Silent Protocol:
0 -VM: General purpose zero knowledge computing layer

Silent Research Labs

Jan 2024

Contents

1 Introduction **1**

2 Decentralized proving through the 0VM **1**

3 The Concept of Anonymous Verification **2**

4 Silent 0VM Design **3**

4.1 Silent 0VM components 3

4.2 Anonymous Relaying 4

5 Root Network **6**

5.1 Root Network Data Flow 6

5.2 The root network Processes 8

5.3 Key Resharing 10

6 Conclusion **11**

1 Introduction

Over the past decade ZK systems have gained in prominence. A zero-knowledge proof (ZKP) is a method of proving the validity of a statement without revealing anything other than the validity of the statement itself. It is a proof system with a prover, a verifier, and a challenge that gives users the ability to publicly share a proof of knowledge or ownership without revealing the details of it. We have seen different applications of zero knowledge systems pop up in recent times like rollups, zkMLs, privacy systems and end interfaces pushing the utility curve of general-purpose zero-knowledge arguments up and to the right. Interactive proof systems and arguments (zero knowledge or otherwise) have made the jump from theory to practice. And this has opened new doors in the design of different protocols that utilize zk proofs to process state update however it has generated additional insights into improvements that needs to be brought forward in order to meet current demand.

There are two main properties that ZK systems bring to the table. First is the property of succinctness where it becomes magnitudes faster for verifying a proof of computational integrity over a set of pre-defined set of constraints compared to processing the given computation itself. Second is the property of "zero knowledge" where hiding certain parts of the computational statement is possible for proving the correctness of the state transition. These helps to build distributed systems that scales and also that preservers and protects privacy.

Zero Knowledge systems in the context of blockchains are mainly used to for scaling and for building privacy systems. These causes two issues, generating a zero-knowledge proof is a computationally intensive process resulting in high computational fees for on-chain verification and becomes responsible for creating system bottlenecks due to strenuous computation required for generating the proofs. One the other hand, on-chain proving systems requires private data from users due to which building privacy systems on public smart contract enabled blockchains developers become inefficient due to existing transaction schematics (private DAOs, secure data transfer layer etc.)

In this paper we are defining a proxy for public blockchains known as 0VM, which is a chain agnostic and consensus agnostic transaction verification and a proving marketplace custom built for enabling scalable ZK applications. Developers can make use of 0VM for inclusion of transaction in a public state machine in a trustless and privacy preserving manner and for generating zk proofs efficiently for different proving systems through GPUs, FPGAs and ASICs.

2 Decentralized proving through the 0VM

Developers can use 0VM to either tap into a decentralized proving market or utilize the properties of anonymous verification facilitated by 0VM. 0VM allows developers and users to verify computation of arbitrary systems defined within the protocol. It allows for distributed proof generation and acts as a proof aggregation engine for a variety of zk applications. It is based on the uniplonk variety to create the processor that allows for verification of proof from different application from different

parties.

0VM allows for high performant proving, while providing availability guarantees that's common across decentralized networks. It does this utilizing off-chain aggregation to prove multiple zero knowledge proofs by uniforming the Verifiers work for families of circuits. Specifically, a single fixed-cost Universal Verifier that can check proofs for circuits of different: sizes, public input lengths, selector polynomials, copy constraints, and even different custom gate sets.

0VM is a chain agnostic and consensus agnostic protocol, that hosts different types of nodes in order to generate and verify zero knowledge proofs for distributed applications. Based on the type of nodes(hardware specifications) a transaction workload could be submitted into the network. The renode layer in 0VM is responsible for passing the workload to the "root network" of the protocol where different prover node based on class specification can process a transaction by signing the message.

The signers initiated within the 0VM makes up the root network that receives "request" from the user through the renode layer and processes workload into a certain layer where they get consumed by the specified category of provers returning an output and only after receiving a majority input into the request. 0VM leverages maximum parallelization through sharding at both the root network level and the network level for processing transactions. This parallel processing capability significantly boosts the network's efficiency and scalability. Based on type of workload that a user or a developer wants to file into, the cost of a transaction or a certain cycle of proving gets determined by the network.

