Technical White Paper

Version 1.0.2 - September 1st, 2021

Introduction

Veritise™ provides verification, identification, data collection and analysis services for companies and individuals. Central to these functionalities is the Veritise™ blockchain that makes these services possible and available on a global scale.

Veritise™ Blockchain is a Next-Gen Enterprise-Grade Technology, based on NEM Symbol architecture, with a network that is optimized for fast, secure and fully auditable payment and transaction processing. Veritise Blockchain is feature-rich with powerful abilities such as Multi-level multi-signature account schemes, Smart Assets, Namespaces, Aggregate Transactions, Metadata Controls and on-chain data storage with strong encryption methods.

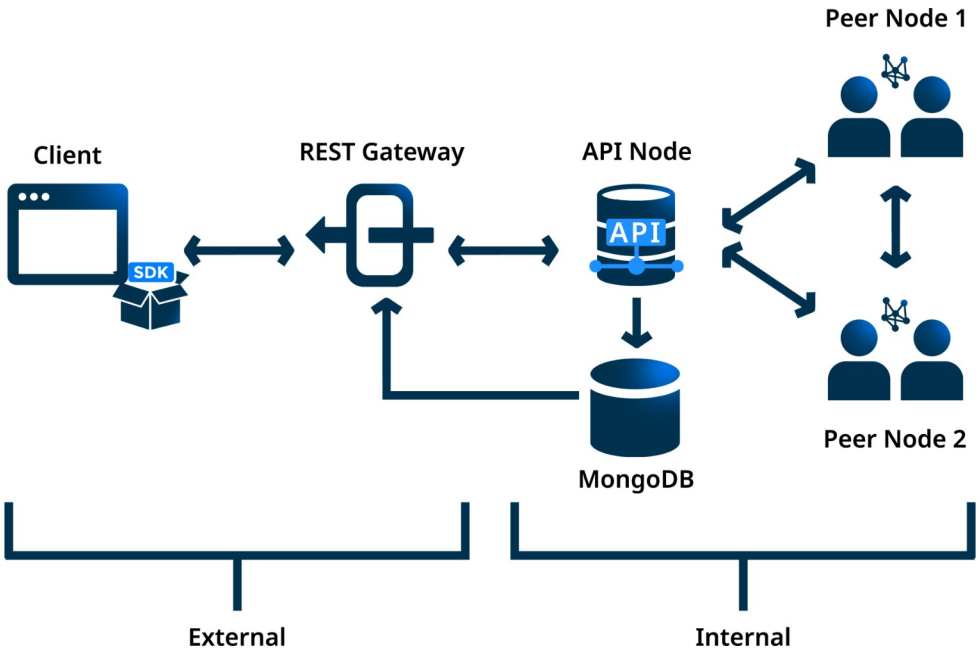
Future use-cases, like fully compliant tokenized securities and other financial instruments are made possible, improving speed, reducing potential fraud, and lowering costs.

The Veritise™ blockchain architecture supports high customization at both the network and individual node levels. Network-wide settings, specified in network, must be the same for all nodes in a network. In contrast, node-specific settings can vary across all nodes in the same network, and are located in node. A plugin/extension approach is used instead of supporting Turing complete smart contracts. While the latter can allow for more user flexibility, it's also more error prone from the user perspective. A plugin model limits the operations that can be performed on a blockchain and consequently has a smaller attack surface.

Additionally, it's much easier to optimize the performance of a discrete set of operations than an infinite set. This enables the Veritise™ blockchain to achieve the high throughput for which it was designed.

Veritise™ Network Topology

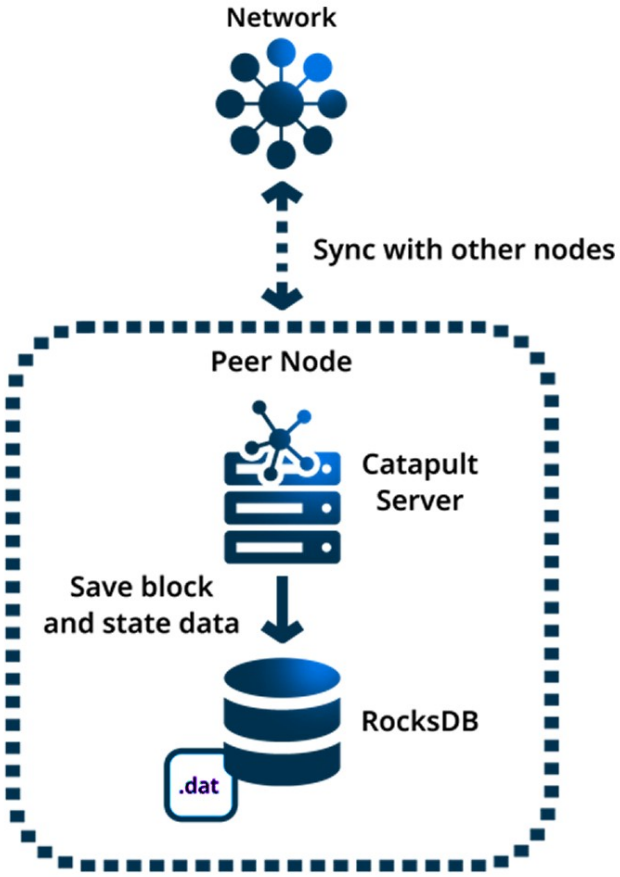
The Veritise™ blockchain infrastructure is a **public blockchain** with network nodes divided into three categories: Peer, API or Dual node. The communication with the network is handled by external component – REST Gateway.



Veritise™ Blockchain Infrastructure

Peer Node:

The peer nodes form the backbone of the Veritise™ blockchain. The role of the node is to verify transactions and blocks, run the consensus algorithm, create new blocks, and propagate the changes through the network.



