the proof and the policy and stores it in shared database 190. Receiver client 210 retrieves the encrypted share ($Enc_{pid_r}(x_A)$) from database 190 (after verifying the certificate), decrypts the share using its private decryption key, and derives and outputs the sender private key share.

Transfer Protocol: ($Transfer \Pi_{Transfer}: Transfer(\mathbb{G}, G, q, sid, pk, pk_{A_{new}} ct_{key})$)

The Transfer Protocol is parameterized with the group description $(\mathbb{G}, G, q)$. The Transfer Protocol interacts with the parties A_{old} , A_{new} and $\{B_i\}_{i \in S}$, where A_{old} transfers its secret key to A_{new} and $\{B_i\}_{i \in S}$ approves this transfer. All parties hold $pk, pk_{A_{new}}, X_{A_{old}}, X$ as input, where pk is a public key to the encryption scheme; $pk_{A_{new}}$ is a public key of A_{new} (which may be accessed from a shared database by querying $pid_{A_{new}}$; $X_{A_{old}}$ is a public share of A_{old} of a signing key (used to prove to B and to A_{new} that the correct secret share is transferred); and X is a public verification key. The parties agree on a fresh session identifier sid and do as follows:

1. A_{old} does as follows:

(a) Samples $\eta \leftarrow \mathcal{R}_{pk_{A_{new}}}$ and sets $ct_{new} = Enc_{pk_{A_{new}}}(x_{old}; \eta)$.

- (b) Sets $\pi_{transfer} \leftarrow \prod_{zk}^{L_{EncDL}[pk_{A_{new}}(\mathbb{G}, G, q)]}(X_{A_{old}}, ct_{transfer}; x_{old}; \eta)$.
- (c) Sends $(broadcast, sid, pid_{A_{old}}, (ct_{transfer}, \pi_{transfer}))$ to $\mathcal{F}_{broadcast}$.
2. $B_i \in B$ does as follows:
- (a) Receives $(broadcast, sid, pid_{A_{old}}, (ct_{transfer}, \pi_{transfer}))$ from $\mathcal{F}_{broadcast}$ and verifies $\pi_{transfer}$; if the proof fails consider A_{old} malicious; otherwise send $(global-broadcast, sid, i, ct_{transfer})$ to $\mathcal{F}_{global-broadcast}$.
- (b) Records $(X_{A_{old}}, pid_{A_{new}})$.
3. A_{new} does as follows:
- (a) Receives $(global-broadcast, sid, ct_{transfer})$ from $\mathcal{F}_{global-broadcast}$, sets $x_{new} = Dec_{pk_{A_{new}}}(ct_{transfer}; sk_{A_{new}})$, and records $(X_{A_{old}}, x_{new})$.

The present disclosure may support additive key derivation for ECDSA signatures.

Definition: “Slightly Enhanced ECDSA Signing Oracle: $\mathcal{G}_{SE-ECDSA}^*$ ”

Parameters include a cyclic group $(\mathbb{G}, G, q)$, a random oracle function $\mathcal{H}: \{0,1\}^* \rightarrow \mathbb{Z}_q$. Biased variables are denoted by $'$. Slightly Enhanced ECDSA Signing Oracle operates as follows:

1. On input (keygen, sid), sample $x \leftarrow \mathbb{Z}_q$, compute $X = x \cdot G$, record and send $(sid, X; x)$.
2. On input (biaskey, $\alpha_{key}, \beta_{key}$) set $x' \leftarrow \alpha_{key} \cdot x + \beta_{key}$ and $X' \leftarrow \alpha_{key} \cdot X + \beta_{key} \cdot G$.
3. On input (pres, ssid), sample $k_0, k_1 \leftarrow \mathbb{Z}_q$, compute $R_0 = k_0 \cdot G$ and $R_1 = k_1 \cdot G$, record and send $(ssid, R_0, R_1; k_0, k_1)$.
4. On input (sign, sid, ssid, msg, $\beta'_{key}, \alpha_{pres,0}, \beta_{pres,0}, \alpha_{pres,1}, \beta_{pres,1}$) do:
 - (a) Retrieve $(sid, X'; x')$ and $(ssid, R_0, R_1; k_0, k_1)$. If no such $(ssid, R_0, R_1; k_0, k_1)$ or sid exist, ignore.
 - (b) Compute $x'' \leftarrow x' + \beta'_{key}$ and $X'' \leftarrow X' + \beta'_{key} \cdot G$.
 - (c) Compute $k'_0 \leftarrow \alpha_{pres,0} \cdot k_0 + \beta_{pres,0}$ and $k'_1 \leftarrow \alpha_{pres,1} \cdot k_1 + \beta_{pres,1}$.
 - (d) Set $R'_0 = k'_0 \cdot G$ and $R'_1 = k'_1 \cdot G$.
 - (e) Set $m = \mathcal{H}(msg)$.
 - (f) Set $\mu = \mathcal{H}(X'', R'_0, R'_1, m; \alpha_{pres,0}, \beta_{pres,0}, \alpha_{pres,1}, \beta_{pres,1})$ and set $R' = R'_0 + \mu \cdot R'_1$.
 - (g) Set $r = R'_{x-ax}$ and compute $s = k^{-1} \cdot (m + r \cdot x'')$.
 - (h) Erase $(ssid, R_0, R_1; k_0, k_1)$ from memory and return $(sid, ssid, msg, \beta'_{key}, \alpha_{pres,0}, \beta_{pres,0}, \alpha_{pres,1}, \beta_{pres,1}; R', s, r)$.

At least one of the disclosed protocols discussed herein may include an identifiable abort, where malicious parties causing failure of protocol are identifiable.

At least one of the disclosed protocols discussed herein may be publicly verifiable, such that any entity (including but not limited to each of the parties), given a protocol transcript (collection of all messages sent from all parties), can check a validity of the protocol execution and of an output thereof.

At least one of the disclosed protocols discussed herein performed over an asynchronous communication channel may include guaranteed output delivery, such that if an adversary does not control a threshold of the parties, then the protocol will execute successfully.

At least one of the disclosed protocols discussed herein may be extended to fit a distributed system serving a plurality of clients concurrently. The distributed parties may hold a single secret share for the underlying TAHE scheme, and the network share of each secret signing key may be stored publicly encrypted under the corresponding TAHE public key.

The disclosed embodiments may include a secure function evaluation for the TAHE, allowing a user to evaluate a private affine function f homomorphically on a tuple of ciphertexts $ct_i = Enc(w_i, r_i)$ without revealing information on function f except for $f(x_1, \dots, x_n)$, even given a secret key of the TAHE and randomizers r_i and plaintexts x_i for the encryptions ct_i .

The TAHE may be tied with elliptic curve group elements, such that the corresponding witnesses thereof satisfy linear relations and have range constraints, using at least one generic zero-knowledge (zk) range proof. The ZK proof may be a novel ZK proof for allowing aggregation of batched statements from multiple provers into a single proof.

In accordance with the presign protocols of the present disclosure, it may be proven that the presigns can be generated offline in a common pool shared with all clients in the network. Upon signing, the clients may use the same pool of presigns to sign transactions.

In some aspects of the present disclosure, the same key may be used for both an ECDSA signature and for a Schnorr signature.

The disclosed embodiments may include one or more performance boosting procedures. One example is batching zk-proofs, such that a single proof can be derived by a prover for multiple statements from multiple parallel executions of at least one of the protocols. Another example is when using Class-Groups, range proofs are redundant and as a result the verification of such batched proofs weakly depends on the batch size (it is linearly dependent, but with a significantly smaller constant compared to the verification time of a single non-batched proof). A further example is amortized decryption, such that it is enough for a single party to combine decryption shares and derive the signature, where the rest of the parties will also validate the signature, and only if verification fails they will have to zk-prove validity of decryption shares and combine decryption shares on their own, such that each party is now $\mathcal{O}(1)$ amortized in the online signing phase. Yet another example is synchronous decryption, where if the subset of parties participating in signing is predetermined and known before the threshold decryption round starts, each party can raise its decryption share to the corresponding Lagrange coefficient; then, combining the decryption shares only consists of n-group multiplication.

Figure. 11 illustrates a diagrammatic representation of a machine in the example form of a computing device 300 within which a set of instructions, for causing the machine to perform any one or more of the methodologies discussed herein, may be executed. In alternative embodiments, the machine may be connected (e.g., networked) to other machines in a Local Area Network (LAN), an intranet, an extranet, or the Internet. Computing device 300 may correspond, for example, to a computing device of a client 210 or party 110 of FIG. 1. The machine may operate in the capacity of a server or a client machine in a client-server network environment, or as a peer machine in a peer-to-peer (or distributed) network environment. The machine may be a personal computer (PC), a tablet computer, a set-top box (STB), a Personal Digital Assistant (PDA), a cellular telephone, a web appliance, a server, a network router, switch or bridge, or any machine capable of executing a set of instructions (sequential or otherwise) that specify actions to be taken by that machine. Further, while only a single machine is illustrated, the term “machine” shall also be taken to include any collection of machines (e.g., computers) that individually or jointly execute a set (or multiple sets) of instructions to perform any one or more of the methodologies discussed herein.

The example computing device 300 includes a processing device 302, a main memory 304 (e.g., read-only memory (ROM), flash memory, dynamic random access memory (DRAM) such as synchronous DRAM (SDRAM), etc.), a static memory 306 (e.g., flash memory, static random access memory (SRAM), etc.), and a secondary memory (e.g., a data storage device 328), which communicate with each other via a bus 308.

Processing device 302 represents one or more general-purpose processors such as a microprocessor, central processing unit, or the like. More particularly, processing device 302 may be a complex instruction set computing (CISC) microprocessor, reduced instruction set computing (RISC) microprocessor, very long instruction word (VLIW) microprocessor, processor implementing other instruction sets, or processors implementing a combination of instruction sets. Processing device 302 may also be one or more special-purpose processing devices such as an application specific integrated circuit (ASIC), a field programmable gate array (FPGA), a digital signal processor (DSP), network processor, or the like. Processing device 302 is configured to execute the processing logic (instructions 326) for performing operations and steps discussed herein.

Computing device 300 may further include a network interface device 322 for communicating with a network 364. Computing device 300 also may include a video display unit 310 (e.g., a liquid crystal display (LCD) or a cathode ray tube (CRT)), an alphanumeric input device 312 (e.g., a keyboard), a cursor control device 314 (e.g., a mouse), and a signal generation device 320 (e.g., a speaker).

