Human Network: A Decentralized Threshold Network with forward secrecy

Human Tech by Holonym Foundation Team

1 Introduction

Personhood is critical infrastructure in society. To access social benefits from a government or travel internationally, you need to prove you are a real person and that you are a specific person.

Yet personhood is under threat from emerging technologies such as AI, surveillance, and cyberattacks. Deepfakes have made it possible to impersonate anyone. It is customary to share government IDs with dozens of websites, hoping none experience a data breach. We trust our sensitive biometrics with centralized companies, hoping they will not mishandle our data or encounter a data breach. We identify ourselves on the web with passwords, yet these are easily stolen or cracked.

Human Network is a threshold network for scalar by elliptic curve point multiplication secured on EigenLayer [Eig23] and Symbiotic [Sym24]. It is a network that allows a group of nodes to perform a scalar by elliptic curve point multiplication, unlocking a new set of decentralized applications not previously feasible that require high privacy and security.

About Human Network

Threshold multiplication allows for two primitives critical to identity: decentralized verifiable oblivious pseudorandom function (VOPRF), and decentralized decryption over ZK-friendly elliptic curves. It can be done asynchronously without communication between nodes, i.e. it is effectively infinitely scalable for practical purposes.

Verifiable Oblivious Pseudorandom Function (VOPRF)

The threshold VOPRF enables high-entropy nullifiers and private keys from low-entropy data. These open up a design space for zero-knowledge privacy-preserving identity and easy-onboarding wallets:

- assigning nullifiers to people based on data that is not cryptographically secure, such as name, date of birth, and address.
- creating private key(share)s from passwords or biometrics with low entropy

- creating wallets that can be recovered by memorable data which inherently has low entropy
- JWTPRF, for permissionlessly and efficiently deriving keys from web accounts.

Human network assists in performing the following pseudorandom function over client input x and a distributed secret k:

$$H(h(x) * k)$$

where h hashes a scalar to a point on an elliptic curve and H hashes a point on the curve to a scalar. To do so obliviously, the network users of the network may apply information-theoretic masks of their inputs using the 2HashDH [JKK14] protocol. In the threshold setting, only the multiplication by k must be distributed.

Provable Encryption On ZK-Friendly Elliptic Curves

ElGamal decryption [Gam84] can be constructed from scalar by curve point multiplication. This allows for compliance with ZK via provable decentralized access control. To satisfy regulatory requirements, companies must hold sensitive data of their users and often inadvertently leak this data to the dark web. Human Network allows for a new model where regular users have privacy and companies can maintain compliance. Threshold ELGamal decryption over BabyJubJub allows users to prove they have encrypted data to the network which only lets a regulator decrypt it when rules are met. This allows for compliance while removing the risks in the current status quo of carte-blanche surveillance. Normal users maintain privacy whereas only the data of suspected criminals is subjected to the current status quo of being seen in plaintext.

In ElGamal decryption, the decryption for a ciphertext of elliptic curve points (C_1, C_2) is $C_2 - C_1 * k$ where k is a scalar representing the private key. In the threshold setting, the multiplication by k needs to be distributed.

2 Preliminaries

2.1 Notation and Writing Conventions

- n-quorum: group of n nodes selected/elected to share the secret after having perform a DKG or resharing protocol.
- t-quorum: group of t nodes selected from a n-quorum to perform the transaction (multiplication, signing...).
- epoch e: period of time during which a n-quorum is selected to share a secret.
- Q_e: selected n-quorum for epoch e.

2.2 Network Assumptions

A node is represented by a unique nonzero identifier. A correspondence between node ids and their public keys is known by the whole network, e.g. a node's id might be a cryptographic hash of its public key. In the case of Human Network, each node can retrieve the RSA public key of other nodes by querying the Human Network Smart Contract. When being part of a n-quorum, a node should also be defined by an index used in the protocols and therefore determined before starting the *DKG* or *resharing* protocol.

We assume a broadcast chanel where any node can see all the messages transmitted.

The content of some of the messages are encrypted so that only the target node can read it but everybody can see it.

