Ontology

TECHNOLOGY WHITE PAPER

Version 0.6.5

Date Updated： 2018/03/14

A New High-Performance Public Multi-Chain Project &
A Distributed Trust Collaboration Platform

Abstract

Throughout history people have established trust through technology, rule of law, and communities.

However, trust and collaboration between entities involves multiple sources and isolated systems, which means it can be costly, and therefore hinders the breadth and depth of collaboration potential. Although the technology has advanced a great deal in recent times, too many factors are still hindering collaboration with trust. These include fragmentation of trust systems, the missing role of the individual, inaccurate identity verification, inability to dispute false information, etc. In areas such as social governance, economic cooperation, and financial services, the cost of establishing trust is huge.

The decentralized, tamper-proof blockchain has brought trust through technology to certain industries, however, further integrative mechanisms are needed to join together diverse trust systems and applications into a single new trust ecosystem.

Ontology establishes the connecting infrastructure for a trust ecosystem, with effective coordination of trust and data sources, as well as providing the infrastructure for distributed application development. [1]

This paper focuses on Ontology's technological framework, key technology principles, and core protocols.

1. Ontology supports various collaborative trust scenarios. It will continuously expand modules and protocols in response to scenario and application needs. This version of the technical white paper describes the architecture and protocols of Ontology in its first stage. The technical white paper will subsequently be updated in accordance with application development.

Contents

1. INTRODUCTION

Ontology is an integrated multi-chain and multi-system framework composed of different industries and regions that pass through Ontology's protocols to allow mapping between different chains and traditional information systems. For this reason, Ontology is also referred to as “Ontology Chain Group” or “Ontology Chain Network”, that is, a connector between blockchains.

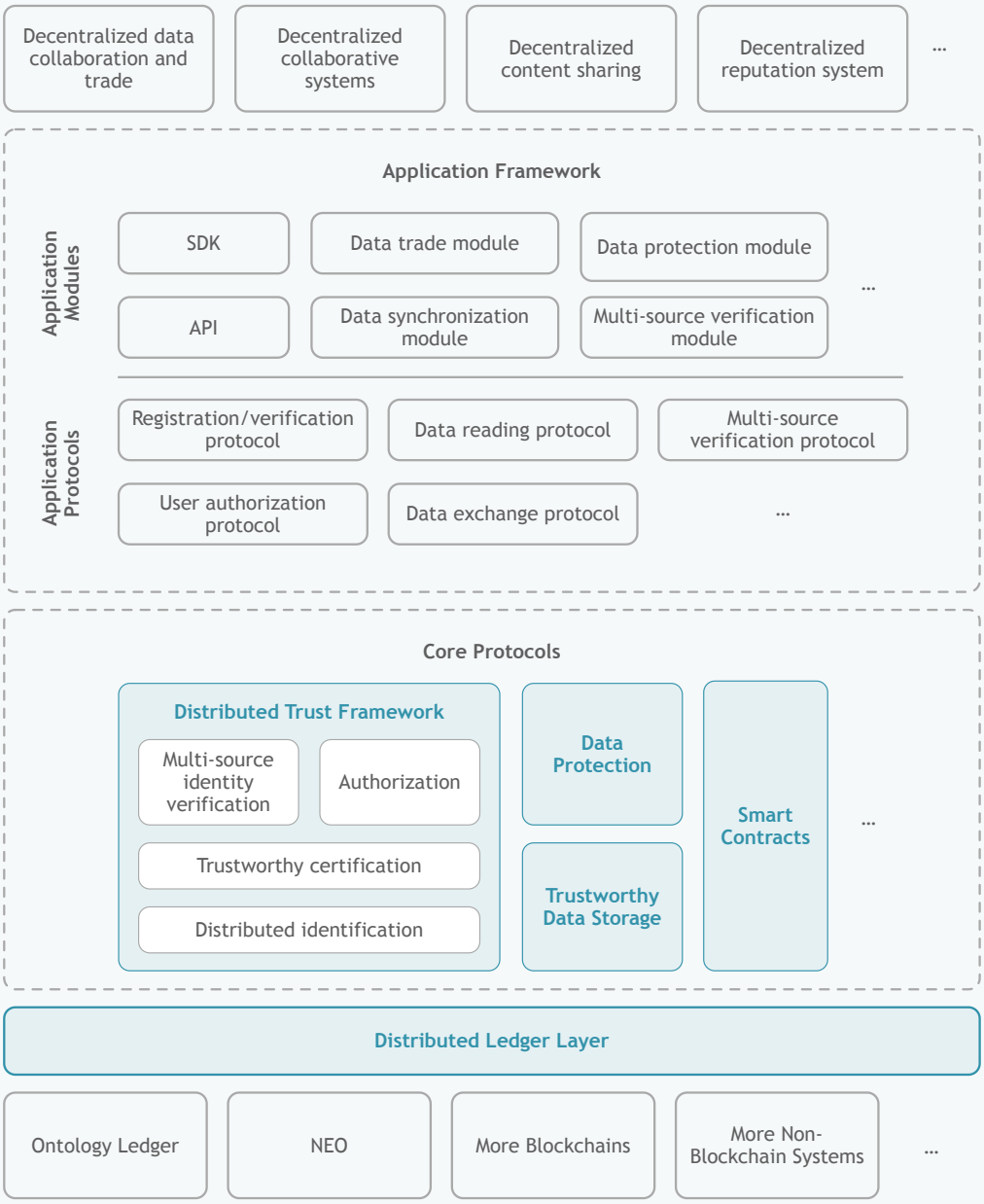


Figure 1.1: Ontology's Technology Framework

At Ontology's core is a complete distributed ledger system, including the Core Ledger, smart contract system, and security system. The distributed ledger is an important underlying storage infrastructure of Ontology; the decentralized, mutually managed, and tamper-proof features of distributed ledger technology are key to creating trust between entities in the network. The distributed ledger includes a consensus and smart contract system, and provides consensus, storage, and smart contract support for the upper application framework. Ontology and its distributed ledger technology use a decoupled architecture (which is the default for the Ontology Ledger) that can support NEO, Ethereum, and other blockchains as the underlying layer. At the ledger level, we creatively propose a shared data contract model that decouples data storage and business logic, and implements different business logic with the use of smart contracts to improve the scalability and flexibility of the architecture on the whole.

The distributed trust framework is the core logic layer of Ontology. We use ONT ID to connect different ID authentication services and provide a source of trust for people, money, goods, and things. ONT ID is decentralized, self-manageable, privacy-protected, and safe and easy to use. The trust framework establishes a distributed trust model and distributed trust delivery system through verifiable claims[1][2], and uses the CL signature algorithm and zero-knowledge proof protocol to assure privacy protection of verifiable claims.

Ontology uses a series of protocol standards. Proposals include an identity protocol, multi-source authentication protocol, user authorization protocol, and distributed data exchange protocol. A range of protocols that are compatible internationally are implemented, such as DID[3] designed by W3C. The cryptographic signature protocol also supports cryptography such as the Chinese national cryptography standard, RSA, and ECDSA. The distributed data exchange system is compatible with the widely used authorization protocols, e.g. Oauth[4] and UMA[5], to enable the architecture to meet the goal of being open and standardized, and to support future extensive ecological cooperation and expansion.

For application service provision, Ontology provides the infrastructure for application developers to directly provide distributed services on top of Ontology without having knowledge of how to develop distributed systems. In short, Ontology provides a series of application frameworks, including APIs, SDKs, and various application modules that allow application service providers with a wide range of technological backgrounds to develop their own dApps and make dApps as a Service (DAAS). This makes blockchain easy to use for all.

Ontology also includes a variety of advanced modules: cryptography and data protection modules, data exchange markets, global transaction databases, hybrid oracle, unlimited consensus engines, and more. In the future, Ontology will develop a community of developers and business partners, continuously enriching applications and modules to promote the technological development of Ontology's ecosystem.

2. GLOSSARY

Ontology Chain Group

Also known as the “Ontology Chain Network”, it is formed by chains of of entities based in disparate industries and regions combining together in Ontology. Each chain uses a separate distributed ledger and collaborates through interactive protocols.

Ontology Distributed Ledger

One or more core public service chains part of Ontology's distributed ledger/blockchain framework. It provides core distributed ledger, smart contract system, and other services for all services Ontology.

Entity

Participants in Ontology who can be identified by their ONT ID.

ONT ID

ONT ID is a decentralized distributed identification protocol based on data from an entity's identity service/s, connected with mapping services and other links. It is decentralized, self-manageable, privacy-protected, and safe and easy to use.

Verifiable Claim

A statement to confirm a claim made by one entity about another (including themselves). The claim is accompanied by a digital signature that can be used by other entities for authentication.

Ontology Trust Framework

Modules which make up Ontology's trust ecosystem including the distributed identity verification protocol, distributed trust model, distributed trust transfer, and other modules.

Multi-source Authentication

Refers to many different verifications covering various aspects of the same entity to create a multi-source verification.

Trust Anchor

An entity that has been entrusted to conduct verification. It acting as a source for trust delivery chains and providing identity verification services.

Distributed Consistent Ledger

An incremental modification data storage mechanism in a decentralized P2P network that is maintained as joint nodes. Transparent and tamper-proof in nature, it provides trusted storage and smart contract support for Ontology.

Consensus

Each node confirms data written onto the ledger according to a protocol to ensure consistency.

Smart Contract

Executable code recorded to the ledger runs through the smart contract engine which runs on the ledger's nodes. The input and output of each execution can also be recorded on the ledger.

Ontology Application Framework

A generic term referring to application modules, protocols, SDKs, and APIs, to facilitate fast, low-cost access to third party dApps.

Hybrid Oracle

Hybrid Oracle is a service that provides credible external data to the blockchain. With it users can predict outcomes of events outside of the blockchain system and record these permanently on the blockchain.

3. CHAIN GROUP STRUCTURE

The aim of Ontology is to build a bridge between the real world and distributed data systems. Due to the diversity, complexity, and specialization of businesses in the real world, considerations in performance, scalability, and applicability must be taken into consideration, of which it is hard to use one public chain/ affiliated chain network to support all scenarios. In practice, different business logics require different chains to meet the needs of different scenarios with different access methods and governance models. Also, many business scenarios do not exist independently, and require diversified interaction with other scenarios. Therefore, different protocols need to be provided between these different chains to support inter-service process cooperation.

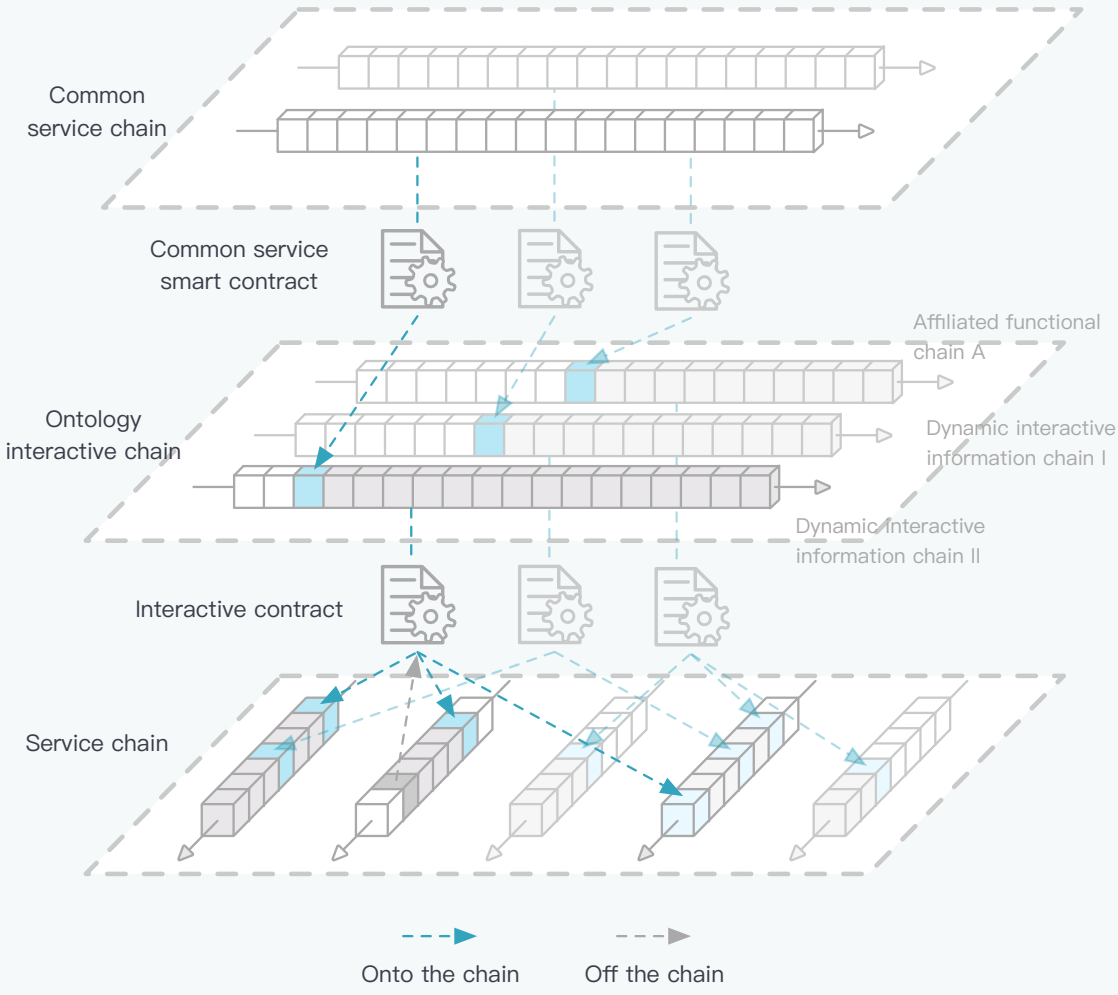


Figure 3.1: Hyper-Converged Chain Group

Based on these requirements and models, Ontology proposes a hyper-converged chain group, which takes the form of a matrix grid. In a horizontal area there may be public chains that provide basic common services, such as entity mapping, protocol support for data exchange, and smart contract services. On one or more public blockchains, each industry, geographical area, and business scenario can have its own unique service chain that can meet its requirements for access, compliance, governance, consensus, etc. Each can also use public chains to provide basic services, such as entity authentication, data exchange protocols, etc., as well as collaborate with other businesses.