Data storage device 328 may include a machine-readable storage medium (or more specifically a non-transitory computer-readable storage medium) 324 on which is stored one or more sets of instructions 326 embodying any one or more of the methodologies or functions described herein, such as instructions for one or more DN party modules 315, which may correspond to DN party modules 130 of FIG. 2 in embodiments. A non-transitory storage medium refers to a storage medium other than a carrier wave. The instructions 326 may also reside, completely or at least partially, within main memory 304 and/or within processing device 302 during execution thereof.

by computer device 300, main memory 304 and processing device 302 also constituting computer-readable storage media. Computer readable storage medium 324 may also store a software library containing methods for one or more of the DN party modules 315. While computer-readable storage medium 324 is shown in an example embodiment to be a single medium, the term “computer-readable storage medium” should be taken to include a single medium or multiple media (e.g., a centralized or distributed database, and/or associated caches and servers) that store the one or more sets of instructions. The term “computer-readable storage medium” shall also be taken to include any medium other than a carrier wave that is capable of storing or encoding a set of instructions for execution by the machine and that cause the machine to perform any one or more of the methodologies of the present disclosure. The term “computer-readable storage medium” shall accordingly be taken to include, but not be limited to, solid-state memories, and optical and magnetic media.

Reference is made to Figure 12, which is a flow diagram of a general method for generating a digital signature, operative in accordance with an embodiment of the present disclosure. In first step 411, AHE DKG is performed. In a step 412, ECDSA/Schnorr presign pool is provided. In a step 413, background processes are performed. In a step 414, ECDSA/Schnorr DKG is performed. In a step 415, ECDSA/Schnorr presign is performed. In a step 416, ECDSA/Schnorr child keys are provided. In a step 417, ECDSA/Schnorr sign is performed. In a step 418, wallet transfer or sharing is performed. In a step 419, network reconfiguration is performed.

While certain embodiments of the disclosed subject matter have been described, so as to enable one of skill in the art to practice the present disclosure, the preceding description is intended to be exemplary only. It should not be used to limit the scope of the disclosed subject matter, which should be determined by reference to the following claims.

CLAIMS

1. A method for generating at least one digital signature, comprising:
 following one or more secure protocols based on a threshold additively homomorphic encryption (TAHE) over a broadcast channel, wherein a threshold of parties of a decentralized network (DN) produces the digital signature by following a Universally Composable (UC)-secure protocol configured to output at least one of an Elliptic curve digital signature algorithm (ECDSA) signature or a Schnorr signature.
2. The method of claim 1, wherein at least one client and the threshold of parties of the decentralized network cooperate to generate the digital signature.
3. The method of claim 1, wherein the threshold of parties of the decentralized network cooperate to generate the digital signature without a client.
4. The method of claim 1, further comprising generating a distributed key using a distributed key generation (DKG) protocol of the one or more secure protocols.
5. The method of claim 1, The method of claim 4, wherein the DKG protocol is a DKG protocol over a synchronous communication channel ("DKG Protocol Sync"), the DKG Protocol Sync being parameterized with a group description $(\mathbb{G}, G, q)$ and interacting with $n + 1$ parties including a client A , and DN parties $B = \{B_i\}_{i \in [n]}$, wherein all parties hold a public key (pk) as input and agree on a fresh session-identifier (sid) that uniquely identifies a session.
6. The method of claim 5, wherein the parties perform the following:
 1. A 's first message:
 - (a) A samples a random $x_A \leftarrow \mathbb{Z}_q$ and computes $X_A = x_A \cdot G$;
 - (b) A sends $(com-prove, pid_A, X_A; x_A)$ to $\mathcal{F}_{com-zk}^{LDL}$;
 2. B_i 's message:
 - (a) each B_i receives $(receipt, pid_A)$ from $\mathcal{F}_{com-zk}^{LDL}$;
 - (b) each B_i samples a random $x_i \leftarrow \mathbb{Z}_q$ and computes $X_i = x_i \cdot G$;

- (c) only on first execution: Each B_i sends $(keygen, sid)$ to $\mathcal{F}_{TAHE}$ and waits for a response from $\mathcal{F}_{com-zk}^{L_{DL}}$; if $\mathcal{F}_{com-zk}^{L_{DL}}$ responds with $(pubkey, sid, \perp, U' \cap U)$ then output $U' \cap U$ and abort; otherwise, if $\mathcal{F}_{com-zk}^{L_{DL}}$ responds with $(pubkey, sid, pk)$ then continue;
- (d) each B_i computes $ct_i = AHE.Enc(pk, x_i; \rho_i)$ for a randomly chosen ρ_i ;
- (e) each B_i sends $(prove, sid, pid_i, X_i, ct_i; x_i; \rho_i)$ to $\mathcal{F}_{agg-zk}^{L_{EncDL}[pk, (\mathbb{G}, G, q)]}$ and $(prove, sid, X_i; x_i)$ to $\mathcal{F}_{agg-zk}^{L_{DL}[(\mathbb{G}, G, q)]}$;
3. B_i 's verification:
- (a) each B_i receives $(proof, sid, X_B, ct_{key})$ from $\mathcal{F}_{agg-zk}^{L_{EncDL}}$; if not, it receives and outputs the set of corrupted parties and aborts;
4. A 's second message:
- (a) A receives $(broadcast, sid, X_B)$ from $\mathcal{F}_{agg-zk}^{L_{DL}}$;
- (b) A sends $(decom-proof, pid_A)$ to $\mathcal{F}_{com-zk}^{L_{DL}}$;
5. B_i 's verification:
- (a) each B_i receives $(decom-proof, pid_A, X_A)$ from $\mathcal{F}_{com-zk}^{L_{DL}}$; if not, it aborts;
6. Output:
- A outputs $X = X_A + X_B$ and records $(keygen, X_B, X; x_A)$;
 - each B_i outputs $X = X_A + X_B$ and records $(keygen, X_A, X, ct_{key}, pk)$;
7. The method of claim 4, wherein DKG protocol is a DKG protocol over an asynchronous communication channel ("DKG Protocol Async"), the DKG Protocol Async being parameterized with a group description $(\mathbb{G}, G, q)$, an access structure Γ_B , a unique session identifier sid , and an encryption key to a TAHE scheme pk , wherein the DKG Protocol Async interacts with a centralized party A_{pidA} and a set of parties $B = \{B_i\}_{i \in [n]}$, wherein the set of parties B can only send messages to parties in A through a functionality $\mathcal{F}_{global-broadcast}$.
8. The method of claim 7 wherein the parties output a public verification key X and record the following:
- X_A : the centralized party's share of a public verification key for an ECDSA/Schnorr signature;

- the distributed party B also records ct_{key} an encryption of the secret share held by distributed party B ;
- the centralized party A_{pidA} also records x_A , its secret share of an ECDSA/Schnorr signature key,

wherein the parties perform the following:

1. each B_i does as follows:

Round 1

- (a) sample $x_i \leftarrow \mathbb{Z}_q$ and $\eta_i \leftarrow \mathcal{R}_{pk}$;
- (b) compute $X_i = x_i \cdot G$, and $ct_i = \text{AHE.Enc}(pk, x_i; \eta_i)$;
- (c) run $\Pi_{agg}^{\Gamma_{B, L_{EncDL}[pk, (\mathbb{G}, G, q)]}}$ on inputs (X_i, x_i, η_i) receiving (X_B, ct_{key}) ;

Broadcast Round:

- (a) send (global-broadcast, sid, i , (pid_A, X_B)) to $\mathcal{F}_{global-broadcast}^{\Gamma_B}$;
2. A_{pidA} does as follows:
 - (a) receive (global-broadcast, sid, (pid_A, X_B)) from $\mathcal{F}_{global-broadcast}^{\Gamma_B}$;
 - (b) verify that $X_B \in \mathbb{G}$, else consider B malicious and abort;
 - (c) sample $x_A \leftarrow \mathbb{Z}_q$ and set $X_A = x_A \cdot G$;
 - (d) run $\Pi_{zk-uc}^{LDL[(\mathbb{G}, G, q)]}(X_A; x_A)$ generating a proof π_{DL} .
 - (e) Send (broadcast, sid, pid_A, X_A, π_{DL}) to $\mathcal{F}_{broadcast}$.

3. Output:

each B_i does as follows:

- (a) receive (sid, pid_A, X_A, π_{DL});
- (b) verify π_{DL} , if fails consider A_{pidA} malicious and abort, otherwise continue;
- (c) set $X = X_A + X_B$;
- (d) output X and record (keygen, pid_A, X, X_A, ct_{key});

A_{pidA} does as follows:

- (a) set $X = X_A + X_B$;
- (b) output X and record (keygen, $pid_A, X, X_A; x_A$).

9. The method of claim 2, further comprising distributing, by a client that holds a private signing key, the private signing key between the client and the decentralized network.

10. The method of claim 1, further comprising pre-signing using a presign protocol of the one or more secure protocols.
11. The method of claim 10, wherein the presign protocol is a presign protocol using an ECDSA scheme over a synchronous communication channel ("Presign Protocol ECDSA/Sync"), the Presign Protocol ECDSA/Sync interacting with $n + 1$ parties including a client A , and DN parties $B = \{B_i\}_{i \in [n]}$, wherein all parties hold a public key (pk) and an encryption of a network share under TAHE (ct_{key}) as input, wherein all parties verify that a session-identifier sid has never been used before.
12. The method of claim 11, wherein all parties further perform the following:

1. A 's message:

- (a) A samples a random $k_A, \rho_1 \leftarrow \mathbb{Z}_q$ and computes $K_A = Com(k_A; \rho_1)$;
 (b) A sends $(prove, sid, pid_A, K_A; k_A, \rho_1)$ to $\mathcal{F}_{zk}^{LDCOM}$;

2. B_i 's message:

Round 1

- (a) B_i receives $(proof, sid, pid_A, K_A)$ from $\mathcal{F}_{zk}^{LDCOM}$, if not, it aborts;
 (b) B_i samples $k_i \leftarrow \mathbb{Z}_q^*$ and computes $R_i = k_i \cdot G$. Denote $k_B = \sum_i k_i$;
 (c) B_i samples $\gamma_i \in [0, q)$, $\eta_{mask_1}^i, \eta_{mask_2}^i, \eta_{mask_3}^i, \eta_{mask_4}^i \in Rpk$, and computes:
 iv. $ct_1^i = AHE.Enc(pk, \gamma_i; \eta_{mask_1}^i)$;
 v. $ct_2^i = AHE.Eval(pk, f_i, ct_{key}; \eta_{mask_2}^i)$, where: $f_i(x) = \gamma_i \cdot x$;
 vi. $ct_3^i = AHE.Enc(pk, k_i; \eta_{mask_3}^i)$;
 (d) B_i sends $(prove, sid \parallel \gamma, pid_i, ct_1^i, ct_2^i; \gamma_i, \eta_{mask_1}^i, \eta_{mask_2}^i)$ to $\mathcal{F}_{agg-zk}^{LEncDH[pk, ct_{key}]}$;
 (e) B_i sends $(prove, sid, pid_i, ct_3^i, k_i, \eta_{mask_3}^i)$ to $\mathcal{F}_{agg-zk}^{LEncDL}$.