We assume an adversary accessing the broadcast chanel as well. It can record all messages and examine them later. We also consider an asynchronous model of communication, meaning that even if nodes are synchronized from one epoch to the next, messages can be arbitrarily delayed by an adversary within an epoch.

2.3 Security Assumptions

Proactive security along with mobile adversary have been introduced [OY91]. This type of adversary is able to corrupt up to a threshold amount of nodes at a given time. For each epoch, an adversary can choose a new subset of t among n nodes to corrupt. In addition, if the adversary is considered as active, it is allowed to control the corrupt nodes' behavior.

An adversary can corrupt up to $t-1$ nodes from the n-quorum Q_e and until the end of that epoch e.

We assume that corrupted nodes remain corrupted from one epoch to the other.

We also assume that an adversary is free to attack a node after it has left an epoch and if it can recover its decryption key, it will be able to decrypt the old messages and thus learn secret information. This could be prevent using a technique named *forward-secure encryption* [CHK03]. We discuss Section 4.3 how we applied such an approach to Human Network.

Indeed, in almost all the resharing protocol from the literature, it is stated that a node should delete its old secret share as soon as the resharing process is complete and verified. Such a step is obviously quite important but not sufficient. If an attacker records all the data transmitted in the public broadcast chanel and corrupts a particular node at a latter epoch, it won't find its former secret share from a previous epoch but using its decryption keys and previously stored data will be able to recover it.

3 Protocols

3.1 Side Protocols

***n*-quorum selection** The n-quorum should be formed at 2 different steps, the *DKG* and the *resharing*. For both protocols we assume that the group of all possible nodes forms what we call the universe U. At epoch e, from that set U, a smaller set Q_e should be selected. We first apply a bunch of rules to U that will limit the candidate to form Q_e. One rule could be, by example that a candidate node should be staking a minimal amount of a certain token. Once the candidates set has been reduced, a randomized selection process should determine the resulting n-quorum Q_e.

***t*-quorum selection** To not alter the inherent security of a threshold configuration and to make the accountability for nodes retribution simpler, this selection of t-quorum might be uniformly distributed. Also, a t-quorum is actually composed of $t+\epsilon$ nodes in reality. We decide to increase its cardinality by ϵ nodes to proactively overcome the case where ϵ nodes are not responding to the request or delaying too much their response. We note that the higher ϵ is, the better failing nodes are tolerated but the more costly the distributed protocol becomes.

Index assignement When a node is joining the network and is part of U, it should be given a nonzero unique index. It will keep this index from one epoch to the other. It is essential that nodes agree on the same list of indexes while performing any kind of protocols. It then implies that all the nodes are synchronized to use the same input values.

Secret Sharing In the widely used protocol of Shamir Secret Sharing (SSS) [Sha79], the secret s is converted into a polynomial of degree $t-1$ with random coefficients where the free-term is set to $P(0) = s$. Each node receives a share corresponding to $P(i)$ where i is the node's unique and nonzero idex. Then, with at least t points from P and using Lagrange interpolation, it is possible to retrieve the secret polynomial, and therefore retrieve secret s. But on the contrary, from a set of less than t points (with none at input 0), one will gain no information on the secret.
Feldman introduced Verifiable Secret Sharing (VSS) [Fel87], a manner of enabling the nodes to verify whether or not their shares are correct. Concretely, in addition to compute the evaluation of its polynomial at all nodes' id, the node will also compute and share commitments of its polynomial's coefficients. These commitments should be publicly distributed such that any node can verify its share but also check that the other nodes are using the same distributed commitments for verification.