Peer Node Communication

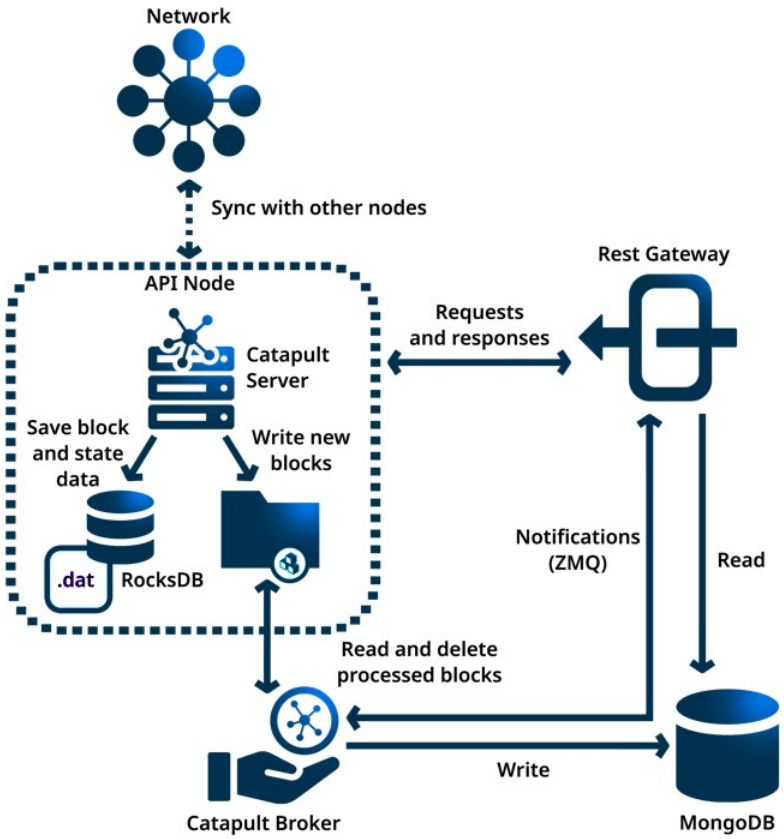
The API nodes push new transactions to the P2P network, where they are included in a block or discarded. After the block is processed, the node saves:

- The binary of each block as a flat-file on disk
- The updated chain state in memory or RocksDB (configurable)

Peer nodes store the chain state in RocksDB. The data structures cached are serialized and stored as values to corresponding keys. For example, a column in this database maps the public keys to addresses. Another one, the account state entries as the values to corresponding address keys.

API Node:

The primary responsibility of an API node is to store the data in a readable form in MongoDB. The catapult-server software allows configuring standalone API nodes or with Peer capabilities (Dual).



Peer + API (Dual) node communication

Instead of writing the data directly into MongoDB, the nodes write it into a file-based queue called spool. A broker service consumes the data from the spool and updates MongoDB accordingly. Once a block is processed, the broker service notifies the changes to catapult-rest instances using ZMQ.

API nodes are also responsible for collecting the co-signatures of aggregated bonded transactions, which are only processed once they are complete. MongoDB stores blocks, transactions, and chain states for high query performance.

The broker service updates the linked MongoDB instance when:

- A new block / a bunch of blocks finish processing.
- New unconfirmed transactions complete processing.

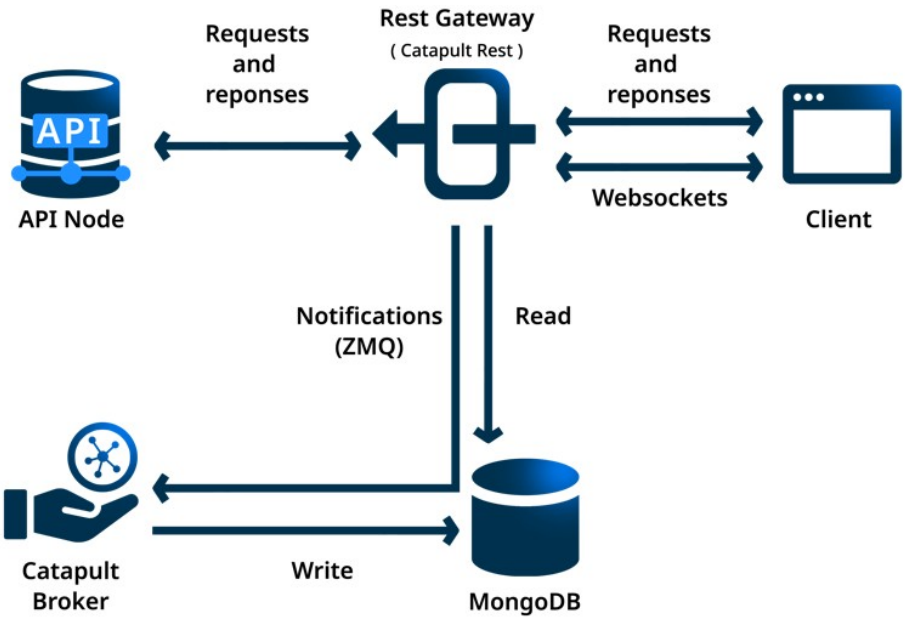
ZeroMQ (ZMQ) is an asynchronous messaging library, which enables real-time subscriptions. It transports notifications from the API node to the ZMQ endpoint, where Catapult REST listens. It is an alternative to REST WebSockets, aimed to be used when performance is critical.

Dual Node:

Dual nodes are simply a superset of Peer and API nodes. They support all capabilities of both node types. Since these nodes support all API node capabilities, they also require a broker.

REST Gateway:

The REST gateways handle JSON API client requests. The gateway reads from MongoDB, formats the response, and returns it to the client. This component is also responsible for returning events to the client using WebSockets.



REST gateway communication

Each REST gateway connects to one API instance to send new transactions requests triggered from the client-side and receive updates in real-time using sockets.

Seed nodes

A freshly booted node is initially isolated and not connected with any peers. It needs to join a network before it can make any meaningful contributions, like validating or producing blocks. In the Veritise™ network, a list of static seed nodes is stored in a peers configuration file.

To join a network, a new node first connects to these nodes. These files don't need to be identical across all nodes in a network.

Node communications

TLS stands for Transport Layer Security, a protocol that provides secure communication between servers. All connections among Veritise™ nodes are made over the newest version of TLS, namely version 1.3 with a custom verification procedure which provides unparalleled privacy and performance compared to previous versions of TLS.

Each node is expected to have a two-level deep X509 certificate chain composed of a root certificate and a node certificate. All certificates must be X25519 certificates. Veritise™ doesn't support any other certificate types.