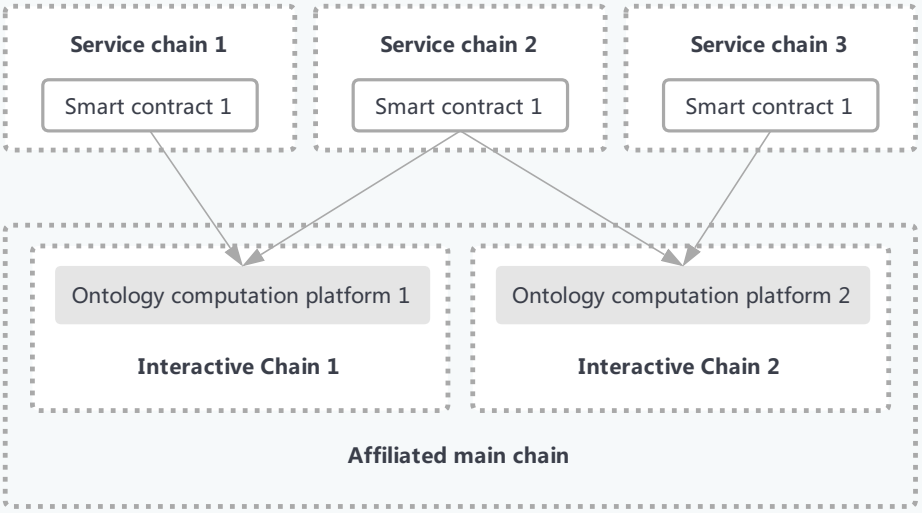


Figure 3.2 Service chains using the Ontology computation platform for cross-chain information exchange

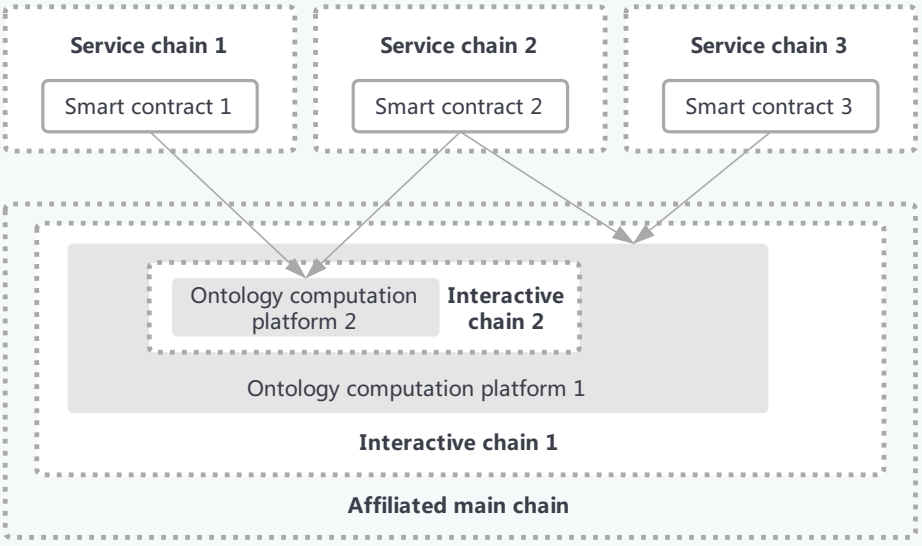


Figure 3.3 Building a dedicated interactive chain using Ontology's computation platform

In addition to the use of the public chain, there will also be collaborations with chains related to the industry of the specific business. Under different collaborations, the service chains involved may also vary, so there may be some small dedicated public/affiliate service chains that cooperate with one or several service chains or service points for specific business requirements. Therefore, in the vertical area, many business collaboration chains will emerge that will include special cross-chain collaborative support for smart contract services, business logic services, etc.

The matrix grid architecture can form a truly autonomous next-generation, versatile network. Different business scenarios can find different ways to apply the right service model through a wide range of collaborative methods.

The various protocols in Ontology are not static and users can choose different protocols according to different business scenarios, industry features, regulatory requirements, and governance requirements. Therefore, in Ontology, protocols will continue to be part of the development process, though the main aim is to maximize usability with different protocols and standards within each scenario to make Ontology have better compatibility and scalability with the world.

Ontology uses its distributed ledger framework to satisfy different scenarios with the implementation of one or more configurable blockchains. The distributed ledger framework can also customize the service chain for specific business scenarios (e.g. different mechanisms for access, governance, consensus, storage, etc.). In addition, Ontology can collaborate with other existing blockchain systems and traditional IT systems through the use of protocols.

4. DISTRIBUTED TRUST FRAMEWORK

4.1. ONTOLOGY IDENTIFICATION PROTOCOL

“Entity” refers to individuals, legal entities (organizations, enterprises, institutions, etc.), objects (mobile phones, automobiles, IoT devices, etc.) in the real world, and “identity” refers to the entity's identity within the network. Ontology uses Ontology Identifier (ONT ID) to identify and manage the entities' identities in the network. In Ontology one entity can correspond to multiple individual identities.

ONT ID is a decentralized identity protocol that links and maps different authentication services of a single entity. It is decentralized, self-manageable, privacy protected, and safe and easy to use. Each ONT ID corresponds to an ONT ID Description Object, which is used to record the attribute information, such as the public key, of the ONT ID. Description objects serve as public information for storage in the distribution layer at the core of Ontology. For privacy reasons, the description object by default does not include information about the identity of any entity.

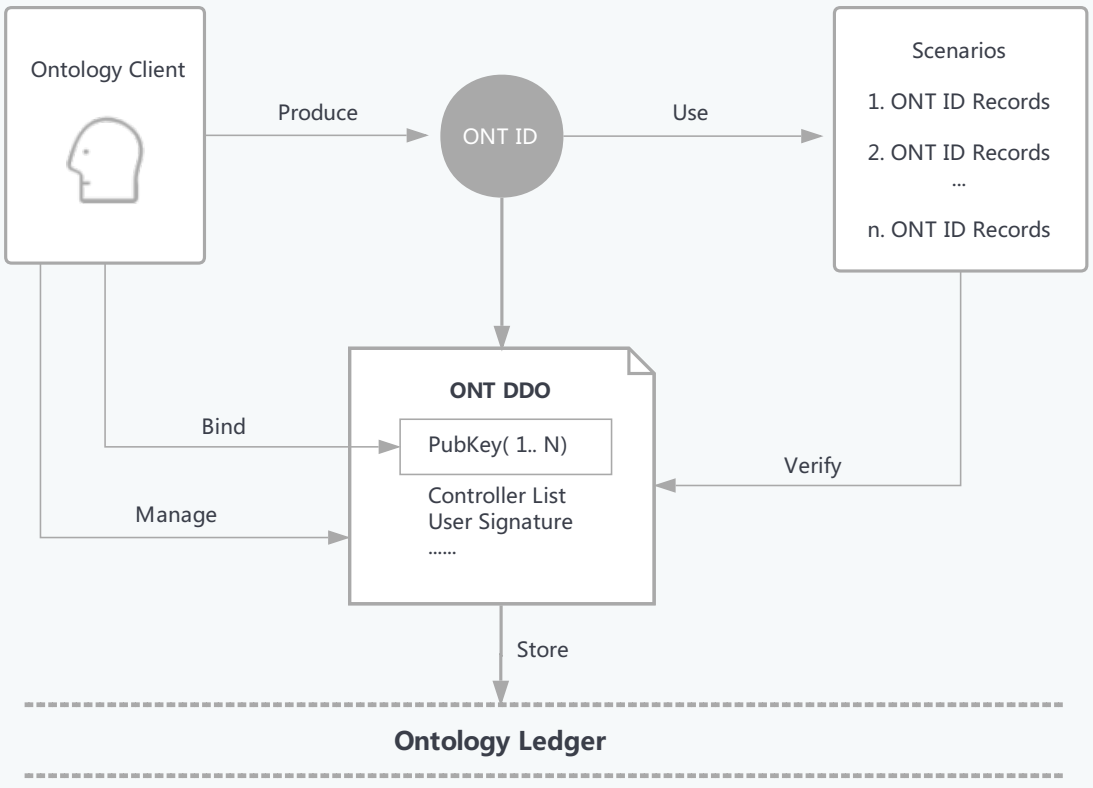


Figure 4.1: ONT ID System

4.1.1. ONT ID Generation

ONT ID is a type of URI[6], generated by each entity. The generation algorithm ensures that there is a very small probability of duplication of two ONT IDs ($\approx \frac{1}{2^{160}}$). At the same time when registering on Ontology, the consensus node checks to see if the ID has already been registered.

4.1.2. Self-Sovereign

Ontology uses digital signature technology to ensure that each entity manages their own identity. ONT ID is registered with the entity's public key to indicate its ownership. An ONT ID's use and its attributes need to be digitally signed by the owner. An entity can determine their own ONT ID's usage, bind a private key, and manage its attributes.

4.1.3. Multi-Key Binding

Ontology supports a variety of domestic and international digital signature algorithm standards, including RSA, ECDSA, and SM2. Private keys bound to an ONT ID need to specify the algorithm used, and at the same time, ONT ID can bind several private keys to meet entity requirements in different application scenarios.

4.1.4. Authorized Control

The owner of the ONT ID may authorize other ONT IDs to exercise management rights over their ONT ID. For example, the attribute information corresponding to the ONT ID may be modified, or another private keys can be bound to an ONT ID if the original key is lost. ONT ID supports fine-grained permission management for each attribute and multiple access controls such as “AND”, “OR”, “m of the n”.

4.2. TRUST MODEL

Ontology's trust model generates trust between entities using both centralized and decentralized trust models. Different trust models can be used according to specific scenarios to meet different needs.

4.2.1. Centralized Trust Model

Under this model one or a group of entities act as the trust anchor and the trust relationship between entities is established based on the trust anchor. The trust anchor specifies the entities it trusts, and the other entities can in turn designate other entities they trust. This format, with the trust anchor as the

source, creates a trust tree of trust delivery is created. Each node in the tree has a path (its "trust path") to the trust anchor, which is its chain of trust. When interacting with other nodes in the tree an entity that recognizes the trust anchor can verify its chain of trust.

As the most mature and widely used trust system, PKI[7] is a centralized trust model. To become a trust anchor a user must first apply for a digital certificate. After approving the application, the certification center writes its identity information and public key into the digital certificate, and then adds the digital signature of the certification authority. The digital certificate issued by the certificate authority authenticates the binding between the user's identity and the public key. Anyone can use the public key of the certification center to verify the authenticity of the certificate, and then use the public key in the certificate to verify the user's signature and confirm its identity. At the same time users who have digital certificates can also issue digital certificates to other users as subordinate certifiers, and the validity of the digital certificates issued by them is guaranteed by the certification center. In the PKI model every participant must unconditionally trust the certificate authority, and the trust relationship is passed from the certificate authority layer by layer between entities though digital signatures.

The centralized trust model has many advantages. The strict method of trust transfer and the clear trust and non-trust boundary are good features in many scenarios and can solve many problems. However, there are certain disadvantages to the centralized trust model. The dependence on the central node may be unsuitable for complex trust relationships, as well as for in cases where there are higher demands for honesty and security. This method of relying on the central node can severely limit application flexibility.

4.2.2. Decentralized Trust Model

In addition to relying on specific central entities to build trust relationships, entities can also build equally strong trust relationships by themselves. Trust transfer is achieved through mutual authentication between entities. Entities will have higher credibility if they receive more authentications from other entities – especially if those other entities have high credibility.

The decentralized trust model is a blurry trust model, and there is no clear trust boundary. Under different scenarios different trust assessment methods can be used to evaluate the trust of entities. It is because of this high degree of flexibility that the model has a wide range of uses in real life.

4.3. VERIFIABLE CLAIM

Verifiable claims are used to prove certain attributes of an entity. They can be stored and transmitted as data units and verified by any entity. A claim includes metadata, claim content, and the issuer's signature, while the content can be any data. Claims are formatted in JSON-LD[8], with signatures following the LD-Signature specification[9].

4.3.1. Life Cycle

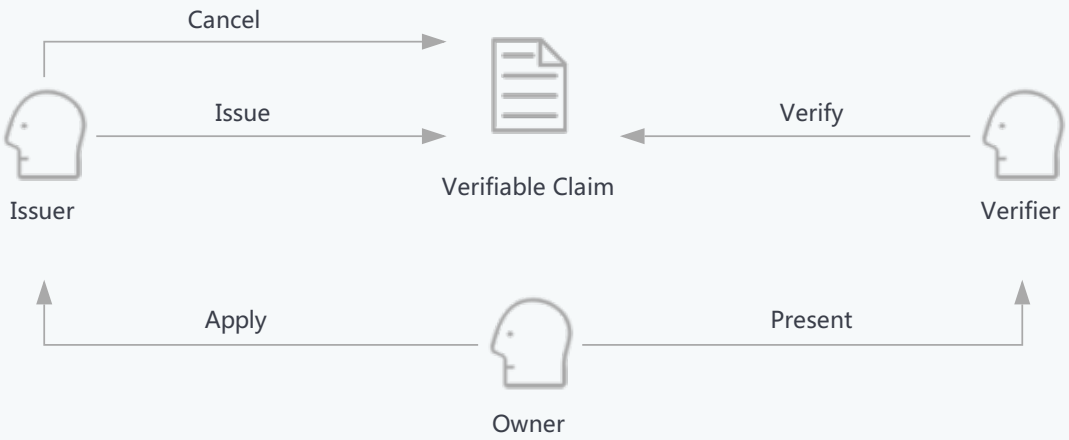


Figure 4.2: Verifiable Claim Lifecycle

Entities associated with a verifiable claim fall into three categories: issuer, holder, and certifier. The life cycle of a verifiable claim includes the following five operations:

- *Issuance: Any entity can issue a verifiable claim of an attribute of another entity. For example, a school may give a student their transcript by issuing a verifiable claim. When a verifiable claim is used a validity period can be set. When the validity period has passed the claim will automatically expire.*
- *Storage: Verifiable claims can be issued as either public claims or private claims. Public claims are stored in a distributed ledger in Ontology, whereas private claims are typically stored on the entity's client and managed by the entity itself.*
- *Presenting: The owner of the verifiable claim can choose to whom the claim is made public and which information is shown without affecting the integrity of the claim.*
- *Verification: The verification of the verifiable claim does not need to interact with the issuer of the claim, it only needs to use the issuer's ONT ID to obtain the public key information from Ontology's distributed ledger. It can then use the public key to verify the digital signature of the claim.*
- *Cancellation: The issuer of the verifiable claim can cancel their claim before the expiry date. The cancelled claim will not be able to be validated.*

4.3.2. Anonymous Claim

In normal circumstances the claim owner exposes the full content of the claim to the verifier when it makes a claim. However, in some scenarios the claim owner may not want to expose certain content of the claim to the verifier. In light of this, Ontologys offers anonymous verifiable claim technology to protect the privacy of its users.

Anonymous Claim technology solves the problem of hiding the holder's information during the process of issuing and presenting a claim. In the anonymous claim protocol an entity receives two verifications of their claim from two different verifiers. Even if the two verifiers wanted to conspire together to leak the information they hold, they would not be able to verify whether the information they received is from the same entity. When making an anonymous claim the issuer does not need to provide the original claim to the verifier, they only need to provide a zero-knowledge proof. The verifier can verify the authenticity of the claim by running a validation algorithm with the issuer's public key, certificate, and an assertion of the attribute values contained in the certificate, e.g. “age > 18” AND “resident of Shanghai”.