3 The Concept of Anonymous Verification

In a public state machine the way to update the state is to send a state update request, while, building a zk application that has private data that cant be sent to the network, the problem of updating the state becomes relevant. This has led to the creation of 3rd party actors in a tx lifecyle known as, 'relayers' who pays for the transaction but retrieves the transaction fee back from the value that is being relayed. This has stopped many types of zk applications(example private voting) from being developed over time.

To formally characterize the problem of providing a relayer a guarantee or a commitment for getting repaid for processing a transaction and including it on chain on behalf of the user before the inclusion takes place, we define the idea of Anonymous Verification.

Anonymous Verification is a communication primitive that enables anonymous inclusion of transaction into the blockchain by providing the following guarantees:

1. Every message m sent over the network by the user U is verified by a group of actors A_x, to check U has balance $> a$ the required amount of gas.

2. A message m is delivered to the network by the relayer, if and only if the associated signature of the verified group of actors is accepted by the relaying network.

Through 0VM we ensures that the 3rd party actors (example relayers) gets paid for their services by extending trust of a accepted network. The user and relayer agrees on the fact that the user will pay using funds already deposited in the network. Once the relayer successfully submits the transaction into the blockchain, they receive their payment as their service fee.

0VM acts as the guarantor for every transaction based on whose guarantee the relayer submits the transaction.

At a high level, 0VM perform the verification in a trustless and secure manner, where the decentralized base of signers helps us, build the concept of trustless anonymous verification into practice throuhg 0VM. The ideal solution to the problem allows the 3rd party including the transaction in the system to receive cryptographically secure authentication as a forward payment guarantee based on which they can conduct their actions trustless anonymous verification. 0VM helps us not only build a decentralized prover network but also allows us to implement an authentication protocol that provides trustless anonymous verification of user data.

4 Silent 0VM Design

0VM provides a architecture that provides trustless anonymous verification and a decentralized base for offering decentralized proving for building zero knowledge applications. It is built on a series of components introduced in Section 4.1. We discuss the interaction flow and protocol design for achieving the properties of anonymous verification through the 0VM in Section 4.2. We leave out details of the verifier circuit hosted by the 0VM through the signer nodes in the root network, for a future article.

4.1 Silent 0VM components

4.1.1 The rooot network

The Root network utilizes the Genarro-Goldfeder 2020 [GG20], to allows efficient threshold-based signing with no trusted dealer. There are three multi-party computation routines namely: key generation, key signing, and key resharing. The key generation ceremony allows the nominated parties to construct the parameters for a new polynomial. The output of the event is conceived as a public key which becomes the representative for this network. When the renode layer (later defined in the paper) delegates an outgoing transaction to be signed or validated by the root network, the root network using the actors delegate the task over to certain set of signers or specialized node that recognize their participation, prepare a copy of the message to be signed, and enter the signing ceremony. They either are responsible for validating a transaction or proving a zero knowledge proof. The key-signing ceremony begins when the required number of participants are present and allows a signature from the public key generated during the key generation process of the protocol for an outgoing transaction to be generated. The renode layer receives the output and makes it compatible

with the recipient chain to satisfy the requirements.

Each ceremonies involves a communication round where each participant checks the validity of all other expected members, as prescribed by the system on top of utilizing a commit-reveal scheme that helps ensure that secret shares for each participant cannot be changed after joining into a ceremony. If signers aborts any process which results in an attributable failure, all participants can make a blame transaction. Finally the Key resharing process allows the parties to add a new member to the network by sharing the secret without updating the system's final public key.

