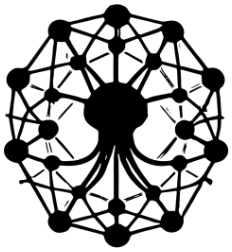

AQUARIUM

Anonymous QUAntum Resistant Internet and Untraceable Money

Abstract

The Anonymous and QUAntum Resistant Internet and Untraceable Money (AQUARIUM) project presents a decentralized, secure blockchain network and cryptocurrency that prioritizes privacy and post-quantum security. The account-based system features unlinkable confidential transactions and supports Turing-complete smart contracts. It uses a scalable succinct blockchain architecture supported by the Gasper Proof of Stake algorithm with sharding, and validated through off-chain recursive zero-knowledge proofs for account and smart-contract state transitions.

Transaction and communication privacy are further enhanced by a Sybil-attack resistant network anonymity layer, based on routing between multiple Riffle groups, and post-quantum cryptography to defend against future quantum computer decryption attempts. The anonymous network layer ensures network accessibility, even under restrictions and regulations.

Aquarium's community is self-governed by a built-in Decentralized Autonomous Organization (DAO) that controls self-development, software upgrades, coin distribution, and inflation rate. The AQUARIUM project aims to create a new generation of privacy-focused, censorship-free, self-administered internet by enabling various post-quantum secure and private blockchain applications.

1. Introduction

1.1 The Challenges of Blockchain Technology

The invention of blockchain, as demonstrated by Satoshi Nakamoto in the creation of Bitcoin [1], has led to significant advancements in the field of cryptocurrency and cryptography. However, several issues still persist in blockchain applications, including privacy, quantum resistance, scalability, environmentally sustainable and secure consensus protocols, blockchain succinctness and fair fund distribution.

The challenges facing blockchain technology are multifaceted and complex. Achieving a balance between decentralization, scalability, and security in a single blockchain architecture, known as the blockchain trilemma, is a significant challenge. Scaling a blockchain network, for instance, may require a trade-off in terms of decentralization or security.

Privacy has become a critical concern in the blockchain space, especially as the technology has gained mainstream attention. Projects like Zcash [2] have attempted to address privacy concerns by implementing zero-knowledge proofs and other privacy-enhancing technologies.

Protecting against quantum computing attacks [3], which threaten cryptographic primitives used in blockchain technology, such as elliptic curve cryptography, is also a major challenge. This poses a significant risk to the security of blockchain networks and the confidentiality of user data.

Addressing these challenges is crucial to developing a comprehensive, secure, private and scalable blockchain system that can meet users' needs and support digital transactions.

1.2 Combining Solutions for the Blockchain Trilemma

The AQUARIUM project addresses the blockchain trilemma by integrating several solutions to achieve a balance between decentralization, scalability, and security. It features unlinkable confidential transactions secured by post-quantum secure primitives, based on the weakest cryptographic assumptions.

The transaction model used in the AQUARIUM project is key to integrating these technologies. The off-chain proof system reduces the transaction size, and enables the use of heavy post-quantum zk-STARKs [4], while avoiding cross-shard communication complexity.

This includes the use of hash-based cryptography, not only for transactions but for the consensus algorithm as well. In particular, the Gasper [5] Proof of Stake algorithm (with RLMD-Ghost) has been specifically redesigned to be quantum-resistant. This algorithm is used in combination with sharding, which allows for the distribution of the transaction load across multiple parallel blockchains run by the same set of block producers and validators.

The AQUARIUM project also employs a Payment Channel Network (PCN) [6] to boost scalability and support efficient instant transactions. Additionally, succinct blockchain summaries based on STARK technology increase decentralization by simplifying the validation process.

1.3 Focus on Privacy

The AQUARIUM project places a strong emphasis on privacy, designing an anonymity network layer to provide uncensored access to the internet, bypassing network restrictions, and enabling a safer web, free of surveillance and censorship. To address the need for a decentralized, secure, and private internet, we have initiated the Anonymous and QUAntum Resistant Internet Shield (AQUARIS) project. The project is entangled with Aquarium, since secure private transactions and security against adaptive corruption of block producers, are dependent on a secure private network layer. Additionally, the anonymity network needs an integrated monetary system to defend against Sybil attacks and incentivize nodes to participate.

1.4 Potential Uses and Future of AQUARIUM

The final parts of the paper discuss the economics, politics, potential uses, and future developments of the AQUARIUM project. It has the potential to be a fundamental building block of a safer and more reliable internet and cryptocurrency, promoting privacy, decentralization, and social coordination through advanced technology. Potential uses include private network applications ranging from existing blockchain projects to entirely new concepts, due to its high level of security and privacy, and providing private and uncensored access to the internet. Future developments may include a decentralized data storage application and the OpiNION project, which aims to guarantee freedom of expression through onion routing privacy and decentralized autonomous organization governance.

2. The Aquarium Blockchain

2.1 Post-quantum security

One of the key features of Aquarium is its use of post-quantum secure cryptographic primitives. These include:

- SHA3-256bit Hash
- Winternitz chains signatures [7] based on SHA3-256
- Poseidon Hash [8]
- AES-128bit encryption standard
- Hash based DRBGs (deterministic random bit generators)
- XMSS (Extended Merkle Signature Scheme) [9], based on Winternitz chains with SHA3-256 for one-time signatures
- Zk-STARK (Zero-Knowledge Succinct Transparent Argument of Knowledge) based on Poseidon hashing to achieve better performance at producing recursive proofs
- STARK proofs for succinct blockchain descriptions based on SHA3-256 (STARKs based on SHA3 are slower to prove but very fast to verify)

The above cryptographic primitives are considered the most secure options currently available as they rely on the weakest cryptographic assumptions.

2.2 The Aquarium ledger

Aquarium's architecture addresses the scalability issue by keeping minimal information and operations on-chain and moving most procedures off-chain. Succinct arguments of knowledge, especially in their recursive form, have reduced the size of signatures and proofs of program execution to that of a single proof. As a result, data availability [10] has become the primary bottleneck for scalability in blockchain systems. To minimize the amount of required data availability, the Aquarium ledger consists only of transaction hashes. Techniques such as shrinking validation data (Merkle paths and large hash-based signatures) in a single proof per shard, maximize the utilization of the sharded architecture. Additionally, multiple transactions can be combined into a single hash through the use of Merkle roots signed by participating accounts, with each transaction's registered entry verifiable through a corresponding Merkle path. This design maximizes the benefits of sharding in a blockchain system.

In specific, the Aquarium book consists of accounts and Turing-complete smart contracts. The state of these accounts and contracts is updated with each transaction or execution cycle. The blockchain tracks account states by storing their hashes and contract transitions by storing their encrypted input data. Unlike the original UTXO architecture, which requires multiple signatures for a single transaction (one per UTXO), Aquarium's account-based protocol uses a single signature per transaction and only registers the hashes of the latest account states on transaction verification. This allows for the use of heavy, post-quantum secure XMSS signatures without consuming valuable blockchain space. All crucial information is kept private and stored off-chain, while the on-chain contract data is encrypted.

2.2.1 Basic concepts and definitions:

- The book is a Merkle tree containing the hashes of the latest states of all active accounts and pointers to the latest contract transitions, to prevent double spending.
- Every block contains the hashes of all recently updated accounts, the recently updated contract transition entries, and the root hash of the active accounts book.
- Block history is a Merkle tree of the hashes of all blocks.
- Accounts remain active (able to update their latest state hash on-chain and thus able to transact) until they are deleted.
- Account owners update the on-chain state hash of their accounts by emitting messages to the network for inclusion in a block. They can also execute a contract command by publishing an encrypted input to the contract and spending the appropriate gas amount.
- Gas is a transparent form of funds generated through coinbase transactions that can be used to pay fees for blockchain messages, or burned to increase a private balance.

Nodes need to store:

- All recent blocks and their contents, from the most recent block to the latest finalized block.
- The block history Merkle root
- The book (all latest account hashes and contract pointers)

Archive nodes need to store:

- The block history (block hashes, not the whole block contents).
- The encrypted contract inputs and contract pointer sequences.

Wallets store:

- The private off-chain part of the most recent state of their own account or contract data.
- A recursive zk-STARK proving the account's or contract's state validity and its inclusion in the block history. (A state can be proven as valid only if it is recursively based on another proven state.)

Private values
Private key seed
(Next) Private key
List of hidden destination addresses
Private Account Balance
List of past unexpired input payments
List of current input payments
List of current input proofs of validity
List of current output payments
Proof of account state validity

Public values
Account public key
Previous account state hash
Next XMSS signature public key
Gas balance
List of past unexpired gas inputs
List of current gas inputs
List of current gas outputs
Gas spent
Merkle hash of all private values
Merkle root hash of all public values

Array 1. *The account data structure*

2.2.2 The payment data structure

Amount:

The amount of balance moving from the source account to the destination account.

Script:

A set of conditions used as prerequisites to unlock the payment in the destination account. It is not Turing-complete.

Expiry block-height:

A block height after which this payment cannot be unlocked and it gets deleted.

Destination:

The destination account address. It remains hidden and confidential.

2.2.3 The account data structure

An account data structure in Aquarium contains the following private data:

Private key seed:

The seed for producing all key-pairs, destination addresses and the account public key.

(Next) Private key:

The next private key is derived by hashing the concatenation of the previous account state's private key with the initial seed. The next XMSS signature private key is derived from it through a DRBG.

List of hidden destination addresses:

The list of hidden addresses to be exposed as destinations for input payments. They are deterministically derived through a verifiable DRBG from the initial account private key seed and the account context (consisting of a valid creation timestamp and the shard id). The latter prevents an attacker from using the same key-pair, to recreate an account with the same destination address in order to receive a transaction twice.

Private account balance:

This variable represents the available funds. It is calculated using the following formula:

Balance = Past inputs + current inputs – past outputs – current outputs

List of past unexpired input payments:

This list is produced by adding elements of the previous account state's past and current input payments and deleting all expired elements.

List of current input payments:

Every current input payment corresponds to an identical output payment of the sender account.

List of current input proofs of validity:

The list of zk-STARKs proving that for every element of the current input payments list there is a valid sender account state or contract transition included in the block history Merkle tree, which includes an unexpired output payment:

- with the correct amount
- directed to a destination included in the hidden addresses list
- with satisfied script conditions

List of current output payments:

The list of the output payments to be sent from this source account to the destination accounts. When the new account state hash is published on the blockchain, the sender account could privately send a proof of payment validity to the receiver account to complete a transaction.

Proof of account state validity:

The proof of account state validity is a recursive zk-STARK which proves:

- A new entry of account creation is published in a block that is included in the block history.

OR Correctness of the transition from the previous account state (the hash of which is included in the book) to the current account state:

- Validity of the previous account state's zk-STARK, recursively
- Inclusion in a block inside the block history
- Correctness of the destination derivation process
- Correctness of the private account balance
- Correctness of the input/output payment lists
- Correctness of the script executions
- Validity of the ZK-STARK proofs of validity of the current input payments, recursively

It is used to construct:

- The next state's validity zk-STARK recursively
- The zk-STARK proof of payment validity recursively, to be sent to the payee off-chain

An aquarium account data structure also contains the following public data:

Account's public key:

Permanent public key which defines account ID on Blockchain. It is derived once by hashing the initial private key seed, with inverted digit sequence.

Previous account state hash:

It is included in a block and inside the book (paired with the public key).

Next XMSS signature public key:

It is the root hash derived from the (Next) XMSS Private key.

Gas balance:

Available gas for paying block producers to include transactions on the blockchain. (Recall that gas is the transparent form of funds generated in coinbase transactions and is used to pay for the computational costs of executing transactions and smart contracts on the blockchain.)

List of past unexpired gas inputs:

These are past outputs from other sender accounts, like input payments, but public. They are stored to avoid double receiving of a payment.

List of current gas inputs:

These are current gas outputs from other sender accounts, like private input payments, but public.

List of current gas outputs:

The list of the output payments in gas to be sent from this source account to the destination accounts.

Gas spent:

The gas fee for the account transition to be included in the next block or burnt to create funds inside the private account balance.

Merkle hash of all private values:

Merkle hash of all private values except private keys

Account state Merkle root hash:

Root hash of all (private and public) values

Note: The validity of each of the above public elements is proven by a Merkle path of inclusion in the sender account state Merkle root and verified by nodes and block producers.

