EdenChain:

The Programmable Economy Platform

Technical Whitepaper
Version 1.2

By James Ahn

Disclaimer

Do not take any action before reading the following disclaimers. This document contains pertinent information about the R&D and businesses of Eden. This material has been prepared for informational purposes only, and is not intended to provide, and should not be relied on for, tax, legal or accounting advice. You should consult your own tax, legal and accounting advisors before engaging in any transaction.

Eden's technical whitepaper and the information posted on its homepage are not guidelines or solicitation documents to attract investments. Any token issued by Eden is neither a security nor an equivalent one used in any jurisdiction. If you have decided to contribute to the development of Eden, your donation to Eden does not include the sale nor exchange of cryptographic money in the form of securities, investment units, or stock of Eden. Eden token holders do not receive dividends or any other income.

US residents and permanent residents cannot donate to Eden and receive tokens due to the uncertainty around US legal regulations. If you provide false information to Eden, this violates the Fundraising Terms and Conditions and Eden may request compensation. You should not download, distribute or make copies of any form of information provided by Eden, such as home pages, technical white papers, etc., wherever there might be any possibility of such breach. The information in the white paper has not been reviewed or approved by regulatory authorities. The publication and distribution of white papers do not imply that they complied with the regulation of the relevant law and its requirements.

To the maximum extent permitted by applicable laws, regulations and rules, Eden entrepreneurs, team members, etc. are not liable for any indirect, incidental or consequential damages related to the Eden project whatsoever.

This document contains information from open market research reports, information from industrial publications, internal research from Eden, etc. While this information is generally considered reliable, there is no assurance for the accuracy or reliability of the information.

All information contained within Eden's homepage, whitepaper, and other related documents is suggested as a model case and is neither binding and will not be executed unless explicitly stated in the funders' terms and conditions.

You may not reproduce or distribute any kind of information or documentation provided by Eden without including this disclaimer.

Abstract

Eden is a blockchain-based platform that can be used to capitalize and trade all types of assets through programmable economy technology. By using blockchain Smart Contracts, you can capitalize all tangible and intangible asset values through tokenization. We use smart contracts to integrate real and virtual economies, creating a fundamentally new programmable economy platform. The use of Eden's programmable economy platform includes the following advantages: 1) It can lower transaction costs due a system that is bereft of a middleman. 2) It can redistribute the profits that the middleman has monopolized. 3) It creates a new market that has never existed through the capitalization of domestic and foreign materials; this in turn can enrich the lives of many by returning the economic benefits back to them.

Currently, there are two major technical problems in achieving programmable economics in blockchain technology: poor performance and the lack of secured connectivity. Eden solves these technical problems and provides components necessary for implementing a programmable economy. Eden implements Merkle Tree + Namespace to solve the performance problem by executing transactions of different Namespaces in parallel and integrates E-Oracle technology for external system interactions. In addition, it provides reliability and privacy, through the adoption of MVT (Median Voter Theorem) as a consensus algorithm, granting robust functions for selection of values that can occur when interacting with external systems. E-Protocol using ECC (Elliptic Curve Cryptography) - TC (Threshold Cryptography) acts as a powerful mechanism that can protect against attack by hackers by encrypting all network data. PoET (Proof-of-Elapsed-Time), the settlement algorithm of ledger data, is designed to enable efficient leader emulation with fewer computing resources.

Eden's key programs are implemented in the SGX Enclave to disable software attacks and hardware attacks, further enhancing secured connectivity when interacting with external systems. In addition, Eden incorporates Global DNS, Multi-Datacenter pattern, and Virtual Private Network (VPN) Hub technology in order to ensure secure services despite the potential occurrence of situations that may disrupt local systems such as natural disasters and large-scaled attacks by hackers. This results in a durable network that can continue to operate from Eden service zones around the world.

Eden can be applied to a diverse array of industries and services, such as Initial Coin Offerings (ICOs), IoT, Shared Economy Platforms, Gaming, Finance, etc. given its ability to capitalize valuable assets and to execute transactions rapidly, securely and at low costs.

1. Introduction

1.1 A Programmable Economy

The Internet, smart phones, big data, cloud services, and artificial intelligence technologies have had a major impact on the way in which we live. These devices and services have become so embedded into our daily lives that they are now virtually indispensable. Similarly, blockchain technology will continue to be integrated into our lives, dramatically altering the manner in which we interact with the world around us.