The root certificate is expected to be self-signed with an account's signing private key. This account is assumed to be a node's unique owner. For security, its private key should not be kept on a running server. The node certificate is signed by the root certificate. It can contain a random public/private key pair. This certificate is used for authenticating TLS sessions and deriving shared encryption keys for encrypting optional. It can be rotated as often as desired. This authentication procedure is performed independently by each partner node. If either node fails the handshake, the connection is immediately terminated.

Veritise™ uses TCP for network communication on the port specified in the node configuration. Communication is cantered around a higher-level packet model on top of and distinct from TCP packets. All packets begin with an 8-byte header that specifies each packet's size and type. Once a complete packet is received, it is ready for further processing.

A mix of long lived and short-lived connections are used. Long lived connections are used for repetitive activities like syncing blocks or transactions. They support both push and request/response semantics.

Network is configured in the way that the connections are allowed to last for 200 block selection rounds before they are eligible for recycling. Connections older than this setting are recycled primarily to allow direct interactions with other partner nodes and secondarily as a precaution against zombie connections. Short lived connections are used for more complex multistage interactions between nodes. For example, they are used for node discovery and time synchronization. Short lived connections help prevent sync starvation, which can occur when all long-lived connections are in use and no sync partners are available.

Cryptography

Veritise™ uses cryptography based on Elliptic Curve Cryptography (ECC). The choice of the underlying curve is important to guarantee security and speed. Veritise™ uses the Ed25519 digital signature algorithm. The algorithm produces short 64-byte signatures and supports fast signature verification. Neither key generation nor signing is used during block processing, so the speed of these operations is unimportant.

Public/Private Key Pair

The private key is a random 256-bit integer. The Public key is also 256-bit integer derived from the private key. The key pair is the Veritise™ blockchain account associated with a mutable state stored on the blockchain.

Accounts and Addresses

Accounts (private and public key pairs) on the Veritise™ network are used as digital identities to identify the users, businesses as well as products and unique objects, to track and trace them, to keep and maintain provenance records and have full traceability of related events.

Each account has a unique address derived from the public key. Normally, the address is shared instead of the public key because it is shorter and gathers information about the network.

A decoded address is a 24-byte value composed of the following three parts:

- network byte
- 160-bit hash of an account's signing public key
- 3 byte checksum

The checksum allows for quick recognition of mistyped addresses. It is possible to send tokens to any valid address even if the address has not previously participated in any transaction. If nobody owns the private key of the account to which the mosaics are sent, the mosaics are most likely lost forever.

Addresses can be represented either in Hexadecimal format (48 characters) or, more commonly, in encoded form in Base32 format (39 characters).

To convert a public key to an address, the following steps are performed:

1. Perform 256-bit SHA3 on the public key
2. Perform 160-bit RIPEMD160 of hash resulting from step 1
3. Perform network version byte to RIPEMD160 hash
4. Perform 256-bit SHA3 on the result, take the first three bytes as a checksum
5. Concatenate output of step 3 and the checksum from step 4
6. Encode result using Base32

Multi-signature Accounts

Veritise™ accounts can be converted to multisig. The cosignatories - other accounts - of the multisig will become the account managers.

From that moment on, the multisig account cannot announce transactions by itself. A multisig cosignatory has to propose a transaction involving the multisig, wrapping it in an Aggregate Transaction.

To record the transaction in the block, the other cosignatories will have to agree.

Public Keys

An account is associated with zero or more public keys. The supported types of keys are:

1. ***Signing:*** ED25519 public key that is used for signing and verifying data. Any account can receive data, but only accounts with this public key can send data. This public key is the only one used as input into an account's address calculation.
2. ***Linked:*** This public key links a main account with a remote staking node. For a Main account, this public key specifies the remote staking node account that can sign blocks on its behalf. For a Remote account, this public key specifies the main account for which it can sign blocks. These links are bidirectional and always set in pairs.
3. ***Node:*** This public key is set on a Main account. It specifies the public key of the node that it can delegate staking on. Importantly, this does not indicate that the remote is actively staking on the node, but only that it has the permission to do so. If either the staking node account or the node owner is honest, the account will be restricted to delegate staking on a single node at a time. An honest staking participant should only send its remote staking private key to a single node at a time. Changing its remote will invalidate all previous remote staking permissions granted to all other nodes (and implies forward security of delegated keys). Any older remote staking private keys will no longer be valid and unable to be used for producing blocks. An honest node owner should only ever remote stake with a remote staking private key that is currently linked to its node public key.
4. ***VRF:*** ED25519 public key that is used for generating and verifying random values. This public key must be set on a Main account for the account to be eligible to participate in staking. A verifiable random function (VRF) uses a public/private key pair to generate pseudo-random values. Only the owner of the private key can generate a value such that it cannot be predetermined by an adversary. Anyone with the public key can verify whether the value was generated by its associated private key.

5. *Voting*: a separate public key that is used for signing and verifying finalization messages. All voting keys are temporary and must be registered with both a start and end epoch. This public key must be set on a Main account for the account to be eligible to vote.
6. *Transport*: This key pair is used by nodes for secure transport over TLS.

Transactions

The Veritise™ network supports two fundamental types of transactions: **basic transactions** and **aggregate transactions**.

Basic transactions represent a single operation and require a single signature. A basic transaction is composed of both cryptographically verifiable and unverifiable data. All verifiable data is contiguous and is signed by the transaction signer. All unverifiable data is either ignored (e.g. padding bytes) or deterministically computable from verifiable data. Each basic transaction requires verification of exactly one signature.

Aggregate transactions are containers of one or more transactions that may require multiple signatures. Aggregate transactions allow basic transactions to be combined into potentially complex operations and executed atomically. This increases flexibility relative to a system that only guarantees atomicity for individual operations while still constraining the global set of operations allowed to a finite set.

The layout of an aggregate transaction is more complex than that of a basic transaction, but there are some similarities. An aggregate transaction shares the same unverifiable header as a basic transaction, and this data is processed in the same way. Additionally, an aggregate transaction has a footer of unverifiable data followed by embedded transactions and cosignatures.

An aggregate transaction can always be submitted to the network with all requisite cosignatures. In this case, it is said to be complete, and it is treated like any other transaction without any special processing.