2.2.4 The on-chain message data structure

Type:

New account state/New contract transition

Account specific data:

- **Public account structure:** This data structure includes only the necessary elements. For instance, if there is no transfer of gas balance between accounts during a transaction, the relevant fields in the message will be left empty.
- **Merkle paths:** The paths proving the inclusion of the above fields in the new account state Merkle root, and the paths proving the inclusion of the previous values of the above fields in the previous account state Merkle root.

Contract specific data:

This data is only included during contract executions and consists of:

- **The contract program hash.** This field can be optionally registered in the first entry of each contract in the blockchain. Alternatively it can be included in the script field of the output payment which initiates the contract, inside an account state.
- **The previous contract entry pointer.**
- **The encrypted contract input blob.** This field includes the contract commands to be executed and their parameters, in the form of an AES encrypted blob.

Signature:

A valid signature from the account on the aforementioned data.

2.3 Scalable quantum resistant unlinkable confidential transactions

The Aquarium transaction protocol involves private, off-chain state transitions and public, on-chain messages to the network. In the Aquarium blockchain, transactions are processed through off-chain transfers of recursive zk-STARKs. These zk-STARKs serve as proofs to ensure the correctness of the off-chain account data structures and the inclusion of transferred UTXOs within them. This approach allows for efficient use of limited blockchain space, as most of the proof-verification process is conducted off-chain and privately between transacting parties.

Wallets initiate transactions by updating their private account state to include the transaction output and emitting a public message to register the Merkle root hash of the new account state on the blockchain. To complete the transaction, the payer must prove to the payee the inclusion of the transaction output in a valid account state registered on the chain through off-chain exchange of recursive zk-STARK proofs. These proofs are necessary for the payee to verify incoming transactions and construct the next proofs for output UTXOs.

Aquarium transactions are designed to be unlinkable and confidential, relying on off-chain, zero-knowledge proofs exchanged only between the transacting parties. This protects transaction metadata from being disclosed to third parties and conceals the sender's identity from the receiver. The only information that can be inferred is the update of the account's state prior to a specific block height, which is only accessible to the receiver. This provides high levels of privacy and anonymity for both parties involved in the transaction.

Finally, zk-STARKs also enable the addition of scripting capabilities to transactions, creating a lightweight, non-Turing-complete smart contract system that can verify the satisfaction of certain conditions on payments. This smart-contract functionality is similar to the cellular automata paradigm of RGB [11] used in the bitcoin network and enriches the transaction protocol's functionality.

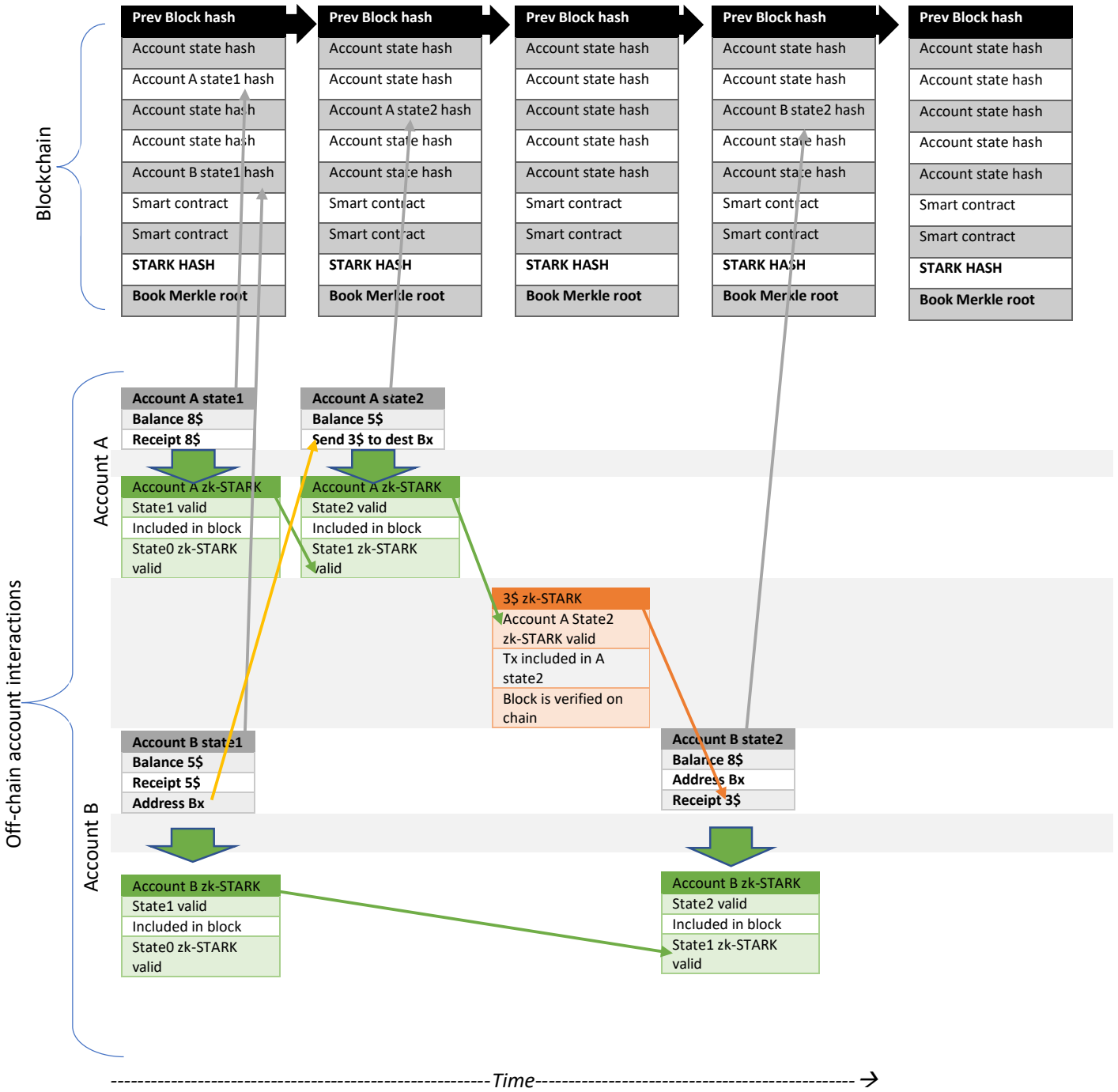

Figure 1. *The Aquarium Transactions Protocol. Depicted are white-grey arrays representing a part of the private account's state data, green boxes representing proofs of valid account states, and orange boxes representing proofs of valid transactions. Grey arrows symbolize signed public messages sent to the network, updating the latest state hashes on-chain. Green arrows represent proof construction from data, while the yellow arrow symbolizes an off-chain private message informing the payer about the payee's destination address. The orange arrow represents the off-chain private transfer of a transaction proof. It is assumed that State 0 of both accounts has a $0 balance.*

2.3.1 The Aquarium transaction protocol outline

Transactions consist of private off-chain state transitions and public on-chain messages to the network. These messages update the hash of the account's state. They follow the protocol outlined below:

1. Wallets initiate transactions by privately updating their account state. This state includes all the account input and output payments. To send an amount, a new account state is created, based on the previous state, which includes:

- an output payment of amount A to the destination of the receiver account
- the reduced balance value B_n (which is private, known only to the account owner), equal to the previous state balance B_{n-1} minus amount A

$$B_n = B_{n-1} - A$$

- the reduced gas balance G_n, equal to the previous gas balance G_{n-1} minus fees F

$$G_n = G_{n-1} – F$$

- the new key-pair for the next XMSS signature.

2. The sender wallet emits the Message, the public part of the transaction, which includes:

- the account public key
- the hash of the new account state
- a Merkle path from the previous XMSS public key to the previous state's hash
- a signature proving ownership of the previous state
- the gas balance update and the associated Merkle paths proving it

3. Block and message relay nodes instead of verifying the private data of the account state, are only able check the public data:

1. They verify the account’s previous state hash using the public key associated with it, which is included inside the book.
2. They verify ownership through XMSS signature verification
3. They check that transacting accounts own sufficient gas.

Note:

Publishing an invalid account state hash (with no valid zk-STARK proof) on-chain is actually allowed, but strongly disincentivized: While an account of this kind can continue to update its state hash until its gas balance runs out, it cannot send or receive payments, since the payee needs a recursive zk-STARK to prove account correctness.

4. When the new account state hash gets published on the blockchain, the sender wallet creates a recursive zk-STARK of account state correctness, which proves:

- the correctness of the new account transition: no payments are used twice and the balance B_n is equal to the previous balance B_{n-1} plus input payments ΣI, minus output payments ΣO, ensuring that no money is created out of thin air.

$$B_n = B_{n-1} + \Sigma I - \Sigma O$$

- the correctness of the script execution process
- inclusion of the current account state hash in a block registered in the block history.
- the validity of the zk-STARK that proved correctness of the previous account state and inclusion in a block, to expand the chain of recursive proofs from the account creation to the most recent account state

5. The payer wallet privately sends output payment data to the payee wallet, along with a recursive zk-STARK proving that:

- this output is included in an account state which is included in a block, inside the block history
- the zk-STARK proof of correctness for the payer's account state, corresponding to this block, was valid

6. The payee wallet includes output data into the receiver account input payments. The payee's account state is updated on the blockchain, by sending a message to the network, containing the information described in step 3.

7. Payee account state is updated and a recursive zk-STARK is constructed to prove:

- the correctness of the account transition
- The correctness of the script execution process
- the inclusion of the new state hash in the block history
- the validity of the previous state’s zk-STARK proof
- the validity of the zk-STARK proof of payment correctness constructed by the payer, as described in step 5

Note: It is possible for an account's state to be updated only when sending funds, consolidating incoming transactions and making outgoing transactions in a single step, reducing the on-chain procedures.

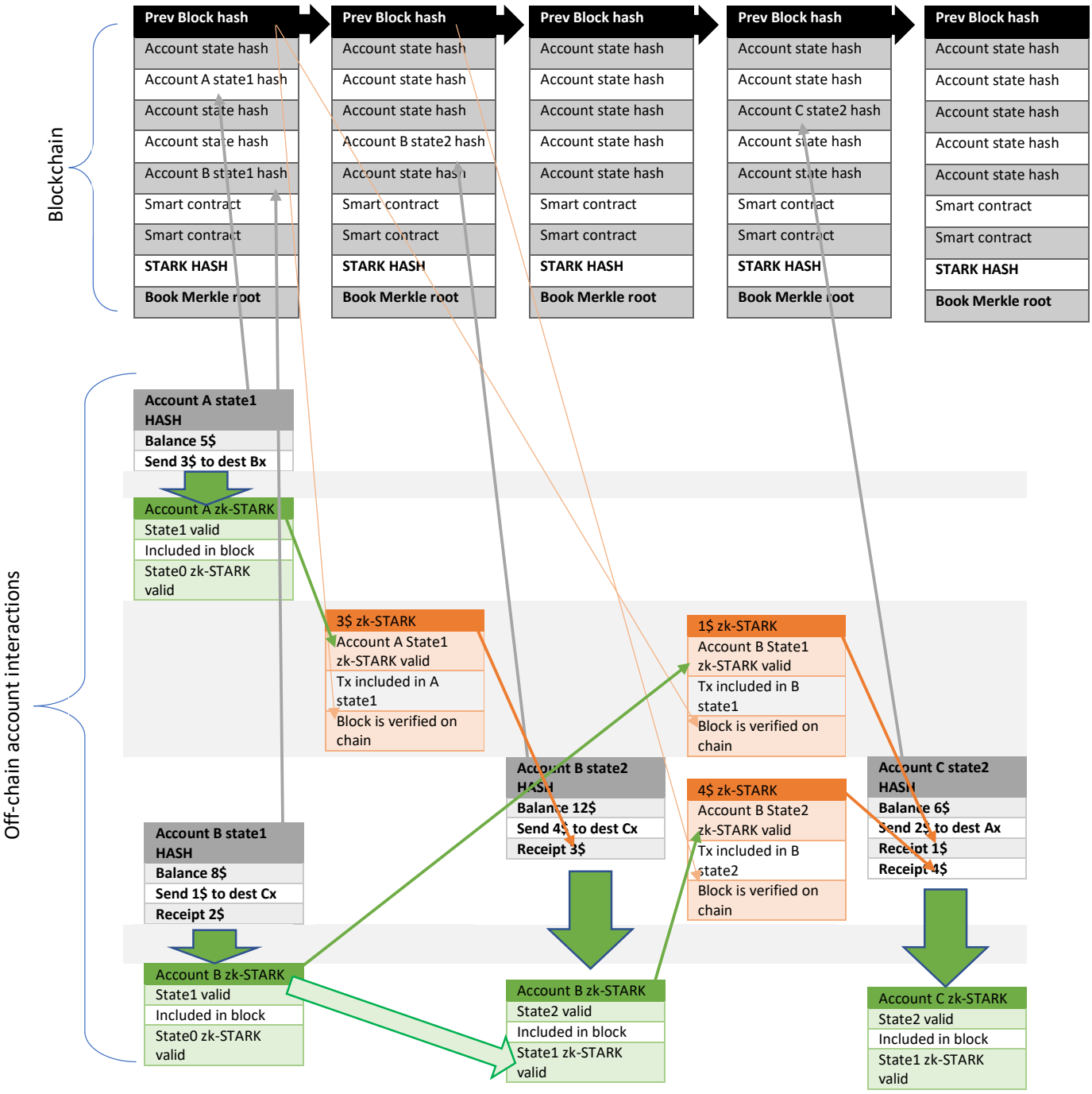

Figure 2. *Illustrating several Aquarium transactions. The white-gray arrays depict a portion of the private account state data, while the green boxes depict proof of valid account states. The orange boxes symbolize proof of valid transactions, and the grey arrows depict signed public messages sent to the network to update the latest state hashes on-chain. The green arrows represent proof construction from data, and the orange arrows represent off-chain private transfers of transaction proofs.*