An anonymous claim is usually an XML or JSON file that contains both public and cryptographic information. The public information includes all the attributes of the anonymous claim, consisting of three parts: the name of the attribute, the type of the attribute, and the value of the attribute. Attributes support a variety of data types, such as strings, integers, dates, and enumeration types. The cryptographic information mainly includes the owner's own master key and the issuer's public information digital signature.

During the presentation of the anonymous verifiable claim the owner proves to the third party verifier that he owns an anonymous claim from an issuer. They can selectively expose some attribute values and hide other attribute values. In addition, they can prove that some hidden attributes satisfy certain logical assertions.

Anonymous claims use the CL signature scheme[10] and Σ-protocol[11] to achieve the features mentioned above.

4.3.2.1. CL Signature Scheme

The CL signature scheme consists of three algorithms, namely the key generation algorithm, signature generation algorithm, and signature verification algorithm. This scheme is provably secure under the strong RSA assumption.

Key generation GEN:

1) *Choose two random secure primes* (p, q) *and compute* $n = pq$.
2) *Choose* $k + 2$ *random quadratic residues* $(R_1, \ldots, R_k, S, Z)$.
3) *Output the private key* $sk = (p, q)$, *and the public key* $pk = (n, R_1, \ldots, R_k, S, Z)$.

Signature generation $SIGN_{sk}(\{m_i\})$:

Input k values $\{m_1, \ldots, m_k\}$.

1) *Compute the Euler totient function* $\varphi(n) = \varphi(p)\varphi(q) = (p-1)(q-1)$.
2) *Choose a large prime e and a large integer v randomly.*

3) *Compute the inverse of e module $\varphi(n)$, denoted as e^{-1}, which satisfies*

$$e \cdot e^{-1} \equiv 1 \pmod{\varphi(n)}$$

4) *Compute an integer*

$$A \equiv (A^e)^{e^{-1}} \equiv \left(\frac{Z}{S^v \cdot \prod_i R_i^{m_i}} \right)^{e^{-1}} \pmod{n}$$

5) *Output signature (A, e, v).*

Signature verification $VERIFY_{pk}(\{m_i\}, A, e, v)$:

Input k values $\{m_1, \ldots, m_k\}$ and the signature (A, e, v).

1) *Check if e and v are in the given interval and e is a prime.*
2) *Check if the signature satisfies*

$$A^e \equiv \frac{Z}{S^v \cdot \prod_i R_i^{m_i}} \pmod{n}$$

4.3.2.2. Σ-protocols

Σ-protocols are efficient zero-knowledge proof protocols, which can be used by a prover P to convince a verifier V that he knows some secret, without conveying the secret or any other information. There are some well-known Σ-protocols, e.g. Fiat-Shamir heuristic[12] and Schnorr signature scheme [13].

Assume P knows a solution to the following system of equations $y_1 = g^x,\ y_2 = h^x$ then he can use the following sigma protocol which in standard notation is written as:

$$SPK\{x : g^x = y_1,\ h^x = y_2\}$$

It can be proven using the following interactive protocol:

1) *P chooses a random integer r.*
2) *P computes (g^r, h^r) and sends them to V.*
3) *V chooses a random integer c and sends it to P.*
4) *P computes $s = r - c \cdot x$ and sends s to V.*
5) *V verifies whether s satisfies the following equations*

$$g^s \cdot y_1^c = g^r, \quad h^s \cdot y_2^c = h^r$$

4.3.2.3. Anonymous Claim Issuance

To issue an anonymous claim to recipient R, the issuer I and recipient R need to run a two-round interactive protocol as follows:

1) *I chooses a random integer n_1 and sends it to R.*
2) *R chooses a random integer v' and computes $U = R_1^{m_0} S^{v'} \pmod{n}$, where m_0 is the master secret. Then R sends U to I.*
3) *I choose a prime e and a integer v'' at random.*
4) *I computes the CL signature (A, e, v'') on the R's pubic attributes $\{m_i\}$ and sends it to R.*
5) *R computes $v = v' + v''$ and save the final anonymous claim (A, e, v).*

The recipient will get a CL signature on his public attribute values from the issuer by running this protocol. The CL signature constitutes the cryptographic information of an anonymous credential.

4.3.2.4. Presentation and Verification

The holder of an anonymous claim can selectively disclose some attributes to the verifier and keep the others private. The anonymous verifiable claim scheme also allows for proving the correctness of certain assertions.

To keep the unrevealed attributes private, the holder needs to prove his knowledge of the unrevealed attributes. Suppose there are *l* attributes $\mathbb{H}_l$ kept private among k attributes $\{m_1, \ldots, m_k\}$. The proof can be constructed as follows:

1) *Choose a random integer r_A and use it to randomize the CL signature*

$$A' = A \cdot S^{r_A}, v' = v - e \cdot r_A$$

2) *Build the proof using the protocol described in section 4.3.2.2:*

$$\pi = SPK\{r_A, e, v', \{m_i : m_i \in \mathbb{H}_l\} : VERIFY_{pk}(\{m_i\}, A', e, v') = TRUE\}$$

3) *Output the proof π.*

To illustrate the proof of a predicate, we show how to prove the inequality predicate of some private attributes. A greater than inequality predicate specifies an attribute value m, and lower bound b. Here we give a protocol to show that $m \geq b$ without leaking the actual m.

1) *Compute $\Delta = m - b$ and decompose it into the sum of squares of for integers:*

$$\Delta = u_1^2 + u_2^2 + u_3^2 + u_4^2$$

2) *Generate four random numbers T_{u_i} to hide these four numbers. Then construct the proof:*

$$\pi_1 = SPK\{(u_i, r_i) : T_i = S^{u_i} Z^{r_i}\}$$

3) *Choose a random number T_Δ to hide the difference Δ, and construct the proof:*

$$\pi_2 = SPK\{(u_1, u_2, u_3, u_4, a) : T_\Delta = \prod_i T_i^{u_i} Z^\alpha\}$$

4) *Construct the proof for $m \geq b$:*

$$\pi_3 = SPK\{(m, r_\Delta : S^b T_\Delta = S^m Z^{r_\Delta}\}$$

5) *Output* (π_1, π_2, π_3).

Combining π and (π_1, π_2, π_3) we get the complete proof. The verifier can use the verification method described in section 4.3.2.2 to check each subproof. The predicate is true if and only if all subproofs are valid.

5. DISTRIBUTED LEDGER

5.1. ONTOLOGY LEDGER

5.1.1. Consensus Mechanism

Ontology Ledger supports the next generation Ontorand Consensus Engine (OCE). OCE is an efficient, consensus-based version of the dBFT consensus protocol and verifiable random function (VRF) consensus engine. It has reached near-infinite scalability and requires a relatively low hashing rate, making network forks highly unlikely. dBFT has demonstrated excellent stability and reliability during its operation on NEO's public chain and other affiliated blockchain projects. OCE's block-creation speed is only limited to internet speed, usually resulting in confirmations within 10 seconds. OCE selects who will participate in consensus validation using a verifiable random function and reaches consensus by using a Byzantine fault tolerance algorithm[14][15][16]. Meanwhile, the seed will be generated by signatures of validator group and direct to the next validator group. OCE also supports pluggable verifiers and online protocol recovery and upgrade.

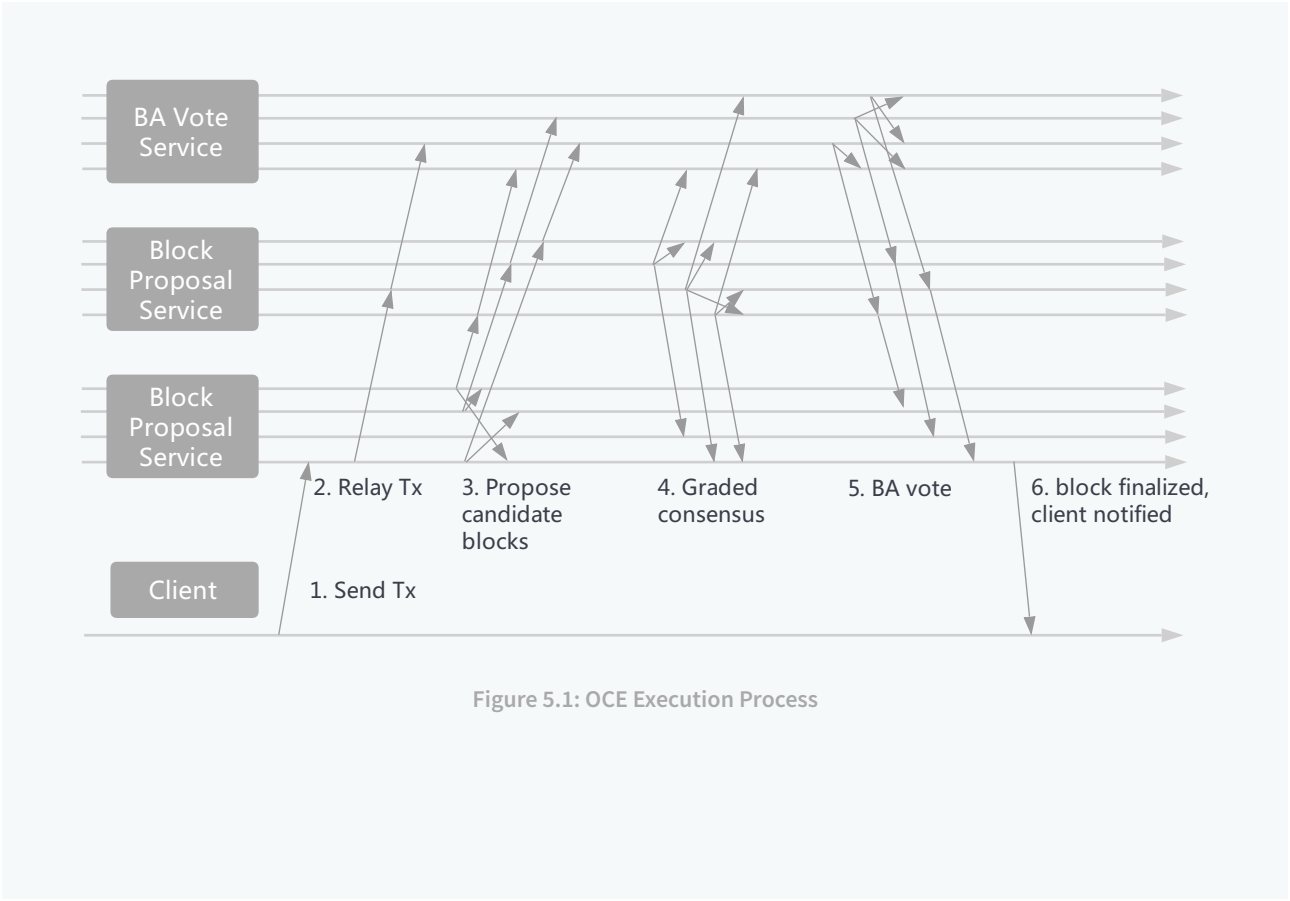


Figure 5.1: OCE Execution Process

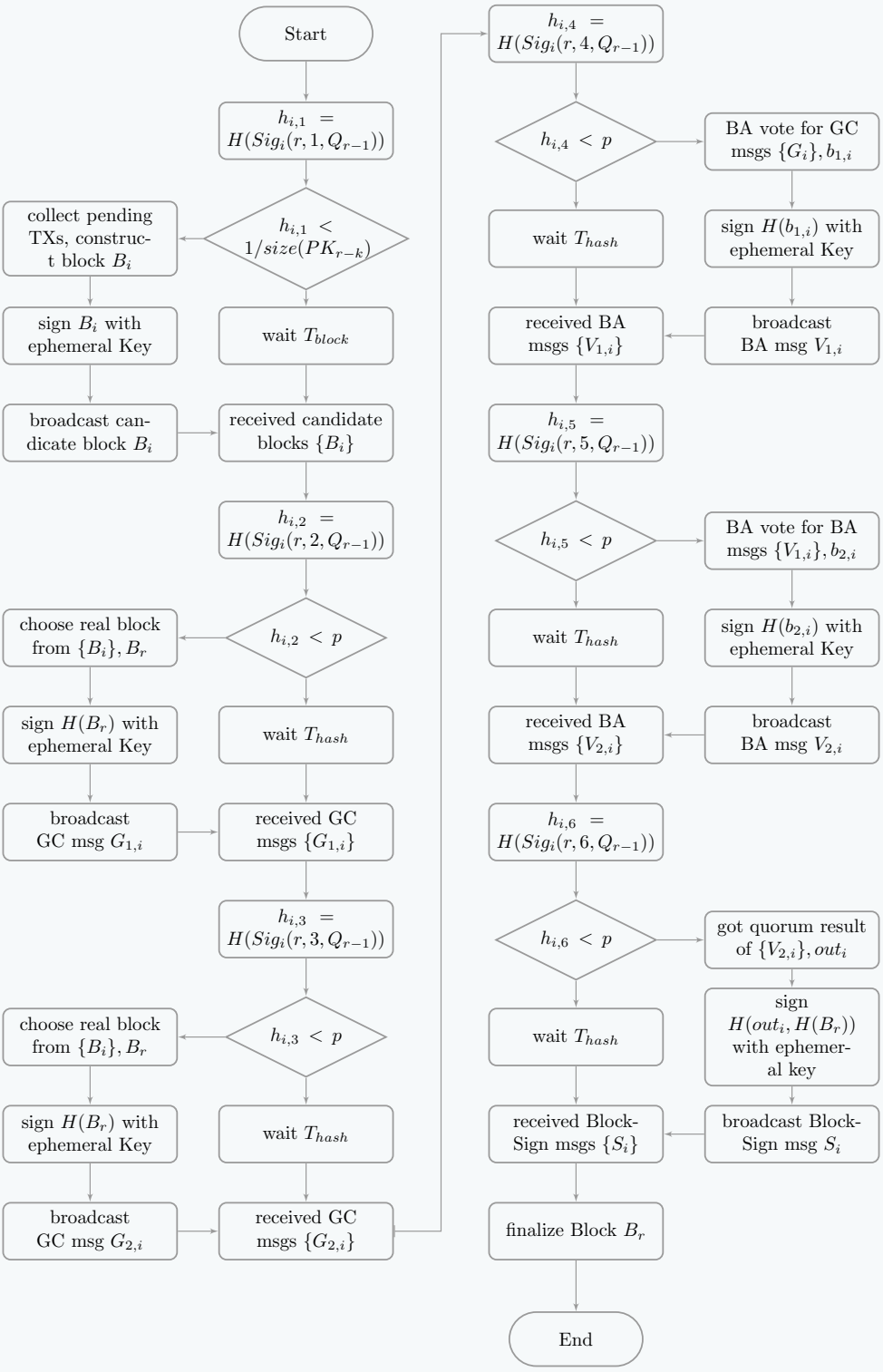


Figure 5.2: OCE Flow Chart

The processing steps and workflow of OCE are shown in figure 5.1 and figure 5.2.