API nodes can also accept bonded aggregate transactions that have incomplete cosignatures. The submitter must pay a bond that is returned if and only if all requisite cosignatures are collected before the transaction times out. Assuming this bond is paid upfront, an API node will collect cosignatures associated with this transaction until it either has sufficient signatures or times out.

Transaction Status Messages

The topics for transaction messages consist of both a topic marker and an optional unresolved address filter. When an unresolved address filter is supplied, only messages that involve the specified unresolved address will be raised. For example, a message will be raised for a transfer transaction only if the specified unresolved address is the sender or the recipient of the transfer. When no unresolved address filter is supplied, messages will be raised for all transactions.

The following transaction messages are supported:

- Transaction: A transaction was confirmed, i.e. is part of a block
- Unconfirmed transaction add: An unconfirmed transaction was added to the unconfirmed transactions cache
- Unconfirmed transaction remove: An unconfirmed transaction was removed from the unconfirmed transactions cache
- Partial transaction add: A partial transaction was added to the partial transactions cache

- Partial transaction remove: A partial transaction was removed from the partial transactions cache
- Transaction status: The status of a transaction changed

The topic for a cosignature message consists of both a topic marker and an optional unresolved address filter. The message is emitted to the subscribed clients when a new cosignature for an aggregate transaction is added to the partial transactions cache. When an unresolved address filter is supplied, messages will only be raised for aggregate transactions that involve the specified address. Otherwise, messages will be raised for all changes.

Data validation

Veritise™ blockchain uses tree structures to store large data associated with a block that cannot be retrieved directly from the block header. This allows light clients to verify if an element (e.g. transaction, receipt statement) exists without demanding the entire ledger history.

Veritise™ Blockchain Features

1. Multi-level multi-signature account schemes

As mentioned before, Veritise™ blockchain accounts can be converted to multi-signature using different level and depth schemes. The cosignatories - other accounts - of the multi-signature account will become the account managers. From that moment on, the multi-signature account cannot announce transactions by itself. A multi-signature scheme cosignatory must propose a transaction involving the multi-signature, wrapping it in an Aggregate Transaction. To record the transaction in the block, the other cosignatories will have to agree.

Veritise™ 's current implementation of multi-signature is "M-of-N", where M is the number of cosignatories required to announce a transaction and N is the total amount of cosignatories for that particular multi-signature account. This means that M can be any number equal to or less than N, i.e., 1-of-4, 2-of-2, 4-of-9, 9-of-10 and so on.

Moreover, the multi-signature account can be a co-signer for another multi-signature account, up to 4 levels. Multi-level multi-signature accounts add "AND/OR" logic to multi-signature transactions, creating multi-level processes and logic implementations as co-signer can be a certain business process application i.e., checking certain business rules.

2. Tokens (Mosaics)

Apart from the VTS token as a central token of the Veritise™ blockchain infrastructure, users/businesses can issue Smart Assets that can be a token, but it

can also be a collection of more specialized assets such as reward points, signatures, status flags, votes, product identifiers or even other currencies.

Each asset has a unique identifier represented as a 64-bit unsigned integer and a set of configurable properties and flags that can be defined during the token creation.

3. Namespaces

Namespaces function similarly to internet domains. Creating a namespace starts with choosing a name that you will use to refer to an account or asset. The name must be unique in the network and may have a maximum length of 64 characters, and the allowed characters are: {a, b, c, ..., z, 0, 1, 2, ..., 9, _ , -.}

Using Veritise™ alias transaction type Veritise™ will be able to link namespaces to accounts and digital assets (tokens). An alias or its linked asset can be used interchangeably when sending a transaction. Using the alias makes long addresses easy to remember and assets (tokens) recognizable. The block receipts store the resolution of the alias for a given transaction.

Namespace functionality can be used to assign identifiers and/or user-friendly labels to Veritise™ accounts and easily navigate through digital ledger transactions as well as initiate transactions using the label/identifier.

4. Account restrictions

On the Veritise™ blockchain infrastructure accounts may configure a set of smart rules to block announcing or receiving transactions given a series of restrictions.

The account owners - plural in case of multi-signature accounts - can edit the account restrictions.

An account can decide to only receive transactions from a list of allowed addresses. Alternatively, the account can define a list of blocked addresses.

Restricting incoming transactions is useful when the account will be only receiving transactions from known addresses, or when the account wants to block transactions coming from unknown senders.

By default, when there are no restrictions set, all the accounts in the network can announce transactions to the unrestricted account. Additionally, an account can decide to apply address restrictions to the outgoing transactions, limiting the accounts allowed that are valid recipients.

5. Operation restriction

An account can be configured to allow/block announcing outgoing transactions with a determined operation type. By doing so, the account increases its security, preventing the announcement by mistake of undesired transactions.

6. Aggregate transactions

Aggregate transactions merge multiple transactions into one, allowing trustless swaps, and other advanced logic. Veritise™ blockchain does this by generating a one-time disposable smart contract.

When all involved accounts have co-signed the AggregateTransaction, all the inner transactions are executed at the same time.

There are several different types of Aggregate transactions predefined on Veritise™:

- **Aggregate complete** - an AggregateTransaction is complete when all the required participants have signed it. The co-signers can sign the transaction without using the blockchain. Once it has all the required signatures, one of them can announce it to the network. If the inner transaction setup is valid, and there is no validation error, the transactions will get executed at the same time. Aggregate complete transactions enable adding more transactions per block by gathering multiple inner transactions.
- **Aggregate bonded** - an AggregateTransaction is bonded when it requires signatures from other participants. Once an aggregate bonded is announced, it reaches a partial state - where it can live up to 2 days (this parameter is a configurable parameter on Veritise™ blockchain) - and notifies its status through WebSockets or HTTP API calls. Every time the cosignatory signs the transaction and announces an aggregate bonded co-signature, the network checks if all the required co-signers have signed. When all signatures are acquired, the transaction changes to an unconfirmed state until the network includes it in a block.

Aggregate transactions are used by Veritise™ to implement business logic and process automation. For example, creating an account, activating it, assigning label/identifier as namespace alias at once.

7. Metadata controls

The Veritise™ blockchain provides the option to associate custom data to an account, digital asset (token) or namespace with a transaction.

Veritise™ uses the following metadata:

- Attach relevant information to accounts (account labelling, account categorisation etc.).
- Validate the value attached to an account to enable the Veritise™ blockchain back-end application to perform an off-chain action (triggered actions, validated actions etc.).