2.3.2 Account creation and gas funding

The process of creating a new account in the Aquarium transaction protocol involves spending a certain amount of gas from an existing account. To create a new account, a user (account A) can either spend its own gas or request another user (account B) to create a new account (account C) on her behalf, by sending an unlinkable conditional payment to account B. This is a mechanism for anonymous account creation. The conditional payment ensures that the funds are returned to the original account (A) if the new account (C) is not created before a specified block height.

In any case, the creation of a new account also initiates a chain of recursive zk-STARK proofs of validity, which is the mechanism used to ensure the integrity of the account's state.

Additionally, a user (account A) can buy gas anonymously from another user (account B) to fund an existing account (account C), by following a similar procedure as described above. The user (account A) sends an unlinkable conditional payment to account B to purchase gas, ensuring that the funds are returned if the gas is not transferred to account C before a specified block height. This provides a mechanism for anonymous gas funding for an existing account.

2.3.3 Transaction aggregation

Transaction aggregation in the Aquarium ledger is a key feature that allows for the optimization of blockchain scalability. By combining multiple account state hashes into one single hash, through the use of their Merkle root signed by participating accounts, the size of the data that needs to be stored on-chain is reduced. The accounts involved in a batch of transactions are motivated to participate due to the reduction in transaction fees.

The size of a batch of transactions remains relatively unchanged as it still contains all signatures and Merkle paths for each transaction. The block producer aggregates the signatures and Merkle proofs from several batches and unbatched transactions into a single STARK proof, reducing the amount of data stored on-chain. The Merkle roots of the batches and a bitmap containing the index of the unspent account state of the hash, are the only information required to be stored on-chain, which provides data availability while keeping all crucial information private and stored off-chain.

Specifically, when registering a batch on the blockchain, nodes validate the signatures and Merkle paths of the batch against its Merkle root. Then, block producers create one STARK proving all of the contained information of all batches and single transactions they receive and they publish just the hashes of them and the STARK, instead of the validation data. In the book, the batch is stored by its hash and an index bitmap indicating which of the included hashes are spent. When an account state hash within a batch needs to be updated, the index bitmap changes to reflect the new situation and nodes can verify if any participant of the batch is attempting to double spend.

The construction of a batch of transactions is a non-standard, off-chain process that can involve different protocols, including centralized and decentralized approaches, depending on the needs and priorities of the parties involved. The standard protocol defines the format for the batch message, block entry, and book entry, and specifies the verification requirements for batches, including the verification of account transitions through the corresponding Merkle paths.

2.3.4 Transaction security and privacy

The Aquarium transaction protocol is designed to ensure the correctness, security, unlinkability, and confidentiality of transactions on the network. In this section, we will provide proofs of these properties.

Proposition 1.

***Security**: The Aquarium transaction protocol ensures that only the owner of an account can spend its balance.*

Proof: In order to update the state of an account on the Aquarium blockchain, a valid signature from the account's private key is required. An adversary without access to the private key of the account cannot produce a valid signature, and as a result, cannot modify the state of the account. Therefore, only the owner of the account, who has access to the private key, can spend the balance.

Proposition 2.

***Correctness**: The Aquarium transaction protocol prevents the creation of money out of thin air and double spending.*

Proof: The Aquarium transaction protocol stores all payments locally until they expire and enforces rules to ensure transaction correctness and prevent double payments. Furthermore, each destination address corresponding to a specific account's public key is unique to that account. The state validity zk-STARK proves the transition correctness, including the derivation of destinations from the seed and context through the DRBG. This ensures that the same destination address cannot be used by different accounts.

Therefore, an adversary cannot produce a valid proof of the state of an account with conflicting transactions or conflicting state inheritance. Without a valid proof of account correctness, a proof of payment correctness cannot be constructed. Consequently, payments without the associated proofs won't be accepted by other users because accounts without proven input payments cannot be considered correct and thus cannot construct proofs of validity for their output payments. Therefore, if the chain of recursive correctness proofs is broken the account becomes non-operational.

Proposition 3.

Unlinkability*: The Aquarium transaction protocol protects the identities of both the sender and the receiver by hiding transaction sources and destinations.*

Proof: The Aquarium transaction protocol conceals transaction input metadata within an encrypted off-chain data structure that is accompanied by a zero-knowledge proof, protecting the identity of the sender. Additionally, destination addresses are hidden and their correspondence to the public key can be proven by transition correctness proof chains, which are zero knowledge, protecting the identity of the receiver. The only information a sender could potentially reveal to a receiver is that the account's state update occurred prior to a specific block height, as the payment zk-STARK proof confirms its inclusion in a block with equal or earlier block height. Despite this, the Aquarium transaction protocol offers full unlinkability between transactions through the use of temporary anonymous accounts as payment proxies. This enables a payer to completely conceal identity metadata from the receiver, thereby avoiding statistical transaction linkage. As a result, it is not possible to link a transaction to a specific sender, ensuring the unlinkability of transactions.

Proposition 4.

Confidentiality*: The Aquarium transaction protocol ensures the confidentiality of transactions by hiding the actual transaction amounts on-chain.*

Proof: The Aquarium transaction protocol obscures transaction metadata through encrypted off-chain data transfers, thus preventing the disclosure of transaction amounts from being revealed on the blockchain.

2.4 Private Turing complete smart contracts

A Turing-complete smart contract system offers more functionality than the lightweight scripting embedded in an account-based transaction system.

Verifiability is a key requirement for a Turing-complete smart contract, as every account interacting with it must be able to verify its execution state data. Storing data completely off-chain is not an option as it would compromise verifiability if an adversary were to withhold data. As such, the input data for a contract is published on-chain while preserving its privacy through encryption.

Zk-STARK proofs are essential for verifying private, Turing-complete smart contracts with post-quantum security. However, if these proofs were stored on-chain, they would consume approximately 100KB of blockchain space per contract execution cycle. To efficiently use blockchain space, a more sophisticated smart contract architecture is needed, as outlined in the account-based scheme.

In particular, a Turing-complete smart contract is initiated with a declaration as a script in an output payment in an account state. To minimize the use of blockchain space, the on-chain entry only stores the transition data necessary to restore the latest contract program memory state from the previous contract entry. These transition data include the pointer to the previous entry or the initiating account state, and the encrypted blob with the input data.

A contract private off-chain data structure includes:

1. The contract program in Web Assembly
2. a set of AES keys to encrypt on-chain data shared among participant accounts
3. Contract private balance
4. List of contract output payments
5. the root hash of the previous transition data structure

A Blockchain entry of the contract transition, consists of:

1. A pointer to the previous transition of the contract
2. the encrypted input data

Private data
AES KEY
Program
Contract private balance
List of contract output payments
Previous transition hash

Public data
Program hash
*Previous transition
AES-encrypted input data (blob)

Array 2. *The private smart contract structure*

2.4.1 The private contract execution protocol outline

In Aquarium, a contract command is executed by a user's account spending gas and using its XMSS signature keys. This updates the contract's state transition on the blockchain. Any wallet expecting payment or interested in the execution results needs access to the contract program, the latest on-chain transition, and the encryption key set to read the contract input data. By scanning the blockchain and reading every transaction that stores an entry for a specific contract, the input data can be decrypted, and the program's execution can be tracked. The program discards irrelevant input data and verifies that the transaction has spent enough gas to support the contract command. If the input operation is successful, the contract's state is updated in the off-chain structure for those interested.

Invalid contract entries may be included in a block and add noise to the chain of valid contract updates. However, any wallet with access to the contract's program and key-set can access the most recent contract state by executing the contract program functions and their parameters included in every blockchain-registered entry. When the blob cannot be decrypted to valid values or the execution is invalid, the invalid transition is recognized as noise and removed from the chain of contract states.

Each contract participant creates a recursive zk-STARK proof of validity or invalidity for each previous on-chain contract entry by following the execution transitions on the chain, which is verifiable by contract participants. The correctness of the new contract transition must be proven: Contract transition correctness means that the contract command was executed correctly, no payments were used twice and that the balance B_n is equal to the previous balance B_{n-1} plus gas input payments ΣI, minus output payments ΣO, ensuring that no money was created out of thin air.

$$B_n = B_{n-1} + \Sigma I - \Sigma O$$

It is important to note that while the input payment amounts for the contract are publicly visible, the actual values of the contract balance and any output amounts or destinations to other accounts remain confidential.

Payment from an Account to a Contract:

The steps for executing a deposit contract command on Aquarium is as follows:

1. A user privately updates her account and the contract's state by executing a contract command and including a gas payment to the contract, constructing the appropriate encrypted blob. The user must have enough gas in her balance to support the on-chain operation and pay for the contract command.
2. The payer broadcasts a message to update her account and contract, including the new contract transition (the previous contract state and encrypted input data) and the gas amount being spent.
3. Nodes verify the account's signature and gas balance.
4. After inclusion in a block, contract participants decrypt the input data, apply the transition by executing the contract command, and construct a new zk-STARK proof of contract validity including the latest transition.

Note: To participate in a contract anonymously, users can exchange private funds for gas using proxy accounts, as described in 2.3.2.

Payment from a Contract to an Account:

A payment from the contract to an account destination updates the contract state on-chain and provides the payee with a recursive proof of payment validity, including the contract's and the participating account's state validity proof.

The steps for executing a withdrawal contract command on Aquarium is as follows:

1. A user privately updates her account and the contract's state by executing a contract command to initiate a payment to a specific account or other contract, constructing the appropriate encrypted blob. The user must have enough gas in her balance to support the on-chain operation.
2. The user broadcasts a message to update her account and contract, including the new contract transition (the previous contract state and encrypted input) and the gas amount being spent.
3. Nodes verify the account's signature and gas balance.
4. After the transaction has been included in a block, contract participants decrypt the input, apply the transition by executing the contract command, and construct a new zk-STARK proof of contract validity, including the latest transition.
5. The user generates a proof of payment validity using the proof of contract validity and the validity proof of the participating account's state, recursively.
6. Finally the user sends this payment and its proof to the receiver. This payment update reflects the contract's state on-chain.