Lagrange Interpolation This technique allows one to recreate the original polynomial from the secret sharing protocol from at least t shares and the set of respective nodes' indexes. Before performing the final summation, each node i should multiply its share to a coefficient computed as follow: $\lambda_i = (\prod_j j)/(\prod_j j - i)$ for all j in the indexes' set except its own index i.
Finally, we can retrieve the original secret as: $s = \sum \lambda_i k_i$

DLEQ Proof These proofs are non-interactive zero-knowledge (NIZK) proofs for the equality (EQ) of discrete logarithms (DL). Used to prove that you know a scalar x for two public values xA and xB. It generates a private nonce and computes a challenge generated using the Fiat-Shamir heuristic [FS87]. This specific approach is a widely used technique to allow the DLEQ proof construction to be non-interactive. In our VOPRF use-case (Section 1) the node should produce such a proof attached to the scalar multiplication to prove its knowledge of the secret scalar. In algorithm 1 we detail the proof generation for input values M and x with n the order if the *secp256k1* curve to prove that $Y = xG$ and $Z = xM$ have the same privet key x. In algorithm 2 we present how the DLEQ proof is verified by any third party. We precise that function H could be any secure hash function as e.g. *SHA512*.

Algorithm 1: DLEQ Proof Generation

Input : $((g, n), M, x)$
Output: (M, Y, Z, A, B, c, s)

Samples random $r \leftarrow \mathbb{Z}_n$
Computes $A \leftarrow g^r$
Computes $B \leftarrow M^r$
Computes $Y \leftarrow q^x$
Computes $Z \leftarrow M^x$
Computes challenge $c \leftarrow H(g, Y, M, Z, A, B)$
Computes $s \leftarrow r - cx$
Returns (M, Y, Z, A, B, c, s)

Algorithm 2: DLEQ Proof Verification

Input : $((g, n), M, Y, Z, A, B, c, s)$
Output: $\{true \text{ or } false\}$

Computes $A' \leftarrow g^s + Y^c$
Computes $B' \leftarrow M^s + Z^c$
Computes challenge $c' \leftarrow H(g, Y, M, Z, A', B')$
Returns $c == c'$

3.2 Distributed Key Generation (DKG)

Each node is defined by a non zero index/id i and we have set Q_e representing all the indexes of the nodes participating to the DKG. We use Feldman VSS [Fel87] to verify whether the shares are correct.

Algorithm 3: Distributed Key Generation

Input : $((g,q),(t,n),Q_e,i,sk_i,(pk_j)_{j\in Q_e})$
Output: $(k_i,(K_j)_{j\in Q_e})$

Round : 1

Samples random $a_{(i,0)} \leftarrow \mathbb{F}_q$

Samples $t-1$ random $\left(a_{(i,1)},\ldots,a_{(i,t-1)}\right)$ and forms $f_i(x) \leftarrow \sum_{z=0}^{t-1} a_{(i,z)}x^z$

Performs a Feldman-VSS on f_i to get $y_{(i,j)} \leftarrow f_i(j)$ and $v_{(i,z)} \leftarrow g^{a_{(i,z)}}$ $\quad \forall (j,z) \in Q_e \times \{0,\ldots,t-1\}$

Stores $v_{(i,0)}$ and $y_{(i,i)}$

Encrypts $y_{(i,j)}$ for all node $j \in Q_e \setminus \{i\}$ with their respective public key: $\boldsymbol{y_{(i,j)}} \leftarrow enc(pk_j, y_{(i,j)})$

Broadcasts $\left((\boldsymbol{y_{(i,j)}}),(v_{(i,z)})\right)$ $\quad \forall (j,z) \in Q_e \setminus \{i\} \times \{0,\ldots,t-1\}$

Round : 2

Receives $\left((\boldsymbol{y_{(j,i)}}),(v_{(j,z)})\right)$ from each node $j \in Q_e \setminus \{i\}$ and $z \in \{0,\ldots,t-1\}$

Decrypts the received partial shares $y_{(j,i)} \leftarrow dec(sk_i, \boldsymbol{y_{(j,i)}})$ $\forall j \in Q_e \setminus \{i\}$

Verifies polynomial f_j and aborts if $g^{y_{(j,i)}} \neq \prod_{z=0}^{t-1} v_{(j,z)}^{i^z}$ $\quad \forall j \in Q_e \setminus \{i\}$

Computes and stores master public key $K \leftarrow \prod_{j\in Q_e} v_{(j,0)}$

Computes and stores private share $k_i \leftarrow \sum_{j\in Q_e} y_{(j,i)}$

Computes public key $K_i \leftarrow g^{k_i}$

Broadcast K_i

Round : 3

Receives public keys $(K_j)_{j\in Q_e\setminus\{i\}}$ from each node

Computes $\lambda_j = (\prod_z z)/(\prod_z z - j)$, with $z \in Q_e \setminus \{j\}$ for all node $j \in Q_e$

Verifies that $\prod_{j\in Q_e} K_j^{\lambda_j} = K$

3.3 Proactivization

In a lot of works investigating secret sharing, a concept of proactive security is developed by the idea of refreshing the nodes' private shares at every epoch.