The Programmable Economy is born when blockchain meets the economic system. The Programmable Economy is a new economic system that capitalizes tangible and intangible values with blockchain technology, and freely trades through the Internet without the need for a middleman. In the existing system, the middleman provides services to buyers and sellers in order to carry out transactions safely and quickly, and in return the middleman receives compensation. In such an economic system, the general public cannot participate in the network, so the middleman monopolizes the profits from such trades. In our Programmable Economy, the buyers and sellers are connected directly with one another, which eliminates the need for a middleman. As a result, transaction costs are lowered, and the entire process is simplified. Participants within the network also receive corresponding revenues that would have otherwise gone entirely to the middleman.

In the Programmable Economy, individuals can utilize blockchain technology to tokenize all types of assets, tangible and intangible, register them on a blockchain to secure their ownership, trade tokens with others, and automate trades by using IoT services.

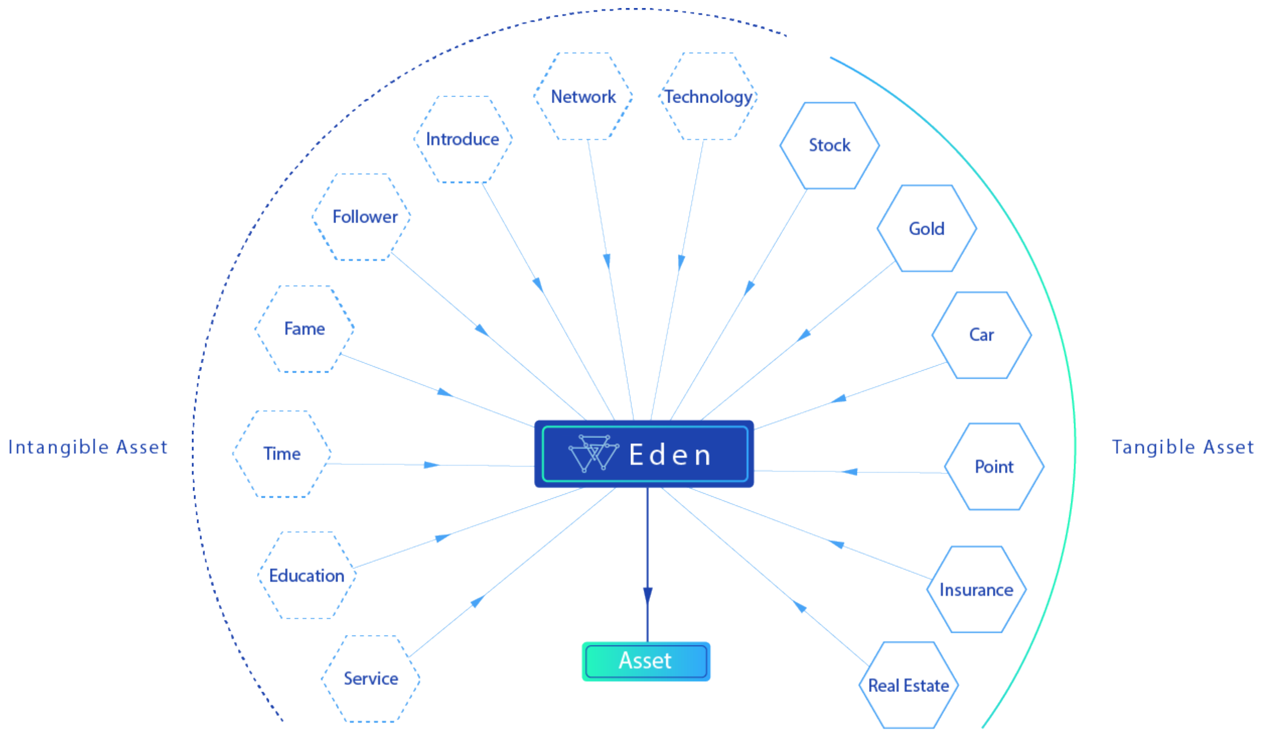


Figure 1 Eden's Programmable Economy

1.2 Tokenization

Tokenization is the process of converting certain types of assets into tokens within the blockchain network (Cameron-Huff, 2017). Throughout the world, there exist a plethora of asset classes, such as stocks, real estate, automobiles, gold, etc. Individuals are able to buy and sell these assets for various purposes.