1) *The client broadcasts a data exchange request through the P2P network with the requesting party's signature attached.*
2) *Any nodes participating in the consensus process can monitor all transactions in the network and store them in a local cache.*
3) *Proposing a new block:*
    a) *Nodes can generate a signature based on the current block height as well as the Q value of the last block, and calculate its corresponding hash value. The hash value can then be utilized to determine whether the node can be a potential proposer of the current block height.*
    b) *If a node has evaluated that it could be a proposer of current block height:*
        i) *It packages current and collected non-consensus-transaction requests and constructs a new block.*
        ii) *It creates a signature based on the data and hash value of the new block, and broadcasts block and signature to the whole P2P network.*
4) *All nodes will wait for T_{block} when monitoring network traffic for block proposals by itself and store the received block proposals in local memory temporarily.*
5) *New block leveled consensus:*
    a) *After T_{block} each node in the network evaluates whether it could be the verification node for the current block proposal.*
    b) *If a node has evaluated itself as a verification node of the current block proposal it:*
        i) *Verifies the correctness and integrity of all block proposals.*
        ii) *Verifies the correctness of the Q value signature of all block proposals.*
        iii) *Verifies the legality of the identity of the block proposer.*
        iv) *Calculates the hash value based on the Q value signature, and selects the block proposal with smallest hash value as the true chosen block by the current block.*
        v) *Generates the signature of the hash value of the chosen block proposal, and broadcasts the block hash and signature to the network.*
    c) *After T_{hash}, the node updates the algorithm step and assesses whether itself should participate in the verification process of the block proposal during this step in the process.*
    d) *If a node has assessed itself as a verification node for the block proposal in the current step it:*
        i) *Verifies the block proposal in the same manner as described above.*
        ii) *Generates the hash value signature of the selected block proposal and broadcasts the block hash value and signature to the P2P network.*
6) *All nodes on the P2P network monitor consensus messages by themselves that were broadcast by block verification nodes.*
    a) *Verify the correctness and integrity of consensus messages.*
    b) *Verify the validity of the identity of the node that broadcast the consensus message.*

c) *Temporarily store the verification message in local memory.*

7) *Byzantine election for the new block.*

a) *After T_{hash}, nodes update the algorithm step and evaluate by themselves whether any of them should engage in the election of a new block.*

b) *If a node determines that it should participate in election, it executes the following steps:*

i) *Counts received polls of consensus verification of different block proposals.*

ii) *Votes on the basis of the result of consensus result if a quorum is reached.*

iii) *Signs the vote, calculates the hash of the selected block, and broadcasts the vote as well as its signature to the P2P network*

c) *All nodes on the network will cast multiple rounds of voting according to the configuration of their blockchain. Each round waits for the T_{hash} interval and operates independently.*

8) *All nodes on the network are constantly monitoring and verifying the validity of voting messages broadcast on the P2P network.*

9) *Generating the final signature of a new block:*

a) *After the last round of byzantine election, nodes will update the algorithm step, and assess whether any of them should cooperate in creating the final signature of a new block.*

b) *If a node has evaluated that it should cooperate in creating the final signature, it executes the following steps:*

i) *According to the vote count, assesses whether the block proposal has reached a quorum-backed consensus.*

ii) *If no quorum-backed consensus is reached, sets the consensus result of the new block to be false.*

iii) *Packs the polling message on the basis of the former assessment and signs it.*

iv) *Broadcasts the final block hash as well as the signature of the polling result to the P2P network.*

10) *All nodes continuously monitor the final signature message of new blocks in the P2P network and verify the validity of signature messages:*

a) *After T_{hash}, nodes calculate the block hash of the current block height in accordance with the received block signature message.*

b) *Nodes compare the calculated block hash with the formerly received block proposal message to get the final block for the current block height.*

Meanwhile, the Q value of the block is calculated based on the final block of the consensus and initializes the consensus process of the block for the next block height.

Ontology Ledger adopts a modularized architecture, which supports a pluggable consensus algorithm which can be easily and quickly replaced by other consensus algorithms, e.g. PoS, DPoS, and Raft, according to different scenarios.

5.1.2. Procedure Protocols

Distributed procedure protocols (distributed transactions) are accomplished through distributed ledger technology, entity cross-chain and cross-system privacy, and specific cross-chain protocols. The identity privacy of entities is protected whilst procedure/transaction steps operate on different blockchains or systems to ensure consistency of the entire transaction.

5.1.3. Attestation Design

Not only data but also data's behavior will be attested to the distributed ledger. This means with every data request that data matching, data retrieval, and data usage are recorded to the ledger, forming a record of the entire process of data whilst ensuring data security and reliability.

5.2. SMART CONTRACT

Ontology Ledger's NeoVM virtual machine, written in Go, is used for executing smart contracts and can implement intelligent control logic to Ontology's application layer framework. The Turing completeness of the NeoVM virtual machine allows for arbitrary logic and a high degree of certainty, which makes sure that same inputs always produce the same outputs, avoiding the container/docker's uncertainty. It is suitable for use in scenarios where certainty is a strong requirement. In addition, NeoVM has high scalability through “deterministic call tree” technology, through which dynamic sharding can be achieved, meaning theoretically unlimited expansion capability.

In addition, using independently developed compilers, other compilers including Java Bytecode and C# MSIL can work with the blockchain virtual machines. This promotes participation amongst smart contract developers, allowing them to write smart contracts using Java, C/C#, Go, and other programming languages without having to learn a new language. The platform’s ease of operation and maintenance makes it easily sharable with other partners (especially information providers), and provides them with smart contract editing capability and accessibility.

The NeoVM virtual machine combines high top language analysis and conversion. The external interface of the virtual machine can be customized with APIs, which can flexibly interact with account and external data. This allows for high performance of native code execution when smart contracts are running. At the same time a virtual machine mechanism for supporting different blockchains has been implemented.

5.3. SHARED DATA CONTRACT MODEL

Application demands from the distributed ledger mainly includes two functions: the first one is the definition and storage of data structures; the other one is business logic processing and interaction with external systems. We have designed a model separating these two parts to make the system have better scalability and flexibility. One part is for data storage (the data contract) and the other part is for the business logic (the controller contract). We call this the “Shared Data Contract Model”.

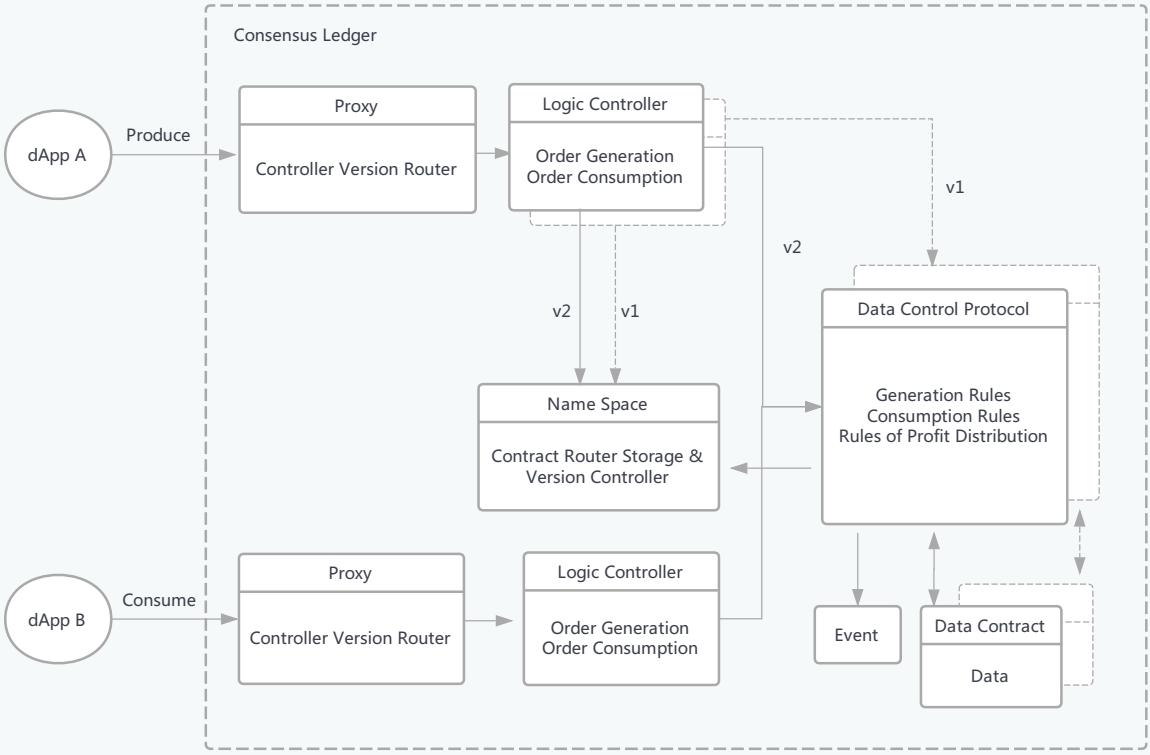


Figure 5.3: Data Sharing Contract Model

The model includes the following design components:

- *Agent controller: Providing an external dApp defined contract entry for other dApps, guaranteeing that even if the contract is upgraded it will not cause inconsistencies in external transfers.*
- *Logic controller: For business logic control.*
- *Namespaces: Data contract address mapping for different versions of different dApps. It ensures data structure upgrades do not affect business logic and can support data traceback between different versions.*
- *Data contract: Provides a basic data structure and storage interface. Similar to DAO, using GET/SET methods.*

- *Data control protocol: General business rules defined by business affiliated parties are digitized into protocol controllers. This protocol should include the following components:*
  - *Connect to the logic controller's permission control*
  - *Interact with the interface of external systems*
  - *Common service logic*
  - *Control logic for the business core ledger*
  - *Event notification mechanism*
- *Event: An event refers to a mechanism for pushing out content during the execution of a smart contract. When each party is in the process of synchronizing the ledger, they can get transaction information by replaying the distributed ledger block back locally. The event is a low-cost storage mechanism and nodes who do not care about the contract node can choose not to implement the contract and not to receive the event information. Nodes corresponding to the smart contract can receive the event information by executing the contract. This can greatly reduce the storage pressure on each node since the chain does not save the actual transaction content.*

This contract model enables dApps to share their data with each other using generic protocols, despite their business. It also makes collaboration across boundaries much easier.

The process of order registration and exchange is shown in figure 5.4.

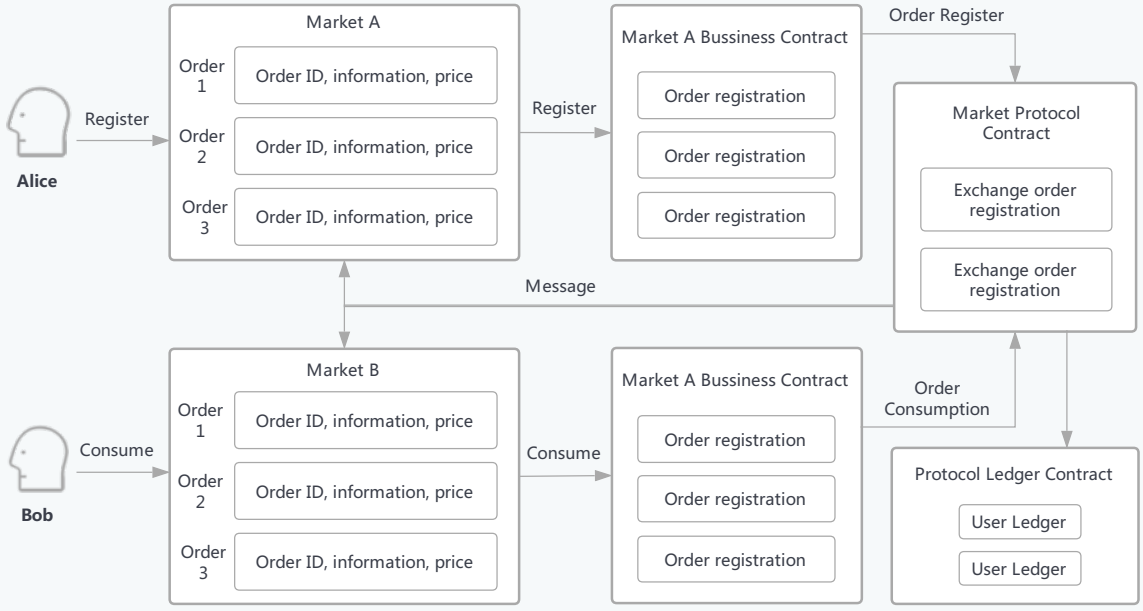


Figure 5.4: Data Exchange Process

Registration:

1) *Alice initiates an order registration request to Market A;*

2) *Market A checks Alice's order, and according to the classification, the classification protocol contract registers the order;*

3) *The market protocol contract examines Market A's authority and the order content. If the criteria is met, then the contract registers the order to the protocol's core ledger;*

4) *After the registration is successful, the protocol contract broadcasts the order registration message, returning the result;*

5) *Market A business protocol contract returns the final result to Alice.*

Transaction:

1) *Market B obtains all transactions by synchronizing with the blockchain.*

2) *Market B obtains the order registration message broadcast by market A through the transaction in the executed block.*

3) *The order information in the order registration message is displayed in Market B.*

4) *Bob initiates the transaction to Market B for the order that Alice registered above;*

5) *Market B checks Bob's transaction request and according to the classification, the protocol initiates the transaction.*

6) *The protocol contract inspects market B's permissions and transaction request information. If the requirements are met, it deals with B's transaction requests and makes the settlement.*

7) *The protocol broadcasts the order's transaction completion information and returns the results to market B.*

8) *Market B returns the final result to Bob.*

9) *Market A receives the protocol contract completion information broadcasted and then sends the transaction completion information to Alice.*

5.4. MERKLE TREE STORAGE MODEL

For digital currency platforms such as Bitcoin and Ethereum, the client usually only focuses on their own account state. To verify the account state without a complete synchronization of the ledger the SPV (simple payment verification) technology is proposed, which can greatly save storage space and reduce network transmission by constructing Merkle proofs[17]. In the Ontology, the client also needs to verify other client's identity information. Therefore, the ledger needs to support Merkle proof construction.

5.4.1. Merkle Hash Tree

A binary Merkle Hash Tree[18][19] is used for efficient auditing. Given an ordered list of n inputs $D[n] = (d_0, d_1, \ldots, d_{n-1})$, the Merkle Tree Hash (MTH) is defined as follows:

$$
\begin{aligned}
MTH() &= sha() \\
MTH(\{d_0\}) &= sha(0x00 \| d_0) \\
MTH(D[n]) &= sha(0x01 \| MTH(D[0:k]) \| MTH(D[k:n])), \quad k < n \leq 2k
\end{aligned}
$$

Where k is the largest power of 2 smaller than n, $D[a:b]$ is the sub-list of D from d_a to d_{b-1}, $\|$ is the concatenation of the two byte arrays.