Multiple techniques exist to re-randomize the shares such that private shares from different epochs could not be combined together.

One method would be to have all the nodes generating a new polynomial sharing zero and adding it to its original private share [HJKY95]. We choose, in the work at hand, to use an alternative approach by having all the nodes re-sharing their private share. They first tune their original private share based on the quorum of nodes and then generate a new polynomial from that private share. Because of the linearity of the polynomial interpolation, each node could then reconstruct locally a new private share of the original secret from the distributed new shares from other nodes. This approach has already been used in several works [GRR98,CKLS02].

At the end of an epoch e, the shared secret is transferred from a n-quorum Q_e to a new one Q_{e+1}. In addition, the value n_e itself could be modified along with the threshold t_e. We introduce n_{e+1} and t_{e+1} as the new configuration for epoch $e+1$.

We also introduce $t^\star$ the number of participants from Q_e for resharing protocol round 1 with $t^\star \geq t_e$ and $T^\star$ the set of their indexes. We note that all the nodes in Q_{e+1} should perform the remaining rounds 2 and 3 of the resharing protocol.

3.4 Scalar Multiplication

A node i should produce here a scalar multiplication of its private share k_i with user's input point ip. We recall that user's input consists of an elliptic curve's point representing a secret value that has been masked beforehand for privacy preservation. The output of this step is composed of the scalar multiplication included in a DLEQ proof allowing the node to prove its knowledge of the private share. Indeed, the generated proof will allow one to verify that the scalar multiplication has been computed with the secret value k_i of the respective public key K_i.

4 Discussions

We are discussing here two security considerations, namely *secrecy* and *robustness*. If our protocols can achieve both characteristics in a mobile adversary scenario, they would be considered as proactively secure. The proofs in the following sections are not cryptographically formal and we focus on proving the correctness of our protocol execution.

As stated previously, we consider the adversary to be active and mobile. The active property means that it is able deviate from the protocols by generating altered data, sending any data of its choice or even sending no data at all. We consider it to be mobile in the sense that it will be able to corrupt up to $t-1$ nodes at each epoch. In other words, a node which is corrupted during epoch e is considered corrupt, and counted against the budget of the adversary in epoch e. Since we assume that corrupted nodes remain corrupted from one epoch to

the other, the amount of corrupted nodes per adversary could grow aggressively if no counter-measures are taken.

4.1 Secrecy

We consider secrecy as the propriety that the secret is never learned by the adversary. To be able to recover secret value s it is sufficient to do it from t shares, as it is stated in Lagrange Interpolation description Section 3.1. In addition to assume that at most $t-1$ nodes could be corrupted by an adversary, it is necessary to show that these particular nodes won't be able to retrieve any additional share from an honest node. We thus reduce the secrecy assumption to the following statement.

Statement 1 *If B is a corrupt node in Q_e and A is an honest node in Q_e, B doesn't learn anything about A's private share.*

Proof. The resulting polynomial F representing the overall secret is equivalent to the sum of all nodes' polynomial $F = \sum_i f_i$.
That being said, since we always have at least $(n - t + 1)$ honest nodes during *DKG* protocol, with $t \leq n$ and we assume that these nodes correctly follow the protocol by generating true random coefficients at *step 2*, we can consider polynomial F's coefficients independent of any adversary knowledge. □

In addition, for *resharing* protocol we have to show that a node in the old quorum Q_e does not learn shares of honest nodes in the new one, except if they are the same node obviously. We recall that the sets of nodes in two consequential epochs could be the same, having different sizes or even being disjoint. We consider in our case that $t-1$ nodes in Q_e and $t-1$ nodes in Q_{e+1} are corrupted at the same time. Therefore, we see that if a corrupted node from Q_e is able to learn the private share of an honest node from Q_{e+1} without being part of Q_{e+1} by itself, the adversary will learn t shares and will be able to retrieve the secret.