4.1.2 Renode Layer

Renode Layer is the component in 0VM that aids both in the process of building meta-transactions processing zero knowledge proofs and posting back and forth from the main network and the root network with the help of relayers. Meta-transactions utilize EIP-712 [BLE17], with which we can work using structured data. By building meta-transactions, users can sign and relay transactions through relayers, who submit them on-chain on their behalf. For processing anonymous verification the user interacts with the system through the end interface, which is a series of contracts, where the 'paymaster' contract is responsible for gas payments in exchange for a service fee, and the 'forwarder' contract verifies the sender's signature and forwards the request to the target contract with which the user wants to interact. For processing zero knowledge proofs for zero knowldge applications (rollups, ZkML circuits) the renode layer is pinged by end application for utilizing the 0VM that wants to communicate with the target application.

4.1.3 Encrypted Storage

An encrypted data storage and data retrieval layer help the actors in the root network to process zero knowledge proof or verification request and to provide anonymous verification for zero knowledge proofs. The system utilizes an offchain HSM module to post private data-store in order to save the data of the user details off chain in the permissioned setting. However, the root network in its permisionless setting encourages the actors in the ceremony to save the data received by the leader node in its encrypted format and decrypt it using threshold decryption to process a certain computation, when requested, eliminating the design dependency on external systems.

4.2 Anonymous Relaying

In this section, we go over the entire process of achieving anonymous verification and how we achieve it. We go through an example of a user sending a relay request and anonymously paying back the relayer using anonymous verification through the Root Network and the Renode layer.

Step 1. The user U owns a master key and utilizes it to deposit the amount a into the renode layer, particularly in the paymaster contract. And while doing so, it creates a relay hub-specific stealth keypair. While interacting with the Renode Layer, the user deposits assets into the paymaster

contract to fuel payments to the relayer. It also provides input data c_{U} accompanying the transaction that helps the system, specifically the root network in the 0VM that helps identify the users relay hub-specific key-pair in an encrypted format,

$$c_{\mathsf{U}} = E(pk_P, spk_{\mathsf{U}}).$$

Step 2. Once the user makes the successful deposit into the renode layer, the paymaster contract emits an event that is listened to by the Root network using an oracle network. The coordinator in the root network logs the details into the encrypted storage layer using asymmetric key encryption.

If this is an initial deposit, the root network adds a new index and stores the following tuple into the encrypted storage network,

$$\{spk_{\mathsf{U}}, B_{\mathsf{U}} = a\}.$$

if spk_{U} already exists, the root network updates the corresponding entry as

$$\{spk_{\mathsf{U}}, B_{\mathsf{U}} = B_{\mathsf{U}} + a, \{\sigma_{\mathsf{U},i}\}\},$$

where $\sigma_{\mathsf{U},i}$ is a list of user's signatures for relaying usages.

Step 3. The paymaster contract in the rneode layer does not store the user's balance when the user has made the deposit. However, it emits the event when the user U selects an active relayer, R_{id} from the renode layer, and initiates a message payload m with the signed information

$$(m, c'_{\mathsf{U}} = E(pk_P, spk_{\mathsf{U}}), \sigma_{\mathsf{U},i} = S(ssk_{\mathsf{U}}, \{m, c'_{\mathsf{U}}, R_{id}\})),$$

where:

- m is the actual transaction payload,
- $c'_{\mathsf{U}} = E(pk_P, spk_{\mathsf{U}})$ is the encryption of stealth public key spk_{U} under the renode trust networks public key pk_P,
- $\sigma_{\mathsf{U},i} = S(ssk_{\mathsf{U}}, \{m, c'_{\mathsf{U}}, R_{id}\})$ is the signature of U on message $\{m, c'_{\mathsf{U}}, R_{id}\}$ with their stealth private key ssk_{U}.

Step 4. The relayers sends the message to the signer in the root network through the renode layer.

Step 5. Once the root network receives the message, the coordinator in the permissioned setting of the network and the leader node in the permissionless setting of the network assembles a group of singers and players to decrypt the commitment to receive the public key, verify the attached signature and check the balance available in the now decrypted public key of the user's relayer hub specific key in the encrypted storage layer. If found honestly, signers would finish the now-ongoing ceremony once the threshold is met and approve the tx-payload with their signature.