As technology has developed, distinct asset classes have begun to emerge, such as copyrights, insurance policies, and derivatives, which all contain tradeable value when certain conditions are met. With blockchain technology, existing assets can be replaced with blockchain-based tokens and their ownership can be registered; the necessary conditions can be specified through smart contracts according to the characteristics of the asset. It is also possible to create new assets by through the combination of different asset types. Assetization is the process of assigning ownership to intangibles such as game items, credit card points, or even one's own influence on social media platforms.

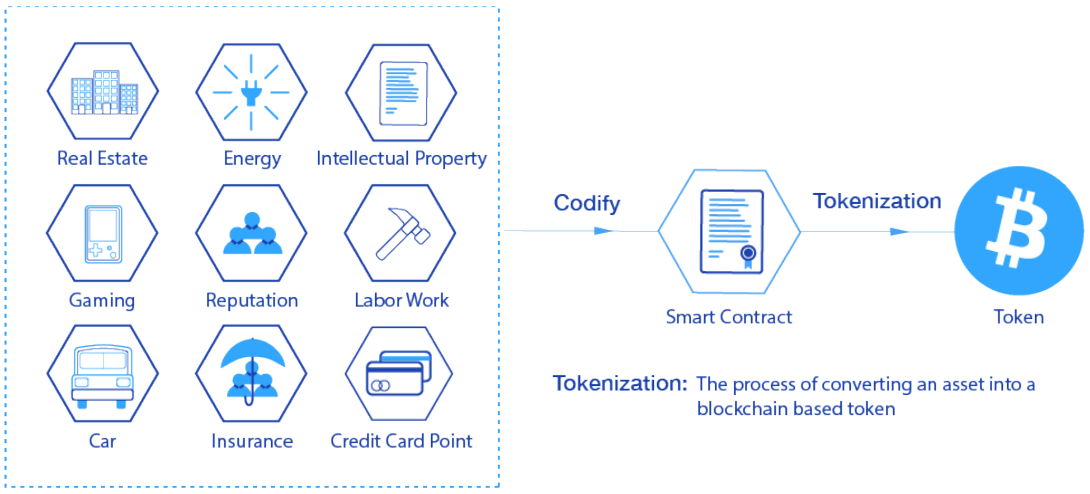


Figure 2 Tokenization

1.3 Smart Contract

Computer Scientist and Cryptographer Nick Szabo explained smart contracts as follows:

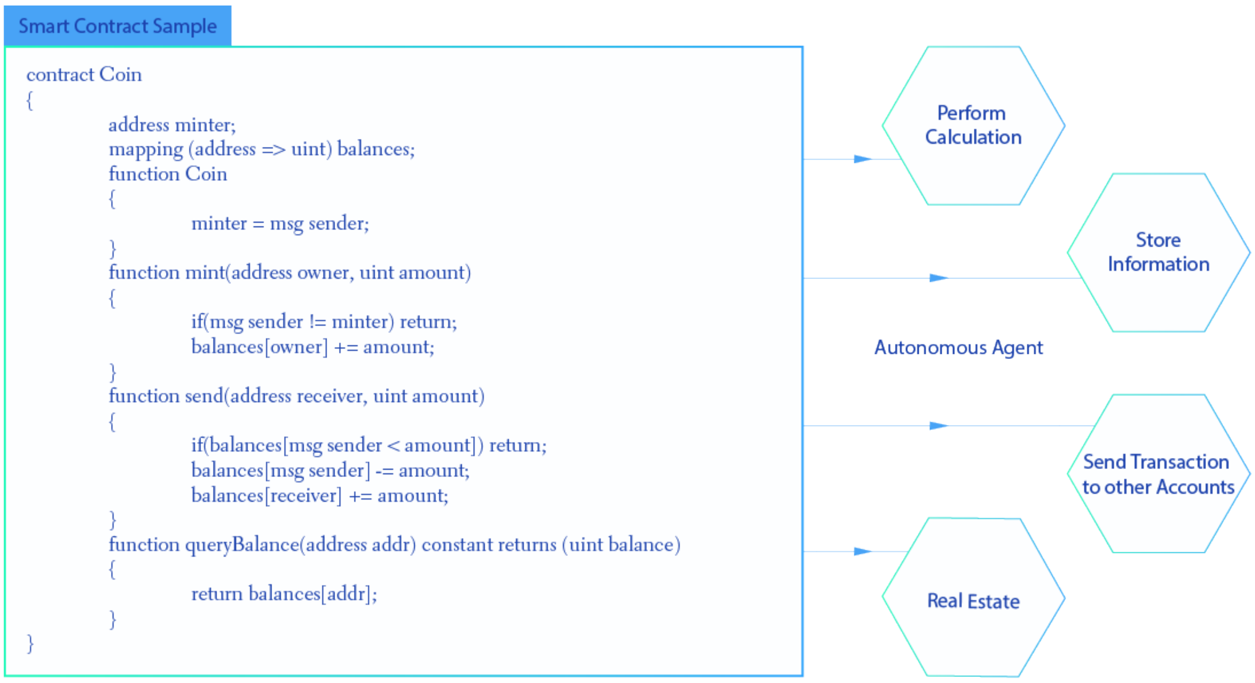
"A set of promises, specified in digital form, including protocols within which the parties perform on these promises." (Szabo, 1996)