5.4.2. Merkle Audit Path

A Merkle audit path for a leaf in a Merkle Hash Tree is the shortest list of additional nodes in the Merkle Tree required to compute the Merkle Tree Hash for that tree. Each node in the tree is either a leaf node or is computed from the two nodes immediately below it (i.e., towards the leaves). At each step down the tree (towards the root), a node from the audit path is combined with the node computed so far. In other words, the audit path consists of the list of missing nodes required to compute the nodes leading from a leaf to the root of the tree. If the root computed from the audit path matches the true root, then the audit path is proof that the leaf exists in the tree.

Given an ordered list of n inputs to the tree, $D[n] = (d_0, d_1, \ldots, d_{n-1})$, the Merkle audit path $PATH(m, D[n])$ for the (m+1)th input $d(m) : 0 \leq m < n$ is defined as follows:

$$
\begin{aligned}
PATH(d, \{d_0\}) &= \{\} \\
PATH(m, D[n]) &= \begin{cases} PATH(m, D[0:k]) + MTH(D[k:n]) & m < k \\ PATH(m-k, D[k:n]) + MTH(D[0:k]) & m \geq k \end{cases}
\end{aligned}
$$

Where $+$ is a concatenation of lists.

Figure 5.5 shows an example of Merkle audit path.

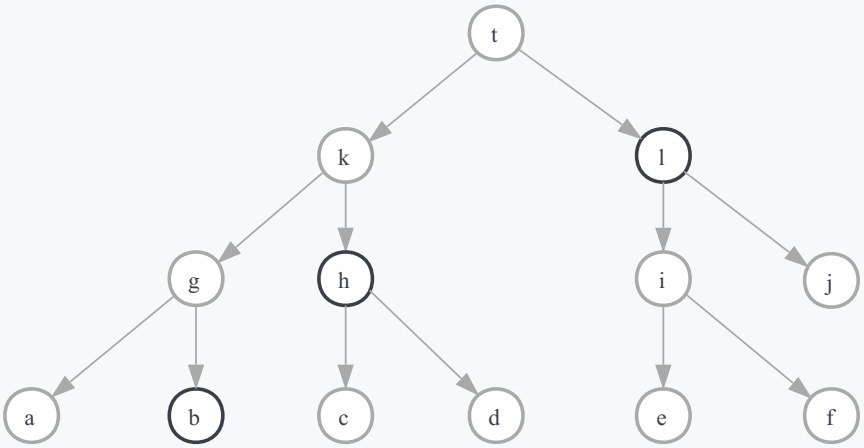


Figure 5.5: Audit Path of a is [b, h, l]

5.4.3. Merkle Consistency Proofs

Merkle consistency proofs prove the append-only property of the tree. A Merkle consistency proof for a Merkle Tree Hash $MTH(D[n])$ and a previously advertised hash $MTH(D[0:m])$ of the first m leaves($m \leq n$), is the list of nodes in the Merkle Tree required to verify that the first m inputs D[0:m] are equal in both trees. The following algorithm can construct the (unique) minimal consistency proof.

Given an ordered list of n inputs to the tree, $D[n] = (d_0, d_1, \ldots, d_{n-1})$, the Merkle consistency proof $PROOF(m, D[n])$ for a previous Merkle Tree Hash $MTH(D[0:m])$ is defined as:

$$
\begin{aligned}
PROOF(m, D[n]) &= SUBPROOF(m, D[n], true) \\
SUBPROOF(m, D[m], true) &= \{\} \\
SUBPROOF(m, D[m], false) &= \{MTH(D[m])\} \\
SUBPROOF(m, D[n], b) &= \begin{cases} SUBPROOF(m, D[0:k], b) + MTH(D[k:n]) & m \leq k \\ SUBPROOF(m-k, D[k:n], false) + MTH(D[0:k]) & m > k \end{cases}
\end{aligned}
$$

Figure 5.6 is an example of Merkle consistency proof.

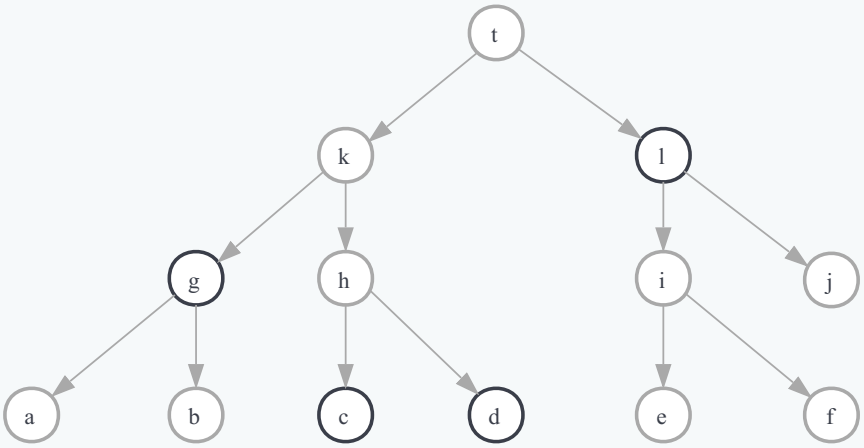


Figure 5.6: $PROOF(3, D[7])$ is [c,d,g,l]

5.4.4. Merkle Patricia Tree

In some scenarios in Ontology we need to quickly prove a transaction's final state for a particular entity, for example to prove the identity of an entity. Using a Merkle proof will require proving each historical transaction one by one, and by using a Merkle Patricia Tree (MPT)[20] efficiency can be greatly improved.

MPT is the combination of Patricia Tree [21] and Merkle Tree. It contains a key-value mapping relation, provides an anti-tamper data structure based on cryptography, and has determinacy, high efficiency, and security.

- *Determinacy: When looking up data the same key-values will ensure finding the same result and will have the same root hash.*
- *Efficiency: New roots can be quickly calculated. When data is changed, the time complexity of insert, search, update, and delete is* $O(\log_2 n)$.
- *Security: When someone maliciously attacks another with a DOS attack and tries to take control of the depth of the tree. Limited tree depth will make this kind of attack unachievable.*

5.5. HYRADAO

HydraDAO is a data prediction and interaction module integrating smart contracts, cross-chain, and cross-data source collaboration. It contains Ontology's DAO (distributed autonomous organization) and cross-chain data interaction (big data/AI) features. Ontology's governance mechanism supports democratic and AI-automated propositions, voting, and verifications. A unique DAO address and polling token will be created during the process, which allows DAO to automatically add funds and results to Ontology. Once polling is complete, DAO will autonomously execute in accordance with the tamper-proof smart contract. The mechanism allows data exchange and governance in Ontology to function with flexibility and supports technology for large-scale automated network operations.

Ontology's Oracle smart contract implementation logic is as follows:

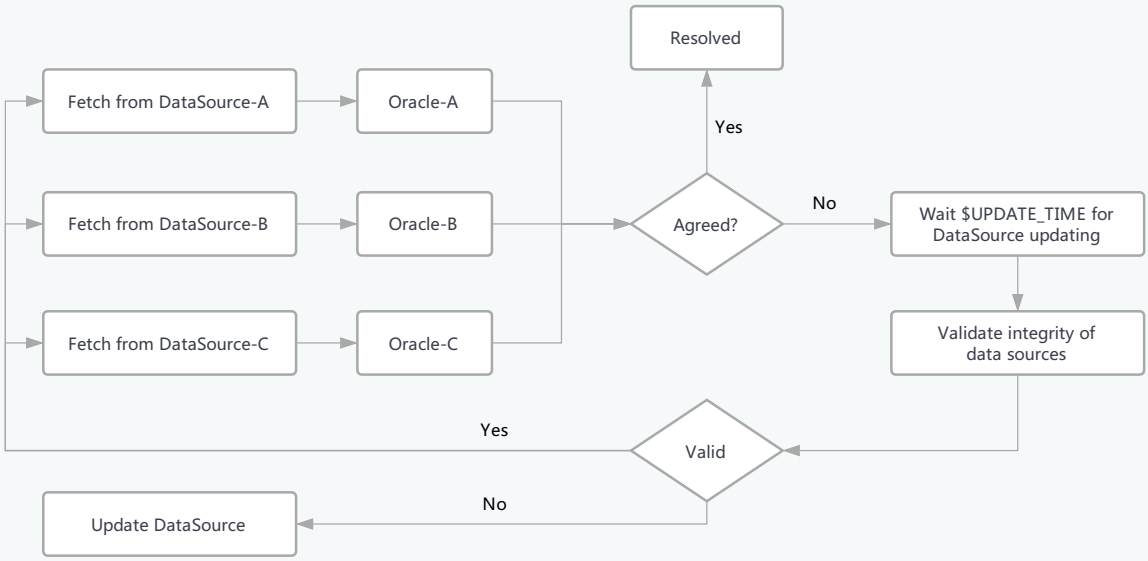


Figure 5.7: Processing Logic of HydraDAO

1) *When any transaction is executed at the beginning or end it is possible to choose to use the Oracle smart contract. Oracle smart contracts and their parameters must be set up at the time the transaction is created and stored on the blockchain as part of the transaction. After the transaction has ended the contract logic is automatically executed to determine the outcome and tokens deposited at the DAO contract address are paid to the participant who provided valid data information. Through Ontology's Distributed Data Exchange Protocol (DDEP) real-time API access to big data computing power is provided along with data market participation in the analysis and prediction of transaction results to verify its integrity. As part of the trust data service, transaction participants can bind trusted data sources or utilize Ontology access services to validate accurate transaction status data before participating in the transaction.*
2) *For transactions that depend only on the status of the blockchain, the Oracle smart contract queries the blockchain data directly from the blockchain interface upon authorization.*
3) *For most transactions related to the real world, Oracle smart contracts require data to get real world transaction results. When an Oracle smart contract needs to access/read data in the real world, each hydra-oracle randomly chooses one of the multiple trust sources or specifies a threshold number for the data anchor. At the end of the transaction the global state is verified and interactively updated via the specified trusted data source[2].*

5.5.1. Built-In DAO Data Prediction

The charm of decentralized systems is not only the reduction of costs and improvement of efficiency, but also the advantage of distributed group decision making. When executing contracts on Ontology there is the option to hook up HydraDAO to activate “public testing”. For example, if a user wishes to buy a certain amount of foreign currency from a foreign currency exchange but the user is not sure if the pending order is correct, he may initiate a DAO project proposal and deposit a certain amount of margin as mortgage. Once a sufficient amount of approval is obtained the proposal goes into the forecast period. Any participant who accepts the proposal can attach a fact statement and their signature. The participant may amend their order at any time, but this requires an additional margin. Once the forecast period is over, the DAO contract will automatically lock the margin in the contract and screen out the best proposals according to the rules of the proposal. Those participants who made the most accurate predictions will be rewarded. User initiated transactions are also automatically executed in accordance with the best output.

Once the proposal is submitted to the DAO the new proposal will be evaluated and decided by the token holders of the entire network. During this period participants can submit facts and amend them at any time. Changing facts may also require voting and ranking. The DAO incentive mechanism for Ontology will ensure the proposal that receives the majority of approvals and is the closest to the fact is the one which receives the maximum reward.

2. All usage of data needs to be authorized by the data owner or relevant authorizing entities, and follow privacy regulations.

HydraDAO automatically starts arbitration after the forecast period is completed. Once arbitration is triggered HydraDAO will calculate and sort all submissions and assign the DAO contract funds to the account of the receiver of the reward. To finish off HyrdraDAO will output a fact status to bind to the smart contract.

5.5.2. External Trusted Data Source

To avoid disputes, pre-defined configurable Oracle smart contracts ensure that value transfer does not occur until smart contract validation conditions are met. Oracle smart contracts can support any JSON API type for easy definition and development.

Trusted data source access process:

1) *Select the data location, data source obtain frequency, expiration date, API certificate, privacy terms, and other input parameters to generate Oracle smart contracts.*

2) *Define the smart contract for other verifiers to evaluate.*

   Descriptive information should include clear instructions to explain the contract's function. The contract's label makes it easy for users to find it on the blockchain and enhances transparency. Detailed "if... then..." descriptions can explain conditions for the transaction to be executed.

3) *Association with real world legal documents*

   Oracle smart contracts use the contract publisher's electronic signatures to sign the binding legal document. Evidence is put on the blockchain by uploading legal documents.

4) *Notifying co-signers of the contract*

   According to the terms of the contract and parties involved, the participants who need to sign are notified by IM/email/phone/etc. The contract will not execute until all signatories sign (before the deadline). If the signatories do not sign before the deadline the contract will fail. When a participant signs the contract the Oracle smart contract assigns the participant a fund escrow address.

5) *Recharging tokens for the Oracle smart contract escrow address*

   Using an external data source incurs costs. Users must pre-credit the address created by the smart contract, which will be controlled by the smart contract.

6) *Complete the contract release*

   Once the contract is signed and the deposit completed the contract will run autonomously on Ontology. Once a transaction is completed, the smart contract from the escrow address will pay funds according to the contract terms. Funds that fail to meet the requirements of the transaction or any fraudulent transactions will automatically be returned to the escrow account.

7) *(Optional) Data open platform, open data access protocol*

   For some open data sharing platforms Oracle smart contracts can interact directly with related off-chain data, based on provided APIs and data usage protocols. Based on the open and transparent conditions, the terms and legal information of the entire process will be publicly displayed, all blockchain users can query/retrieve, and the non-profit nature of such data will simplify the

configuration process for Oracle smart contracts. There will be no need to maintain the payment relationship, just to maintain data and status updates.

6. CORE PROTOCOLS

6.1. MULTI-SOURCE AUTHENTICATION PROTOCOL

Multi-source authentication is different from existing single-factor authentication systems. Ontology can provide entities with multi-source authentication systems which integrate the authentication of external identity trust sources and the endorsement of entities in Ontology. Not only can an entity provide information on who they are, they can also provide what they own, what they want, what skills they have, and other information related to their identity to create a comprehensive identity portfolio.

Multi-source authentication protocol includes the following two modes:

- *External trust certification: Ontology binds ONT ID to an external trust source with a self-signed verifiable claim. Any entity can verify the identity of an entity by verifying the external trust source bound to the ONT ID. The trustworthiness of an entity's authentication is determined by the trustworthiness and acceptance of external trust bound to the ONT ID.*
- *Authentication between Ontology entities: Entities in Ontology can also authenticate each other by issuing claims between themselves.*

6.1.1. External Trust Certification

External trust certification supports the following two methods: self-introduction and importing through trust anchors.