Statement 2 *If B is a corrupt node in Q_e and A is an honest node in Q_{e+1}, B doesn't learn anything about A's private share.*

Proof. For *resharing* protocol, the number of nodes from last epoch participating in *round 1* is at least the threshold value, there will always be at least one honest node participating. Similarly to *Statement* 1, the assumption holds. □

A final aspect of secrecy worth being investigated would be when an attacker corrupts disjoint sets of nodes from different epochs in the objective to gather shares from t nodes that previously was in a same n-quorum at a specific epoch $e^\star$. We consider the worst case where the attacker had corrupted $t-1$ nodes during $e^\star$ and succeed to corrupt one of the remaining $(n - t + 1)$ nodes in a later epoch. By definition, that remaining node was honest during $e^\star$ and therefore, had successfully deleted its share from $e^\star$ at the end of it. We recall

the assumption that the attacker is recording all the messages distributed over the public broadcast chanel at any time. Thus, by corrupting the remaining node, the attacker would be able to decrypt the recorded partial shares and retrieve the node's private share. To prevent such an attack, we highlight that the encryption scheme used to encrypt the partial shares should imperatively be a *forward-secure* kind of scheme, such that the decryption key owned by the late corrupted node can't be used to decrypt the partial shares (see Section 4.3).

4.2 Robustness

Robustness, in general use, would ensure that the adversary cannot make the scalar multiplication computation or any other protocol fail. In addition to act badly, an adversary may cause an honest party to behave incorrectly too by sending it incorrect messages. We precise that we only consider active adversaries here since in the case of passive ones, all protocols are robust by definition, as the adversary cannot introduce corruptions into the shares.

In our specific use cases and in blockchain ecosystem more generally, when node operators are required to stake some amount of assets to be authorized to join the protocol, we are considering different assumptions. Indeed, we are reducing the robustness assumption to the ability to identify any falling nodes. We argue that the assets committed during staking is a sufficient guarantee that a node won't deviate from the protocol if it risks to be detected, slashed and thus lost all its assets. We note that a detailed and formal slashing strategy will be presented at a later stage.

We distinguish here between two categories of protocol deviation. One which impacts the correct execution of the protocol and the other which doesn't. We decide to only consider the first case since we assume that a node which slightly deviates from the protocol without causing any consequence on the final outputs won't affect robustness as we define it. By example, in *dkg* protocol *step 1*, if a node decides to not generate its secret randomly, we are facing a situation that could affect its own private share secrecy but not the overall robustness of the protocol. Therefore, we omit such kind of node behavior and focus on analyzing how a corrupt node could deviate and make the protocol fail. The fundamental reason integrity is preserved is because of the way we use commitments.

Statement 3 *If B is a corrupt node in Q_e deviating and failing dkg protocol, it will be noticed and identified by any honest node participating to the same DKG protocol.*

Proof. The first step where node B could act badly is during *Round 1* when generating the partial shares but our protocol makes use of the Feldman VSS. Node B will have to commit on its polynomial f_B such that each node can verify at *step 9* that the received partial share has correctly been generated from polynomial f_B. In addition, we force node B to publicly broadcast its polynomial commitments such that all the nodes are verifying f_B using the same commitments and node B cannot fool one by using a different polynomial

to generate both the partial share and the respective commitments. Any attempt in failing the encryption at *step 5* by using an incorrect public key by example, will also be noticed since the later decrypted value won't be validated by thr receiver node at *step 9*.