The pseudocode of the above can be depicted as follows:

1. Decrypts c'_{U} to get $spk_{\mathsf{U}} = D(sk_P, c'_{\mathsf{U}})$
2. Verifies the signature, i.e $\{True\} = V(spk_{\mathsf{U}}, \{m, c'_{\mathsf{U}}, R_{id}\}, \sigma_{\mathsf{U},i})$
3. Checks if spk_{U} has enough funds B_{U} for max fee payment
4. Only then would it approve tx payload with its signature, and sends it to the selected relayer, R_{id} with the following

$$(m, h(m, c'_{\mathsf{U}}, \sigma_{\mathsf{U},i}), \sigma_P = S(sk_P, \{m, h(m, c'_{\mathsf{U}}, \sigma_{\mathsf{U},i})\}))$$

5. Waits for the renode layer to notify the renode trust network that the tx has been successfully posted, the actual fee is deducted, and the balance B_{U} is updated in the database

Step 6. The relayer R_{id} posts the tx approval to the relay hub contract within the renode layer, which helps the relayer gets a refund guarantee,

$$(m, h(m, c'_{\mathsf{U}}, \sigma_{\mathsf{U},i}), \sigma_P, txsig_{R_{id}}).$$

Step 7. Once the renode layer successfully observes the tx being mined, it transfers the balance from the paymaster contract to the renode Rid for completing the transaction in a permissionless manner.

Remark. *The process of anonymous verification assumes all existing entities may try to take advantage if possible. For instance, a user may claim that her balance is deducted without her consent. In this case, Root Network should be able to provide her signature for the request, that's why the list of these signatures are stored as* $\{\sigma_{\mathsf{U},i}\}$.

5 Root Network

5.1 Root Network Data Flow

The root network is responsible for processing the workload and returning a request whether be it for the anonymous verification protocol or for proving a zk proof request for arbitrary zero knowledge applications. In the Root Network, all processes - keygen, key sign, and key reshare - have a similar data flow. In the permissioned setting the coordinator and later the leader node initiates a request and checks if it has enough signers or nodes before starting the process. The coordinator runs the join-party scheme and waits for other signers to join within a given time. If a player fails to join the party within that time, the other players blame that player and continue the process. The process is finalized if all players show up on time. If a node cannot process the shares received from other players within a given time, it blames the party that failed to provide the share. After the process

is completed, the coordinator sends notifications to all parties involved. Moreover, before a player quits the process, it broadcasts the generated signature to all parties involved in the same process.

To protect sensitive shares from being leaked to a third party, the Root Network uses libp2p to ensure the network infrastructure is secure and private. Before working together in a joint-party scheme, the Root network also ensures that individual signers verify their identities by proving knowledge of their private keys. Moreover, all signers working together in the network use public key encryption for transmitting messages within the protocol and conduct message passing directly without any relay.

5.1.1 Message Security and integrity check during the message processing

The Root Network, to ensure smooth coordination of processes and avoid signing failures, uses a mechanism called 'blame signaling' to identify bad signers or players within the network. The coordinator works with all nodes in the network impartially. The messages are verified and transferred between the nodes/signers and the coordinator by checking and verifying their shares, avoiding man-in-the-middle attacks or network availability issues. Peers or signers within the network can detect an abort in the signing process and blame a particular signer. This can lead to the signer losing their stake in the network once it becomes permissionless. The coordinator continues the signing process and completes it only after receiving enough shares from the participants. In a permissionless setting, if the leader node does not receive enough shares from the local party, it can request the shares from the peers who claim to have them without halting the signing process.

After receiving a share for a specific message, the coordinator checks its validity by verifying the signatures and ensuring that it meets the majority requirement within the network. Once this requirement is satisfied, the coordinator signs the message and moves the system forward. The process for a given message gets finalized once the coordinator signs it. By default, the coordinator, or leader node in a permissionless setting, will notify all parties involved in the process upon completion.