6.1.1.1. Self-Introduction

Users bind trust through social media, e-banking, etc., taking advantage of existing trust systems. The principle is very simple, first the user adds a proof address of an external trust source to Ontology. Then the user provides a credible declaration on the proof address, the format of which is as follows:

- *Claim creation and expiration time.*
- *Claim content: including the claim type, ONT ID, social media type, social media username, etc.*
- *Signature: a public key already contained in the ONT ID.*

When the third party needs to verify the user's external identity, it first reads the certification address of the user's trust source in Ontology, then goes to the address to obtain a verifiable claim, and finally verifies the verifiable claim.

6.1.1.2. Importing through Trust Anchors

Trust anchors are usually government organizations and other public institutions, enterprises, non-profit organizations, and individuals with authority who have been verified. Trust anchors use their own authentication methods to authenticate entities and issue verifiable claims to the authenticated entities. The claim does not need to contain the identity information of the authenticated entity; only the ONT ID and the authentication service are recorded, and the authentication result can be provided. The authentication model is as follows:

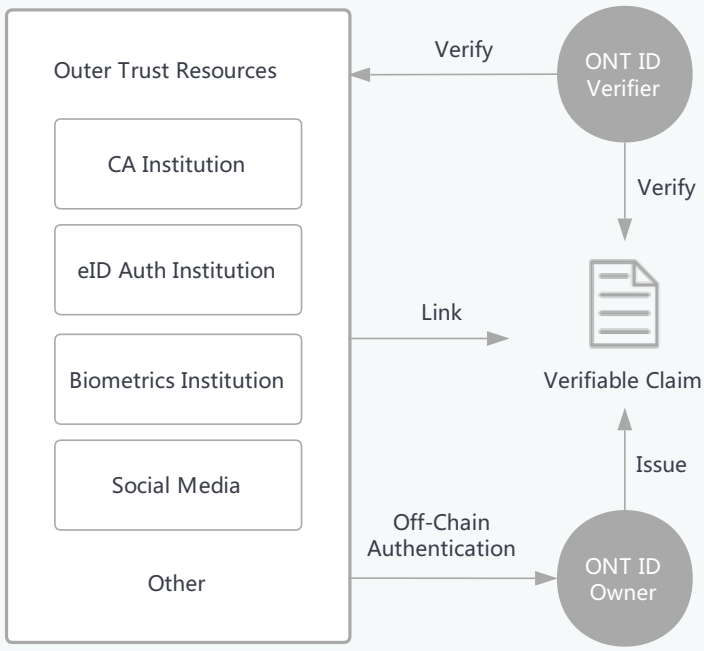


Figure 6.1: Importing through Trust Anchors

6.1.2. Identity Authentication between Ontology Entities

Entities in Ontology can also authenticate identity through entities that have passed identity authentication (from an external trust source) in Ontology, as showed in Figure 6.2.

Due to the versatility of verifiable claims, any entity in Ontology can make a claim about anything regarding another entity. This takes identity authentication in Ontology far beyond the traditional concept of identity authentication. By collecting claims from government institutions, schools, hospitals, banks, businesses, families, friends, partners, community leaders, colleagues, and teachers, we can create a more multi-faceted authentication far more effective than traditional systems.

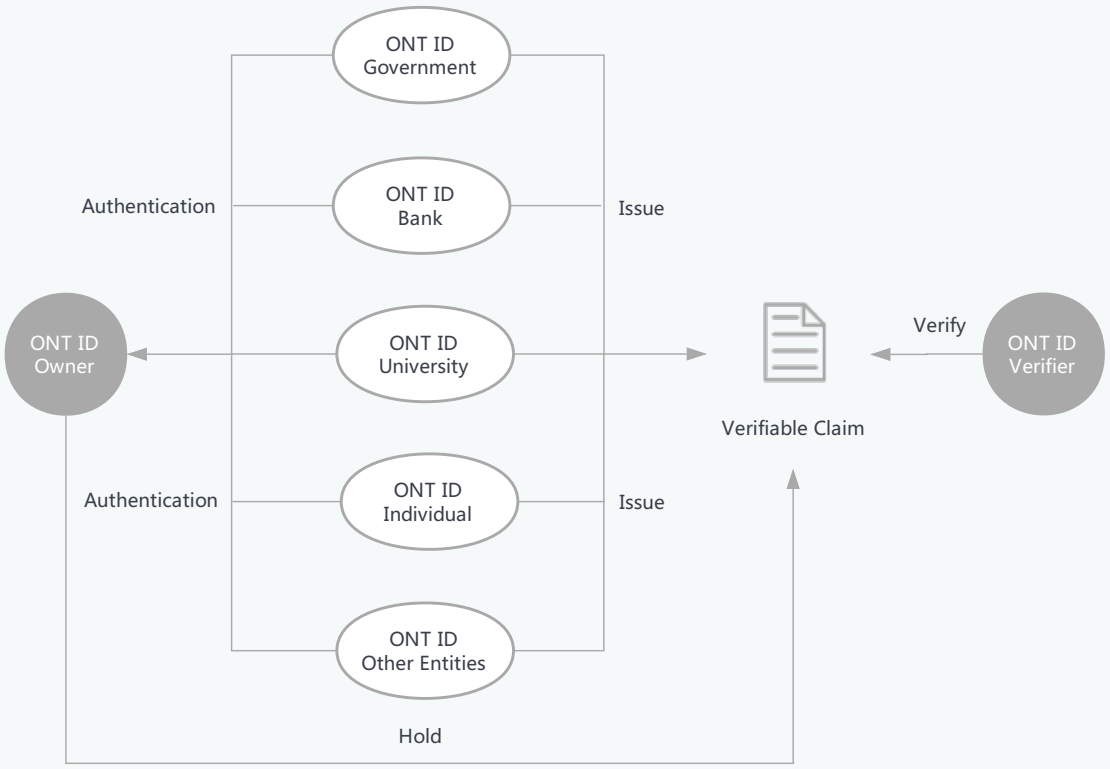


Figure 6.2: Authentication between Entities

6.2. USER AUTHORIZATION PROTOCOL

In Ontology user has complete control over their own data. Any access or transaction involving the data needs to be authorized by the user. In light of this, we have devised a set of user authorization protocols to protect users' data privacy. The protocol uses verifiable claims to perform asynchronous and verifiable authorization, and supports delegated authorization and fine-grained access control.

6.2.1. Roles

The main roles in the user authorization protocols are:

- *User: The entity that has ownership of a resource and is capable of granting access to the resource.*
- *Resource Requester: The requester who needs to obtain the user data or other resources.*
- *Resource Provider: The service provider that provides the user data or other resources.*
- *Authorization Server: the server who receives and processes the authorization requests, and can provide delegated authorization service for users.*

6.2.2. Authorization

The user authorization protocol for application scenarios is divided into three phases:

1) *Mutual registration: The user authorizes the resource provider to register the specified resource to the authorization server, and the authorization server register its address to the resource provider.*

2) *Access control policy setting: After the mutual registration is complete, the user can access the authorization server and set the resource's access control strategies.*

3) *Authorization: Resource requester initiates an access authorization request. If the authorization condition is met, the requester will receive an authorization certificate which would be used to request data from the resource provider.*

6.2.3. Mutual Registration

Mutual registration means that the authorization server registers their address with the resource provider so that when the resource provider processes the data access request it returns the address of the authorization server to the requester. At the same time, the resource provider also needs to register its own address with the authorization server so that users can access the user's resource controls.

Before performing authorization operations, user needs to cooperate with the resource provider and the authorization server to complete the mutual registration process.

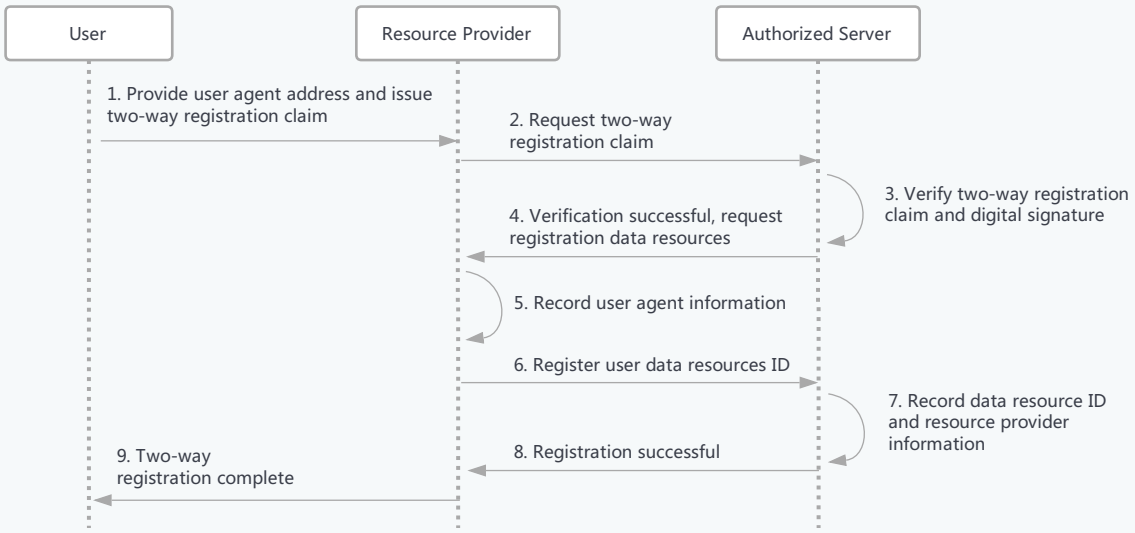


Figure 6.3: Two-Way Registration Process

The process is briefly explained as follows:

1) *The mutual registration process is initiated by user, the user self-signs a mutual registration claim containing their ONT ID, the address of the resource provider, and the ONT ID and address of the authorization server.*

2) *The resource provider handshakes with the authorization server using the mutual registration claim and both parties verify the digital signature.*
3) *The resource provider records the authorization server's access address.*
4) *The authorization server records the resource ID provided by the resource provider.*

6.2.4. Access Control Strategy

With attribute-based access control, users can set flexible access control strategies that restrict specific resources from being accessed by requesters that meet specific attribute conditions. An access control strategy can be described as a boolean expression, such as the strategy $A \vee (B \wedge C)$, which is a combination of three attribute conditions. When requesting authorization, the requester must provide proof of attributes in a verifiable claim that satisfies the strategy.

6.2.5. Authorization Certificate

When receiving an authorization request, the authorization server notifies the owner to perform access control. For requesters who meet the access control strategies, the owner issues a certificate of authority for them. Within the validity period of the certificate, the requester can access the data repeatedly without requesting authorization again.

6.2.6. Delegated Authorization

Authorization servers can act as delegates for users by allowing them to set up their access control strategies. In this pattern the servers can process authorization requests and issue certificates without interacting with users. To prove the validity of the certificates, users need to issue delegate claims for the authorization server.

6.3. DISTRIBUTED DATA EXCHANGE PROTOCOL

The downsides of centralized data exchange include data caching, use of data without user authorization, and data copyright protection. Ontology proposes a Distributed Data Exchange Protocol (DDEP), which defines a set of protocol specifications for data transactions between entities.

To protect the equity of both parties in the transaction, a middleman acting as a “guarantor” is introduced into the transaction process of the agreement to ensure the settlement process is handled securely and smoothly. The intermediary is responsible for keeping the buyer's funds and transferring the funds to the seller or the buyer based on the final trading result. Since the middleman is responsible for

the final settlement of the transaction it is fair and secure. It works on a distributed ledger contract with public and decentralized management features to ensure it can suitably play role of intermediary.

6.3.1. Roles

The main roles in the distributed data exchange protocol are:

- *Data requester: Data agencies/businesses/individuals who want to buy data.*
- *Data provider: Data agencies/business/individuals who want to sell data, both raw and processed. The data needs to meet local government laws and regulations.*
- *User Agent: Responsible for interacting with users to meet user authorization requirements for data transaction. User agents can be diversified (enterprise OA systems, internet platforms, or even simple SMS gateways), but they need to be fully implemented as defined in the application protocol framework's user license protocol.*
- *Data owner: The data subject, which can be institutions/businesses/individuals.*

6.3.2. User Authorization

In the data exchange architecture, since transaction data needs to be authorized by the data owner, the authorization process fully complies with the user authorization protocol described in section 6.2.

6.3.3. Secure Transaction

Secure transaction protocols through smart contracts provide centralized third-party assurance services for trading activities and allows for a secure and smooth transaction process that protects the equity of data requesters and providers.

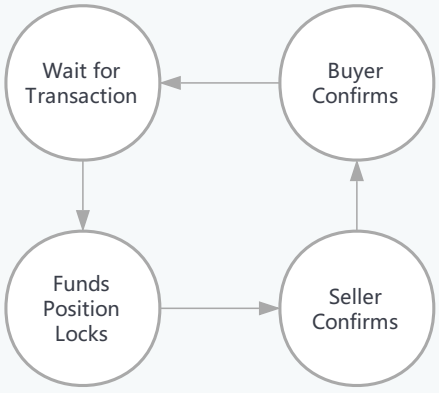


Figure 6.4: Guarantor Transaction Process

The implementation process of the secure smart contract transaction is as follows:

1) *The data provider enters a pending order, writes the product information including the resource ID, data characteristics, provider's account, price, and other features, and the contract waits a requester starting a transaction.*
2) *The data requester transfers to the contract the specified amount and the contract checks whether the amount transferred meets the sale requirements.*
   a) *If it passes the contract enters a funds-locked state*;
   b) *If the check fails, an error message is returned to the transferee and the transaction status returns to its previous state.*
3) *After the provider provides the data, the contract confirms and sets an expiration date. If no other action has been taken at the end of the validity period, the contract automatically enters the settlement process (step 5).*
4) *After receiving the data, the requester confirms with the contract.*
5) *The contract transfers the funds to the provider's account to complete this transaction, and waiting for the next transaction.*

6.3.4. Data Exchange Process

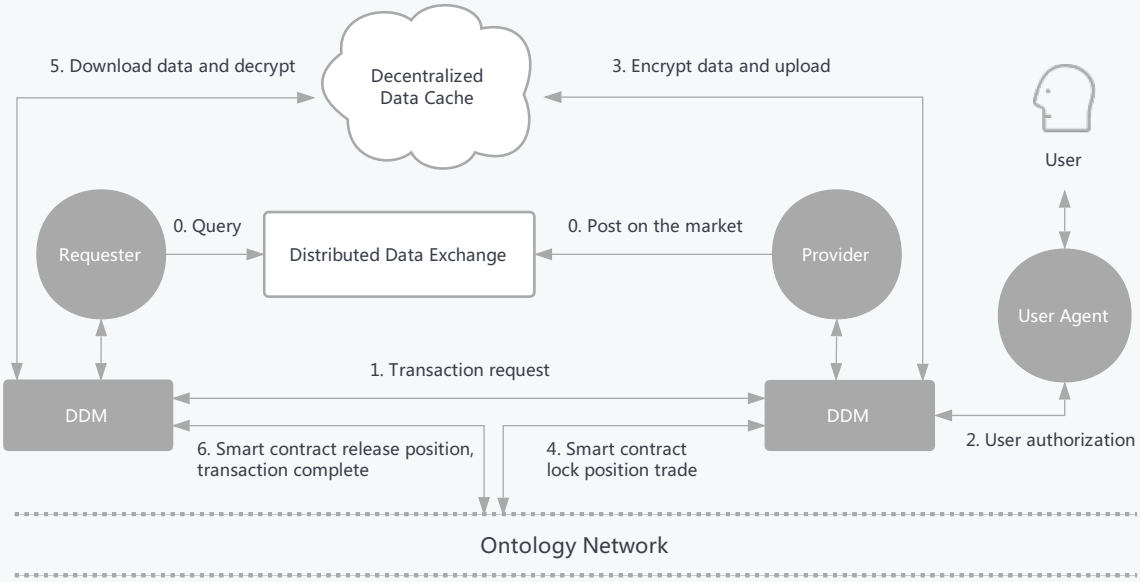


Figure 6.5: Data Exchange Process

Pre-order: transaction preparation

Data product release: The data provider publishes their data product information in the marketplace. Requesters can browse the marketplace, search for data products, and choose data they would like to

buy. Meta Data information should include but is not limited to: data resource introduction, keywords, data resource hash, contract payment address, and other information.