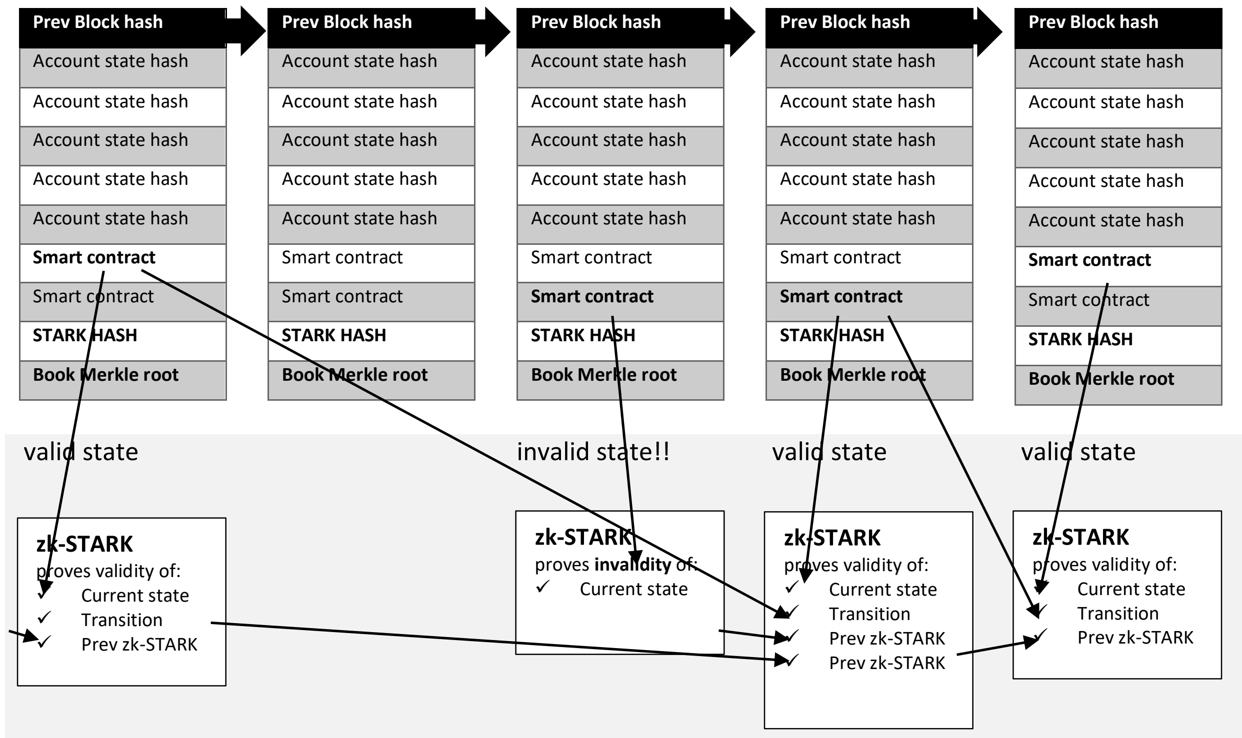

Figure 3. *A private contract execution chain. It contains an invalid state.*

2.4.2 Forward secrecy

Forward secrecy is an important feature in the execution of public functions of a private contract to ensure the security of sensitive data. It is achieved through the use of key-evolving schemes that prevent past data from being accessible to accounts executing the contract.

One example of a key-evolving scheme is the substitution of the AES key with its hash in each contract execution cycle. This ensures that new participants have access only to the new key and are unable to access previous contract states.

More complex key-evolving schemes can be implemented, such as deriving multiple keys from the previous key using a Deterministic Random Bit Generator (DRBG). One key is provided to the new participants, but if they are no longer involved in the contract, the next state update uses an alternative key, effectively locking them out of the contract and ensuring the confidentiality of past data.

The Implementation of forward secrecy is crucial for maintaining the privacy and security of sensitive data in a contract, and it is an essential aspect of contract execution privacy.

2.4.3 Privacy-focused dApps and Turing-completeness

It is known that dApps (decentralized applications) can be created by interconnecting transparent, Turing-complete contracts. The same can be achieved with private contracts, as long as they are managed by a common group of contract members who have access to all of the contracts' encryption keys.

In order for multiple private contracts to connect and form a Turing-complete dApp, their programs must be able to execute using both the input data of both contracts and they must be written in such a way that they can understand how the input data affects each contract and how the outputs of one program can provide inputs for the other.

In particular, cooperative contract members execute all interconnected contracts by following all recorded transitions on-chain and recovering their latest states. Consequently, each contract can refer to the state of any other contract, and a contract command can also execute another contract's command, in the aforementioned setting where all members of these contracts have access to all programs and keys.

When it comes to a dApp consisting of contracts managed by different groups, with different encryption keys, the situation becomes more complex. To achieve Turing-completeness in this setting, a coordinator contract is used to facilitate the exchange of funds and messages between different private contracts. The coordinator's keys must be available to all participants of the underlying communicating contracts.

The coordinator contract handles all Input data and functions that affect more than one participating contract in the dApp. Then, the other contracts interpret the data, with results only available to them through separate encryption keys, executing themselves individually and privately.

For example, imagine a dApp for selling lemons and oranges from different farms. There is a coordinator contract that sends the income from lemon sales to one farm and the income from orange sales to the other. Then, members of each farm execute their private contracts privately to share the earnings among them. In this example, members from both farms can decode the dApp's input data for the coordinator contract, but the lemon contract participants have no access to the orange contract, and vice versa. This means that one group has no information about the other group's members' wealth, they only know the total amount the other farm received.

Proposition 5:

A collection of interconnected private contracts on Aquarium can form a Turing-complete dApp.

Proof:

Recall that a Turing-complete system is one that can simulate a Turing machine and can perform any computation that can be expressed as an algorithm. Each private contract on Aquarium has the capability to perform computations by executing a Web Assembly program and updating its state based on input data, which is recorded on the blockchain in an encrypted form and is therefore only accessible to members with the corresponding encryption keys.

On Aquarium, interconnected private contracts can share input data and intermediate results through a common coordinator contract, which can pass the data to the private contracts, enabling them to perform coordinated computations. Since Web Assembly is a Turing-complete language, by executing a series of interconnected private contract functions on Aquarium, in a similar way to how subroutines are used in traditional Web Assembly programs, it is possible to simulate the behavior of a Turing machine. Therefore, a collection of interconnected private contracts on Aquarium can form a Turing-complete dApp.

Proposition 6:

A dApp on Aquarium that is composed of private contracts managed by different groups with different encryption keys will be able to maintain the privacy of each group, for data not affecting the contract execution in other groups.

Proof:

Each private contract on Aquarium encrypts its input data and output results with a unique encryption key that is only accessible to its members. Interconnected private contracts share input data and any intermediate results through a common coordinator contract encrypted by a key known to all members of the communicating contracts. The coordinator contract is responsible for passing input data and intermediate results to the private contracts, but it does not have access to the decryption keys for the private data of the other contracts. Therefore, the private data of each group remains encrypted and inaccessible to any other group. The coordinator contract only has access to the shared inputs of each contract in an encrypted form and only the participants of the contract can decrypt and understand the data. Therefore, the dApp maintains the privacy of each group and ensures that some data is accessible only to the intended participants.

2.5 Proof of Stake

Aquarium Blockchain uses a cutting-edge proof-of-stake protocol based on Gasper with RLMD-GHOST [12]. This protocol offers several benefits over traditional proof-of-stake systems, including increased decentralization, sharding optimization, and both BFT (Byzantine Fault Tolerance) [13] transaction finality and dynamic availability.

2.5.1 Post-quantum secure Gasper

One of the key features of Aquarium's Gasper variant is its use of non-aggregatable, post-quantum secure cryptographic primitives, specifically XMSS for block production and Winternitz chain signatures for attestations. These signatures are considered the most secure against quantum attacks, making the Aquarium blockchain highly resistant to quantum computing threats.

A quantum-secure, decentralized implementation of Gasper is challenging compared to weaker, less secure or less decentralized proof-of-stake protocols. This is because Gasper demands thousands of attestations per block and those attestations typically consist of aggregatable BLS signatures in current implementations.

The signature scheme used in Aquarium attestations is based on signatures from two Winternitz chains of 256 SHA3-256 repetitions (64 bytes per signature). This scheme can sign messages up to 8 bits in length. However, a single signature on an 8-bit hash does not provide security, as the adversary could find a colliding message for that hash with a probability of 1/256.

To address this issue, Aquarium uses a multi-signer approach. In this approach, multiple independent validators are considered to be cooperating in the signing process. Each validator computes the hash of the concatenation of her public key with the common message to be signed (in this case, the block hash to be attested), and then signs the first byte of this hash.

This multi-signer approach significantly increases the security of the protocol. For over 1000 validators assigned to every slot, the probability of finding a second block hash that corresponds to the signatures of over 1/20 of the attesters is negligible.

In Ethereum, a block is considered justified if it is signed by more than 2/3 of the validators. In Aquarium, a block is considered justified if it is signed by more than 2/3 + 1/20 of the validators. This threshold is adjusted upward to maintain the property that 1/3 of the validators should be slashable in order to subvert a finalized block. This may result in a slight trade-off in terms of the network’s ability to finalize blocks when between 2/3 and 2/3 + 1/20 of the validators are online, but it offers increased long-term security against potential quantum attacks.

To maintain practical block sizes and keep the number of attestations per block between 1000 and 4000, the length of the epoch can be extended to 64 or more slots, depending on the size of the validator set. This ensures that the total overhead on the beacon chain remains between 64KB and 256KB per slot.

Proposition 7:

For large numbers of attestations n (e.g. n>1000), the probability of an attacker successfully forging a counterfeit block hash b' sharing k=n/20 or more identical attestations with the original block hash b, is negligibly small.

Proof:

SHA-3 256-bit is a collision-resistant and pre-image resistant hash function. This means that it is computationally infeasible for an attacker to find two distinct messages m1 and m2 such that H(m1)=H(m2) or to find a message m' such that H(m') = H(m) given only the value H(m), where H(x) denotes the SHA3-256 hash of x. Since the attester signs only the first byte of the hash of the concatenation of her public key p with the block hash b, the probability P of an adversary successfully constructing another block hash b' with an identical attestation from the same attester is 1/256.

$$P\left(S\big(H(b\ XOR\ p)\big) = S\big(H(b'\ XOR\ p)\big)\right) = \frac{1}{256}$$

The probability P of an attacker constructing a counterfeit block hash b' that shares k identical attestations from k different attesters, with the original block hash b, is given by the binomial probability formula:

$$P(k) = \left(\frac{n!}{(k! * (n-k)!)}\right) * \left(\frac{1}{256}\right)^{k} * \left(1 - \frac{1}{256}\right)^{n-k}$$

where n is the number of attestations to the original block hash b and k is the number of common attestations with the forged block hash b'. In this case, we are considering the specific case where k > n/20.

$$P\left(k > \frac{n}{20}\right) = \Sigma_{k=\frac{n}{20}}^{n} \left(\left(\frac{n!}{(k! * (n-k)!)}\right) * \left(\frac{1}{256}\right)^{k} * \left(1 - \frac{1}{256}\right)^{n-k}\right)$$

As n increases, the value of the binomial coefficient increases, while the probability of success (1/256) remains constant. However, as k is a fixed value of n/20, the binomial probability P(k) decreases as n increases. For large values of n (e.g. n>1000), the binomial probability P(k) becomes significantly small and can be considered negligible. For example, when n = 1000, the probability P of k > n/20 is equal to $6.997 \bullet 10^{-39}$.

In summary, due to the collision resistance of the sha3 256-bit hash function, it is computationally infeasible for an attacker to construct a counterfeit block hash b' that shares k = n/20 or more attestations with block hash b, for large values of n, making the proposed scheme secure.

2.5.2 Sharding

The Aquarium project is working towards scalability through sharding, which is the division of a blockchain's workload, network load, and storage into shards. Ethereum's upcoming sharding upgrade, only focuses on data sharding which is based on Data Availability Sampling (DAS) technique [14]. Data sharding can bring a block size of 16MB with full data availability every 12 seconds. Data Availability Sampling involves erasure coding [15], which can be verified through polynomial commitments and data availability verification distribution to committees. The main blockchain in Aquarium which supports full sharding, is connected to multiple shard chains, as shown in the initial Eth2 proposal.

Block attesters, who validate transactions and produce blocks, are randomly assigned to produce blocks for the shard chains in Aquarium. They then post cross-links of the shard blocks on the main chain. To ensure quantum secure data availability for the shard chains, Aquarium adapts the Data Availability Sampling scheme, now based on STARKs instead of polynomial commitments, to prove the erasure coding.

A key difference between Aquarium's sharding and Ethereum's Is the ability for zero-knowledge cross-shard transactions, eliminating the need for inter-shard communication mechanisms. Communication between accounts happens off-chain and each shard only tracks gas balances and signatures, allowing the number of shards to dynamically increase as they reach their maximum transaction capacity for adjustable scalability.

Shard blocks in Aquarium are constructed using message data such as account state updates or contract transitions, while main blocks are made up of signed cross-links and attestations. The shard blocks include the previous shard block hash, the current Merkle root hash of the shard's part of the book, and recent account state hashes and contract encrypted inputs committed by messages. Main chain blocks, on the other hand, contain the previous main block hash, the book root hash, and the hashes of all cross-linked shard blocks. To validate the shard blocks, Aquarium uses message validation data such as Merkle path proofs and XMSS signatures to create STARK proofs of validity.