In short, it refers to an "automated trade agreement" that functions without human intervention if the conditions specified in the contract are met. The Chamber of Digital Commerce described Szabo's concept of smart contract as four elements: Model of Smart Contract, Automation, Data Processing Protocol and Software Code (Chamber of Digital Commerce, 2016). Furthermore, such smart contracts are composed of six steps in total:

1. Identify Agreement
2. Set Conditions
3. Code the Business Logic
4. Encrypt & Blockchain Technology
5. Execution & Processing
6. Network Updates

Bocconi University Professor Vincenzo Morabito divided smart contracts into two types: deterministic and nondeterministic (Morabito, 2017). A deterministic smart contract is a type of contract that does not have any data from outside in its execution, while a non-deterministic smart contract is a form that requires external data in its execution (Morabito, 2017).

Non-deterministic smart contracts are more vulnerable to security breaches relative to deterministic smart contracts because non-deterministic smart contracts need to retrieve data from external systems rather than from blockchain networks. However, these non-deterministic smart contracts can be integrated with external systems to create various types of smart contracts and automation agents, a very important element of smart contracts.



```
contract Coin
{
        address minter;
        mapping (address => uint) balances;
        function Coin
        {
                minter = msg sender;
        }
        function mint(address owner, uint amount)
        {
                if(msg sender != minter) return;
                balances[owner] += amount;
        }
        function send(address receiver, uint amount)
        {
                if(balances[msg sender < amount]) return;
                balances[msg sender] -= amount;
                balances[receiver] += amount;
        }
        function queryBalance(address addr) constant returns (uint balance)
        {
                return balances[addr];
        }
}
```

Figure 3 Smart Contract

1.3.1 Bitcoin Smart Contract

Bitcoin provides a programming language called "Script". Script is a bytecode stack-based language similar to Forth, and it is designed as Turing-Incomplete, which intentionally does not support loop or recursion in order to ensure the execution of script programs without risk of hacking (Lamela-Seijas, 2017).

Script supports various functions such as stack manipulation, string manipulation, bitwise manipulation, and basic cryptography. However, Script is inherently limited in use due to various factors. The maximum size of a script is 10,000 bytes due to the limitation of Bitcoin's block size, and the number of available opcodes is limited to 201 (Harding, 2015).

1.3.2 Ethereum Smart Contract

Ethereum supports a Turing-Complete language, and it uses Loop and Recursion, which is a more flexible implementation than Bitcoin's Script language. Script prohibits Loop and Recursion in order to deliberately delay the execution of the program and to prevent long wait times. Ethereum solved this problem by introducing the concept of "Gas" (Wood, 2017). Gas can be understood as a kind of royalty fee concept that uses ether encryption to execute the smart contract. Each time someone executes calculations in an Ethereum Smart Contract, it runs in a way that subtracts a certain amount from the gas paid in advance, so an individual cannot deliberately delay the execution or repeat the calculation indefinitely.

An interesting feature of the Ether Smart Context language is that it has the reactive property, this ensures that smart contract programs cannot be executed by themselves and can be executed by other transactions. It is designed to have the reactive characteristic in order to avoid a DoS attack.