Mutual registration: If the data product requires the authorization of the data owner, the data provider needs to complete mutual registration with the user-appointed User Agent before releasing the data. For more details refer to the mutual registration above.

1) Transaction request

After finding the data they would like to purchase the requester verifies the identity of the provider through Ontology and for more details refers to the multi-source authentication protocol. Before the transaction request is initiated, the requester first deposits funds to the contract address, sends a purchase data request to the provider, and attaches the information required for the user's authorization. The request includes but is not limited to transaction information and ONT ID information.

2) Authorization

After receiving the request from the requester, the data provider accesses the User Agent and initiates an authorization request. At this point the User Agent can authenticate the identity of the requester on demand via Ontology, and perform authorization according to the access control policies provided in advance by the Owner. If the Owner does not set an access control policy, the User Agent notifies the Owner for authorization. If the authorization request is rejected, the transaction should be terminated.

3) Uploading data

The data provider generates a one-time session key according to the symmetric-key algorithm supported by the requester, uses it to encrypt the data and data characteristic values of the transaction, and sends the ciphertext to an intermediate storage system, e.g. IPFS.

4) Locking position

The data provider calls the smart contract to check the funds deposited by the requester. If the amount is correct, the position gets locked until the transaction is completed or cancelled. Meanwhile, the provider encrypts the session key with the requester's public key and sends it to the requester through a secure channel.

5) Receiving data

After receiving the notification of the smart contract event (event referring to section 5.3), The requester then gets the ciphertext from the intermediate storage, decrypts it with the session key, calculates and verifies the characteristics of the plaintext, and move onto step 6) if the verification succeeds.

6) Transaction confirmation

When the data trade contract is complete, the contract funds are transferred to the data provider's account.

Exception handling mechanism: The exception handling mechanism can be customized according to different business scenarios. For example: if the data has not yet been confirmed by the requester within the specified period the provider can make the contract unlock the funds, or the smart contract can automatically unlock the funds.

6.3.5. Privacy Protection

In some data exchange cases participants may want to hide their transactions. We propose using the stealth address technique to make it infeasible to correlate the recipient's token address and ONT ID. The requester generates a stealth address whose private key can be only recovered by the data provider based on the his token address.

Suppose the data provider's token address is $S = s \cdot G$. The requester generates a random number r, compute $R = r \cdot G$. Then the stealth address $E = Hash(r \cdot S) \cdot G + S$, which will be used as the output address in a token transaction. Only the provider can calculate the private key $e = Hash(s \cdot R) + s$ with his own private key s.

7. ONTOLOGY APPLICATION FRAMEWORK

7.1. APPLICATION FRAMEWORK MODEL

Ontology's application framework provides a rich set of application protocols and modules that enable third-party dApp developers to quickly build decentralized applications without having to understand the complexities of the underlying distributed ledger. Ontology's application framework has a high degree of scalability and can be expanded according to scenarios' needs.

Figure 7.1 shows how dApps interact with Ontology through its application framework to achieve decentralized trust:

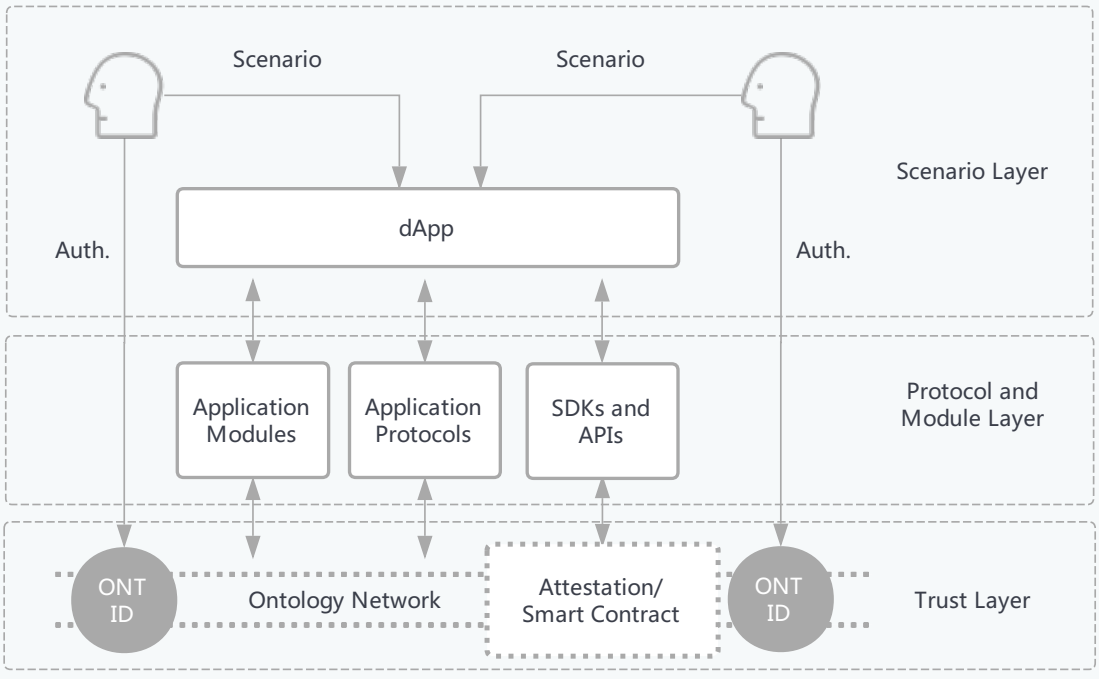


Figure 7.1: Application Framework Model

- *Trust Layer: Ontology takes on the responsibility of a trust system through its identity, attestation, and smart contract systems, etc.*
- *Module and protocol layer: This layer enables the upper-layer scenarios to make better use of Ontology through application modules, application protocols, SDKs, and APIs.*

- *Application layer: dApps for different scenarios and the teams behind them can focus on scenario development, user services, etc. Ontology can help solve trust issues.*

7.2. DATA MARKETPLACE

The future is a digital world and a data exchange market fit for the future will not be limited to only the transfer of data ownership. Trusted collaborative computation will also become an important mode of collaboration. Products in the data marketplaces include data, data forecasting, and data computing resources, etc. Participants include data consumers, providers, and processors such as big data services using deep learning and other AI technology, which together will form a data collaboration ecosystem. This complex data collaboration ecosystem requires a matching infrastructure.

Ontology's Distributed Data Exchange Protocol (DDEP), Data Trading Module (DDM), and a series of cryptographic modules together form a complete distributed data trading and collaboration framework, allowing for global scale trade sizes and exchange needs across industries for Ontology dApp users. Layers of data exchanges service providers allow for a variety of data exchange types across industries.

7.3. DATA DEALER MODULE

Data Dealer Module (DDM) is based on the distributed data exchange protocol and is an important application module in Ontology. DDM enables both the dApp developers and data exchange participants to rapidly utilize data exchange. The modules can use RESTful API, RPC, SDK, etc., and can support multiple types of protocols.

DDM has a variety of types to meet different requirements including data transaction server, single-user client, multi-user client, and light wallet client. There are four major components for ONT identity management, data resource management, smart contract transactions, and P2P communication.

The DDM data transaction module has the following advantages and features:

- *Decoupling from access control module. The module only manages the binding of the data resource and the data resource owner's control server. It does not participate in the configuration and verification of access rights. It not only maintains the equity privacy protection for the data owner, but also enhances the data provider's credit to avoid unnecessary disputes.*
- *Protecting data privacy. Modules do not store the actual data, and with transaction data encrypted, vendors' data precipitation concerns are eliminated.*
- *In addition to data directory retrieval, a data requester can also broadcast his orders which will be received by data providers.*
- *Designed in accordance with the "single module, single function" principle, it easily supports flexible scenarios with cryptographic security modules and user authorization modules.*

7.4. CRYPTOGRAPHY AND SECURITY MODULES

7.4.1. Secure Multiparty Computation

In a typical collaborative computation scenario, two or more participants may want to conduct computation using private data without sharing it with others. Current approaches used do not work in this scenario. For example, if application development company A wants to integrate powerful machine learning algorithms into their application, current systems will transfer their data to ML algorithm provider P. Then P will run their algorithm on the data and return the results back to A. P will know everything about A's data.

For this kind of scenario, we utilize secure multiparty computation (MPC) techniques. The first secure two-party computation protocol was introduced by Andrew Yao, which can be used to solve Yao's Millionaires' Problem: two millionaires who are interested in knowing who is richer without letting the other know their actual wealth. The general problem can be summarized as, n participants each hold some private data x_i and want to compute the value of a public function on that private data: $f(x_1, \ldots, x_n)$. The goal is to design a protocol such that all information each party can learn is what they can learn from the output and their own input. Thus, there is no additional data leakage for every participant.

Two major approaches for MPC have been developed over the years: those based on Yao's garbled circuits[22][24] and those based on secret sharing[23]. In the remainder of this section, we will focus on the latter approach.

To put it simply, in a (t,n)-threshold secret sharing scheme, a secret is partitioned into n shares. Each party gets only one share of that secret. One can recover the secret if and only if he gets at least t shares. MPC protocols based on secret sharing usually partition the intermediate results into different shares and distribute them to the participants. In the end, each participant will reconstruct the value $f(x_1, \ldots, x_n)$ from the shares they received.

7.4.2. Fully Homomorphic Encryption

In a typical data exchange scenario, the data provider wants to give the requester access permission to his data, rather than transfer the whole data to the requester. It is critical that a company is able to maintain the privacy of its data while permitting limited access to its data. Recent advances in cryptography, namely fully homomorphic encryption, provide very promising solutions to this problem[25][26][27][28]. For example, the company first uses its public key to encrypt data m and then sends ciphertext to the other party who will do complicated computation f on the ciphertext. This new ciphertext is an encryption of

$f(m)$. Finally, the new cipher text will be sent back to the enterprise who can decrypt using its private key and get $f(m)$.

Formally, a fully homomorphic encryption scheme consists of three elementary algorithms which are common to all public key encryption schemes: key generation, encryption, and decryption, plus homomorphic addition $CAdd$ and multiplication $CMul$. Suppose C_1(resp. C_2) is ciphertext of message M_1(resp. M_2), then:

$$Decrypt(CAdd(C_1, C_2)) = M_1 + M_2$$
$$Decrypt(CMul(C_1, C_2)) = M_1 \times M_2$$

Many complicated operations can be built as an arithmetic circuit consisting of these two elementary operations.

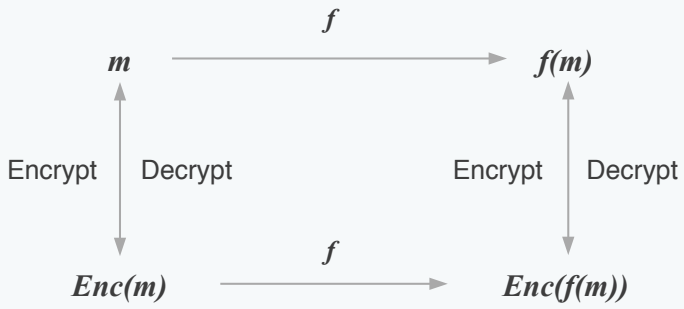


Figure 7.2: Homomorphic Encryption of Plain Text

Therefore, by using FHE scheme, we can run homomorphic operation f on the ciphertexts and the decryption algorithm will give $f(m)$ back to us (as illustrated in the above graph).

We plan to support various FHE schemes such as BGV and FV in the future.

7.4.3. Digital Copyright

Based on data's digital characteristics, Ontology provides data storage and lifecycle management functions. Firstly, Ontology designs the life cycle traceability mechanism. Ontology establishes a digital identity for each piece of data to track the entire process of registration, request, authorization, and transaction. Secondly, Ontology ensures data's copyright and transaction information are all recorded on the distributed ledger.

Ontology uses digital watermarking technology to directly embed the identification information into the digital carrier. Without affecting the use value, Ontology can easily identify attackers and prevent them from making modifications. With the hidden information in the carrier, the requester can easily

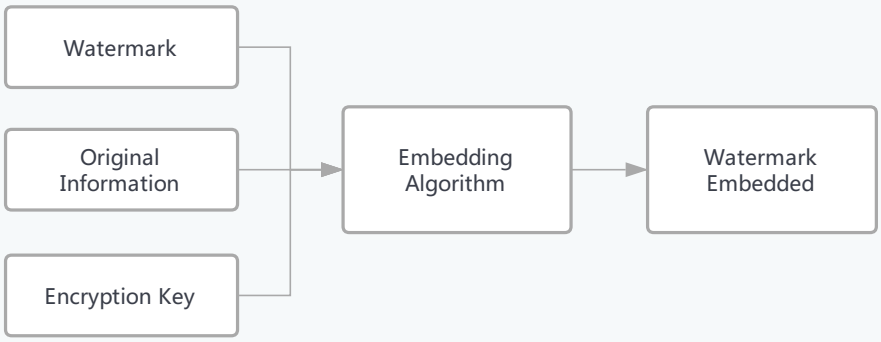
confirm the content creator and determine whether the carrier has been tampered with. Digital watermarking is an effective method for implementing anti-counterfeit traceability, copyright protection, hidden identification, authentication, and secure covert communication.