Metadata is uniquely identified by the tuple {signer, target-id, metadata-key}. Including a signer in this composite identifier allows multiple accounts to specify the same metadata without conflict.

The value linked to an identifier is a string up to 4096 characters.

Metadata entries are stored on the Veritise™ blockchain - like the message of a regular TransferTransaction - but also as a key-value state. This feature reduces the reading time of client applications; metadata allows information to be accessed by keys instead of processing the entire account transaction history off-chain to obtain the latest transaction message value.

8. Message

On the Veritise™ blockchain network, transfer transactions can hold a message up to 4096 characters (configurable parameter) in length, making them suitable for timestamping data permanently on the blockchain. By default, the messages attached are visible to all blockchain network participants. Encrypted messages are only accessible by the sender and the recipient. Veritise™ blockchain network uses Bouncy Castle's AES block cypher implementation in CBC mode to encrypt and decrypt messages.

Consensus

Veritise™ implemented a Proof of Stake (PoS) consensus that attempts to award users preferentially relative to hoarders. It strives to calculate a holistic score of an account's importance without sacrificing performance and scalability.

The algorithm considers the following factors when calculating an account's importance, the measure that will ultimately be used to choose the next staking node:

- **Stake:** The total amount of harvesting mosaic held, since owners with larger balances have the incentive to see the ecosystem flourish. Only accounts holding more than 10.000 VTS tokens (high-value accounts) are eligible for staking.
- **Transactions:** The total amount of fees paid by an account. This encourages being an active account in the network.
- **Nodes:** The number of times an account has been the beneficiary of the fees collected by a node. Thus, the network incentivizes accounts which run nodes.

Periodically, an importance score based on these three factors is calculated for all high-value accounts. The importance score determines an account's probability to produce the next block and be remunerated for the consensus.

Partial scores

The network calculates first the following partial scores for all high-value accounts at the end of each importance period (720 blocks, roughly 3 hours. See *importanceGrouping* in Network configuration):

- **Stake Score (S):** Account's balance divided by the balance of all high-value accounts, at the end of the period.
- **Transaction Score (T):** Total amount of fees paid by the account divided by the total amount of fees paid by all high-value accounts during the period.
- **Node Score (N):** Number of times the account has been the beneficiary of a node fee divided by the number of times all high-value accounts have been the beneficiary of a node fee, during the period.
- **Activity Score (A):** Average of the T and N scores weighted 80% and 20% respectively, divided by the account's balance. Dividing by the account's balance gives some boost to small accounts because their importance score will depend more on their activity and less on their stake.

  An absolute activity score (A') is calculated first:

$$A' = \frac{10000}{Balance}(0.8T + 0.2N)$$

And the actual activity score (A) is calculated by dividing A' by the sum of the absolute activity scores of all high-value accounts.

The importance score is then calculated based on the above partial scores.

Importance score

The importance score I is calculated as the average of the S and A scores, weighted by an activity factor γ:

$$I = \gamma A + (1 - \gamma) S$$

In the Veritise™ network γ is 0.05 (5%)

Finally, among all accounts eligible for staking, the probability that a particular one is chosen is proportional to its effective importance score, which is defined as the smaller of the previous two importance scores.

Settlement finality

Veritise™ blockchain technology comes with the deterministic block finalization mechanism implemented at the protocol level that allows checkpoints to be set that can never be rolled back by any party - network Information Is Immutable Indefinitely by design.

The approach is modeled after GRANDPA used by Polkadot. GRANDPA is highly influenced by CASPER protocol, which itself is influenced by PBFT. Traditionally, PBFT uses three types of messages: pre-prepare, prepare and commit. Pre-prepare messages, which are used to start rounds, are not used by Veritise™. Instead, elapsed network time is used to start rounds. Prepare and commit messages in PBFT roughly correspond to prevote and precommit messages in the Veritise™ network.

Network Fees

By default, fees will be paid in the underlying currency of the Veritise™ network - VTS tokens. All network fees (token creation, namespace rental, transfer transaction etc.) are configurable on the Veritise™ network.

The fee associated with a transaction primarily depends on the size of the transaction. The effective fee deducted from the account sending the transaction is calculated as the product of the size of the transaction and a fee multiplier set by the node that produces the block.

$$\mathbf{effectiveFee = transaction::size * block::feeMultiplier}$$

Veritise™ node owners will be able to configure the fee multiplier to all positive values, including zero. This multiplier will create a competition between nodes and will allow to keep fee structure relative to the Blockchain economy and demand for the services. The `fee_multiplier` is stored in the block header when the node produces a new block, determining which was the effective fee paid for every transaction included.

Before announcing the transaction, the sender must specify during the transaction definition a `max_fee`, indicating the maximum fee the account allows to spend for this transaction.

If the `effective_fee` is smaller or equal to the `max_fee`, a harvester could opt to include the transaction in the block. The nodes can set their transaction inclusion strategy:

- **Prefer-oldest:** Preferred for networks with high transaction throughput requirements. Include first the oldest transactions.
- **Minimize-fees:** Philanthropic nodes. Include first the transactions that other nodes do not want to include.
- **Maximize-fees:** Most common in public networks. Include first transactions with higher fees.

To ensure that the transaction will get included without setting a `max_fee` unnecessarily high, the sender of the transaction can ask the REST Gateway for the median, average, highest, or lowest multiplier of the network over the last N blocks.

For example, the sender could set the transaction `max_fee` as follows:

$$\textbf{maxFee=transaction::size} * \textbf{network::medianFeeMultiplier}$$

Network Rewards

Veritise™ blockchain network configuration has a predetermined network fee sink account that will receive a percentage of the staking rewards (block fees and inflation). This fee is set to 10% and is used to support the further network development and different Reward Programs.

There is also predetermined fee sink for Namespace and SubNamespace rental fees and Token issuance fees that will used also for Veritise™ development and market penetration activities.

Moreover, each node operator can set a beneficiary account to share a percentage (up to 25%) of the harvesting rewards. The node operators can use this feature to create incentive structures for their node supporters.

Staking methods

There are different staking modes available on whether the staking account owns the node as well as the desired security level: Local, Remote and Delegated.