Transactions in Aquarium occur off-chain through zero-knowledge proofs of inclusion in the block history. The sender includes the transaction in their new data structure and sends a message to the shard chain's relay nodes. The shard block producer includes the new account state hash in a new shard block and cross-links it to the main chain. The sender then sends proof of payment validity to the receiver, who checks it against the main chain's block history to include the payment in their account data structure. To prevent double-spend attacks, accounts with the same destination in different shards are not allowed, as the destination is context-aware and dependent on the shard where the account is located. This mechanism allows transactions to occur without the need for a wallet to be aware of other shard contents, ensuring that transaction information about the source or destination's shard is not leaked.

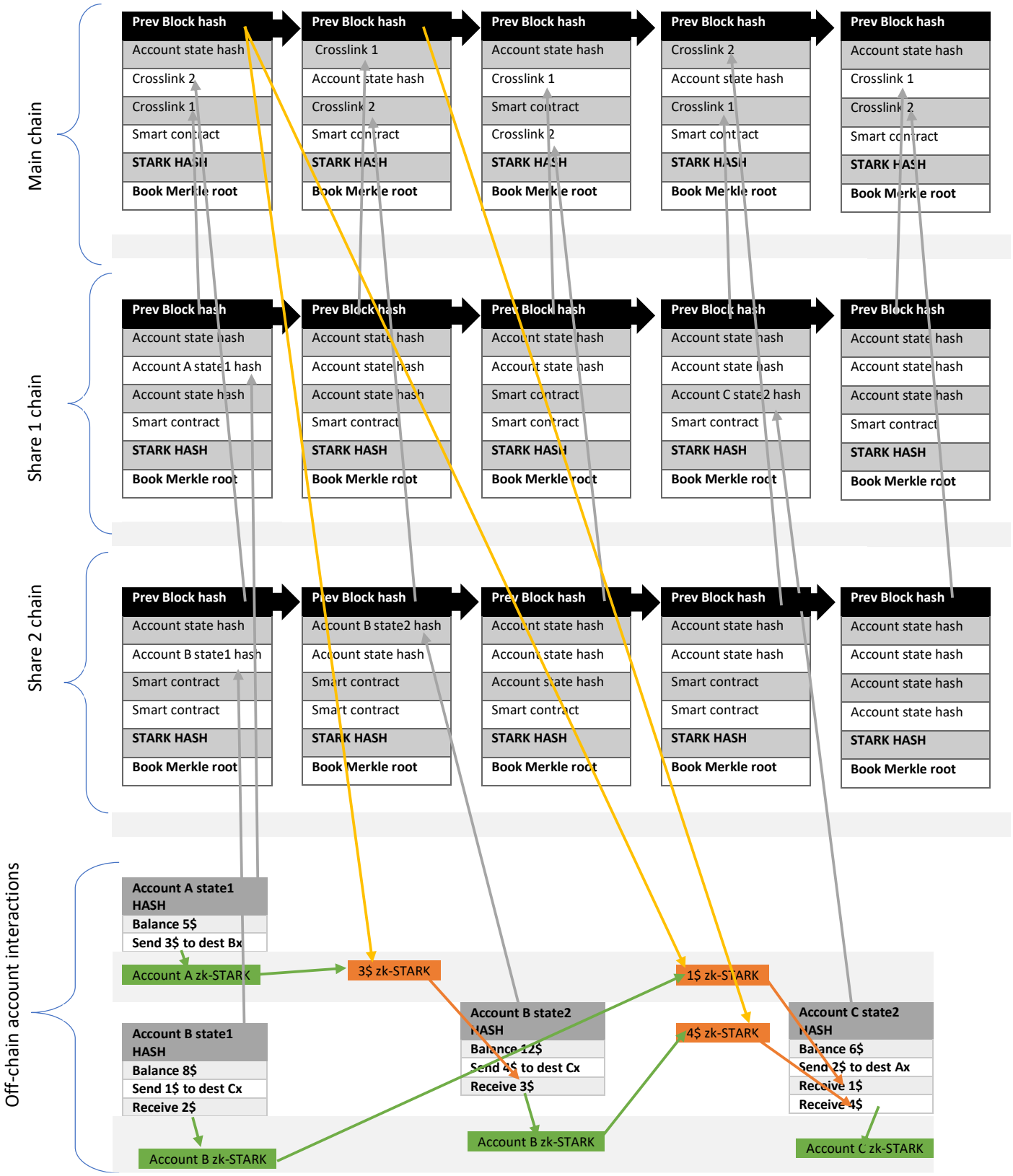

Figure *4.* *Simplified Aquarium Transactions between Shards. White-gray arrays symbolize a portion of the private account's state data. Green boxes indicate proofs of valid account states. Orange boxes symbolize proofs of valid transactions. Gray arrows depict signed public messages sent to the network to update the latest state hashes on the chain. Green arrows represent proof construction from data. Golden arrows represent proof of inclusion in a block. Orange arrows represent off-chain, private transfer of transaction proofs.*

Summary of the Aquarium transaction protocol supporting sharding:

1. The sender incorporates the transaction into a new data structure.
2. The sender sends a message to relay nodes on the shard chain where the account resides.
3. A shard block producer verifies the new account state hash and adds it to a shard block linked to the main chain.
4. The sender creates a proof of payment validity and sends it to the receiver.
5. After the block is finalized, the receiver on another shard verifies the proof against the main chain block history, adding the payment to their own data structure.

Summary of the modified proof of payment validity for sharding:

- Validity of the account state confirmed through a recursive zk-STARK.
- Inclusion of the payment within the account state.
- Recursive zero-knowledge proof of the account state's inclusion in a shard block linked to the main chain (without revealing which shard chain).
- Recursive zero-knowledge proof of the cross-link's inclusion in a main chain block.

The scalability of Aquarium's sharding is substantial due to the limited on-chain data required. With 64 shard blocks, each with 256 KB of space and even if a STARK proof takes up 100 KB per shard block, there would still be more than 150K available space in each shard block and 9.5 MB available in total. Without transaction batching, there is space for 300K transactions per block (25 KTPS) with a 32-byte hash per transaction.

With transaction batching, the TPS rate depends on batch size. Assuming 128 transactions per batch with 48 bytes each (32-byte hash and 16-byte bitmap), the TPS rate would be around 1.5 MTPS, surpassing centralized solutions.

However, downloading and verifying transaction batches is a demanding task for block producers. In high-throughput scenarios, this task should be handled by specialized machines, with block producers only responsible for verifying and signing their proofs.

2.5.3 Blockchain immutability

The Aquarium initiative aspires to develop an immutable blockchain resistant to quantum computing attacks. To accomplish this goal, the system must have a robust defense mechanism against long-range attacks. One proposed solution is the Ouroboros key-evolving scheme [16], which involves the removal of old keys. However, this approach is vulnerable to a supermajority of selfish users who may keep backups of old keys and sell them to attackers in the future.

To address this vulnerability, the block rewards on the Aquarium blockchain are gradually released. Initially, only half of the reward is accessible, and the remainder is halved each year. This means that a reward cannot be spent all at once, effectively deterring validators from selling old keys to attackers since this would negatively impact their unspent balance. Validators cannot alter the private keys for their locked block rewards, so they cannot sell them. Their financial incentive is to delete their old keys to protect their locked balance from being invalidated.

By having block rewards locked for an extended period, the possibility of leaked keys is significantly reduced, making a long-range attack more expensive and therefore infeasible. This leads to an immutable blockchain. Technically, this is implemented by constructing a large XMSS tree, with its Merkle root serving as a common public key for all XMSS signatures required by the validator over the lifetime of a single reward, and linking the reward to a specific validator.

2.5.4 Blockchain succintness

The Aquarium project aims to make blockchain validation simpler, increase decentralization, and reduce the amount of data needed to be stored and processed, by creating periodic, concise summaries of the blockchain. These summaries, constructed by block producers, provide light nodes with a condensed history of the blockchain, enabling them to determine the legitimate blockchain by comparing different forks using the summaries.

The Aquarium project uses STARKs Instead of SNARKs for its proof system, which differs from the Mina protocol [17], and its proof of stake algorithm is adapted to Gasper. To adapt this concept to Gasper, the BFT finalized sections of the chain with the smallest validator set are equivalent to the chain sections with the lowest density in Ouroboros Samisika.

The validity of the blockchain summary is proven through recursive STARKs, which verify the following for each block:

• The validity of the block producer's signature

• The validity of shard STARKs, which prove the identity of each shard block producer and the validity of signatures or Merkle paths of included messages

• The validity of shard block interlinks

• The validity of the active account and contract (book) tree reorganization

• The validity of the previous block's STARK proof, and therefore the validity of the previous blockchain description

Block producers receive additional rewards for publishing these proofs quickly, and this incentivizes them to minimize the latency of proof creation, potentially leading to a collaborative parallel STARK generation process. This is similar to mining pools, but without the resource-intensive competition of Proof of Work.

This succinct design benefits both light nodes and full nodes. Light nodes only need to download the latest blockchain proofs and signatures and to validate the availability of all data, while the actual validity of the data is proven by the STARK proof. Full nodes benefit from being able to delete unnecessary past block data.

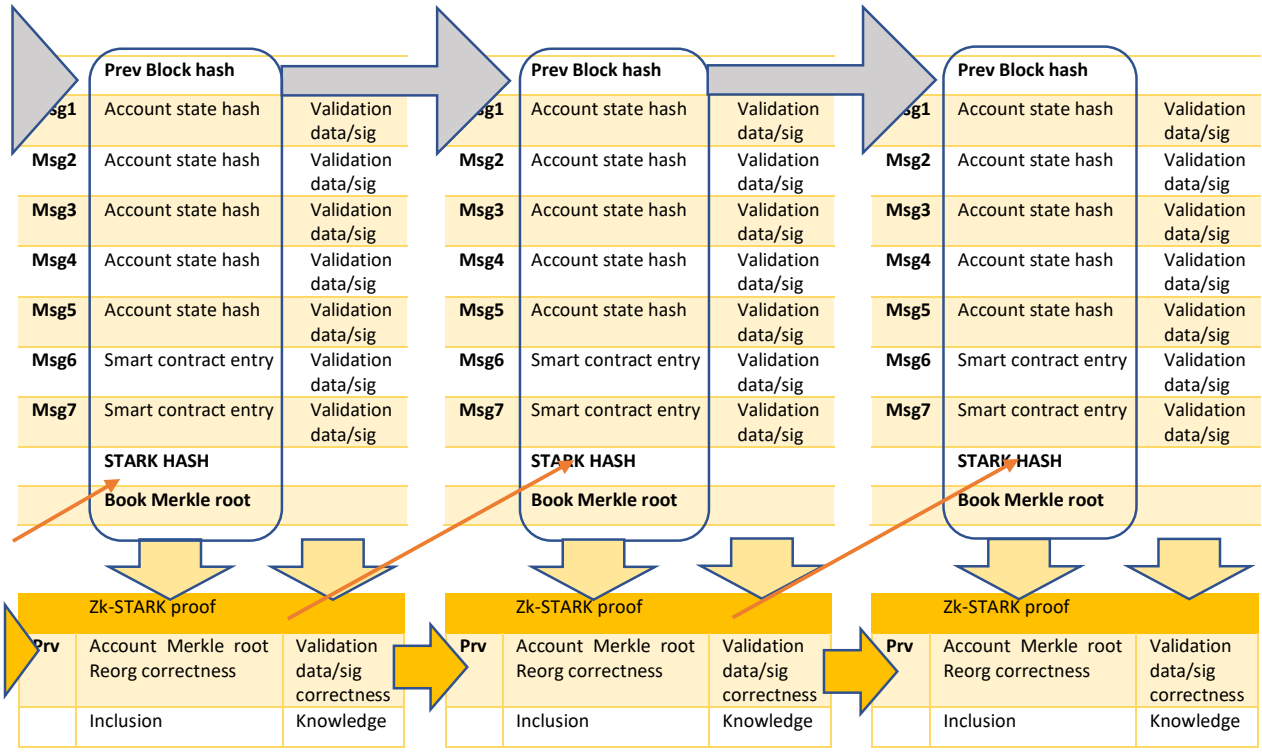

Figure 5. *The Succinct Blockchain – A visual representation where round rectangles represent blocks, and orange boxes represent proofs.*

2.5.5 Egalitarian collaborative staking

The Aquarium project has a mechanism for collaborative staking that utilizes secure multi-party computation (MPC) with the BGW scheme (based on Shamir secret sharing) [18]. Collaborative staking involves the computation of entire Winternitz hash chains, with shares stored locally by all parties involved. When it comes time to sign, the wallets open their shares for the relevant hash pairs in the Winternitz chains. The MPC process for deriving signing shares and public keys from private keys requires multiple communication rounds and thus has a high latency, but it can be optimized by computing multiple chains in parallel. On the other hand, collaborative signing is quick and efficient.