The main features of digital watermarking technology used in Ontology:

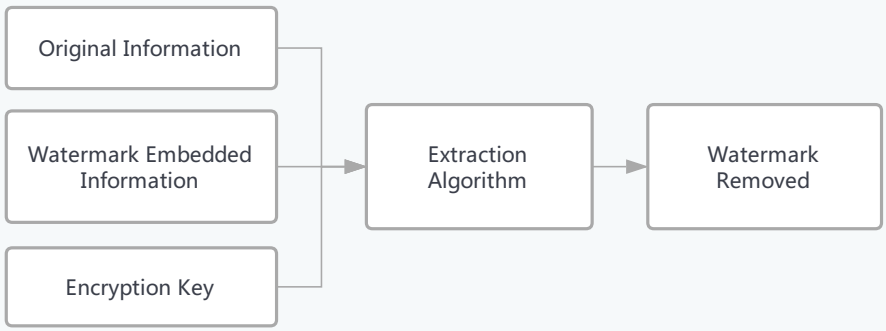
- *Vindicability: The watermark provides complete and reliable evidence of attributes of the copyrighted product. Using the watermarking algorithm users can identify the owner's information embedded into the protected object (such as the registered user number, product logo, text, etc.), and extract it when needed. Watermarks can be used to determine whether the object is protected, monitor the spread of protected data, conduct authentication, and prevent illegal copying.*
- *Imperceptibility: The embedded watermark is not able to be seen. The ideal situation is that the watermarked image is visually identical to the original image, which is what most watermarking algorithms can accomplish.*
- *Robustness: After undergoing a variety of unintentional or intentional signal processing, the digital watermark still maintains integrity and can be accurately identified. Robustness is extremely important for watermarks. A digital watermark should be able to withstand different physical and geometric distortions, including intentional distortions (e.g. malicious attacks) and unintentional distortions (such as image compression, filtering, scanning and copying, noise pollution, dimensional changes, etc.). A robust watermarking algorithm should be able to extract the embedded watermark from the watermarked image or prove the existence of the watermark. If a specific watermark embedding and detection algorithm is missing, the data product copyright protection mark should be difficult to be forged. If the attacker attempts to delete the watermark, multimedia products will destroy the image.*

Ontology's data watermarking feature contains two modules: one for embedding watermarks and the other for removing them, as showed in Figure 7.3.

Digital watermarking will be used as a security system for documents, bills, and digital identity. Through digital watermarking the authenticity of a document can be proven, and protected from illegal copying.



(a) Data Watermark Embedding Process



(b) Data Watermark Removal Process

Figure 7.3: Data Watermark Module

7.5. USER AUTHORIZATION CONTROLLER

The user authorization control (UAC) module is based on the user authorization protocol. Using granular access control, UAC modules assist the data owner with the authorization of their own data. Any transactions related to data will notify the data owner to conduct data transaction authorization. UAC modules provide RESTful APIs for external use, supporting the use of relational databases (Mysql or Oracle). They have two major functions:

- *User Data Authorization Access Policy Settings.*
- *User Data Authorization Access Control.*

7.5.1. Authorization Policy Setting

The user can request the resource provider to register his data by using the RESTful API provided by the UAC module. After registration the user can search and review the registered data by using the UAC module and conduct data access and control policy settings. For some specific frequently accessed data with low privacy requirements, users can set up authorization hosting on UAC modules.

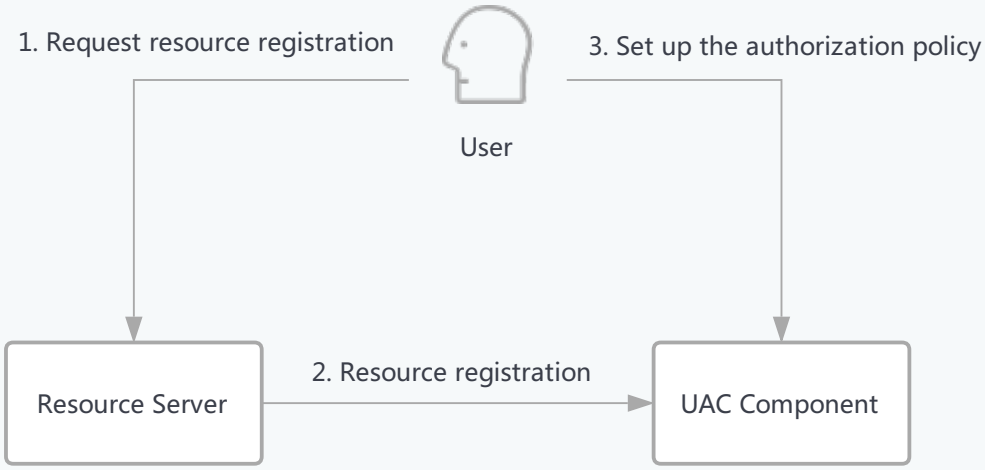


Figure 7.4: Setting Authorization Access Policy

7.5.2. Access Control

In a non-hosted mode, the UAC module acts as the protector of user data and requires the user to authorize data transmission to a requester. In the hosted mode, the UAC module makes authorization decisions on behalf of users and provides authorization receipts to the users. Both modes inform users of the details of data transactions, including who is involved, what time the transaction occurs, and what data is transferred.

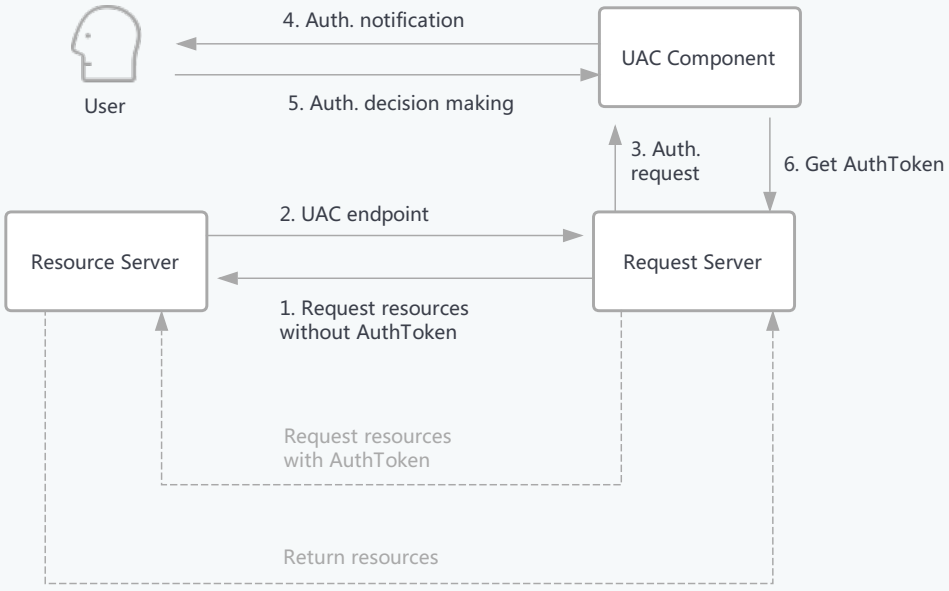


Figure 7.5: Authorization Access Control

7.6. CLAIM MANAGEMENT MODULE

Verifiable claim management is an important and basic module for the trust anchor in Ontology. The claim management module is developed in accordance with the distributed trust system's requirement. This module provides RESTful APIs to support the use of relational databases (MySQL or Oracle). Key features of its modules include, but not limited to: the issuance, verification, querying, and cancellation of trusted claims.

7.7. GLOBALDB

GlobalDB is a pluggable distributed key-value database. Its underlying layer is TiBD, an open-source distributed NewSQL database based on Google's Spanner/F1.

GlobalDB is a database optimized for blockchain/distributed ledger and IPFS. GlobalDB provides SQL compatibility, storage sharding, distributed transaction, horizontal scalability, and error recovery capability. It could be applied in joint applications of blockchain and big data, blockchain and artificial intelligence(AI).

GlobalDB has four major features: distributed transactions, storage sharding, load balancing, and SQL on KV.

7.7.1. Distributed Transactions

GlobalDB provides a complete set of distributed transactions that support state sharding and off-chain services. The transaction model is based on Google Percolator [29] for optimization.

GlobalDB's transactional model adopts optimistic locking. Distributed transactions only detect conflicts when they are submitted. If the traditional methods conflict they need to be retried. This model will be inefficient in the case of severe conflicts and the optimistic locking model has higher efficiency in most scenarios.

Since distributed transactions must be conducted in two phases and the underlying data needs to do consistent replication, if a transaction is very large the , it would slow the commit process and clog the consistent replication process. To avoid that, Ontology has limitations as below:

1) *A single KV entry does not exceed 6MB.*
2) *The total number of KV entries does not exceed 30W.*
3) *The total size of the KV entry does not exceed 100MB.*

7.7.2. Storage Sharding

GlobalDB automatically shards the underlying data based on the key's range. Each region is a key range $[StartKey, EndKey)$. When the total amount of the fragmented key-value exceeds a certain threshold, it will automatically split.

7.7.3. Load Balancing

The load balancer PD schedules the load on the cluster based on the state of the storage cluster. The scheduling is based on region, the policy configured by the PD is the scheduling logic, and the whole process is completed automatically.

7.7.4. SQL on KV

GlobalDB automatically maps the SQL structure to a KV structure. In a nutshell GlobalDB does two things:

- *One line of data is mapped to a KV. The key is prefixed to the TableID, and the line ID is suffixed.*
- *An index is mapped to a KV. The key constructs a prefix with TableID+IndexID and a suffix with index values.*

Data or indexes in one table will have the same prefix, so in the key space of TiKV, the key-value will be in an adjacent position. GlobalDB will configure the corresponding dirty data management strategy to achieve a higher performing data readability.[30]

GlobalDB is highly configurable and adaptable, and could be utilized simultaneously in on-chain and off-chain real-time high-performance business. It will serve as a core component in Ontology to provide a solid foundation for the underlying distributed ledger.

8. POSTSCRIPT

This white paper includes an overview of technologies involved in Ontology. However, as the technology itself advances, the Ontology team would be constantly updating its content in the future.

Meanwhile, Ontology will create an open, collaborative, and creative technology ecosystem. The team of Ontology welcomes developers around the globe to join the family, to participate and advance Ontology's technology.

REFERENCE

[1] Burnett, Daniel C. et al. "Verifiable Claims Data Model" Verifiable Claims Working Group, W3C Editor's Draft, 2017, https://w3c.github.io/vc-data-model/.

[2] McCarron, Shane et al. "Verifiable Claims Use Cases" Verifiable Claims Working Group, W3C Working Group Note, October 2017, https://w3c.github.io/vc-use-cases/.

[3] Reed, Drummond et al. "Decentralized Identifiers (DIDs)" W3C, Credentials Community Group, 2017, https://w3c-ccg.github.io/did-spec/.

[4] Hardt , D.. "The OAuth 2.0 Authorization Framework" RFC6749, October 2012.

[5] Machulak, Maciej and Machulak Richer. "User-Managed Access (UMA) 2.0 Grant for OAuth 2.0 Authorization" User-Managed Access Work Group, Draft Recommendation, 2017, https://docs.kantarainitiative.org/uma/wg/oauth-uma-grant-2.0-09.html.

[6] Jones, M., et al. "Uniform Resource Identifier (URI): Generic Syntax" RFC3986, January 2005.

[7] Adams, Carlisle, and Steve Lloyd. "Understanding PKI : concepts, standards, and deployment considerations." Boston: Addison-Wesley, 2003.

[8] Sporny, Manu et al. "JSON-LD 1.0" W3C, W3C Recommendation, January 2014, http://www.w3.org/TR/json-ld/.

[9] Longley, Dave, et al. "Linked Data Signatures". W3C Digital Verification Community Group, Draft Community Group Report. 2017, https://w3c-dvcg.github.io/ld-signatures/.

[10] Camenisch, Jan, and Anna Lysyanskaya. "A signature scheme with efficient protocols." international workshop on security (2002): 268-289.

[11] R., Cramer. "Modular design of secure yet practical cryptographic protocols." Ph. D thesis, Universiteit van Amsterdam, Netherlands, 1997.

[12] Fiat, Amos, and Adi Shamir. "How to prove yourself: practical solutions to identification and signature problems." international cryptology conference(1987): 186-194.

[13] Schnorr, Clauspeter. "Efficient signature generation by smart cards." Journal of Cryptology 4.3 (1991): 161-174.

[14] Abdelmalek, Michael, et al. "Fault-scalable Byzantine fault-tolerant services." symposium on operating systems principles 39.5 (2005): 59-74.

[15] Castro, Miguel, and Barbara Liskov. "Practical Byzantine fault tolerance." operating systems design and implementation (1999): 173-186.

[16] Borran, Fatemeh, and Andre Schiper. "Brief announcement: a leader-free byzantine consensus algorithm." international symposium on distributed computing (2009): 479-480.

[17] Laurie, B., et al. "Certificate Transparency" RFC6962, June 2013.

[18] Merkle, Ralph C.. "A digital signature based on a conventional encryption function." Lecture Notes in Computer Science (1989).

[19] US patent 4309569, Ralph C. Merkle, "Method of providing digital signatures", published Jan 5, 1982, assigned to The Board Of Trustees Of The Leland Stanford Junior University.

[20] Matthew, S.. "Merkle Patricia Trie Specification" Ethereum, October 2017, https://github.com/ethereum/wiki/wiki/Patricia-Tree.

[21] Morrison, Donald R.. "PATRICIA—Practical Algorithm To Retrieve Information Coded in Alphanumeric." Journal of the ACM 15.4 (1968): 514-534.

[22] Yao, Andrew C. "Protocols for secure computations." Foundations of Computer Science, 1982. SFCS'08. 23rd Annual Symposium on. IEEE, 1982.

[23] Shamir, Adi. "How to share a secret." Communications of the ACM 22.11 (1979): 612-613.

[24] Damgård, Ivan, et al. "Practical covertly secure MPC for dishonest majority–or: breaking the SPDZ limits." European Symposium on Research in Computer Security. Springer, Berlin, Heidelberg, 2013.

[25] Gentry, Craig. "A Fully Homomorphic Encryption Scheme." Stanford University, 2009.

[26] Brakerski, Zvika, Craig Gentry, and Vinod Vaikuntanathan. "(Leveled) fully homomorphic encryption without bootstrapping." ACM Transactions on Computation Theory (TOCT) 6.3 (2014): 13.

[27] Halevi, Shai, and Victor Shoup. "Algorithms in helib." International Cryptology Conference. Springer, Berlin, Heidelberg, 2014.

[28] Fan, Junfeng, and Frederik Vercauteren. "Somewhat Practical Fully Homomorphic Encryption." IACR Cryptology ePrint Archive 2012 (2012): 144.

[29] Peng, Daniel, and Frank Dabek. "Large-scale Incremental Processing Using Distributed Transactions and Notifications" Proceedings of the 9th USENIX Symposium on Operating Systems Design and Implementation, 2010.

[30] Shen, Li. "Understand TiDB technology insider" PingCAP, 2017, https://pingcap.com/blog-cn/tidb-internal-2/.

CONTACT US

 Email: contact@ont.io

 Telegram: OntologyNetwork

 Twitter: OntologyNetwork

 Facebook: ONTnetwork

 Reddit: OntologyNetwork

 Discord：https://discord.gg/vKRdcct

 Medium: OntologyNetwork

 LinkedIn: Ontology Network