The second step where a malicious node could deviate from the protocol in the objective to fail final verification at *step 16* or even fool the scalar multiplication at a later stage, in public key commitment at the end of *Round 2*. We recall that having nodes committing on their respective public key during *dkg* protocol will guarantee that the later scalar multiplication has been performed with the same key and therefore the correct private share. So any honest node that correctly computed the *master* public key K at *step 10* based on the polynomials' commitments $v_{(j,0)}$ will be able to validate all the keys as a set. We precise that values $v_{(j,0)}$ are considered as correct as that step since the protocol would have been aborted at *step 9* otherwise. One may argue that if node B decides to send a bad K_B value which doesn't correspond to its private share, it will cause *step 16* abortion without being identified as the faulty node. Since the polynomials' commitments are publicly distributed at the end of *Round 1*, we argue that any node can actually compute each other node j's public key using the following formula: $K_j \leftarrow \prod_{z=1}^{n} \prod_{k=0}^{t-1} v_{(z,k)}^{j^k}$
To not overload the protocol we decided to not include this step to the last round of *dkg* protocol but we save it in case *step 16* is failing and faulty nodes should be identified. □

Statement 4 *If B is a corrupt node in Q_e deviating and failing Resharing protocol, it will be noticed and identified by any honest node participating to the same Resharing protocol.*

Proof. As presented in Section 3.3, *Resharing* protocol is quite similar to *dkg* one. The attack openings will be exactly the same and the proof provided for *Statement* 3 holds. This ensures that all the new nodes have shares of the same secret and that these shares can be used to correctly compute the scalar multiplication of the secret in a distributed manner. □

Statement 5 *If B is a corrupt node in Q_e deviating and failing Scalar Multiplication protocol, it will be noticed and identified by any honest node participating to the same Scalar Multiplication protocol.*

Proof. In such a case, an adversary could make use of node B to provide an dummy value in the objective make the final scalar multiplication incorrect. Depending on the scenario, it may even happen that no one is noticing that the final result is not the expected one. The security here relies on the DLEQ proof described at Section 3.1 where node B is forced to submit the scalar multiplication computed using its correct private share. We recall that node B's public key had to be committed and verified in *dkg* protocol *Round 3*. Therefore, during DLEQ proof generation (*algorithm* 1), value Y should correspond to

the public key broadcasted at the end of *dkg* protocol *Round 2*. That being said, let's now suppose that node B is including to the DLEQ proof a scalar multiplication $Z^\star$ computed by replacing its secret share x such that $Z^\star = g^{x^\star}$. The incorrect DLEQ proof is thus of the form $(M, Y, Z^\star, A, B, c^\star, s)$ with $c^\star = H(g, Y, M, Z^\star, A, B)$ that should be computed using $Z^\star$ since DLEQ verification algorithm (algorithm 2) will use it to compute c' step 3. Node B will then face a dilemma when computing and providing value $s\star$ about which of the two private shares x or $x^\star$ to use.
If $s^\star = r - c^\star x$ then for DLEQ verification step 2:

$$\begin{aligned} B' &= M^{s^\star} + Z'^{c^\star} \\ &= M^{r - c^\star x} + M^{c^\star x^\star} \\ &\neq B \end{aligned}$$

Or else $s^\star = r - c^\star x^\star$ then for DLEQ verification step 2:

$$\begin{aligned} A' &= g^{r - c^\star x^\star} + Y^{c^\star} \\ &= g^{r - c^\star x^\star} + g^{c^\star x} \\ &\neq A \end{aligned}$$

We thus have shown that in either cases the DLEQ proof will be verified and an incorrect scalar multiplication result will always be rejected. □

4.3 Forward-Secure Public-Key Encryption (FSPKE) Scheme

As a complementary approach to proactivization induced by key resharing, we adopt an FSPKE type of encryption scheme for, on the one hand, *DKG* and *Resharing* communication, and on the other hand, secret keys local storage. We implemented scheme from Canetti, Halevi and Katz [CHK03] and applied sight modifications. This paper introduces a binary-tree encryption (BTE) technique, based on hierarchical identity-based encryption (HIBE) scheme of Gentry and Silverberg [GS02]. The scheme is composed of four algorithms, namely *Keygen*, *Update*, *Encryption* and *Decryption*. We highlight that in its original version [CHK03], the total number of periods N is bounded. Indeed, such a scheme allows one to update its secret key from one period to the other such that the updated secret key won't allow to decrypt a ciphertext encrypted for any other period. We also note that updating the secret key won't change its respective public key which should remain the same over the periods.