5.1.2 Increased performance through Parallel and Batch Processing

The Root Network's coordinator or leader node allows signers to work on multiple signing tasks in parallel. Once the coordinator receives requests and messages from the renode layer, it starts a ceremony for a specific message. It waits for others to join, but at the same time, other nodes can join a new ceremony for a different message due to the unique identification of each message in the network. Based on the message ID, signers can authorize and sign the request using their key shares in the network. If a node is slightly late to work on a ceremony with a given message ID, and the group has already been formed and completed, the coordinator will return an error code to notify the node that they are not part of the key sign party. This design enables signers to work on different message processing threads, where 'Session ID' is applied to allow the underlying communication management service to dispatch the signing messages to the corresponding process thread without facing any risks from side-channel attacks.

The Root Network also supports batch processing, where signers can send multiple messages to the coordinator in a single signing request. This feature helps to reduce network communication between nodes dramatically. The coordinator stores the intermediate result of each signing round for different messages and packs them together as a bulk message to send to peer nodes. On receiving the bulk P2P message, the receiver verifies the correctness of the batch and processes each message one by one.

5.2 The root network Processes

The Root Network relies on Genarro-Goldfeder [GG20] for its key generation and key signing process; we recall the protocols in Section 5.2.1 and Section 5.2.2, respectively.

5.2.1 Ceremony: Key Generation

The key generation protocol is largely as follows:

- Each Player P_i selects $u_i \in_R \mathbb{Z}_q$ and computes $[KGC_i, KGD_i] = Com(g^{u_i})$ and broadcasts KGC_i. Each Player P_i broadcasts E_i, the public key for Paillier's cryptosystem.
- Each Player P_i broadcasts KGD_i. Let y_i be the value decommitted by P_i. The player P_i performs a (t, n) Feldman-VSS of the value u_i , with y_i as the free term in the exponent. The public key is set to $y = \prod_i y_i$. Each player adds the private shares received during the n Feldman VSS protocols. The resulting values x_i are a (t, n) Shamirs secret sharing of the secret key $x = \sum_i u_i$. Note that the values $X_i = g^{x_i}$ are public.
- Let $N_i = p_i q_i$ be the RSA modulus associated with E_i . Each player P_i proves in ZK that he knows x_i using Schnorrs protocol [Sch91], that N_i is square-free using the proof of Gennaro, Micciancio, and Rabin [GMR98], and that h_1, h_2 generate the same group modulo N_i.

The Root Network requires all players to collaborate to generate key pairs. In the keygen process, all players should have their clocks synchronized, allowing them to join the key generation process at almost the same time. Each signer or player in the distributed key generation process establishes communication channels to share secrets and generate the protocol key pair. Malicious players are identified and blamed. Each signer stores their secret shares with a backup and forms the public key by summation over the shares generated by the scheme.

5.2.2 Key Signing, Root network message verification and signature generation

The Root Network runs on input m (sent by the user utilizing the relaying network).

Let $S \subseteq [1 \dots n]$ be the set of players participating in the signature procedure. We assume that $|S| = t + 1$, where t is the threshold value.

For the signing protocol we can share any ephemeral secrets using a $(t, t + 1)$ secret sharing scheme, and do not need to use the general (t, n) structure.

We note that using the appropriate Lagrangian coefficients $\lambda_{i,S}$ each player in S can locally map its own (t,n) share x_i of x into a $(t,t+1)$ share of $x, w_i = (\lambda_{i,S})(x_i)$, i.e. $x = \sum_{i\in S} w_i$. Since $X_i = g^{x_i}$ and $\lambda_{i,S}$ are public values, all the players can compute $W_i = g^{w_i} = X_i^{\lambda_{i,S}}$.