Round 2

- (e) B_i receives $(prove, sid \parallel \gamma, pid_i, ct_1, ct_2;)$ from $\mathcal{F}_{agg-zk}^{LEncDH[pk, ct_{key}]}$, and $(prove, sid, R_B, ct_3)$ from $\mathcal{F}_{agg-zk}^{LEncDL}$.

- (f) Otherwise, if B_i receives $(malicious, sid, U')$ from $\mathcal{F}_{agg}^{L_{EncDH}[pk, ct_{key}]}$ and/or $(malicious, sid, U'')$ from $\mathcal{F}_{agg-zk}^{L_{EncDL}}$, it records the malicious parties (M and/or M');
- (g) B_i computes $ct_4^i = AHE.Eval(pk, f_i', ct_1; \eta_{mask_4}^i)$ where: $f_i'(x) = k_i \cdot x$;
- (h) B_i sends $(prove, sid \parallel "k", pid_i, ct_3^i, ct_4^i, k_i, \eta_{mask_3}^i, \eta_{mask_4}^i)$ to $\mathcal{F}_{agg-zk}^{L_{EncDH}[pk, ct_1]}$;

Proof verification:

- (b) B_i receives $(proof, sid \parallel "k", pid_i, ct_3, ct_4)$ from $\mathcal{F}_{agg-z}^{L_{EncDH}[pk, ct_1]}$; otherwise, if B_i receives $(malicious, sid, U'')$ from $\mathcal{F}_{agg}^{L_{EncDH}[pk, ct_1]}$, it records the malicious parties and aborts;

3. A 's verification:

- (d) A receives $(proof, sid \parallel R_B, ct_3)$ from $\mathcal{F}_{agg-z}^{L_{EncDL}}$, if not, it aborts;
- (e) A receives $(proof, sid \parallel "y", ct_1, ct_2)$ from $\mathcal{F}_{agg-zk}^{L_{EncDH}[pk, ct_{key}]}$, if not, it aborts;

4. Output:

- A records $(presign, sid, R_B, ct_1, ct_2; k_A, \rho_1)$, where ct_1 and ct_2 are encryptions of γ and $\gamma \cdot x_B$;
- B_i records $(presign, sid, R_B, K_A, ct_1, ct_2, ct_4)$, where ct_4 encrypts $\gamma \cdot k_B \bmod q$.

13. The method of claim 10, wherein the presign protocol is a presign protocol using a Schnorr scheme over a synchronous communication channel ("Presign Protocol Schnorr/Sync"), the Presign Protocol Schnorr/Sync interacting with $n + 1$ parties including a client A , and DN parties $B = \{B_i\}_{i \in [n]}$, wherein all parties have an encryption of a network share under TAHE (ct_{key}) as input, wherein all parties verify that a session-identifier (sid) has never been used before.

14. The method of claim 13, wherein all parties further perform the following:

1. A 's message:

- (a) A samples a random $k_A, \rho \leftarrow \mathbb{Z}_q$ and computes $K_A = Com(k_A; \rho)$;
- (b) A sends $(prove, sid, pid_A, K_A; k_A, \rho)$ to $\mathcal{F}_{zk}^{L_{DCom}}$;

2. B_i 's message:

- (a) B_i receives $(proof, sid, pid_A, K_A)$ from $\mathcal{F}_{zk}^{L_{DCom}}$, if not, it aborts;

- (b) B_i samples $k_i^0, k_i^1 \leftarrow \mathbb{Z}_q^*$ and computes $R_i^0 = k_i^0 \cdot G$ and $R_i^1 = k_i^1 \cdot G$; denote $k_B^0 = \sum_i k_i^0$ and $k_B^1 = \sum_i k_i^1$;
- (c) B_i samples $\eta_i^0 \in R_{pk}$, and computes $ct_{k_0}^i = AHE.Enc(pk, k_i^0; \eta_i^0)$ and $ct_{k_1}^i = AHE.Enc(pk, k_i^1; \eta_i^1)$;
- (d) B_i sends $(prove, sid, pid_i, R_i^0, ct_{k_0}^i; k_i^0, \eta_i^0)$ and $(prove, sid, pid_i, R_i^1, ct_{k_1}^i; k_i^1, \eta_i^1)$ to $\mathcal{F}_{agg-z}^{LEncDL[(\mathbb{G}, G, q)]}$;
- (e) B_i sends $(prove, sid, pid_i, R_i^0; k_i^0)$ and $(prove, sid, pid_i, R_i^1; k_i^1)$ to $\mathcal{F}_{agg-zk}^{LDL[(\mathbb{G}, G, q)]}$;
3. B_i 's verification:
- (a) B_i receives $(proof, sid, R_B^0, ct_{k_0})$ and $(proof, sid, pid, R_B^1, ct_{k_1})$ from $\mathcal{F}_{agg-zk}^{LEncDL[(\mathbb{G}, G, q)]}$;
- (b) Otherwise, if B_i receives $(malicious, sid, U'')$ from $\mathcal{F}_{agg-zk}^{LEncDL[(\mathbb{G}, G, q)]}$, it records malicious parties U and aborts;
4. A 's verification:
- (f) A receives $(proof, sid, R_B^0)$ and $(proof, sid, R_B^1)$ from $\mathcal{F}_{agg-zk}^{LEncDL[(\mathbb{G}, G, q)]}$, if not, it aborts;
5. Output:
- A records $(presign, sid, R_B^0, R_B^1, K_A; k_A, \rho)$;
 - B_i records $(presign, sid, R_B^0, R_B^1, K_A, ct_{k_0}, ct_{k_1})$;
15. The method of claim 10, wherein the presign protocol is a presign protocol using an ECDSA scheme over an asynchronous communication channel ("Presign Protocol ECDSA/Async"), the Presign Protocol ECDSA/Async parameterized with: a group description $(\mathbb{G}, G, q)$, an access structure Γ_B , a unique session identifier sid , an encryption key to a TAHE scheme pk , an ECDSA verification key X , a centralized party share of the verification key X_A , and an encryption of a share of a distributed party of a signing key ct_{key} , the Presign Protocol ECDSA/Async interacting with a centralized party A_{pidA} and a set of parties $B = \{B_i\}_{i \in [n]}$, wherein the set of parties B can only send messages to parties in A through a functionality $\mathcal{F}_{global-broadcast}$.
16. The method of claim 15, wherein each party B_i outputs:
- ct_γ : an encryption of a randomizer γ ;

- $ct_{\gamma \cdot key}$: an encryption of the randomizer γ multiplied by the signing key share of the blockchain x_B ;
 - $ct_{\gamma \cdot k_0}, ct_{\gamma \cdot k_1}$: an encryption of a randomized nonces $k_0 \cdot \gamma$, and $k_1 \cdot \gamma$, respectively;
 - $R_{B,0}, R_{B,1}$: points on the curve equal to $k_0 \cdot G$, and $k_1 \cdot G$, respectively,
1. each B_i does as follows:

Round 1

(a) Sample a random $\gamma_i \leftarrow \mathbb{Z}_q$ and $\eta_1^i, \eta_2^i \leftarrow \mathcal{R}_{pk}$, and compute:

- i. $ct_\gamma^i = \text{AHE.Enc}(pk, \gamma_i, \eta_1^i)$;
- ii. $ct_{\gamma \cdot key}^i = \text{AHE.Scale}(pk, ct_{key}, \gamma_i; \eta_2^i)$;

(b) Run $\Pi_{agg}^{\Gamma_B, L_{EncDH}}[pk, ct_{key}]$ on inputs $(ct_\gamma^i, ct_{\gamma \cdot key}^i; \gamma_i; \eta_1^i, \eta_2^i)$ receiving $(ct_\gamma, ct_{\gamma \cdot key})$;

Round 2

(f) Sample a random $k_0^i, k_1^i \leftarrow \mathbb{Z}_q$ and $\eta_{3,0}^i, \eta_{3,1}^i \leftarrow \mathcal{R}_{pk}$, and compute:

- i. $ct_{\gamma \cdot k_0}^i = \text{AHE.Scale}(pk, ct_\gamma, k_0^i, \eta_{3,0}^i)$;
- ii. $ct_{\gamma \cdot k_1}^i = \text{AHE.Scale}(pk, ct_\gamma, k_1^i, \eta_{3,1}^i)$;
- iii. $R_{B,0}^i = k_0^i \cdot G$;
- iv. $R_{B,1}^i = k_1^i \cdot G$;

(g) Run $\Pi_{agg}^{\Gamma_B, L_{ScaleDL}}[pk, (\mathbb{G}, G, q), ct_\gamma]$ on inputs $(ct_{\gamma \cdot k_0}^i, R_{B,0}^i; k_0^i, \eta_{3,0}^i)$ and $(ct_{\gamma \cdot k_1}^i, R_{B,1}^i; k_1^i, \eta_{3,1}^i)$ receiving $(ct_{\gamma \cdot k_0}, R_{B,0}), (ct_{\gamma \cdot k_1}, R_{B,1})$;

Broadcast Round:

(b) Send $(\text{global-broadcast}, \text{sid}, i, (\text{pid}_A, ct_\gamma, ct_{\gamma \cdot key}, ct_{\gamma \cdot k_0}, ct_{\gamma \cdot k_1}, R_{B,0}, R_{B,1}))$ to $\mathcal{F}_{global-broadcast}^{\Gamma_B}$;

2. Output:

- A_{pid_A} receives $(\text{global-broadcast}, \text{sid}, (\text{pid}_A, ct_\gamma, ct_{\gamma \cdot key}, ct_{\gamma \cdot k_0}, ct_{\gamma \cdot k_1}, R_{B,0}, R_{B,1}))$ from $\mathcal{F}_{global-broadcast}^{\Gamma_B}$;
- A_{pid_A} verifies that $R_{B,0}, R_{B,1} \in \mathbb{G} \setminus \{0\}$ and $ct_\gamma, ct_{\gamma \cdot key}, ct_{\gamma \cdot k_0}, ct_{\gamma \cdot k_1} \in \mathcal{C}_{pk}$, else it aborts and considers B malicious;
- Each party B_i records $(\text{sid}, X, ct_\gamma, ct_{\gamma \cdot key}, ct_{\gamma \cdot k_0}, ct_{\gamma \cdot k_1}, R_{B,0}, R_{B,1})$.

17. The method of claim 10, wherein the presign protocol is a presign protocol using a Schnorr scheme over an asynchronous communication channel (“Presign Protocol Schnorr/Async”), the Presign Protocol Schnorr/Async parameterized with: a group description $(\mathbb{G}, G, q)$, an access structure Γ_B , a unique session identifier sid , and an encryption key to a TAHE scheme pk , the Presign Protocol Schnorr/Async interacting with a centralized party A_{pidA} and a set of parties $B = \{B_i\}_{i \in [n]}$, wherein the set of parties B can only send messages to parties in A through a functionality $\mathcal{F}_{global-broadcast}$.