Complexity. Obviously, integrating such an advanced encryption scheme instead of a more basic one as RSA, will require more computation and communication power. The beauty of this FSPKE scheme and its binary tree construction is that computation complexity of *encryption* and *decryption* functions relies on the binary tree depth. So at most, parameters grow logarithmically with the total number of time periods. While each period will be affected to a node from

the binary tree following a preorder transversal, runtime of the two functions will increase depending how depth the current node is. *Keygen* function is, on the other hand, constant time no matter how large is the total amount of planed periods given as input. Finally, *Update* runtime depends on the fact that current node is a leaf or not. So by considering only complete binary tree, the ratio of current period corresponding to a leaf node (around 1/2) remains the same no matter the total number of time periods.

On the aspect of communication complexity, size of some elements is also growing logarithmically with the total number of time periods. The ciphertext consists of a vector of group elements and the amount of these elements varies from 1 to l, the binary tree depth. Secret key is actually represented as a stack of secret keys and both the number of secret keys in the stack and the size of each secret key depends on l similarly to ciphertext. Concretely, the amount of group elements is a secret key stack varies from 1 to l^2.

Summary

Human Network is a threshold network for scalar by elliptic curve point multiplication built as an actively validated service on EigenLayer and Symbiotic. It is a network that allows a group of nodes to perform a scalar by elliptic curve point multiplication. This unlocks two decentralized cryptographic primitives: decentralized VOPRF and decentralized ElGamal over ZK-friendly curves. These in turn unlock a new set of decentralized applications not previously feasible that require high privacy and security.

Algorithm 4: Resharing Scheme

Input : $((g,q),(t_{e+1},n_{e+1}),T^\star,Q_{e+1},i,sk_i,(pk_j)_{j\in Q_{e+1}},K)$
Output: $(k_i,(K_j)_{j\in N})$

Round : 1

Computed by $t^\star$ nodes in Q_e

Computes $\lambda_i = (\prod_j j)/(\prod_j j - i)$, for $j \in T^\star \setminus \{i\}$

Computes $a_{(i,0)} \leftarrow \lambda_i k_i$

Samples $t_{e+1}-1$ random $\left(a_{(i,1)},\ldots,a_{(i,t_{e+1}-1)}\right)$ and forms

$$f_i(x) \leftarrow \sum_{z=0}^{t_{e+1}-1} a_{(i,z)}x^z$$

Performs a Feldman-VSS on f_i to get $y_{(i,j)} \leftarrow f_i(j)$ and
$v_{(i,z)} \leftarrow g^{a_{(i,z)}}$ $\quad \forall (j,z) \in Q_{e+1} \times \{0,\ldots,t_{e+1}-1\}$

Encrypts $y_{(i,j)}$ for all node $j \in Q_{e+1}$ with their respective public key:
$\boldsymbol{y_{(i,j)}} \leftarrow enc(pk_j, y_{(i,j)})$

Broadcasts $\left((\boldsymbol{y_{(i,j)}}),(v_{(i,z)})\right)$ $\quad \forall (j,z) \in Q_{e+1} \times \{0,\ldots,t_{e+1}-1\}$

Round : 2

Computed by all nodes in Q_{e+1}

Receives $\left((\boldsymbol{y_{(j,i)}}),(v_{(j,z)})\right)$ from each node $j \in T^\star$ and $z \in \{0,\ldots,t_{e+1}-1\}$

Decrypts the received partial shares $y_{(j,i)} \leftarrow dec(sk_i, \boldsymbol{y_{(j,i)}})$ $\quad \forall j \in T^\star$

Verifies polynomial f_j and aborts if $g^{y_{(j,i)}} \neq \prod_{z=0}^{t-1} v_{(j,z)}^{i^z}$ $\quad \forall j \in T^\star$

Aborts if $\prod_{j\in T^\star} v_{(j,0)} \neq K$

Computes and stores private share $k_i \leftarrow \sum_{j\in T^\star} y_{(j,i)}$

Computes public key $K_i \leftarrow g^{k_i}$

Broadcast K_i

Round : 3

Computed by all nodes in Q_{e+1}

Receives public keys $(K_j)_{j\in Q_{e+1}\setminus\{i\}}$ from each node in Q_{e+1}