- **P**hase 1. Each Player P_i selects $k_i, \gamma_i \in_R \mathbb{Z}_q$; computes $[C_i, D_i] = Com(g^{\gamma_i})$ and broadcasts C_i . Define $k = \sum_{i\in S} k_i, \gamma = \sum_{i\in S} \gamma_i$. Note that

$$k\gamma = \sum_{i,j\in S} k_i\gamma_j \mod q,$$

$$kx = \sum_{i,j\in S} k_i w_j \mod q.$$

- **Phase 2.** Every pair of players P_i , P_j engages in two multiplicative-to-additive share conversion subprotocols, MtA [GG18] and MtAwc [GG20]. Note that the first message for these protocols is the same and is only sent once.

  - P_i , P_i run MtA with shares k_i, γ_j respectively. Let α_{ij} [resp. βij] be the share received by player P_i [resp. P_j] at the end of this protocol, i.e. $k_i\gamma_j = \alpha_{ij} + \beta_{ij}$. Player P_i sets $\delta_i = k_i\gamma_i + \sum_{j\neq i} \alpha_{ij} + \sum_{j\neq i} \beta_{ij}$. Note that the δ_i are a $(t,t+1)$ additive sharing of $k\gamma = \sum_{i\in S} \delta_i$
  - P_i, P_j run MtAwc with shares k_i, w_j respectively. Let μ_{ij} [resp. ν_{ij}] be the share received by player P_i [resp. P_j] at the end of this protocol, i.e.

$$k_i w_j = \mu_{ij} + \nu_{ij}$$

    . Player P_i sets $\sigma_i = k_i w_i + \sum_{j\neq i} \mu_{ij} + \sum_{j\neq i} \nu_{ij}$. Note that the σ_i are a $(t,t+1)$ additive sharing of $kx = \sum_{i\in S} \sigma_i$.

- **Phase 3.** Every player P_i broadcasts

  - δ_i and the players reconstruct $\delta = \sum_{i\in S} \delta_i = k\gamma$. The players compute $\delta^1 \mod q$.
  - $T_i = g^{\sigma_i} h\ell_i$ with $\ell_i \in_R \mathbb{Z}_q$ and proves in ZK that he knows σ_i, ℓ_i.

- **Phase 4.** Each Player P_i broadcasts D_i. Let Γ_i be the values decommitted by P_i. The players compute $\Gamma = \prod_{i\in S} \Gamma_i$, and

$$R = \Gamma^{\delta^{-1}} = g^{(\sum_{i\in S} \gamma_i)k^{-1}\gamma^{-1}} = g^{\gamma k^{-1}\gamma^{-1}} = g^{k^{-1}}$$

  as well as $r = H'(R)$.

- **Phase 5.** Each player P_i broadcasts $\overline{R}_i = R^{k_i}$ as well as a zero-knowledge proof of consistency between R_i and $E_i(k_i)$, which each player sent as the first message of the MtA protocol in Phase

2. If

$$g \neq \prod_{i \in S} \overline{R}_i$$

the protocol aborts.

- **Phase 6.** Each player P_i broadcasts $Si = R^{\sigma_i}$ as well as a zero-knowledge proof of consistency between S_i and T_i, which each player sent in Phase 3. If

$$y \neq \prod_{i \in S} S_i$$

the protocol aborts.

- **Phase 7.** Each player P_i broadcasts $s_i = mk_i + r\sigma_i$ and set $s = \sigma s_i$. If the signature (r, s) is correct for m, the players accept, otherwise they abort.

Through the Root Network, multiple players can collectively sign messages that can be verified by a single public key. Before the key generation protocol starts, the coordinator in the network assembles online players and invites them to participate in the ceremony for the signature generation process. Signers involved in the key sign process participate in the signature generation within a given time frame. If any player does not cooperate or fails the message signing process, other parties will flag them. Once a signature is generated, the coordinator batches them and generates the output from the ceremony. In a permissionless setting of the network, a player can be struck if they fail to participate.