18. The method of claim 17 wherein each party B_i outputs:

- $K_{B,0}, K_{B,1}$: points on the curve equal to $k_0 \cdot G$, and $k_1 \cdot G$, respectively;
- The distributed parties also output ct_{k_0}, ct_{k_1} : encryptions of k_0, k_1 , respectively,
- 1. each B_i does as follows:

Round 1

(a) B_i samples a random $k_0^i, k_1^i \leftarrow \mathbb{Z}_q$ and $\eta_0^i, \eta_1^i \leftarrow \mathcal{R}_{pk}$, and computes:

- i. $K_{B,0}^i = k_0^i \cdot G$;
- ii. $K_{B,1}^i = k_1^i \cdot G$;
- iii. $ct_{k_0}^i = \text{AHE.Enc}(pk, k_0^i, \eta_0^i)$;
- iv. $ct_{k_1}^i = \text{AHE.Enc}(pk, k_1^i, \eta_1^i)$;

(b) Run $\Pi_{agg}^{LEncDL[pk, (\mathbb{G}, G, q)]}$ on inputs $(ct_{k_1}^i, K_{B,0}^i; k_0^i, \eta_0^i)$ and $(ct_{k_1}^i, K_{B,1}^i; k_1^i, \eta_1^i)$ receiving $(ct_{k_0}, K_{B,0}), (ct_{k_1}, K_{B,1})$;

Broadcast Round:

(c) Send $(\text{global-broadcast}, sid, i, (pid_A, K_{B,0}, K_{B,1}))$ to $\mathcal{F}_{global-broadcast}$;

2. Output:

- A_{pidA} receives $(\text{global-broadcast}, sid, (pid_A, K_{B,0}, K_{B,1}))$ from $\mathcal{F}_{global-broadcast}^{\Gamma_B}$;
- A_{pidA} verifies that $K_{B,0}, K_{B,1} \in \mathbb{G} \setminus \{0\}$, else it aborts and considers B malicious;
- Both parties output $K_{B,0}, K_{B,1}$;
- Parties in B also record ct_{k_0}, ct_{k_1} .

19. The method of claim 1, further comprising signing a message using a sign protocol of the one or more secure protocols.
20. The method of claim 19, wherein the sign protocol is a sign protocol using an ECDSA scheme over a synchronous communication channel (“Sign Protocol ECDSA/Sync”), the Sign Protocol ECDSA/Sync interacting with $n + 1$ parties including a client A , and DN parties $B = \{B_i\}_{i \in [n]}$, wherein all parties hold a public key (pk), network public nonce shares (R_B^0, R_B^1), and a commitment on A ’s share of the nonce for signature (K_A), as input, wherein all parties verify that a session-identifier (sid) has never been used before.

21. The method of claim 20, wherein all parties further perform the following:

1. A ’s message:

- (a) A computes $R = (k_A)^{-1} \cdot R_B = (k_A^{-1} k_B \cdot G)$ and $r = R_{x-axis} \bmod q$. Denote $k = k_A^{-1} k_B$;
- (b) A samples $\rho_2 \in \mathcal{R}_{pp}$ and computes $U_A = Com(k_A \cdot x_A; \rho_2)$;
- (c) A sets $a_1 = r \cdot k_A \cdot x_A + m \cdot k_A$ and $a_2 = r \cdot k_A$, where $m := \mathcal{H}(msg)$;
- (d) A homomorphically evaluates ct_1, ct_2 on its private function $f_A(x_A, x_A) := a_1 x_1 + a_2 x_2$:
 - $ct_A \leftarrow AHE.Eval(pk, f_A, ct_1, ct_2; \eta_{eval})$;
- (e) A sends message and the following proofs:
 - A sends $(prove, sid, pid_A, K_A, R_B; k_A, \rho_1)$ to $\mathcal{F}_{zk}^{L_{DComDL}[pp, (\mathbb{G}, G, q)]}$;
 - A sends $(prove, sid, pid_A, K_A, U_A; k_A, x_A, \rho_1, \rho_2)$ to $\mathcal{F}_{zk}^{L_{DComRatio}[pp, (\mathbb{G}, G, q), X_A]}$;
 - A computes $C_1 = (r \odot U_A) \oplus (m \odot K_A)$ and $C_2 = (r \odot K_A)$ and sends $(prove, sid, pid_A, ct_A, C_1, C_2; a_1, r \cdot \rho_2 + m, r \cdot \rho_1 + m, \eta)$ to $\mathcal{F}_{zk}^{L_{DComEval}[pp, pk, ct_1, ct_2]}$;

2. B_i ’s verification and output:

- (a) B_i receives msg and the following proofs, otherwise it aborts:
 - $(proof, sid \parallel pid_A, K_A, R_B)$ from $\mathcal{F}_{zk}^{L_{DComDL}[pp, (\mathbb{G}, G, q)]}$;
 - $(proof, sid \parallel pid_A, K_A, U_A)$ from $\mathcal{F}_{zk}^{L_{DComRatio}[pp, (\mathbb{G}, G, q), X_A]}$;

- $(proof, sid \parallel pid_A, ct_A, C_1, C_2)$ from $\mathcal{F}_{zk}^{LDComEval[pp, pk, ct_1, ct_2]}$;
 - (b) B_i verifies that values used in the proofs are consistent with values obtained previously by B_i , specifically:
 - There are records $(keygen, X_A, X, ct_{key}, pk)$ and $(presign, sid, R_B, K_A, pt')$; and
 - $C_1 = (r \odot U_A) \oplus (m \odot K_A)$ and $C_2 = (r \odot K_A)$, where $r = R_{x-axis} \bmod q$ and $m = \mathcal{H}(msg)$;
 - (c) B_i sends $(decrypt, pk, ct_A)$ and $(decrypt, pk, ct_4)$ to $\mathcal{F}_{TAHE}$ and waits for its response. Let the responses be $(decrypted, pk, ct_A, pt_A, U_A)$ and $(decrypted, pk, ct_4, pt_4, U_4)$ respectively; if $pt_A = \perp$ or $pt_4 = \perp$ then output $U_A \cup U_4$ and abort; otherwise, compute $s' = pt_4^{-1} \cdot pt_A \bmod q$ (which is equal to $(\gamma k_B)^{-1} \cdot ((rk_A x_A + mk_A)\gamma + rk_A \gamma x_B) = k^{-1}(rx + m) \bmod q$ as required) and output $s = \min\{s', q - s'\}$ (to ensure uniqueness of the signature);
3. Output:
- B_i outputs $\sigma = (r, s)$.
22. The method of claim 19, wherein the sign protocol is a sign protocol using a Schnorr scheme over a synchronous communication channel ("Sign Protocol Schnorr/Sync"), the Sign Protocol Schnorr/Sync interacting with $n + 1$ parties including a client A , and DN parties $B = \{B_i\}_{i \in [n]}$, wherein all parties have a public key (pk), network public nonce shares (R_B^0, R_B^1) , a commitment on A 's share of the nonce for signature (K_A), an encryption of a network share (ct_{key}), and encryptions of nonces k_0, k_1 (ct_{k_0}, ct_{k_1}) as input, wherein all parties verify that a session-identifier (sid) has never been used before.
23. The method of claim 22, wherein all parties further perform the following:
1. A 's message:
 - (a) A computes $R_A = k_A \cdot G$;
 - (b) A sends $(prove, sid, pid_A, K_A, R_A; k_A, \rho)$ to $\mathcal{F}_{zk}^{LDComDL[(G, G, q)]}$;
 - (c) A calls the random oracle $\mathcal{H}(msg)$ and receives m ;
 - (d) A calls the random oracle $\mathcal{H}(msg, R_A, R_B^0, R_B^1)$ and receives μ ;
 - (e) A computes $R = R_A + R_B^0 + \mu \cdot R_B^1$;
 - (f) A calls the random oracle $\mathcal{H}(R, X, m)$ and receives e ;

- (g) A computes $z = K_A + e \cdot x_Z$;
- (h) A sends msg and z to $\{B_i\}_{i \leq \eta}$;
2. B_i 's verification and output:
- (a) B_i receives msg and z from A , and $(proof, sid, pid_A, K_A, R_A)$ from $\mathcal{F}_{zk}^{L_{DCoM}DL[(\mathbb{G}, G, q)]}$; otherwise, it aborts;
- (b) B_i calls the random oracle $\mathcal{H}(msg)$ and receives m ;
- (c) B_i calls the random oracle $\mathcal{H}(m, R_A, R_B^0, R_B^1)$ and receives μ ;
- (d) B_i computes $R = R_A + R_B^0 + \mu \cdot R_B^1$;
- (e) B_i calls the random oracle $\mathcal{H}(R, X, m)$ and receives e ;
- (f) B_i computes $ct_B = z \oplus ct_k \oplus (e \odot ct_{key})$;
- (g) B_i sends $(decrypt, pk, ct_B)$ to $\mathcal{F}_{TAHE}$ and receives $(decrypted, pk, ct_B, pt_B, \mathcal{U}A)$; otherwise, it records the malicious parties and aborts;
- (h) B_i verifies that $pt_B \cdot G = R + e \cdot X$; if not, it aborts;
3. Output:
- B_i outputs $\sigma = (R, pt_B)$.
24. The method of claim 19, wherein the sign protocol is a sign protocol using an ECDSA scheme over an asynchronous communication channel ("Sign Protocol ECDSA/Async"), the Sign Protocol ECDSA/Async parameterized with a group description $(\mathbb{G}, G, q)$, a public generator (H) used for Pedersen commitments, an access structure Γ_B , a unique session identifier (sid) , an encryption key to a TAHE scheme pk , an ECDSA verification key X , a centralized party share of the verification key X_A , an encryption of a share of a distributed party of a signing key ct_{key} , an output of a uniquely determined previous presign protocol $presX, sid = (ct_\gamma, ct_{\gamma \cdot key}, ct_{\gamma \cdot k_0}, ct_{\gamma \cdot k_1}, R_{B,0}, R_{B,1})$, and a message to be signed (msg) , the Sign Protocol ECDSA/Async interacting with a centralized party A_{pidA} and a set of parties $B = \{B_i\}_{i \in [n]}$, wherein the set of parties B can only send messages to parties in A through a functionality $\mathcal{F}_{global-broadcast}$, wherein the parties output a valid ECDSA signature $\sigma = (r, s)$ with verification key X .
25. The method of claim 24, wherein:
- A_{pidA} does as follows:

- (a) calls the random oracle $\mathcal{H}(msg)$ and receives m ;
- (b) samples $k_A, \alpha, \beta \leftarrow \mathbb{Z}_q$ and $\rho_0, \rho_1, \rho_2, \rho_3 \leftarrow \mathcal{R}_{pk}$ and computes:
- $C_k = \text{Pedersen.Com}_{G,H}(k_A; \rho_0)$;
 - $C_\alpha = \text{Pedersen.Com}_{G,H}(\alpha; \rho_1)$;
 - $C_\beta = \text{Pedersen.Com}_{G,H}(\beta; \rho_2)$;
 - $C_{kx} = \text{Pedersen.Com}_{X_A,H}(k_A; \rho_3)$;
- (c) calls the random oracle $\mathcal{H}(msg)$ and receives m ;
- (d) runs protocols $\prod_{zk-uc}^{L_{DCom}[C_k,H]}(G; k_A^{-1}, -k_A^{-1}, \rho_0)$, $\prod_{zk-uc}^{L_{DCom}[C_\alpha,H]}(G; \alpha^{-1}, -\alpha^{-1}, \rho_1)$ and $\prod_{zk-uc}^{L_{DCom}[C_\beta,H]}(C_\beta; \beta, \rho_2)$, generating proofs π_k and π_α, π_β ;
- (e) calls a random oracle $\mathcal{H}(sid, \mathbb{G}, G, q, H, X, presX_{sid}, C_k, C_{kx}, X_A, C_\alpha, C_\beta, \pi_k, \pi_\alpha, \pi_\beta)$ and receives μ_k ; then computes:
- $ct_\gamma \cdot k = (ct_\gamma \cdot k_0) \oplus (\mu_k \odot ct_\gamma \cdot k_1)$;
 - $R'_B = (R_{B,0}) + (\mu_k \cdot R_{B,1})$;
- (f) samples $\eta_0, \eta_1 \leftarrow R_{pk}$, and computes:
- $ct_{\alpha,\beta} = \text{AHE.Eval}(pk, (ct_\gamma, ct_\gamma \cdot k), (0, \beta, \alpha); \eta_0)$;
 - $R_B = (\alpha \cdot R'_B) + (\beta \cdot G)$;
 - $R = k_A^{-1} \cdot R_B$ and $r = R_{x-axis}$;
 - $\alpha_1 = r \cdot k_A \cdot X_A + m \cdot k_A$ and $\alpha_2 = r \cdot k_A$;
 - $ct_A = \text{AHE.Eval}(pk, (ct_\gamma, ct_{\gamma \cdot key}), (0, \alpha_1, \alpha_2); \eta_1)$;
 - $C_1 = (r \odot C_{kx}) \oplus (m \odot C_k)$ and $C_2 = r \odot C_k$;
- (g) generate the following proofs:
- $\pi_{kx} \leftarrow \prod_{zk}^{L_{DComEq}[(G,H),(X_A,H)]}(C_k, C_{kx}; k_A, \rho_0, \rho_3)$;
 - $\pi_R \leftarrow \prod_{zk}^{L_{DComDL}[(G,H),(\mathbb{G},G,q)]}(C_k, R_B; k_A, \rho_0)$;
 - $\pi_{R_B} \leftarrow \prod_{zk}^{L_{VecDComDL}[(G,H),(\mathbb{G},(R'_B,G),q)]}(C_\alpha, C_\beta, R_B; (\alpha, \beta), \rho_1, \rho_2)$;
 - $\pi_{ct_{\alpha,\beta}} \leftarrow \prod_{zk}^{L_{DComEval}[G,H,pk,(ct_\gamma, ct_{\gamma \cdot k})(\mathbb{G},G,q)]}(ct_\gamma \alpha, \beta, (C_\beta, C_\alpha); (\beta, \alpha), \rho_0, \rho_1, \eta_0)$;
 - $\pi_{ct_A} \leftarrow \prod_{zk}^{L_{DComEval}[G,H,pk,(ct_\gamma, ct_{\gamma \cdot key})(\mathbb{G},G,q)]}(ct_A, (C_1, C_2); (\alpha_1, \alpha_2), \rho_3 \cdot r + \rho_0 \cdot m, r \cdot \rho_0, \eta_1)$;

(h) send $R, R_B, C_k, C_\alpha, C_\beta, C_{kx}, ct_A, ct_\alpha, \beta, \pi_k, \pi_\alpha, \pi_\beta, \pi_{kx}, \pi_R, \pi_{R_B}, \pi_{ct_{\alpha,\beta}}, \pi_{ct_A}$;

2. Each B_i does as follows:

Round 1

(a) call the random oracle $\mathcal{H}(msg)$ and receives m ;

(b) receive $R, R_B, C_k, C_\alpha, C_\beta, C_{kx}, ct_A, ct_\alpha, \beta, \pi_k, \pi_\alpha, \pi_\beta, \pi_{kx}, \pi_R, \pi_{R_B}, \pi_{ct_{\alpha,\beta}}, \pi_{ct_A}$;

(c) call the random oracle $\mathcal{H}(sid, msg, \mathbb{G}, G, qH, X, presX, sid, C_k, C_{kx}, X_A, C_\alpha, C_\beta, \pi_{kx}, \pi_\alpha, \pi_\beta)$ and compute:

$$i. \quad ct_{Y,k} = (ct_{Y,k_0}) \oplus (\mu_k \odot ct_{Y,k_1});$$

$$ii. \quad R'_B = (R_{B,0}) + (\mu_k \cdot R_{B,1});$$

$$iii. \quad C_1 = (r \odot C_{kx}) \oplus (m \odot C_k);$$

$$iv. \quad C_2 = r \odot C_k;$$

(h) verify the proofs; if fail abort and consider A_{pidA} malicious;

(i) send $(decrypt, pk, ct_A)$ and $(decrypt, pk, ct_\alpha, \beta)$ to $\mathcal{F}_{TAHE}$;

Broadcast Round:

(d) receive $(decrypt, pk, ct_A, pt_A)$ and $(decrypt, pk, ct_\alpha, \beta, pt_\alpha)$ from $\mathcal{F}_{TAHE}$;

(e) compute $s' = pt_\alpha^{-1} \cdot pt_A \bmod q$ and $s = \min\{s', q - s'\}$;

(f) send $(global-broadcast, sid, i, (r, s))$ to $\mathcal{F}_{global-broadcast}^{\Gamma_B}$;

4. Output:

- A_{pidA} receives $(global-broadcast, sid, i, (r, s))$ from $\mathcal{F}_{global-broadcast}^{\Gamma_B}$;
- A_{pidA} verifies that (r, s) is a valid ECDSA signature, if the verification fails it aborts and considers B malicious;
- Both A_{pidA} and B output (r, s) .

26. The method of claim 19, wherein the sign protocol is a sign protocol using a Schnorr scheme over an asynchronous communication channel ("Sign Protocol Schnorr/Async"), the Sign Protocol Schnorr/Async parameterized with: a group description $(\mathbb{G}, G, q)$, an access structure Γ_B , a unique session identifier (sid) , an encryption key to a TAHE scheme pk , a public verification key X , an encryption of the share of the distributed party of the private signing key ct_{key} , an output of a unique presign $pres_X = (K_{B,0}, K_{B,1})$, and a message to be signed (msg) , the Sign Protocol Schnorr/Async interacting with a centralized party A_{pidA} and a set of parties

$B = \{B_i\}_{i \in [n]}$, wherein the set of parties B can only send messages to parties in A through a functionality $\mathcal{F}_{global-broadcast}$, wherein each party B_i outputs a valid Schnorr signature (z, e) to the verification key X .

27. The method of claim 26, wherein:

1. A_{pid_A} does as follows:

- (a) samples $k_A \leftarrow \mathbb{Z}_q$ and computes $K_A = k_A \cdot G$;
- (b) calls the random oracle $\mathcal{H}(\text{sid}, X, pres_{X, \text{sid}}, K_A, \text{msg})$ and receives μ_k ;
- (c) computes $K = (K_{B,0}) + (\mu_k \cdot K_{B,1}) + K_A$;
- (d) calls the random oracle $\mathcal{H}(K, X, \text{msg})$ and receives e ;
- (e) computes $z_A = k_A + e \cdot x_A$;
- (f) sends $(\text{broadcast}, \text{sid}, (z_A, K_A))$ to $\mathcal{F}_{broadcast}$;

2. each B_i does as follows:

Round 1

- (a) receive $(\text{broadcast}, \text{sid}, (z_A, K_A))$ from $\mathcal{F}_{broadcast}$ and verify that $K_A \in \mathbb{G}$ else consider A_{pid_A} malicious and abort;
- (b) call the random oracle $\mathcal{H}(\text{sid}, X, pres_{X, \text{sid}}, K_A, \text{msg})$ and receives μ_k ;
- (c) B_i computes:
 - i. $ct_k = ct_{k_0} \oplus \mu_k \cdot ct_{k_1}$;
 - ii. $K = (K_{B,0}) + (\mu_k \cdot K_{B,1}) + K_A$;
- (d) calls the random oracle $\mathcal{H}(K, X, \text{msg})$ and receives e ;
- (e) check that $z_A G = K_A + e \cdot X_A$. If not, consider A_{pid_A} malicious and abort;
- (f) compute $ct_B = ct_k \oplus (e \cdot ct_{key}) \oplus z_A$;
- (g) send $(\text{decrypt}, pk, ct_B)$ to $\mathcal{F}_{TAHE}$;

Broadcast Round:

- (a) receive $(\text{decrypted}, pk, ct_B, pt_B)$ from $\mathcal{F}_{TAHE}$;
- (b) set $\sigma = (pt_B, e)$;
- (c) send $(\text{global-broadcast}, \text{sid}, i, \sigma)$ to $\mathcal{F}_{global-broadcast}$;

3. Output:

- A_{pid_A} receives $(\text{global-broadcast}, \text{sid}, i, \sigma)$ from $\mathcal{F}_{global-broadcast}$;

- A_{pidA} verifies that σ is a valid signature, else it aborts and considers B malicious;
 - both parties output σ .
28. The method of claim 1, wherein each DN party has a respective voting power, the method further comprising changing the voting power of at least one DN party using a network reconfiguration routine.
29. The method of claim 28, wherein the network reconfiguration routine comprises a reconfiguration protocol with a varying threshold (“Reconfiguration Protocol Varying Threshold”), the Reconfiguration Protocol Varying Threshold parameterized with a TAHE encryption scheme, a public verifiable secret sharing scheme (PVSS), a computational security parameter κ , a statistical security parameter σ , an initial threshold t and a final threshold t' , the Reconfiguration Protocol Varying Threshold interacting with a transferring quorum $B = \{B_i\}_{i \in [n]}$ and a receiving quorum $B' = \{B'_{i'}\}_{i' \in [n']}$, wherein the transferring quorum holds a t -out-of- n secret sharing $[sk]_i$ on a secret key of a TAHE (sk), wherein the transferring quorum and the receiving quorum hold a public key (pk) to the encryption scheme, verification keys of the transferring quorum $\{vk_i\}_{i \in [n]}$, and an encryption of the secret key $ct_{sk} = \text{Enc}_{pk}^b(sk; \eta)$, wherein the Reconfiguration Protocol Varying Threshold outputs a fresh t' -out-of- n' secret shares $[sk]_{i'}$ on the secret key (sk) held by B' and public verification keys $\{vk'_{i'}\}_{i' \in [n']}$ corresponding to the secret shares, wherein when the Reconfiguration Protocol Varying Threshold begins the parties already executed PVSS.Setup and PVSS.Gen successfully, and holds public keys for the new parties $\{pk_{i'}\}_{i' \in [n']}$.
30. The method of claim 19, wherein:
1. $B_i \in B$ does as follows:

Round 1:

 - (a) Sample a mask $r_i \leftarrow [-D(\mathcal{K}_K) \cdot 2^{\sigma\omega}, D(\mathcal{K}_K) \cdot 2^{\sigma\omega}]$.
 - (b) Calculate $ct_{mask}^i = \text{Enc}_{pk}^{b \cdot 2^\sigma}(r_i; \eta_i)$. In addition generate proof $\pi_{mask}^i \leftarrow \Pi_{zk}^{L_{sk}^{masked}[pk, b \cdot 2^\sigma]}(ct_{mask}^i; r_i; \eta_i)$.
 - (c) $\left(\{ct_{share}^{i, j'}\}_{j' \in [n]}, \{C_{i', j'}\}_{j' \in [n]}, \pi_{share}^{i'} \right) \leftarrow \text{PVSS.Dist}\left(\text{pp}, \{pk_{j'}\}_{j' \in [n]}, \text{Pack}_b\left(\left[[r_i]\right]_{b \cdot 2^\sigma}\right)\right)$.