Computes $\lambda_j = (\prod_z z)/(\prod_z z - j)$, with $z \in Q_{e+1} \setminus \{j\}$ for all node $j \in Q_{e+1}$

Verifies that $\prod_{j\in Q_{e+1}} K_j^{\lambda_j} = K$

Algorithm 5: Scalar Multiplication

Input : $(g,n),k_i,ip$
Output: $d_i = \{ip,K_i,Z_i,A_i,B_i,c_i,s_i\}$

Computes $d_i \leftarrow dleq\_gen((g,n),ip,k_i)$

Returns $d_i = \{ip,K_i,Z_i,A_i,B_i,c_i,s_i\}$

References

[CHK03] Ran Canetti, Shai Halevi, and Jonathan Katz. A forward-secure public-key encryption scheme. Cryptology ePrint Archive, Paper 2003/083, 2003. https://eprint.iacr.org/2003/083.

[CKLS02] Christian Cachin, Klaus Kursawe, Anna Lysyanskaya, and Reto Strobl. Asynchronous verifiable secret sharing and proactive cryptosystems. Cryptology ePrint Archive, Paper 2002/134, 2002. https://eprint.iacr.org/2002/134.

[Eig23] Team EigenLayer. Eigenlayer: The restaking collective. EigenLayer Whitepaper, 2023. https://docs.eigenlayer.xyz/assets/files/EigenLayer_WhitePaper-88c47923ca0319870c611decd6e562ad.pdf.

[Fel87] Paul Feldman. A practical scheme for non-interactive verifiable secret sharing. In *28th Annual Symposium on Foundations of Computer Science, Los Angeles, California, USA, 27-29 October 1987*, pages 427–437. IEEE Computer Society, 1987.

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: practical solutions to identification and signature problems. In *Proceedings on Advances in Cryptology—CRYPTO '86*, page 186–194, Berlin, Heidelberg, 1987. Springer-Verlag.

[Gam84] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Advances in Cryptology, Proceedings of CRYPTO '84, Santa Barbara, California, USA, August 19-22, 1984, Proceedings*, volume 196 of *Lecture Notes in Computer Science*, pages 10–18. Springer, 1984.

[GRR98] Rosario Gennaro, Michael O. Rabin, and Tal Rabin. Simplified VSS and fast-track multiparty computations with applications to threshold cryptography. In Brian A. Coan and Yehuda Afek, editors, *Proceedings of the Seventeenth Annual ACM Symposium on Principles of Distributed Computing, PODC '98, Puerto Vallarta, Mexico, June 28 - July 2, 1998*, pages 101–111. ACM, 1998.

[GS02] Craig Gentry and Alice Silverberg. Hierarchical ID-based cryptography. Cryptology ePrint Archive, Paper 2002/056, 2002.

[HJKY95] Amir Herzberg, Stanislaw Jarecki, Hugo Krawczyk, and Moti Yung. Proactive secret sharing or: How to cope with perpetual leakage. In *Advances in Cryptology - CRYPTO '95, 15th Annual International Cryptology Conference, Santa Barbara, California, USA, August 27-31, 1995, Proceedings*, volume 963 of *Lecture Notes in Computer Science*, pages 339–352. Springer, 1995.

[JKK14] S. Jarecki, A. Kiayias, and H. Krawczyk. Round-optimal password-protected secret sharing and t-pake in the password-only model. International Conference on the Theory and Application of Cryptology and Information Security, 2014. https://eprint.iacr.org/2014/650.

[OY91] Rafail Ostrovsky and Moti Yung. How to withstand mobile virus attacks (extended abstract). In Luigi Logrippo, editor, *Proceedings of the Tenth Annual ACM Symposium on Principles of Distributed Computing, Montreal, Quebec, Canada, August 19-21, 1991*, pages 51–59. ACM, 1991.

[Sha79] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, nov 1979.

[Sym24] Team Symbiotic. Introduction to symbiotic. Symbiotic Whitepaper, 2024. https://docs.symbiotic.fi/.