5.3 Key Resharing

Following [Can+20], key-refresh phase proceeds as follows.

Each party $\mathcal{P}_i$ samples a Paillier modulus N_i obtained as a product of safe-primes, as well as ring-Pedersen parameters (s_i, t_i). Then, $\mathcal{P}_i$ samples a secret sharing $\left(x_i^1, \ldots, x_i^n\right)$ of $0 \in \mathbb{F}_q$, computes $\boldsymbol{X}_i = \left(X_i^1 = g^{x_i^1}, \ldots, X_i^n = g^{x_i^n}\right)$, and broadcasts $\boldsymbol{X}_i, N_i, s_i, t_i$ to all. After receiving all the relevant values, party $\mathcal{P}_i$ encrypts each x_i^k under $\mathcal{P}_k$'s Paillier public key N_k and obtains ciphertexts C_i^k, for all $k \neq i$, which he sends to all parties.

Then, each $\mathcal{P}_i$ refreshes to a new private key-shares $x_i^* = x_i + \sum_\ell x_\ell^i \bmod q$, updates public key-shares of all parties $X_j^* = X_j \cdot \prod_\ell X_\ell^j$, and stores new $(N_1, s_1, t_1), \ldots, (N_n, s_n, t_n)$. Concurrently, each $\mathcal{P}_i$ (locally) samples $y_i \leftarrow \mathbb{F}_q$ and communicates the value $Y_i = g^{y_i}$. For malicious security, the aforementioned process is augmented with the following ZKP's:

- N_i is a Paillier-Blum Modulus.
- ZK-Proof that s_i belongs to the multiplicative group generated by t_i in $\mathbb{Z}_{N_i}^*$.
- Schnorr PoK for the discrete logarithms of $X_i^1, \ldots, X_i^n$ and Y_i.

- Verify that the plaintext value of C_j^i is equal to the discrete logarithm of X_j^i, for all $j \neq i$.

For the security analysis, reader may refer to the original paper [Can+20].

6 Conclusion

This paper introduced the design and implementation of 0VM. It provides a decentralized infrastructure for processing and generating zero knowledge proofs for general purpose zero knowledge applications along side allowing an end agent to get verified and trustless certainty for a future event. We showed that by leveraging three independent components, untrusted players operating under the Root network, the renode layer contains the set of smart contracts that allow users to interact with the relaying network and the relayers themselves, and finally, unrestricted access to decentralized storage. The protocol can provide distributed solving and can achieve anonymous verification without requiring any middle party or trusted guarantor with a single point of failure. 0VM is designed not to preclude arbitrary relayer services, ensuring no collusion between the Relayer and Root Network. The Root Network design can be easily extended to support any chain. During the initial days, the root Network is set up to be permissioned and later is updated to a permissionless. We presented a framework for achieving decentralised zero knowledge proof generation and for achieving anonymous verification ensuring furthur development of zero knowledge applications on the public blockchain..

References

[BLE17] Remco Bloemen, Leonid Logvinov, and Jacob Evans. "EIP-712: Ethereum typed structured data hashing and signing". In: *Ethereum Improvement Proposals* 712 (2017).

[Can+20] 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*. 2020, pp. 1769–1787.

[GG18] Rosario Gennaro and Steven Goldfeder. "Fast multiparty threshold ECDSA with fast trustless setup". In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 2018, pp. 1179–1194.

[GG20] Rosario Gennaro and Steven Goldfeder. "One round threshold ECDSA with identifiable abort". In: *Cryptology ePrint Archive* (2020).

[GMR98] Rosario Gennaro, Daniele Micciancio, and Tal Rabin. "An efficient non-interactive statistical zero-knowledge proof system for quasi-safe prime products". In: *Proceedings of the 5th ACM Conference on Computer and Communications Security*. 1998, pp. 67–72.

[Sch91] Claus-Peter Schnorr. “Efficient signature generation by smart cards”. In: *Journal of cryptology* 4.3 (1991), pp. 161–174.