Smart contracts written in languages supported by Ethereum run on the Ethereum Virtual Machine (EVM). EVMs that run smart contracts in a non-secure environment in which anyone can create and register smart contracts in a publicly-permissive blockchain runs quarantined in a redundant network environment. When problems arise while running a particular smart contract, EVMs roll back to the state before execution and do not return the Gas already used.

1.3.3 NXT Smart Contract

NXT adopted the Proof-of-Stake (POS) consensus algorithm to secure performance and scalability with first-generation blockchain technology. It has been designed with a client-server architecture and connects with internal and external systems using APIs.

Unlike Bitcoin and Ethereum, NXT is prepared by using smart APIs provided by NXT. The NXT API provides the basic functions necessary for various fields such as file storage, messaging, and trading. Therefore, the general logic is written using a programming language such as Javascript, Python, or Java in order to create an NXT smart contract, while functions such as input, delete, and update data in a block, are completed using the NXT API.

The types and functions of smart contracts that can be created in NXT are more limited than other blockchain technologies supporting language-based smart contracts such as dummy because they depend on the type and content of the APIs provided.

2.Technical Motivation

Smart contracts are a key element in the programmable economy since tokenization is realized through smart contracts. These smart contracts should have the ability to rapidly defend against attacks, misuse, and tampering by hackers because they deal with a large number of transactions and sensitive data.

Smart contracts operate as autonomous actors whose behavior is completely predictable (Christidis, 2016). This allows smart contracts to be used for reliable transaction processing.

Distributed Systems Professor Aad van Moorsel categorized the issues that could be generated by smart contracts into four categories: codifying issue, security issue, privacy issue, and performance issue (Alharby & Moorsel, 2017). The codifying issue explains the difficulty in creating a completely seamless smart contract, while the security issue is related to attacks using bugs or weaknesses within the smart contract itself. The performance issue is caused by the scalability problems of the blockchain executing the smart contract. Finally, the privacy issue describes the problems that arise as the content of the smart contract is made public.

There are a number of technical issues the need to be addressed in order for a smart contract to be used as trusted autonomous actors. When these issues are solved, smart contracts can be used in various fields of the programmable economy as true smart contracts.

2.1 Technical Issues

There are two major technical issues in the implementation of the programmable economy: throughput speed and security. Both of these issues restrict the full functionality of a programmable economy. In current blockchain processing techniques, there exists a great deal of difficulty in using the techniques in conjunction with actual services due to both the slow speeds and the vulnerability to attacks carried out by hackers.

2.2 Performance & Scalability

Two of the most discussed technical issues in Bitcoin and Ethereum's respective blockchain technologies are performance and scalability. According to Coinbase co-founder Fred Ehrsam, Facebook can handle 157,000 requests per second, while Ethereum can only handle 7 transactions per second (TPS) in the case of tokens.

Technical issues related to performance and scalability must be solved in order for the continued growth and adoption of blockchain technology. In addition, higher levels of scalability and performance are necessary for smart contracts that require far more computations than traditional blockchain transactions. For example, if a smart contract is interfaced with an external system, the slow processing of the smart contract can become a bottleneck in the blockchain itself, degrading the performance and scalability of the overall system, and seriously undermining system responsiveness. Issues related to performance and scalability may become more salient over time as the number of transactions conducted over blockchain-based platforms and smart contract applications continue to increase. If these obstacles continue to be present, blockchain may suffer from a lack of mainstream adoption by both enterprises and consumers.