- (d) Send $\left(\text{broadcast}, \text{sid}, i', \left(\text{ct}_{mask}^i, \pi_{mask}^i, \{ \text{ct}_{share}^{i,j'} \}_{j' \in [n]}, \{ C_{i,j'} \}_{j' \in [n]}, \pi_{share}^i \right) \right)$
to $\mathcal{F}_{broadcast}$.

Broadcast Round:

- (a) Receive
 $\left(\text{broadcast}, \text{sid}, j, \left(\text{ct}_{mask}^j, \pi_{mask}^j, \{ \text{ct}_{share}^{j,j'} \}_{j' \in [n]}, \{ C_{j,j'} \}_{j' \in [n]}, \pi_{share}^j \right) \right)$
from $\mathcal{F}_{broadcast}$.
- (b) Verify the proofs. If any of them fail consider j malicious. Else send
(validate, sid, i, j) to $\mathcal{F}_{ACS}^{([n]_{t+1})^B}$.

Round 3:

- (a) Receive S_B from $\mathcal{F}_{ACS}$ and set $\text{ct}_{masked-key} = \text{ct}_{sk} \oplus \sum_{j \in S_B} \text{ct}_{mask}^j$.
- (b) Set $(\text{ds}_i, \pi_{ds}^i) \leftarrow \text{TAHE.TDec}_{pk}(\text{ct}_{masked-key}, \text{vk}_i; [\text{sk}]_i)$ and send
 $\left(\text{broadcast}, \text{sid}, i, (\text{ds}_i, \pi_{ds}^i) \right)$ to $\mathcal{F}_{broadcast}$.

Broadcast Round:

- (a) Upon receiving $\left(\text{broadcast}, \text{sid}, j, (\text{ds}_j, \pi_{ds}^j) \right)$ verify that the proof π_{ds}^j is valid;
if it fails consider j malicious.
- (b) Collect a subset $S_B^i \in ([n]_{t+1})$ with validated proofs and calculate $\text{sk}_{masked} =$
 $\text{TAHE.Rec}_{pk}^b(\{ \text{ds}_j \}_{j \in S_B^i})$.
- (c) Send
 $\left(\text{global-broadcast}, \text{sid}, i, \left(\text{sk}_{masked}, \{ \{ \text{ct}_{share}^{i,j'} \}_{j' \in [n]}, \{ C_{j,j'} \}_{j' \in [n]} \}_{j \in S_B} \right) \right)$ to
 $\mathcal{F}_{global-broadcast}$.

2. $B'_{i'} \in B$ does as follows:

- (a) Upon receiving
 $\left(\text{global-broadcast}, \text{sid}, \left(\text{sk}_{masked}, \{ \{ \text{ct}_{share}^{i,j'} \}_{j' \in [n]}, \{ C_{j,j'} \}_{j' \in [n]} \}_{j \in S_B} \right) \right)$ from
 $\mathcal{F}_{global-broadcast}$, set $([r_j]_{i'}) \leftarrow \text{PVSS.DecShare}(\text{pp}, \text{pk}_{i'}, \text{sk}_{i'} \text{ct}_{share}^{i,i'})$.
- (b) Calculate $[sk]_{i'} = \Delta \text{sk}_{masked} - \sum_{j \in S_B} [r_j]_{i'}$.
- (c) Calculate $\text{vk}'_{j'} = \text{Com}_{pp}(\Delta \text{sk}_{masked}) - \sum_{j \in S_B} C_{j',j}$.
- (d) Record $([sk]_{i'}, \{ \text{vk}_{j'} \}_{j' \in [n]})$.

31. The method of claim 28, wherein the network reconfiguration routine comprises a reconfiguration protocol with a constant threshold (“Reconfiguration Protocol Constant Threshold”), the Reconfiguration Protocol Constant Threshold parameterized with a TAHE encryption scheme, a public verifiable secret sharing scheme PVSS, a computational security parameter κ , and a threshold t , the Reconfiguration Protocol Constant Threshold interacting with a transferring quorum $B = \{B_i\}_{i \in [n]}$ and a receiving quorum $B' = \{B'_{i'}\}_{i' \in [n']}$, wherein the transferring quorum holds a t -out-of- n secret sharing $[sk]_i$ on a secret key of a TAHE sk , wherein the transferring quorum and the receiving quorum hold a public key (pk) to the encryption scheme, and verification keys of the transferring quorum $\{vk_i\}_{i \in [n]}$, wherein the Reconfiguration Protocol Constant Threshold outputs a fresh t -out-of- n secret shares $[sk]_{i'}$ on the secret key sk held by B' and public verification keys $\{vk'_{i'}\}_{i' \in [n']}$ corresponding to the secret shares, wherein when the Reconfiguration Protocol Constant Threshold begins the parties already executed PVSS.Setup and PVSS.Gen successfully, and holds appropriate public keys for the new parties $\{pk_{i'}\}_{i' \in [n']}$.

32. The method of claim 31, wherein $B_i \in B$ does as follows:

Round 1:

- (a) Samples $r_i \leftarrow \mathcal{K}_\kappa$.
- (b) $(\{ct_{Share,old}^{i,j}\}_{j \in [n]}, \{C_{i,j}\}_{j \in [n]}, \pi_{Share,old}^i) \leftarrow \text{PVSS.Dist}(\text{pp}, \{pk_j\}_{j \in [n]}; r_i)$.
- (c) $(\{ct_{Share,new}^{i,j'}\}_{j' \in [n]}, \{C_{i,j'}\}_{j' \in [n]}, \pi_{Share,new}^i) \leftarrow \text{PVSS.Dist}(\text{pp}, \{pk_{j'}\}_{j' \in [n]}; r_i)$.
- (d) Send $\left(\text{broadcast}, \text{sid}, i, \left(\begin{array}{l} \{ct_{Share,old}^{i,j}\}_{j \in [n]}, \{C_{i,j}\}_{j \in [n]}, \pi_{Share,old}^i \\ \{ct_{Share,new}^{i,j'}\}_{j' \in [n]}, \{C_{i,j'}\}_{j' \in [n]}, \pi_{Share,new}^i \end{array} \right) \right)$ to $\mathcal{F}_{broadcast}$.

Round 2:

- (c) Upon receiving $\left(\text{broadcast}, \text{sid}, j, \left(\begin{array}{l} \{ct_{Share,old}^{j,j^*}\}_{j^* \in [n]}, \{C_{j,j^*}\}_{j^* \in [n]}, \pi_{Share,old}^j \\ \{ct_{Share,new}^{j,j'}\}_{j' \in [n]}, \{C_{j,j'}\}_{j' \in [n]}, \pi_{Share,new}^j \end{array} \right) \right)$ from $\mathcal{F}_{broadcast}$.

- (d) Verify the proofs and check the parameters of $\pi_{share,old}^j$ and $\pi_{share,new}^j$ contain the same public commitment to the point at zero. If any of the checks fail consider j malicious, else send (validate, sid,pid) to $\mathcal{F}_{ACS}^{([n]_{t+1})^B}$.

Round 3:

- (e) Receive (authorized-subset,sid, S_B). For every $j \in S_B$ set $r_{j,i} = \text{PVSS.DecShare}(\text{pp}, \text{pk}_i, \text{sk}_i, \text{ct}_{share,old}^{j,i})$.
- (f) Set $[\overline{sk}]_i = [\text{sk}]_i + \sum_{j \in S_B} r_{j,i}$.
- (g) $(\{\text{ct}_{share}^{i,j'}\}_{j' \in [n]}, \{C'_{i,j'}\}_{j' \in [n]}, \pi_{share}^i) \leftarrow \text{PVSS.Dist}(\text{pp}, \{\text{pk}_{j'}\}_{j' \in [n]}, [\overline{sk}]_i)$.
- (h) Send (broadcast, sid, $i, (\{\text{ct}_{share}^{i,j'}\}_{j' \in [n]}, \{C'_{i,j'}\}_{j' \in [n]}, \pi_{share}^i)$) to $\mathcal{F}_{broadcast}$.

Round 4:

- (c) Receive (broadcast,sid, $j, (\{\text{ct}_{share}^{j,j'}\}_{j' \in [n]}, \{C'_{j,j'}\}_{j' \in [n]}, \pi_{share}^j)$).
- (d) Verify the proofs $\pi_{share}^{j'}$ and check that the commitment in the public parameters of $\pi_{share}^{j'}$ is equal to $\text{vk}_j + \sum_{j^* \in S_B} C'_{j^*,j}$. If any of the checks fail consider j malicious, else send (validate, sid, i) to $\mathcal{F}_{ACS}^{([n]_{t+1})^B}$.