Low-balance wallets can participate in the validation process collectively by utilizing MPC to create the staking account keys and generating collaborative signatures by opening their shares. The block rewards are allocated to a contract shared among the cooperative wallets in proportion to their contributions in the staking process.

Light wallets can easily join the staking process with trustless light nodes, which only download block hashes from other nodes and request succinct blockchain descriptions (STARK proofs) to resolve disagreements among nodes. These wallets run on light nodes and collaborate with at least one full-node staking wallet to provide the necessary STARK proofs as needed. This approach ensures an equal distribution of rewards among Aquarium users and avoids the dangers of centralized solutions like conventional stake pools.

2.6 The payment Channel Network

The payment channel network (PCN) in Aquarium is designed to address the latency issue in transaction finalization, which is a drawback of the first layer of the platform. To bypass the capital bottlenecks, which is a common issue in PCNs, the scalability of the first layer makes it possible to enable frequent, low-cost on-chain reorganization of the payment channels, preserving the financial balance of the network and making it more effective and efficient.

The PCN consists of Virtual Channels [19], which are payment channels constructed based on the Generalized Channels scheme [20]. These channels allow for the transfer of Turing-complete functionality from the first layer to the second, while preventing transaction linkage and preserving the anonymity of transactions by redirecting and rerouting payments without leaving any traces on the blockchain.

However, if a Virtual Channel collapses due to misbehavior, it may trigger on-chain channel closures. In other cryptocurrencies, this can result in the exposure of accounts, which undermines the anonymity of transactions. In Aquarium, the underlying smart contracts are anonymously funded and programmed to send funds back to predefined secret account destinations, preserving unlinkability.

Additionally, the PCN in Aquarium has a unique mechanism for channel closures, preventing channel closures during times when the blockchain is not constantly finalizing. This ensures that no member of a payment channel loses funds during potential network splits caused by disasters or other adverse events. In case of user abuse in a channel, the other user will only be able to close the channel when the network becomes reunited. This makes payment channels connecting members within the same network segment a reliable transaction medium even under extreme conditions.

3. Aquaris (Anonymous Quantum Resistant Internet Shield)

The issue of online anonymity has long been a concern for individuals seeking to protect their privacy and freedom of speech, and a subject of much research and discussion in the field of computer science and cryptography. The anonymity trilemma [21] refers to the trade-off between three essential features of anonymous communication systems: low latency, high bandwidth, and secure anonymity. Achieving all three of these features simultaneously is challenging, and most anonymous communication systems have fallen short in this regard.

Dining Cryptographers Networks (DC nets) [22] are one approach to achieving strong anonymity, but they are limited in scope as they only provide strong anonymity in small anonymity sets and have high bandwidth requirements. Verifiable shuffles such as Riffle [23] aim to provide group anonymity, but they face scalability challenges as they require an increase in latency or bandwidth to accommodate larger anonymity sets.

The widely used Onion Routing (Tor) network [24] and I2P are designed for low latency, but they offer only weak anonymity compared to DC nets and shuffles. They use onion routing to encrypt data and route it through multiple intermediary nodes, making it difficult for an observer to determine the origin and destination of the communication. Despite achieving enormous anonymity sets, Tor and I2P are vulnerable to Sybil attacks, where an adversary operates multiple nodes to control a large portion of the network, compromising its privacy. Tor and I2P architectures also leave them vulnerable to traffic analysis, a type of attack where an adversary tracks communication patterns to infer information about a user's activities.

Lokinet [25] addresses the issue of Sybil attacks by incorporating blockchain technology to incentivize nodes to operate honestly and prevent such attacks. Aquaris takes this a step further by offering provable anonymity in a large anonymity set (up to 10^6) and using post-quantum secure encryption to ensure communication remains secure in the face of quantum computers. The integration of blockchain technology in Aquaris, similar to Lokinet, prevents Sybil attacks and incentivizes nodes to operate securely and effectively. Onion tunnels with post-quantum secure encryption through Riffle groups, optimized for low latency, provides stronger security guarantees against resourceful adversaries.

3.1 Quantum resistance

Aquaris tunnels are established through the use of the Kyber Key Encapsulation Protocol [26] between the tunnel ends. This protocol is used to agree on a common AES-256 key for transmitting information. Authentication is based on the poly1305 [27] and chains of XMSS key-pairs. To achieve forward secrecy, the AES key is frequently changed and replaced by its hash. The bootstrapping of the hybrid Riffle shuffle in Aquaris uses a verifiable cryptographic shuffle, which is based on the Ring Learning with Errors (RLWE) scheme [28]. This scheme, along with the rest of the primitives used in the Riffle protocol, offers quantum security.

3.2 Aquaris nodes availability

Nodes earn funds from user payments. Users send payments to the entry groups, which then pay the exit groups. The payment policy is such that service is provided first and payment follows. This incentivizes anonymity nodes to offer open access to their services to low data usage users, who can change relay groups.

For low data usage, users can switch entry groups to avoid payments. However, for high data usage, this strategy is limited by the self-protection mechanism of the entry groups in the P2P network. The mechanism allows the entry groups to see the user's IP and deny access if they don't receive payments for three consecutive epochs.

3.3 Defense against Sybil attacks

To defend against a Sybil attack, in which an adversary operates multiple nodes in an attempt to de-anonymize users, the Aquaris network requires a validator to participate in staking within the Aquarium network. Operating an Aquaris server requires staking a significant amount of funds, making it cost-prohibitive for an adversary to gain access to a sufficient number of servers to compromise the network's anonymity. Conversely, operating only a few servers is incentivized by higher income generated from user payments.

However, concentration of servers in one individual or organization could potentially reduce the network's anonymity. To address this, Aquaris provides financial incentives for servers that declare they belong to the same party. These declarations have two implications:

1. These servers never participate in the same group.
2. These servers can run on the same node and serve as convergence points for different groups.

3.4 Low latency group anonymity: Riffle

Riffle is a privacy-enhancing protocol with proven strong anonymity, designed for secure and anonymous communication within groups. It is based on the anytrust model, which ensures that colluding servers do not have the power to compromise the privacy of the group if at least one server in the group is honest. To achieve its goals, Riffle implements two distinct protocols: the hybrid shuffle for sending and Private Information Retrieval (PIR) for receiving.

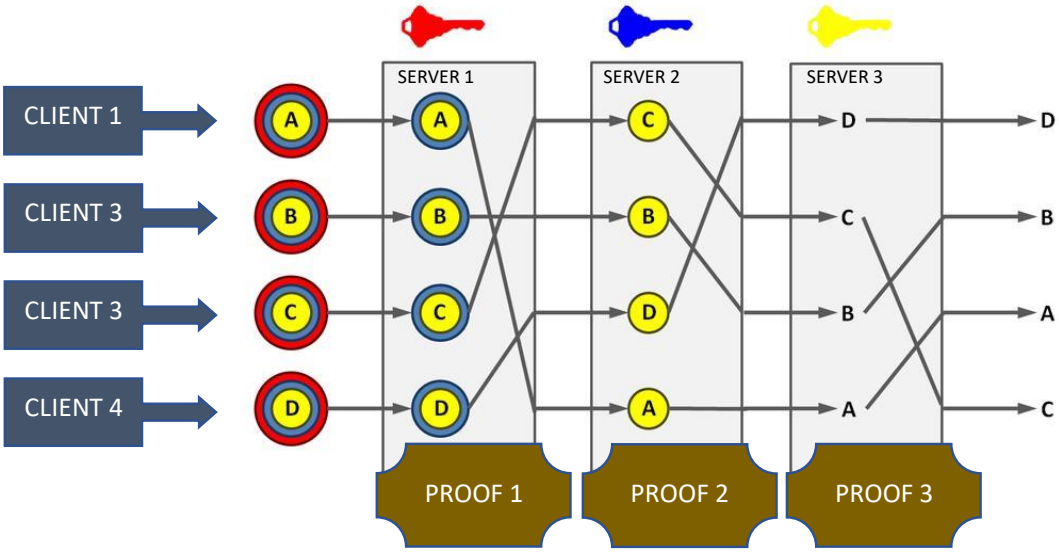

Figure 6. *Illustration of a simple verifiable shuffle with 3 servers and 4 clients. Each client sends a message with multiple layers of encryption depicted in different colours. In Rjiffle hybrid shuffle, public key encryption and zk-proofs are utilised only during the setup phase to establish authenticated encryption channels verified through the accusation protocol.*
Red de mezcla.png © 2008 Primepq Creative Commons Attribution-Share Alike 3.0 Unported license. https://creativecommons.org/licenses/by-sa/3.0/deed.en

The Hybrid Shuffle protocol consists of two distinct phases: the setup phase and the transmission phase. The Hybrid Shuffle protocol utilizes a slow verifiable shuffle based on public key cryptography during the setup phase and an efficient shuffle based on symmetric key cryptography during the transmission phase.

Setup Phase:

1. Clients encrypt their symmetric keys to be used with the final server using the final server's public key.
2. They then add the symmetric keys to be used with the previous server in the sequence and encrypt the ciphertext and new keys with the previous server's public key.
3. This process is repeated with each subsequent server until the first server is reached.
4. Clients send the final ciphertexts to the first server in the sequence.
5. The first server decrypts the outermost layer of encryption using its private key to recover the symmetric keys and permutes the remaining ciphertexts.
6. The decrypted ciphertexts are then sent to the second server, which performs the same operations.
7. The process continues until the final server is reached.

The correct functioning of each server during the setup phase is proven through zero-knowledge proofs.

Transmission Phase:

1. Clients use their secret symmetric keys to encrypt their messages with authenticated encryption. They encrypt their messages using the keys shared with the final server, then encrypt the ciphertext using the keys shared with the previous server, and the process is repeated.
2. The resulting onion ciphertexts are sent to the first server, which decrypts and authenticates them.
3. The first server then permutes the messages using the same permutation used in the setup phase and signs them.
4. The signed messages are then sent to the second server, which performs the same operations.
5. The process continues until the messages are decrypted by the final server and sent to their destination or uploaded to the replicated database of the servers.

If a server encounters unauthenticated messages or different permutations, it exposes the signed message of the previous server and runs the accusation protocol. Thus, the protocol ensured verifiability without requiring computationally intensive protocols during transmission phases.

It should be noted that the latency of the shuffle is proportional to the number of servers in the group. More servers result in more hops for the data to pass through in a serialized manner, leading to increased latency.

Private Information Retrieval (PIR) [29, 30] is a protocol that can be parallelized. All servers in the system share a replicated database, and when a client requests an entry from the database, they can cooperatively access it without knowing which entry they are accessing.

In Chor's PIR scheme specifically:

1. The client shares two secret bitmasks with each server. The first bitmask has a length equal to the number of entries in the database and the second has a length equal to the bitlength of each entry. If the first bitmasks are XORed together, they result in 0s in all positions except for one position, where there is a 1. This 1's position represents the position of the entry to be retrieved.
2. Each server constructs a list of all the entries of their own copy of the database, which correspond to positions with 1, in the first bitmask received from the client.
3. Each server XORs the ciphertext produced by every entry of the list constructed in 2 with each other, resulting in a ciphertext with the size of one entry.
4. Each result ciphertext is XORed again with the second secret bitmask.
5. The final ciphertexts of all servers are sent to one of them to be XORed together and sent to the client.
6. The client XORs this result with the second secret bitmasks and retrieves the hidden database entry without any of the servers knowing which entry was retrieved.

Due to the homomorphic properties of the XOR gate, the message can be retrieved intact and, if the bitmasks are randomly generated, the servers cannot distinguish which entry was retrieved unless they all collude.

As an optimization, the clients do not directly send the bitmasks to all servers. Instead, they send PRNG seeds to all servers except one, where they send a bitmask. This allows for low latency data retrieval regardless of the number of servers involved.