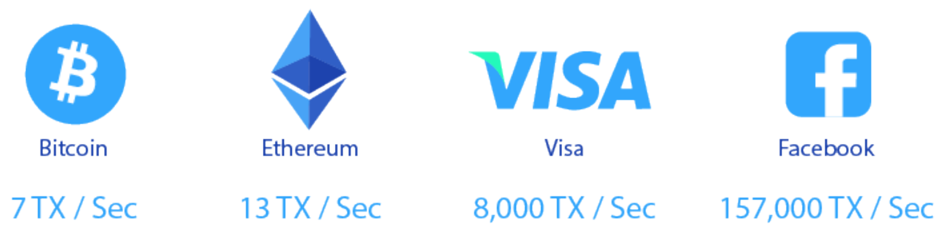


Figure 4 TPS Performance

2.3 Lack of Secured Connectivity

Economist Mark Flood states that it is important to understand that a financial network is an edge in which vertices are connected, and that connecting these edges forms a smart contract (Flood, 2017). Distinct systems often require require a high degree of connectivity in order to execute certain tasks. In order for smart contracts to create more abundant application services and to further realize a programmable economy, a high level of connectivity that can interact with external systems is required. For example, suppose that a cargo owner moves a cargo from point A to point B, and then the owner creates a smart contract to pay for the GPS information attached to the cargo. In order for a smart contract to be implemented, GPS information attached to the freight car should be transmitted to the blockchain where the smart contract is stored at a predetermined time interval. As a result, the smart contract is executed, and the freight car driver can receive the promised cost. In other words, a system that can be securely interlocked with an external system can be provided to the smart contract to provide various benefits promised by the original smart contract. Smart contracts must be able to work with a variety of external systems in order to process contracts automatically. An important point is that system interaction not only means interlocking with external systems but also means ensuring stability for interacting with external systems.

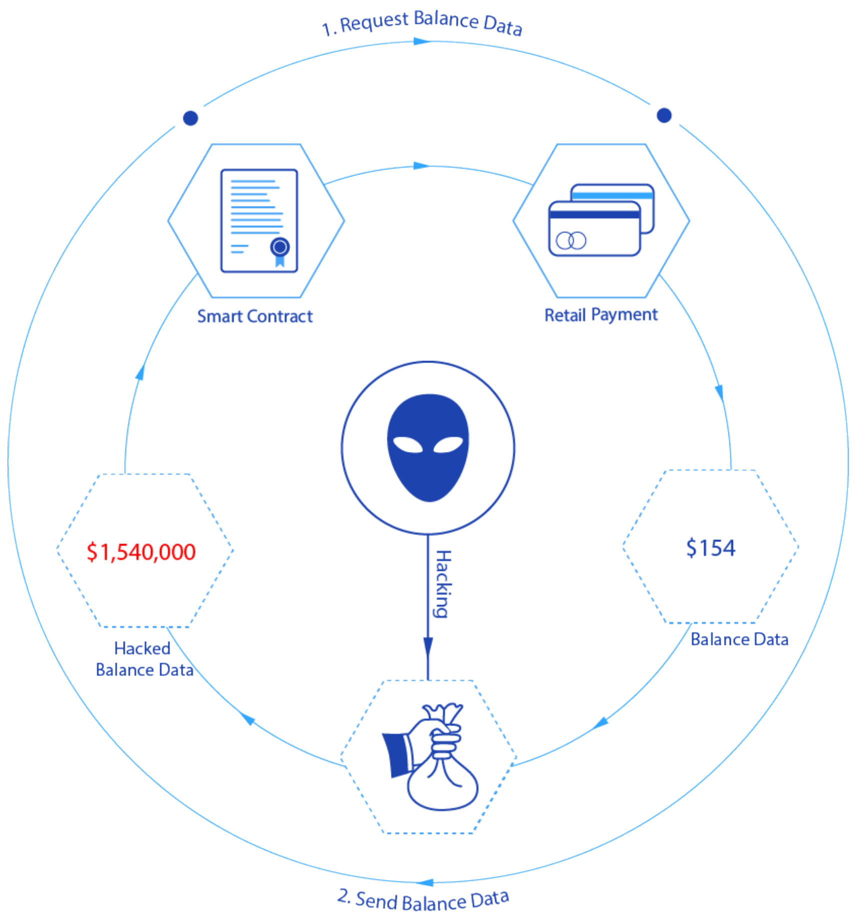


Figure 5 Lack of Secured Connectivity

Ethereum allows Oracle to interoperate with external systems, but it does not guarantee the safety of data from external systems (Alharby & Moorsel, 2017). Secured Connectivity protects against the attacks by hackers that may occur during the interaction between smart contracts and external systems, which guarantees stability. Secured Connectivity is an important technical issue that must be addressed for the scalability and automation of smart contracts, but it is currently not provided by blockchain technologies such as Bitcoin and Ethereum.

3.Eden: The Programmable Economy Platform

3.1 Introduction to Eden

Eden is a blockchain-based programmable economy platform that provides high-performance processing speeds which address the aforementioned technical issues. Eden has the capacity to develop a variety of automated services using smart contracts, enabling interoperability with external systems. The core technologies implementing programmable economy smart contracts, have a greater technological and economic value than non-deterministic smart contracts that require off-chain integration rather than deterministic smart contracts that are operated only on-chain.