Local staking

This is the simplest to set up, and the most insecure method. It requires changing a node's configuration, so it is only available to node owners. It is enabled by filling-in the appropriate staking properties in the node configuration file.

As it can be seen, the staking participants account's private key is stored in Node properties, since it is needed to sign off created blocks. This is a security concern since this account contains funds and the configuration file might be accessed by uninvited actors if the node is compromised. Funded accounts' private keys should always be stored offline.

Therefore, this method is strongly discouraged. Remote or delegated staking are recommended instead.

Remote staking

Node owners can use a remote account to act as proxy and sign off the newly created blocks, while staking rewards are still collected by their main account. The remote

account has no funds, so the fact that its private key is exposed in a configuration file on the node is not a concern. The importance score is still based on the main account.

In this setup the main account is still called the Staking participant, for simplicity, whereas the remote account is called a Proxy.

Remote harvesting is enabled just like local harvesting but using the remote account's private key in the Node properties and announcing an AccountKeyLink transaction that links the remote and main accounts.

Delegated staking

Eligible accounts not owning a node can still benefit from harvesting by requesting a node to stake and participate in consensus for them. The account's importance score is used and any collected fees are divided among the account and the node's beneficiary (as explained in the Rewards section). It is a advantageous agreement to both the account and the node.

It is then said that the account delegates staking to the node, but the account is still considered the staking participant.

Delegated staking is enabled similarly to remote staking but, since the account has no access to the node's configuration, it announces a PersistentDelegationRequest transaction instead. Upon receiving the request, the node may or may not grant it, depending on its configuration and the rest of requests received.

As with remote staking a proxy remote account is used so the main account's private key is never put at risk.

Use case implementation examples

First Scenario – Product Authenticity Verification

Client Company Perspective

1) Client Company signs up for an account at the Veritise.com website and Veritise takes the company through a verification process to confirm they are the real company.
2) Once confirmed, the client company gets access to the full account and receives a blockchain wallet that is specifically assigned to them. We'll call this the client company wallet from now on.
3) The client company chooses to pay for a monthly subscription plan in order to start using product authenticity services.
4) The client company selects subscription plan will include up to 1000 QR code generations per month.
5) The client company wants to start by registering 10 product items in order to protect them from counterfeiting.
6) The client company logs in and uses the web interface to generate unique QR codes for 10 products. Each product will have 2 QR codes (one for public key, one for private key).
7) Through a transaction between client company wallet and the product wallets, a link is established which ensures the product is truly originating from said company (this is automatically done by the backend).
8) Client company sends the QR codes that Veritise generated to the print shop for label printing or to package manufacturer of their choice so the QR codes are attached to the product, public key QR code on the outside of the package, private key QR code on the inside of package or on the backside of a tamperproof label.

Customer perspective

1) Customer walks into the store and sees client company product.
2) Customer scans the Veritise QR code.
3) If the customer does not have the Veritise app installed yet, it will redirect customer to Veritise.com webpage and shows authenticity and product data. Customer will have the option to download the Veritise mobile app from there as well for future use.
4) If the customer already has the Veritise app installed, the app will show the authenticity and product data directly.
5) The customer verifies that the product is genuine and purchases it.
6) The customer arrives home, unboxes the product and scans the private key QR code with the Veritise App, which triggers a blockchain transaction (see patent).
7) The app indicates that the customer is now successfully the registered owner and also asks the customers if he/she wants to register warranty with the client company.
8) When 'yes' is clicked, the customer personal info is immediately sent to client company (via Veritise back-end) and the customer's product is now protected and under warranty.

Scenario implementation on Veritise Network

The full implementation diagram/scheme is presented at *Annex 1 "First Scenario - Product Authenticity Verification"* describing recommended/proposed Veritise platform and Veritise blockchain network communication and interaction schemes as well as identifying Blockchain network features that are used to implement the Use Case.

Second Scenario – CV & Portfolio Verification

Portfolio Verification from the Client Company & Future Client Perspective

1) The Client Company wants to prove to their future clients that their portfolio is genuine, meaning they indeed built certain projects for clients and received a high approval rating for doing so.
2) The client company logs into the system and creates a portfolio profile (name of client, contact email, scope of work, etc.) and then
3) The Veritise system checks whether Client has Veritise account and:
    3.a. If Yes, Veritise system sends client with the request to confirm that the portfolio information is correct and to give them a rating for work done.
    3.b. If No, then Veritise system sends a request to crate Veritise profile and Veritise Blockchain identity.
4) The client can either approve or deny the request using the Veritise App which will generate a wallet uniquely identifying the client.
5) If the client approves the request, a transaction on the blockchain occurs which guarantees immutably that the portfolio item of the client company is genuine.
6) A future client of the client company can use the Veritise app to verify the portfolio item and to check the approval rating.

Scenario implementation on Veritise Network

The full implementation diagram/scheme is presented at *Annex 2 "Second Scenario: Portfolio Verification from the Client Company"* describing recommended/proposed Veritise platform and Veritise blockchain network communication and interaction schemes as well as identifying Blockchain network features that are used to implement the Use Case.