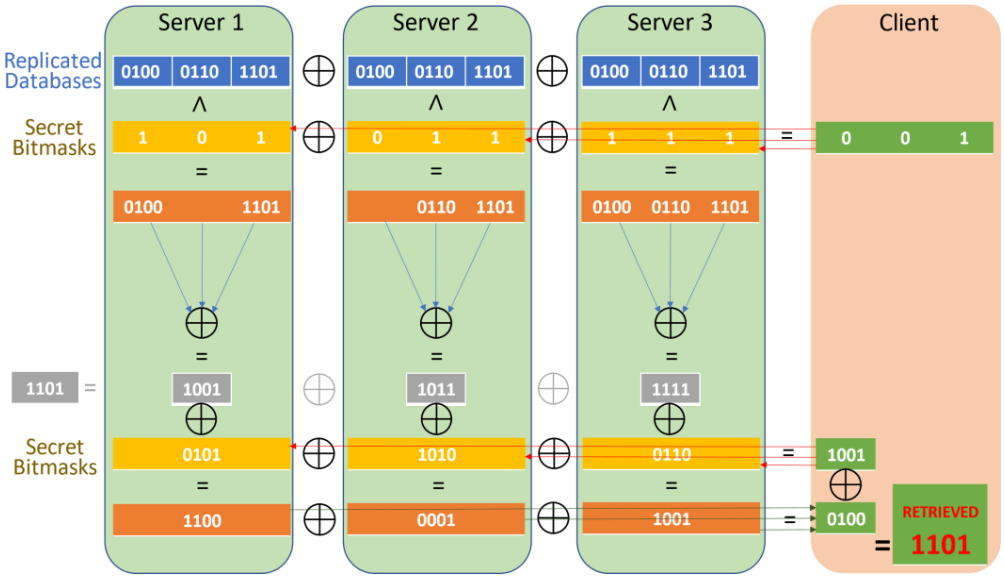

Figure 7. Illustrating an example run of Chor's Private Information Retrieval algorithm. The client (right) sends the secret bitmasks to the 3 servers (red arrows), to retrieve the 3d database entry. The bitmasks, depicted in gold, define the position of the entry to be retrieved (up) and hide this position from the servers (down). If the bitmasks (up) XORed give 1 at position n (depicted in green), the nth entry will be retrieved. Finally the servers send the combined ciphertexts to the client (green arrows) and the client XORes them with his secret bitmasks to retrieve the entry. The servers cannot distinguish which entry the client retrieved, unless they collude.

The Aquaris variant of Riffle introduces an inverted PIR process for low latency anonymous sending. This process begins with clients establishing anonymous channels with all servers through the hybrid shuffle of Riffle. The clients then run the PIR protocol through the shuffled channels, retrieving data from one replicated database to another shared database, instead of directly retrieving the data. They then publish the data to the first replicated database in the position from where they anonymously retrieve it to the second database. Since their retrieval requests come through the shuffle and PIR is provably anonymous, there is no way to correlate the positions of messages in the first database to the second.

In forward PIR, the retrieved message is masked with secret bitmasks to hide its contents from the servers using an XOR operation. The client must then decrypt it using the masks. In inverted PIR, the XOR operations sequence must be inverted as well. The client XORs the message with the secret bitmasks before sending it, so it appears unmasked in the second database. Messages in the second database can be replicated and anonymously retrieved within the group, or directly routed to external destinations without replication.

The sending procedure in Aquaris groups using the inverse PIR process is as follows:

1. All clients communicate with all servers and run the RLWE-based verifiable shuffle to share keys with the servers and establish permutations.
2. After all servers have received all clients' keys in permuted order, each client onion-encrypts another secret symmetric key to be anonymized. Specifically, they encrypt each key with the public key of the intended server, and then onion-encrypt it with all symmetric keys previously exchanged with the servers to route this new key through the anonymization shuffle.
3. Clients send their inverse PIR requests by onion-encrypting their seeds and bitmasks, beginning with the second symmetric key they shared with the intended server. The resulting data is then onion-encrypted with all previously exchanged symmetric keys and sent through the shuffle.
4. Each server decrypts one layer of encryption, permutes the resulting ciphertext, and sends it to the next server.
5. The last server sends the final ciphertexts to all servers. All servers decrypt their ciphertexts and recover the inverse PIR requests.
6. Clients encrypt one more PRNG seed with one server's public key and directly send it to be used as a traffic substitution in case the client's connection is lost.
7. To send a message with low latency using the inverse PIR scheme, clients encrypt the message by XORing it with all the second bitmasks they previously exchanged with the servers anonymously. They then upload the message to the first database.
8. If a client does not upload a message to the first database, the selected server puts a message in the database as dictated by the PRNG using the client's seed.
9. All servers perform the inverse PIR using the bitmasks and seeds they anonymously received previously from clients. They upload the retrieved messages to the intended positions in the second database, from where they can be privately retrieved within the group. Additionally, the messages can be routed outside the group according to the addresses they include.

A detailed diagram of the above procedure is placed on Appendix A.

This optimization provides several benefits, including low latency and improved security. Essential data, such as the destination address, is less vulnerable to deanonymization due to malicious misbehavior of the first server in the shuffle, as shuffling only occurs during the setup phase and does not include the essential data. Additionally, PIR allows clients to upload their encrypted message to the first database through any of the involved servers, eliminating the risk of a single point of failure. In the event of a lost connection, one of the servers can provide a PRNG meaningless flow to cover the absence, reducing communication overhead compared to sending properly encrypted cover packages through the shuffle in similar situations, as proposed in the Riffle presentation.

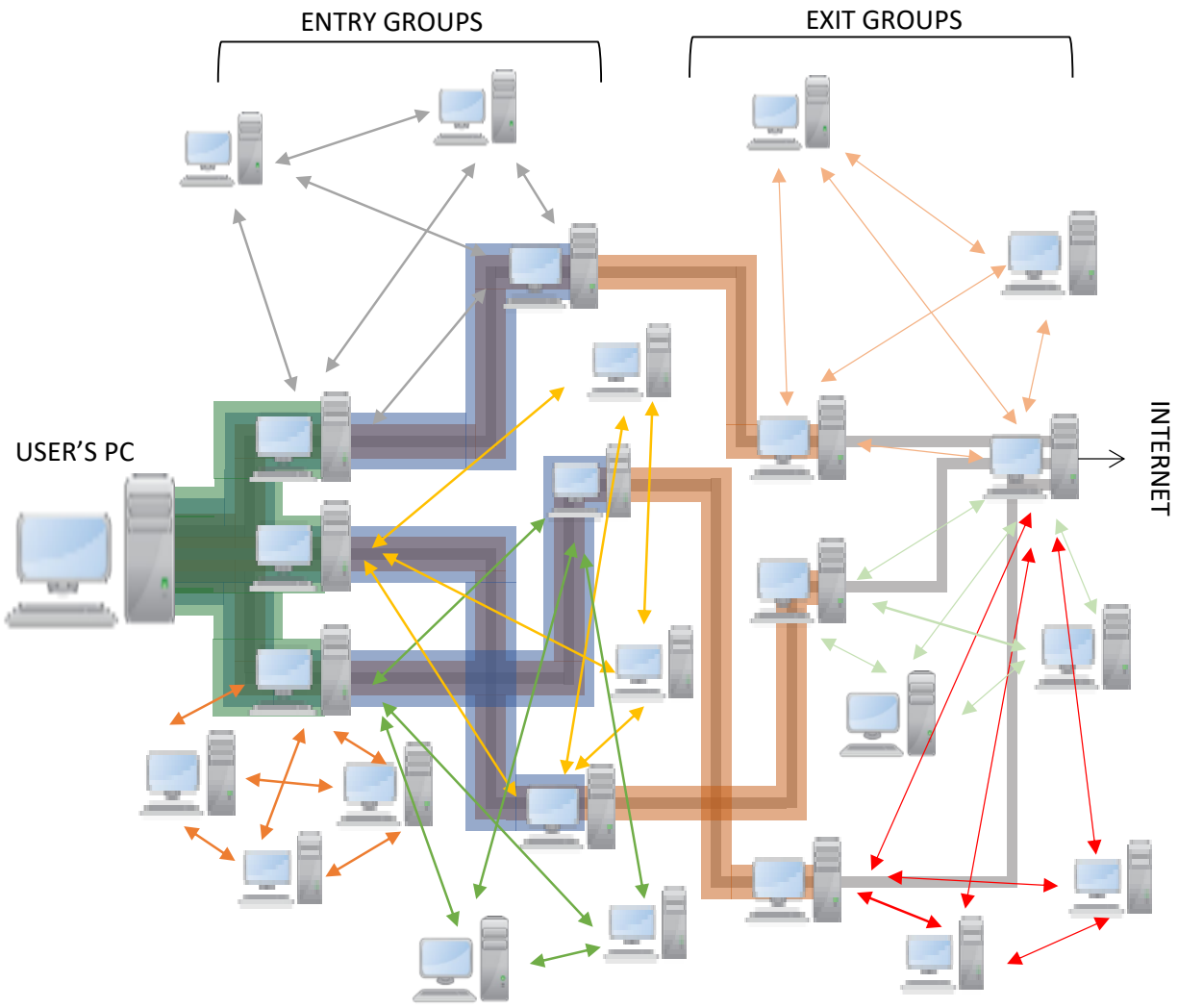

Figure 9. *The diagram depicts the Aquaris onion routing network through Riffle groups. Internal communication within the Riffle groups is illustrated using arrows of different colours, each representing a different group. The tunnels are represented as pipes of different colours. The final node is a convergence node, which participates in three groups simultaneously.*

3.5 From group anonymity to network anonymity

The Aquaris network consists of two layers of Riffle groups. Each group is made up of 24 servers selected randomly through the use of a verifiable random function (VRF) and can accommodate up to 10^3 clients. The VRF is seeded with the last finalized block hash, which ensures that even if a resourceful adversary successfully compromises a third of the servers in the network, the probability of compromising all servers in a single group is less than $3.54*10^{-12}$ or $(1/3)^{24}$. This makes Riffle's anytrust model effective in providing provable anonymity within the Aquaris network.

Clients seeking to join the network publish a join request and are assigned to groups through the VRF, resulting in a random assignment of clients and making it impossible for an adversary to compromise the anonymity set of a single group.

The Riffle groups are divided into two layers: entry groups and exit groups. Data is routed from the entry groups to the exit groups and vice versa, with each group in one layer connected to 10^3 groups in the other layer, and each connection originating from a different group. This results in an anonymity set of 10^6 if each group in the first layer can provide perfect anonymity among 10^3 clients, and each group in the second layer can provide perfect anonymity among 10^3 distinct groups in the first layer.

The network routes user data anonymously by dividing it into fixed-sized packages, routing them across different tunnels, and then reassembled on the convergence exit nodes (nodes which run multiple servers participating in multiple groups) to be routed to the outer internet or an internal network (such as for transactions). Onion routing is applied through one group in the first layer and one group in the second layer. The data is transmitted through each tunnel with symmetric authenticated encryption established by the Kyber key encapsulation protocol. In specific, each Aquaris client:

1. Participates in the setup phase of the hybrid shuffle.
2. Anonymously shares secret symmetric keys with every server.
3. Sends encrypted data including secret bitmasks and PRNG seeds to every server through the shuffle.
4. Directly sends a PRNG seed to one or more servers for use on accidental disconnections.
5. Establishes an encrypted tunnel with one of the servers.
6. Through this tunnel, uploads data to be inverse PIRed from the first to the second database. This data has a second layer of encryption by being XORed with the secret bitmasks.
7. Through this double encrypted tunnel shaped by the inverse PIR process for sending and forward PIR for receiving, inside the aforementioned tunnel, the client connects to a group of the second layer.
8. Repeats the above procedure with the selected group of the second layer. Outgoing data can be routed to the external internet or retrieved within the network through this group, and incoming data can be anonymously retrieved by the client.

To reduce latency, the size of data packages transmitted through the Riffle groups is limited to 256 bytes. To replicate the database, with up to 1000 clients in a group, divided equally among 24 entry nodes, this results in 42 clients * 256 bytes or approximately 10.5 KB for broadcast transmission and 256 KB to download during the first hop, which can be transferred in a fraction of a second. The clients are evenly distributed among different servers within a group to achieve optimal efficiency. After the setup phase and the inverse PIR establishment phase, communication between groups is directly broadcasted from the exit server in the entry group to all servers in the exit groups and vice versa, reducing the communication latency of the protocol, to 6 hops:

1. The client sends the data package to the entry server in the entry group.
2. The entry server broadcasts the package to all servers in the entry group.
3. The servers in the entry group perform XOR operations on all data and send the result to the exit server.
4. The exit server broadcasts the combined result to all servers in the specified exit group.
5. The servers in the exit group perform XOR operations on all data and send the result to the exit server.
6. The exit server sends the anonymized data to its destination.