Conventional blockchain technologies such as Bitcoin and Ethereum are unsafe because they are exposed to hacker attacks. This is due to the lack of guaranteed trust in interacting with external systems that is normally required for non-deterministic smart contracts. Eden uses the E-Bridge layer to retrieve data from multiple data sources when a non-deterministic smart contract is interfaced with an external system. Eden encrypts this data and incorporates the Median Voter Theorem (MVT) to secure trust and to defend against attacks from hackers.

Performance is a vital technical issue that is essential to the implementation of smart contracts. Eden combines namespaces with Merkle Tree, isolates transactions based on its particular namespace, and secures performance and scalability by constructing an execution system capable of parallel processing by namespace. This enables the ability to carry out a vast number of transactions simultaneously in parallel.

Eden supports Solidity, the most popular smart contract language at this time, as well as Dyerium's EVM since smart contracts are heavily used for sensitive services such as payment and settlement. Stable and reliable smart contract programming languages are more important than the creation of an unnecessary new type of programming language. The emergence of a new type of language that requires a long time to be validated through constant testing and can be exposed to serious security vulnerabilities during this validation period. Eden constructs a blockchain using EVM and Solidity, which are verified and continually improved by the community.

Since Eden guarantees secured connectivity, one can fully realize a programmable economy by trading various blockchain-based assets on the Internet. Tokenization of any type of asset can publicly prove ownership of the asset and the owner of a particular asset can take part in both P2P transactions and transactions on exchanges.

3.2 Architecture

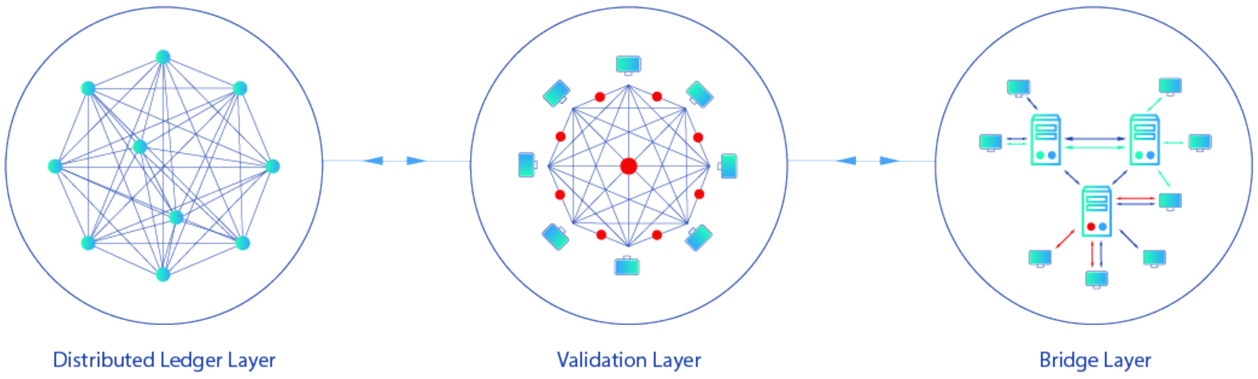


Figure 6 Eden Architecture

Eden is a permissioned blockchain consisting of three layers: a distributed ledger layer, a validation layer, and a bridge layer. The distributed ledger layer is a place in which data used in the blockchain are separately stored, and only data of transactions agreed to in the validation layer are processed. Distributed ledger data can be added through transactions. The validation layer is where a transaction is executed and verified; this layer includes an EVM that runs the smart contract. The validation layer has a transaction scheduling function, which has a significant impact on the performance and scalability of Eden.

The bridge layer is used to securely import data needed by an on-chain smart contract within the blockchain in cooperation with an off-chain. In the bridge layer, nodes naturally exist on-chain and off-chain, and an E-Protocol that implements can encryption technique called Elliptic Curve Cryptography –Threshold Cryptography (ECC-TC) is used for reliable communication between these nodes.

3.3 Permissioned Blockchain

A permissioned blockchain has evolved as an alternative to permissionless blockchain technology, which allows anyone to join a network, such as Bitcoin and Ethereum. Permissioned blockchain technology must be authorized by a network administrator through an authentication process in order to participate in the network.