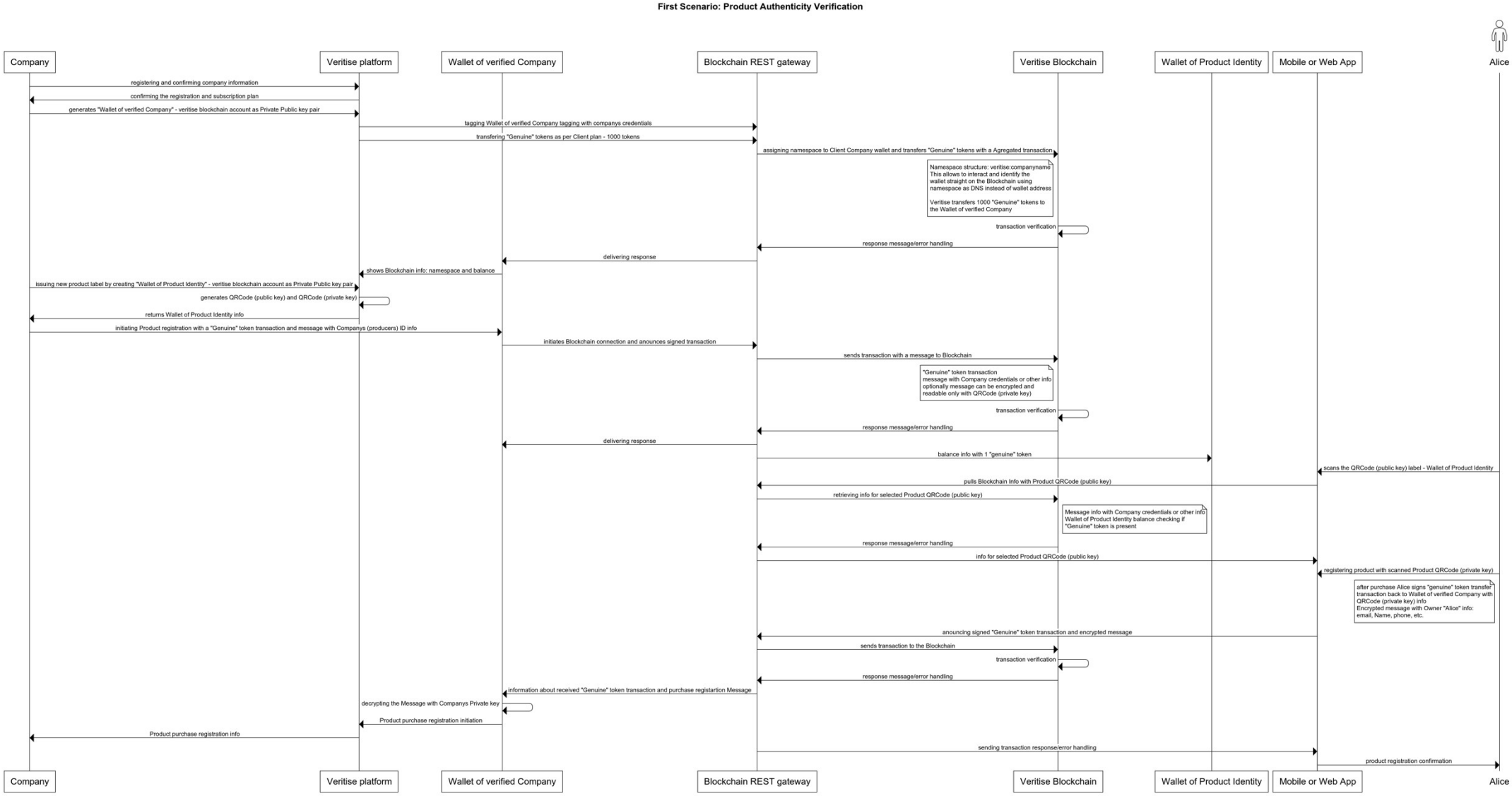
Third Scenario - CV Verification from the Employee and Employer Perspective

1) An employee wants to add his new employer to his CV and wants to do this in such a way that the reference can be verified by others, such as headhunters, or a future employer.
2) The employee signs up for a Veritise account, buys the correct subscription plan and creates a CV item profile that contains job description, employer contact information etc.
3) The Veritise system checks whether employer has a profile on Veritise platform and:
    3.a. If Yes, then sends a request to the employer to confirm the CV information and the employer can approve or deny the request using the Veritise App.
    3.b. If No, then Veritise system sends a request to crate Veritise profile and Veritise Blockchain identity.
4) If employer approves, Veritise system will establish a link between employer and employee via a transaction on the blockchain.
5) The employee now can have anyone verify his new CV entry by them using the Veritise system/App.

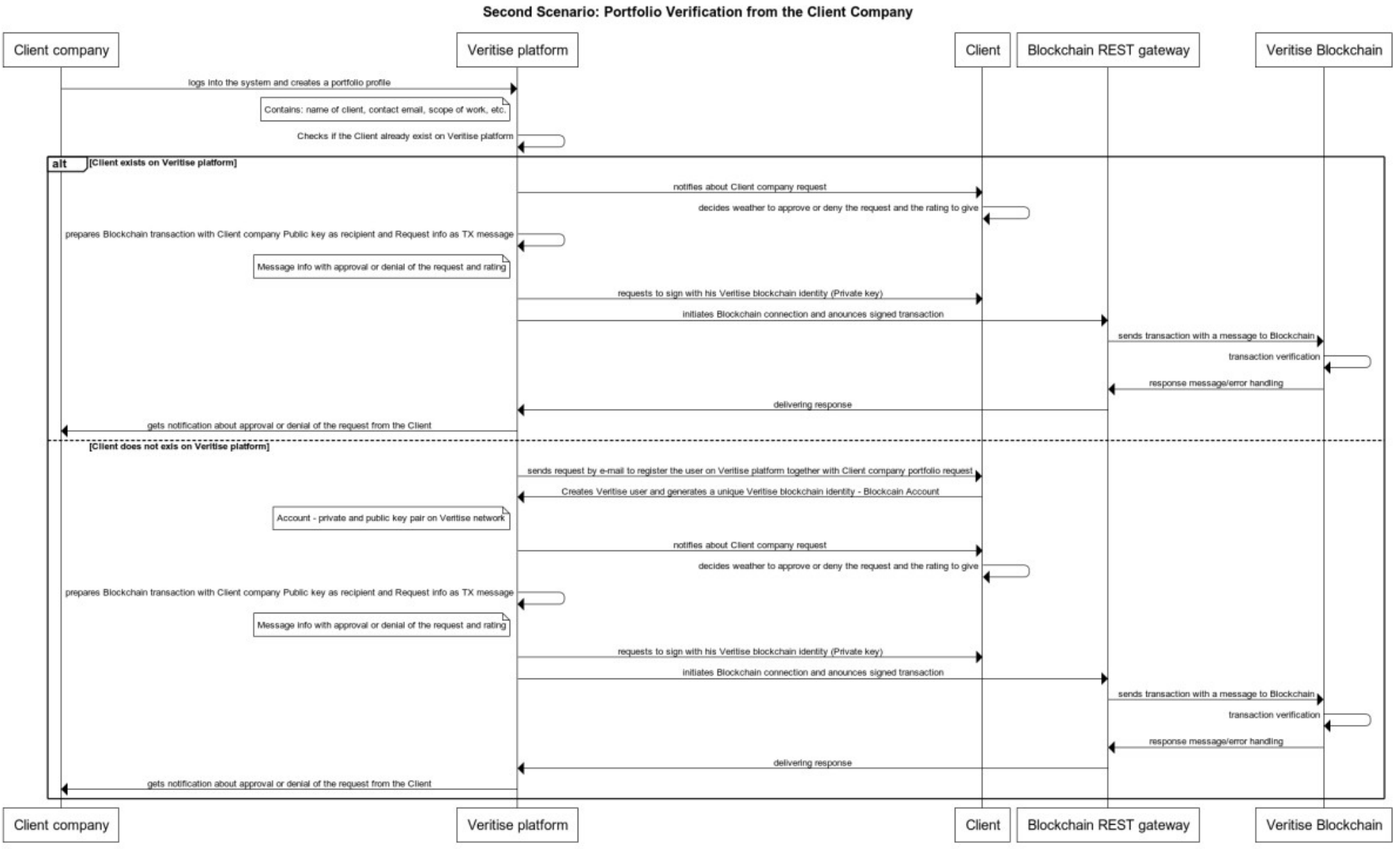
Scenario implementation on Veritise Network