Broadcast Round:

- (c) Receive (authorized-subset,sid, $\bar{S}_B$) from $\mathcal{F}_{ACS}$.
- (d) Send
$$\left(\text{global-broadcast,sid, } i, \left(\left\{ \left\{ \text{ct}_{share}^{j,j'}, C'_{j,j'} \right\}_{j' \in [n]} \right\}_{j \in \bar{S}_B}, \left\{ \left\{ \text{ct}_{share,new}^{j^*,j'}, C'_{j^*,j'} \right\}_{j' \in [n]} \right\}_{j \in S_B} \right) \right) \text{ to } \mathcal{F}_{global-broadcast}.$$

2. $B_i \in B$ does as follows:

- (g) Receive
$$\left(\text{global-broadcast,sid, } i, \left(\left\{ \left\{ \text{ct}_{share}^{j,j'}, C'_{j,j'} \right\}_{j' \in [n]} \right\}_{j \in \bar{S}_B}, \left\{ \left\{ \text{ct}_{share,new}^{j^*,j'}, C'_{j^*,j'} \right\}_{j' \in [n]} \right\}_{j \in S_B} \right) \right)$$
- (h) For every $j^* \in S_B$, set $r'_{j^*,i'} = \text{PVSS.DecShare}(\text{pp}, \text{pk}_{i'}, \text{sk}_{i'}, \text{ct}_{share,new}^{j^*,i'})$.
- (i) For every $j \in \bar{S}_B$, set $\bar{r}_{j,i'} = \text{PVSS.DecShare}(\text{pp}, \text{pk}_{i'}, \text{sk}_{i'}, \text{ct}_{share}^{j,i'})$.
- (j) Set $[\text{sk}]'_{i'} = \frac{1}{\Delta^3} \sum_{j^* \in S_B} \sum_{j \in \bar{S}_B} \Delta \lambda_{i',j^*}^{S_B} \Delta \lambda_{0,j}^{\bar{S}_B} \bar{r}_{j,j^*} - \sum_{j^* \in S_B} r'_{j^*,i'}$.

(k) Set $vk'_{j'} = \sum_{j^* \in S_B} \sum_{j \in S_B} \Delta \lambda_{j',j^*}^{S_B} \Delta \lambda_{0,j}^{S_B} C'_{j,j^*} - \sum_{j^* \in S_B} C_{j^*,j'}$.

(l) Record $([sk]_{i'}, \{vk_{j'}\}_{j' \in [n]})$.

33. The method of claim 1, wherein a plurality of clients and the threshold of parties of the decentralized network cooperate to generate the digital signature, the method further comprising transferring ownership of a resource from a sender client of the plurality of clients to a receiver client of the plurality of clients using an ownership transfer protocol of the one or more secure protocols.
34. The method of claim 33, wherein the ownership transfer protocol is parameterized with a group description $(\mathbb{G}, G, q)$, and interacts with parties A_{old} , A_{new} and $\{B_i\}_{i \in S}$, where A_{old} transfers its secret key to A_{new} and $\{B_i\}_{i \in S}$ approves the transfer, wherein all parties hold: a public key (pk) to the encryption scheme; a public key of A_{new} ($pk_{A_{new}}$); a public share of A_{old} of a signing key ($X_{A_{old}}$); a public verification key (X) as input, and wherein the parties agree on a fresh session identifier (sid).
35. The method of claim 34, wherein the parties further perform the following:
1. A_{old} does as follows:
 - (a) Samples $\eta \leftarrow \mathcal{R}_{pk_{A_{new}}}$ and sets $ct_{new} = \text{Enc}_{pk_{A_{new}}}(x_{old}; \eta)$.
 - (b) Sets $\pi_{transfer} \leftarrow \prod_{zk}^{L_{\text{EncDL}}[pk_{A_{new}}(\mathbb{G}, G, q)]}(X_{A_{old}}, ct_{transfer}; x_{old}; \eta)$.
 - (c) Sends $(\text{broadcast}, sid, pid_{A_{old}}, (ct_{transfer}, \pi_{transfer}))$ to $\mathcal{F}_{broadcast}$.
 2. $B_i \in B$ does as follows:
 - (a) Receives $(\text{broadcast}, sid, pid_{A_{old}}, (ct_{transfer}, \pi_{transfer}))$ from $\mathcal{F}_{broadcast}$ and verifies $\pi_{transfer}$; if the proof fails consider A_{old} malicious; otherwise send $(\text{global-broadcast}, sid, i, ct_{transfer})$ to $\mathcal{F}_{global-broadcast}$.
 - (b) Records $(X_{A_{old}}, pid_{A_{new}})$.
 3. A_{new} does as follows:
 - (a) Receives $(\text{global-broadcast}, sid, ct_{transfer})$ from $\mathcal{F}_{global-broadcast}$, sets $x_{new} = \text{Dec}_{pk_{A_{new}}}(ct_{transfer}; sk_{A_{new}})$, and records $(X_{A_{old}}, x_{new})$.

36. The method of any one of claims 1 to 35, wherein at least one secure protocol of the one or more secure protocols comprises an identifiable abort.
37. The method of any one of claims 1 to 35, wherein at least one secure protocol of the one or more secure protocols is publicly verifiable.
38. The method of any one of claims 1 to 35, wherein at least one secure protocol of the one or more secure protocols is performed over an asynchronous communication channel and comprises guaranteed output delivery.
39. The method of any one of claims 1 to 35, further comprising:
performing a generic transformation to securely transform a plaintext space of the TAHE, to align the plaintext space with an ECDSA/Schnorr group of order q .
40. The method of claim 1, further comprising a secure function evaluation for the TAHE, allowing a user to evaluate a private affine function f homomorphically on a tuple of ciphertexts $ct_i = Enc(w_i, r_i)$ without revealing information on function f except for $f(x_1, \dots, x_n)$, even given a secret key of the TAHE and randomizers r_i and plaintexts x_i for the encryptions ct_i .
41. The method of claim 1, wherein the TAHE is tied with elliptic curve group elements, such that corresponding witnesses thereof satisfy linear relations and have range constraints, using at least one generic zero-knowledge (zk) range proof.
42. The method of claim 1, wherein the parties of the decentralized network hold a single secret share, for an underlying TAHE scheme, and a network share of each secret signing key is stored publicly encrypted under a corresponding TAHE public key.
43. The method of any one of claims 10 to 18, wherein the presign protocol is configured to generate at least one presign offline in a common pool of presigns shared with all clients in the network, such that upon signing the clients could use the common pool of presigns to sign transactions.
44. The method of any one of claims 1 to 43, further comprising:
performing at least one performance boosting procedure selected from the group consisting of:
(i) batching zero knowledge proofs;

- (ii) redundant range proofs when using Class-Groups;
 - (iii) amortized decryption; and
 - (iv) synchronous decryption.
45. The method as in any one of claims 1 to 44, wherein the decentralized network is selected from the group consisting of: a blockchain system; a directed acyclic graphs DAG-based consensus platform; and a blockchain-DAG hybrid.
46. The method as in any one of claims 1 to 45, wherein the digital signature is used for at least one application selected the group consisting of: digital wallets; future transactions; bridging blockchains; wallet transfer; and decentralized autonomous organization (DAO).
47. The method of claim 46, wherein a public key of a digital wallet generated on a first blockchain is configured to be used as a digital wallet on a second blockchain.
48. The method of claim 47, wherein a policy is specified upon generation of the digital wallet on the first blockchain, and wherein the first blockchain enforces the policy when using the digital wallet for the second blockchain participating in signing requests only on transactions valid with the specified policy.
49. The method of claim 46, wherein a future transaction comprises a signed transaction of a client configured to be executed when a predetermined condition is met, wherein the blockchain is configured to delay execution of the transaction.
50. The method of claim 46, wherein a decentralized autonomous organization (DAO) comprises at least one child key derived from a parent key, wherein the child key comprises fewer permissions than the parent key.
51. A system for generating a digital signature, comprising:
a decentralized network (DN) comprising a plurality of parties, configured to follow one or more secure protocols based on a threshold additively homomorphic encryption (TAHE) over a broadcast channel,

wherein a threshold of the plurality of parties of the DN produces the digital signature by following a Universally Composable (UC)-secure protocol configured to output at least one of an Elliptic curve digital signature algorithm (ECDSA) signature or a Schnorr signature.

52. The system of claim 51, wherein at least one client and the threshold of parties of the decentralized network cooperate to generate the digital signature.
53. The system of claim 51, wherein the threshold of parties of the decentralized network cooperate to generate the digital signature without a client.
54. The system of claim 51, comprising a network distributed key generation (DKG) module, configured to generate a distributed key using a distributed key generation (DKG) protocol of the one or more secure protocols.
55. The system of claim 54, wherein the DKG protocol is a DKG protocol over a synchronous communication channel (“DKG Protocol Sync”), the DKG Protocol Sync being parameterized with a group description $(\mathbb{G}, G, q)$ and interacting with $n + 1$ parties including a client A , and DN parties $B = \{B_i\}_{i \in [n]}$, wherein all parties hold a public key (pk) as input and agree on a fresh session-identifier (sid) that uniquely identifies a session.
56. The system of claim 54, wherein DKG protocol is a DKG protocol over an asynchronous communication channel (“DKG Protocol Async”), the DKG Protocol Async being parameterized with a group description $(\mathbb{G}, G, q)$, an access structure Γ_B , a unique session identifier sid , and an encryption key to a TAHE scheme pk , wherein the DKG Protocol Async interacts with a centralized party A_{pidA} and a set of parties $B = \{B_i\}_{i \in [n]}$, wherein the set of parties B can only send messages to parties in A through a functionality $\mathcal{F}_{global-broadcast}$.
57. The system of claim 52, configured to distribute, by a client that holds a private signing key, the private signing key between the client and the decentralized network.
58. The system of claim 51, comprising a network ECDSA/Schnorr presign module, configured for pre-signing using a presign protocol of the one or more secure protocols.

59. The system of claim 58, wherein the presign protocol is a presign protocol using an ECDSA scheme over a synchronous communication channel (“Presign Protocol ECDSA/Sync”), the Presign Protocol ECDSA/Sync interacting with $n + 1$ parties including a client A , and DN parties $B = \{B_i\}_{i \in [n]}$, wherein all parties hold a public key (pk) and an encryption of a network share under TAHE (ct_{key}) as input, wherein all parties verify that a session-identifier sid has never been used before.
60. The system of claim 58, wherein the presign protocol is a presign protocol using a Schnorr scheme over a synchronous communication channel (“Presign Protocol Schnorr/Sync”), the Presign Protocol Schnorr/Sync interacting with $n + 1$ parties including a client A , and DN parties $B = \{B_i\}_{i \in [n]}$, wherein all parties have an encryption of a network share under TAHE (ct_{key}) as input, wherein all parties verify that a session-identifier (sid) has never been used before.
61. The system of claim 58, wherein the presign protocol is a presign protocol using an ECDSA scheme over an asynchronous communication channel (“Presign Protocol ECDSA/Async”), the Presign Protocol ECDSA/Async parameterized with: a group description $(\mathbb{G}, G, q)$, an access structure Γ_B , a unique session identifier sid , an encryption key to a TAHE scheme pk , an ECDSA verification key X , a centralized party share of the verification key X_A , and an encryption of a share of a distributed party of a signing key ct_{key} , the Presign Protocol ECDSA/Async interacting with a centralized party A_{pidA} and a set of parties $B = \{B_i\}_{i \in [n]}$, wherein the set of parties B can only send messages to parties in A through a functionality $\mathcal{F}_{global-broadcast}$.
62. The system of claim 58, wherein the presign protocol is a presign protocol using a Schnorr scheme over an asynchronous communication channel (“Presign Protocol Schnorr/Async”), the Presign Protocol Schnorr/Async parameterized with: a group description $(\mathbb{G}, G, q)$, an access structure Γ_B , a unique session identifier sid , and an encryption key to a TAHE scheme pk , the Presign Protocol Schnorr/Async interacting with a centralized party A_{pidA} and a set of parties $B = \{B_i\}_{i \in [n]}$, wherein the set of parties B can only send messages to parties in A through a functionality $\mathcal{F}_{global-broadcast}$.
63. The system of claim 51, comprising a network ECDSA/Schnorr sign module, configured for signing a message using a sign protocol of the one or more secure protocols.