The inverse procedure for receiving follows the same steps.

Proposition 8:

The Aquaris network provides strong anonymity with a theoretical maximum anonymity set of $10^6 - a$, where a represents the number of clients controlled by the adversary, even in a worst case scenario where the adversary controls 1/3 of the servers in the network.

Proof:

The Aquaris network consists of two layers of Riffle groups, each group consisting of 24 servers randomly selected using a verifiable random function (VRF). Even if an adversary manages to compromise one-third of all servers, the probability P of compromising n=24 servers within a single group is less than:

$$\mathrm{P}(\mathrm{n} = 24) < \left(\frac{1}{3}\right)^{24} = 3.54 * 10^{-12}$$

If anyone of the servers in a Riffle group is honest, the group's anonymity remains intact due to the proven strong anonymity of the Riffle system.

Clients are assigned to groups through the VRF, making it infeasible for an adversary to compromise the anonymity set of a single group. Each group in one layer connects to 10^3 groups in the other layer, with each channel connecting to a different group. This results in a theoretical anonymity set of 10^6, assuming perfect anonymity among $10^3 - a / 10^3$ clients in each group in the first layer and perfect anonymity among 10^3 distinct groups in the second layer. Therefore, the Aquaris network provides strong anonymity with a theoretical anonymity set of $10^6 - a$.

Note: However, in practice, clients may connect to more than one groups to increase their bandwidth. This reduces the anonymity set to the number of the clients k, where k is the number of users utilizing the same number of tunnels simultaneously (k-anonymity).

3.6 Defence against traffic analysis

The Aquaris network provides defense against traffic analysis through the proven anonymity guarantees offered by the Riffle system, identical package sizes, and synchronized group behavior. Each tunnel in the network is resistant to traffic analysis as a result.

It is important to note that within a Riffle group, all clients share the same data rate. This allows for k-anonymity to be established against a global adversary, which is perfect anonymity among th clients consuming the same bandwidth.

To further increase anonymity, a client can send obfuscating traffic, using more channels than needed, which serves to hide the specific data rate and conceal the client's k-set. This provides additional defense against traffic analysis.

3.7 Security against adaptive adversary

The anonymity nodes in the Aquaris network operate within a Trusted Execution Environment (TEE) or Secure Enclave [31], providing protection against routing data exposure to the node operators. As a result, in comparison to existing anonymous networks that are vulnerable to adaptive adversaries who bribe and compromise nodes to collaborate in de-anonymizing users, the compromise of an Aquaris node requires the compromise of its Trusted Execution Environment. This means that an adaptive adversary who possesses significant funds and seeks to launch a Sybil attack must first breach the security of the nodes' enclaves and obtain leaked data.

3.8 Access from restrictive network environments

There are two ways to block access to an onion routing network: blocking entry nodes' IP addresses and connection filtering based on the identification of the underlying connection properties.

To defend the network against IP blocking, Aquaris utilizes a group of entry nodes with hidden IP addresses (bridges). Under conditions of bridge scarcity, these entry nodes share their IP addresses only after being paid, with the price defined by the market. As the demand for hidden entry nodes increases, due to more addresses being requested by the adversary, more incentives exist for new users to run them, preventing totalitarian governments from unveiling and blocking these addresses.

Concealing Aquaris connection properties is achieved by encapsulating its encrypted tunnels inside ordinary HTTPS connections.

4. Economics and politics:

The original vision of Satoshi Nakamoto for Bitcoin was to promote a nearly equal distribution of the cryptocurrency among internet users. This was to be achieved through a decentralized mining process, where every computer chip would contribute to mining using approximately the same computing power.

However, this vision failed to materialize due to the evolution of mining techniques, which utilized specialized hardware such as GPUs and ASICs. This led to a significant centralization of Bitcoin funds, with some countries benefiting more from mining due to lower electricity costs. As a result, this also led to a substantial waste of energy, contributing to negative environmental impacts.

PoS (Proof-of-Stake) algorithms, utilized by the next generation of cryptocurrencies, allow different principles in their block reward distributions. These cryptocurrencies incentivize contributions to network services, including decentralized storage, computation clouds, wireless connections, anonymity services, and more [32, 33, 34, 35]. Additionally, they employ more complex governance principles, to improve decentralization.

4.1 Inflation, Block reward distribution and Governance

The economics and politics of a cryptocurrency, play a crucial role in its success and adoption. These aspects include inflation, block reward distribution, and governance.

Inflation refers to the increase in the supply of a cryptocurrency, which leads to a decrease in its purchasing power. Aquarium's inflation rate is limited by a predefined maximum emission curve, meaning that there is a finite amount of coins that can be minted. The initial supply of Aquarium coins is 10 trillion, and the maximum supply is capped at 30 trillion. The inflation rate starts at 2 trillion coins in the first year and gradually decreases by 10% each year.

Block reward distribution refers to the process of how new coins are added to the network and who receives them. In the case of Aquarium, 50% of the coinbase funds are allocated to stakers as a reward for their contributions to the network, and the remaining 50% is distributed according to a governance plan agreed upon by a built-in decentralized autonomous organization (DAO). This DAO is a special smart contract that implements proposals for the evolution of the Aquarium project and its distribution plans, which can be updated annually.

Governance, in the context of cryptocurrency, refers to the decision-making process of a decentralized network. Aquarium's governance system operates through a voting process, where all active validators cast their votes by sending a special payment output to their preferred proposal account during each voting epoch. The process eliminates 50% of the proposed plans in each cycle and results in one remaining proposal that reflects the balance of different voter interests and intentions. To prevent permanent unfair distribution against minority voters, defeated voters are given an advantage in the following elections.

Funds can be distributed to a variety of destinations, such as the Aquarium's self-improvement team, humanitarian or ecological organizations, or even burned by sending it to null addresses. This not only aligns with the original vision of Satoshi Nakamoto for an open financial system, but also serves as self-advertising for the Aquarium ecosystem, increasing profits for its userbase. Additionally, the option to burn a portion of funds can help to protect price stability by reducing inflation.

The economics and politics of Aquarium are designed to ensure decentralization, fairness, and transparency in its operations, contributing to its long-term success and adoption.

5. Potential of the Aquarium Project

The Aquarium Project is an innovative cryptocurrency aimed at revolutionizing the blockchain industry. Its architecture strikes a balance between scalability and decentralization, allowing for the processing of large volumes of data and transactions while preserving its decentralized network structure. The project boasts post-quantum security and unlikable transactions that ensure the confidentiality and security of users' sensitive information, and the proof-of-stake consensus mechanism helps maintain the immutability of the blockchain, making it resistant to tampering.

One of the most significant features of the Aquarium network is its Aquaris anonymity network, which provides a high level of privacy for users. The network is based on proven anonymity, utilising Riffle, and achieves low latency due to its innovative inverted Private Information Retrieval (PIR) architecture. The Aquaris anonymity network offers proven low latency anonymity for large anonymity sets, preserving the privacy of transactions. It has the potential to not only serve as a network layer for cryptocurrency transactions but also as a secure and private gateway to the internet, particularly in areas with restricted internet access.

The Aquarium network's architecture enables the creation of decentralized applications (dApps) managed by various groups with distinct encryption keys, enhancing privacy and security for each group's data and transactions. The project also includes a second-level payment channel network (PCN) that resolves the limitations faced by traditional cryptocurrencies and blockchain networks. The PCN provides a secure and efficient transaction medium for the cryptocurrency, even during network splits, and utilizes a scalable first layer for frequent channel reorganization, ensuring capital balance and efficient instant transactions.

The project has explored the feasibility of a decentralized data storage application, which would use erasure coding, the Aquaris anonymity network as a defense line against adaptive corruption, and the financial finality principle of proof-of-stake, to guarantee the integrity of critical files and wallet data. An additional innovative project, OpiNION, seeks to enhance freedom of expression through onion routing privacy and publicly administrated handling of abusive content through decentralized autonomous organizations. This project has the potential to revolutionize the web into a self-administered and censorship-free network where privacy, content quality, and freedom of expression are ensured.

The Aquarium project is a highly ambitious and promising project that has the potential to create a new era of privacy-focused and self-administered internet. The name "Aquarium" was chosen as a metaphor for the project's design as a closed ecosystem, where each component contributes to the overall system, much like the organisms in an aquarium working together to maintain balance.

6. References – citation

1. Nakamoto, S. (2008). Bitcoin: A Peer-to-Peer Electronic Cash System.
2. Ben-Sasson, E. et al. (2014). Zerocash: Decentralized Anonymous Payments from Bitcoin.
3. Fernández-Caramés, T. et al. (2020). Towards Post-Quantum Blockchain: A Review on Blockchain Cryptography Resistant to Quantum Computing Attacks.
4. Ben-Sasson, E. et al. (2018). Scalable, transparent, and post-quantum secure computational integrity.
5. Buterin, V., et al. (2020) Combining GHOST and Casper.
6. Erdin, E., et al. (2021) An Evaluation of Cryptocurrency Payment Channel Networks and Their Privacy Implications.
7. Buchmann, J., et al. (2011) On the Security of the Winternitz One-Time Signature Scheme.
8. Grassi, L. et al. (2019). POSEIDON: A New Hash Function for Zero-Knowledge Proof Systems.
9. Buchmann, J. et al. (2011). XMSS – A Practical Forward Secure Signature Scheme based on Minimal Security Assumptions.
10. Al-Bassam, M., et al. (2019). Fraud and Data Availability Proofs: Maximizing Light Client Security and Scaling Blockchains with Dishonest Majorities.
11. Orlovsky, M. (2020). RGB Technology Guide: Scalable & confidential Bitcoin/LN smart contracts built with client-side validation paradigm.
12. D'Amato, F. et al. (2023). A Simple Single Slot Finality Protocol For Ethereum
13. Castro, M., et al. (1999) Practical Byzantine Fault Tolerance.
14. Sel, D. et al. (2018). Towards Solving the Data Availability Problem for Sharded Ethereum.
15. Alhaddad, N. (2021). Succinct Erasure Coding Proof Systems.
16. Badertscher, C. et al. (2018). Ouroboros Genesis: Composable Proof-of-Stake Blockchains with Dynamic Availability.
17. Bonneau, J. et al. (2020). Mina: Decentralized Cryptocurrency at Scale.
18. Asharov, G. & Lindell, Y. (2022). A Full Proof of the BGW Protocol for Perfectly-Secure Multiparty Computation.
19. Dziembowski, S. et al. (2019). Multi-Party Virtual State Channels.
20. Aumayr, L. et al. (2020). Generalized Channels from Limited Blockchain Scripts and Adaptor Signatures.
21. Das, D. (2017). Anonymity Trilemma: Strong Anonymity, Low Bandwidth Overhead, Low Latency—Choose Two.
22. Golle, P., et al. (2004) Dining Cryptographers Revisited.

23. Kwon, A., et al. (2016) Riffle: An Efficient Communication System With Strong Anonymity.
24. Choudhary, D. & Bhagat, R. (2018). The Onion Routing.
25. Jefferys, K., et al. (2018) Loki: Private Transactions, Decentralized Communication.
26. Avanzi, R., et al. (2021) CRYSTALS-Kyber Algorithm Specifications and Supporting Documentation (version 3.01).
27. Bernstein, D. (2005) The Poly1305-AES message-authentication code.
28. Costa, N. et al. (2017). Proof of a shuffle for lattice-based cryptography.
29. Chor, B., et al. (1998) Private Information Retrieval.
30. Olumofin, F. (2010). Revisiting the Computational Practicality of Private Information Retrieval.
31. Sabt, M. et al. (2015). Trusted Execution Environment: What It is, and What It is Not.
32. Dimitri, N. (2021). Monetary Dynamics With Proof of Stake.
33. Karantias, K. et al. (2019). Proof-of-Burn.
34. Protocol Labs. (2017). Filecoin: A Decentralized Storage Network.
35. Matzutt, R. et al. (2020). Utilizing Public Blockchains for the Sybil-Resistant Bootstrapping of Distributed Anonymity Services.

7. Appendix A: Sending through inverse PIR in Aquaris groups