The full implementation diagram/scheme is presented at *Annex 3 "Third Scenario - CV Verification from the Employee and Employer Perspective"* describing recommended/proposed Veritise platform and Veritise blockchain network communication and interaction schemes as well as identifying Blockchain network features that are used to implement the Use Case.

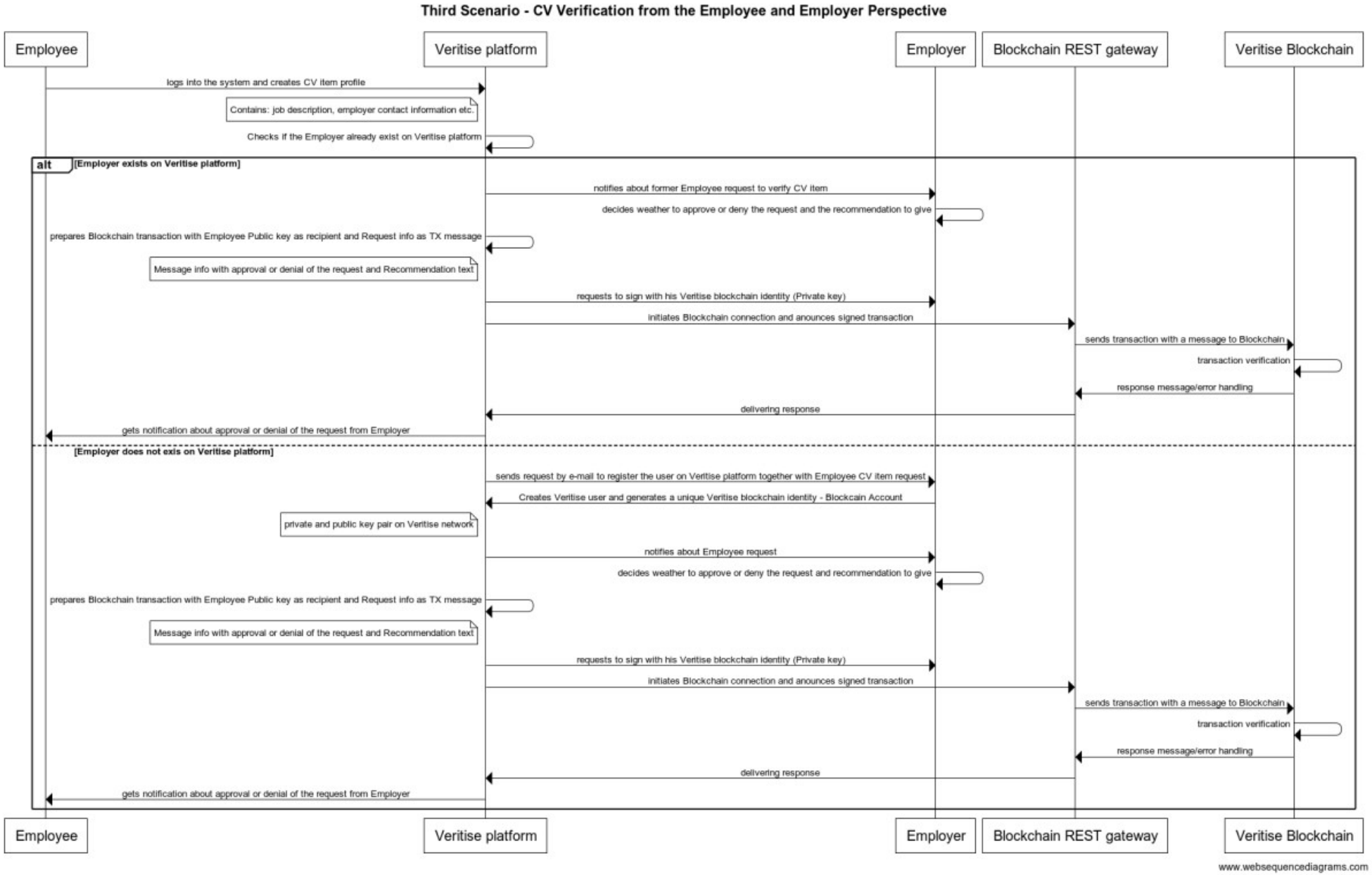
Annex 1 - First Scenario - Product Authenticity Verification



Annex 2 – Second Scenario: Portfolio Verification from the Client Company



Annex 3 - Third Scenario - CV Verification from the Employee and Employer Perspective



Tokenomics

- Total Supply: 300 Million VTS tokens
- Circulating Supply 1st year: 75M to 200M maximum depending on private and public token sales
- Extremely low inflation (maximum of 1% per year). This inflation is required in order to:
    - Tmaintain supply and compensate for lost keys/wallets/funds
    - reward Server Nodes that maintain the blockchain network during time transaction volume is still ramping up
    - protect value for token holders
- 400+ TPS (Transactions Per Second) on Public network and 3000+ TPS on private network
- Block Generation Target Time: 15 seconds with confirmed transaction times of less than 5 seconds
- Up to 4096 characters of on-chain data per transaction can be stored

- End of Technical White Paper -

- content subject to change -