64. The system of claim 63, wherein the sign protocol is a sign protocol using an ECDSA scheme over a synchronous communication channel (“Sign Protocol ECDSA/Sync”), the Sign Protocol ECDSA/Sync interacting with $n + 1$ parties including a client A , and DN parties $B = \{B_i\}_{i \in [n]}$, wherein all parties hold a public key (pk), network public nonce shares (R_B^0, R_B^1), and a commitment on A ’s share of the nonce for signature (K_A), as input, wherein all parties verify that a session-identifier (sid) has never been used before.
65. The system of claim 64, wherein the sign protocol is a sign protocol using a Schnorr scheme over a synchronous communication channel (“Sign Protocol Schnorr/Sync”), the Sign Protocol Schnorr/Sync interacting with $n + 1$ parties including a client A , and DN parties $B = \{B_i\}_{i \in [n]}$, wherein all parties have a public key (pk), network public nonce shares (R_B^0, R_B^1), a commitment on A ’s share of the nonce for signature (K_A), an encryption of a network share (ct_{key}), and encryptions of nonces k_0, k_1 (ct_{k_0}, ct_{k_1}) as input, wherein all parties verify that a session-identifier (sid) has never been used before.
66. The system of claim 64, wherein the sign protocol is a sign protocol using an ECDSA scheme over an asynchronous communication channel (“Sign Protocol ECDSA/Async”), the Sign Protocol ECDSA/Async parameterized with a group description $(\mathbb{G}, G, q)$, a public generator (H) used for Pedersen commitments, an access structure Γ_B , a unique session identifier (sid), an encryption key to a TAHE scheme pk , an ECDSA verification key X , a centralized party share of the verification key X_A , an encryption of a share of a distributed party of a signing key ct_{key} , an output of a uniquely determined previous presign protocol $presX, sid = (ct_\gamma, ct_{\gamma \cdot key}, ct_{\gamma \cdot k_0}, ct_{\gamma \cdot k_1}, R_{B,0}, R_{B,1})$, and a message to be signed (msg), the Sign Protocol ECDSA/Async interacting with a centralized party A_{pidA} and a set of parties $B = \{B_i\}_{i \in [n]}$, wherein the set of parties B can only send messages to parties in A through a functionality $\mathcal{F}_{global-broadcast}$, wherein the parties output a valid ECDSA signature $\sigma = (r, s)$ with verification key X .
67. The system of claim 64, wherein the sign protocol is a sign protocol using a Schnorr scheme over an asynchronous communication channel (“Sign Protocol Schnorr/Async”), the Sign Protocol Schnorr/Async parameterized with: a group description $(\mathbb{G}, G, q)$, an access structure Γ_B , a unique session identifier (sid), an encryption key to a TAHE scheme pk , a public

verification key X , an encryption of the share of the distributed party of the private signing key ct_{key} , an output of a unique presign $pres_X = (K_{B,0}, K_{B,1})$, and a message to be signed (msg), the Sign Protocol Schnorr/Async interacting with a centralized party A_{pidA} and a set of parties $B = \{B_i\}_{i \in [n]}$, wherein the set of parties B can only send messages to parties in A through a functionality $\mathcal{F}_{global-broadcast}$, wherein each party B_i outputs a valid Schnorr signature (z, e) to the verification key X .

68. The system of claim 51, wherein each DN party has a respective voting power, the system comprising a network reconfiguration module, configured for changing the voting power of at least one DN party using a network reconfiguration routine.
69. The system of claim 68, wherein the network reconfiguration routine comprises a reconfiguration protocol with a varying threshold ("Reconfiguration Protocol Varying Threshold"), the Reconfiguration Protocol Varying Threshold parameterized with a TAHE encryption scheme, a public verifiable secret sharing scheme (PVSS), a computational security parameter κ , a statistical security parameter σ , an initial threshold t and a final threshold t' , the Reconfiguration Protocol Varying Threshold interacting with a transferring quorum $B = \{B_i\}_{i \in [n]}$ and a receiving quorum $B' = \{B'_i\}_{i \in [n']}$, wherein the transferring quorum holds a t -out-of- n secret sharing $[sk]_i$ on a secret key of a TAHE (sk), wherein the transferring quorum and the receiving quorum hold a public key (pk) to the encryption scheme, verification keys of the transferring quorum $\{vk_i\}_{i \in [n]}$, and an encryption of the secret key $ct_{sk} = \text{Enc}_{pk}^b(sk; \eta)$, wherein the Reconfiguration Protocol Varying Threshold outputs a fresh t' -out-of- n' secret shares $[sk]_{i'}$ on the secret key (sk) held by B' and public verification keys $\{vk'_{i'}\}_{i' \in [n']}$ corresponding to the secret shares, wherein when the Reconfiguration Protocol Varying Threshold begins the parties already executed PVSS.Setup and PVSS.Gen successfully, and holds public keys for the new parties $\{pk_{i'}\}_{i' \in [n']}$.
70. The system of claim 68, wherein the network reconfiguration routine comprises a reconfiguration protocol with a constant threshold ("Reconfiguration Protocol Constant Threshold"), the Reconfiguration Protocol Constant Threshold parameterized with a TAHE encryption scheme, a public verifiable secret sharing scheme PVSS, a computational security parameter κ , and a threshold t , the Reconfiguration Protocol Constant Threshold interacting with

a transferring quorum $B = \{B_i\}_{i \in [n]}$ and a receiving quorum $B' = \{B'_{i'}\}_{i' \in [n']}$, wherein the transferring quorum holds a t -out-of- n secret sharing $[sk]_i$ on a secret key of a TAHE sk , wherein the transferring quorum and the receiving quorum hold a public key (pk) to the encryption scheme, and verification keys of the transferring quorum $\{vk_i\}_{i \in [n]}$, wherein the Reconfiguration Protocol Constant Threshold outputs a fresh t -out-of- n secret shares $[sk]_{i'}$ on the secret key sk held by B' and public verification keys $\{vk'_{i'}\}_{i' \in [n']}$ corresponding to the secret shares, wherein when the Reconfiguration Protocol Constant Threshold begins the parties already executed PVSS.Setup and PVSS.Gen successfully, and holds appropriate public keys for the new parties $\{pk_{i'}\}_{i' \in [n']}$.

71. The system of claim 51, wherein a plurality of clients and the threshold of parties of the decentralized network cooperate to generate the digital signature, the system comprising an ownership transfer verification module, configured for transferring ownership of a resource from a sender client of the plurality of clients to a receiver client of the plurality of clients using an ownership transfer protocol of the one or more secure protocols.
72. The system of claim 71, wherein the ownership transfer protocol is parameterized with a group description $(\mathbb{G}, G, q)$, and interacts with parties A_{old} , A_{new} and $\{B_i\}_{i \in \mathcal{S}}$, where A_{old} transfers its secret key to A_{new} and $\{B_i\}_{i \in \mathcal{S}}$ approves the transfer, wherein all parties hold: a public key (pk) to the encryption scheme; a public key of A_{new} ($pk_{A_{new}}$); a public share of A_{old} of a signing key ($X_{A_{old}}$); a public verification key (X) as input, and wherein the parties agree on a fresh session identifier (sid).
73. The system of any one of claims 51 to 72, wherein at least one secure protocol of the one or more secure protocols comprises an identifiable abort.
74. The system of any one of claims 51 to 72, wherein at least one secure protocol of the one or more secure protocols is publicly verifiable.
75. The system of any one of claims 51 to 72, where at least one secure protocol of the one or more secure protocols is performed over an asynchronous communication channel and comprises guaranteed output delivery.

76. The system of any one of claims 51 to 72, configured for performing a generic transformation to securely transform a plaintext space of the TAHE, to align the plaintext space with an ECDSA/Schnorr group of order q .
77. The system of claim 51, configured for performing a secure function evaluation for the TAHE, allowing a user to evaluate a private affine function f homomorphically on a tuple of ciphertexts $ct_i = Enc(w_i, r_i)$ without revealing information on function f except for $f(x_1, \dots, x_n)$, even given a secret key of the TAHE and randomizers r_i and plaintexts x_i for the encryptions ct_i .
78. The system of claim 51, wherein the TAHE is tied with elliptic curve group elements, such that corresponding witnesses thereof satisfy linear relations and have range constraints, using at least one generic zero-knowledge (zk) range proof.
79. The system of claim 51, wherein the parties of the decentralized network hold a single secret share, for an underlying TAHE scheme, and a network share of each secret signing key is stored publicly encrypted under a corresponding TAHE public key.
80. The system of claim 58, wherein the presign protocol is configured to generate at least one presign offline in a common pool of presigns shared with all clients in the network, such that upon signing the clients could use the common pool of presigns to sign transactions.
81. The system of any one of claims 51 to 80, configured for performing at least one performance boosting procedure selected from the group consisting of:
 - (i) batching zero knowledge proofs;
 - (ii) redundant range proofs when using Class-Groups;
 - (iii) amortized decryption; and
 - (iv) synchronous decryption.
82. The system as in any one of claims 51 to 81, wherein the decentralized network is selected from the group consisting of: a blockchain system; a directed acyclic graphs DAG-based consensus platform; and a blockchain-DAG hybrid.

83. The system as in any one of claims 51 to 82, wherein the digital signature is used for at least one application selected the group consisting of: digital wallets; future transactions; bridging blockchains; wallet transfer; and decentralized autonomous organization (DAO).
84. The system of claim 83, wherein a public key of a digital wallet generated on a first blockchain is configured to be used as a digital wallet on a second blockchain.
85. The system of claim 84, wherein a policy is specified upon generation of the digital wallet on the first blockchain, and wherein the first blockchain enforces the policy when using the digital wallet for the second blockchain participating in signing requests only on transactions valid with the specified policy.
86. The system of claim 85, wherein a future transaction comprises a signed transaction of a client configured to be executed when a predetermined condition is met, wherein the blockchain is configured to delay execution of the transaction.
87. The system of claim 83, wherein a decentralized autonomous organization (DAO) comprises at least one child key derived from a parent key, wherein the child key comprises fewer permissions than the parent key.