Newly emerging blockchain technologies such as Kadena, Tendermint, and Chain, adopted permissioned networks, and Hyperledger, a blockchain open source project for the Linux Foundation, also adopted the permissioned blockchain technology. Eden is configured with a permissioned blockchain to run smart contracts quickly and efficiently in a trusted environment.

In a permissionless blockchain network such as Ethereum, a smart contract runs on all nodes. This leads to significant problems surrounding performance and efficiency. All smart contracts are stored in an EVM within a blockchain network and are executed according to certain conditions. If a million, or a hundred million, smart contracts exist in an Ethereum blockchain, serious performance problems may occur. Miners within the network may prioritize running smart contracts from which they can gain higher profits, that is, those with higher gas, so all smart contracts may not be executed. Although individually running smart contracts on all full nodes are based on the philosophy of the permissionless blockchain technology, individually running and validating the smart contracts on all of the nodes at all times is not considered efficient.

Eden is a permissioned blockchain that builds and runs a trustworthy environment for smart contract execution. This is achieved with the use of a platform that not only ensures safety but also increases efficiency through the use of nodes within namespaces which helps guarantee 100 percent processing of all transactions.

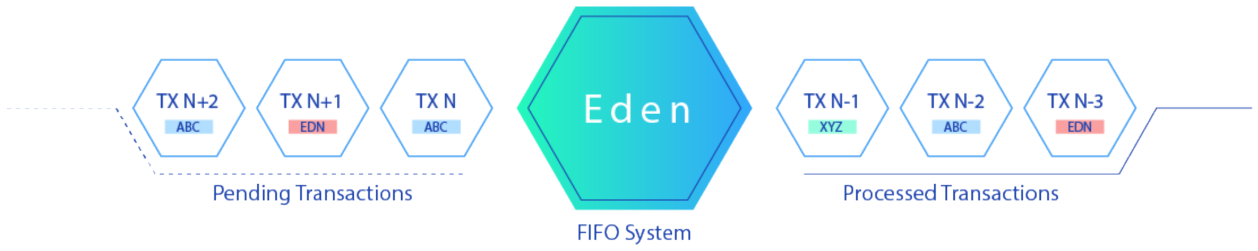


Figure 7 100% Transaction Guarantee

3.3.1 Block Withholding

Kyber Network CEO Loi Luu described a fraud technique called block withholding and its associated risks (Luu, 2015). Block withholding is an attack technique that prevents a miner who has found a solution to a problem in a POW consensus algorithm from receiving an incentive by not disclosing the answer to a mining pool. This technique can be used effectively when one mining group seeks to attack another. For example, if Miner Group A wants to attack a newly formed Miner Group B, Miner Group A selects a high-performance computer to penetrate into Miner Group B and launches a block withholding attack. Members of Miner Group B will stop receiving incentives, and thus their motivation to participate in the mining group will be weakened, which eventually leads to disorganization within Miner Group B. These types of block withholding attacks have been identified in the Bitcoin Forum and in a number of recent papers (Eyal, 2015).

In order to avoid such issues, Eden operates a fair and stable system using a permissioned blockchain. The purpose of Eden is not to secure incentives through mining but to create a programmable economy. Since public confidence in Eden is essential in the creation of a programmable economy, the system is transparently operated and is devoid of any foul play.

3.4 High Availability

Since Eden is a permissioned blockchain, a consideration of service availability is necessary. Given that an Eden server is operated by a small number of authorized agencies or companies, the server operation can be terminated when many hackers attack the servers or when there is a natural disaster such as an earthquake. Eden must be able to guarantee high availability in order to ensure that the services for users and businesses alike can continue to operate at all times regardless of any external threat.

Eden utilizes cloud services to ensure a high degree of availability and operates an Eden system with a multi-datacenter pattern using a global DNS and a load balancer. The same system that provides the Eden service is configured and operated in each service zone across major continents such as Asia, North America, and Europe, and it can provide a stable service despite attacks from hackers and or the occurrence of natural disasters.

A network between service zones deployed on each of the continents is composed of a Virtual Private Network (VPN). Cloud services provide connectivity between data centers across continents with high-speed dedicated lines, enabling fast networking and a data center-to-data center configuration. A multi-datacenter pattern is a pattern provided by the cloud service provider Amazon. It is used by a number of Internet companies such as the Apache Foundation, Netflix, CloudFoundry, and Attlasian, and is also recommended by Microsoft Azure (Meshenberg, Gopalani, and Kosewski, 2013).

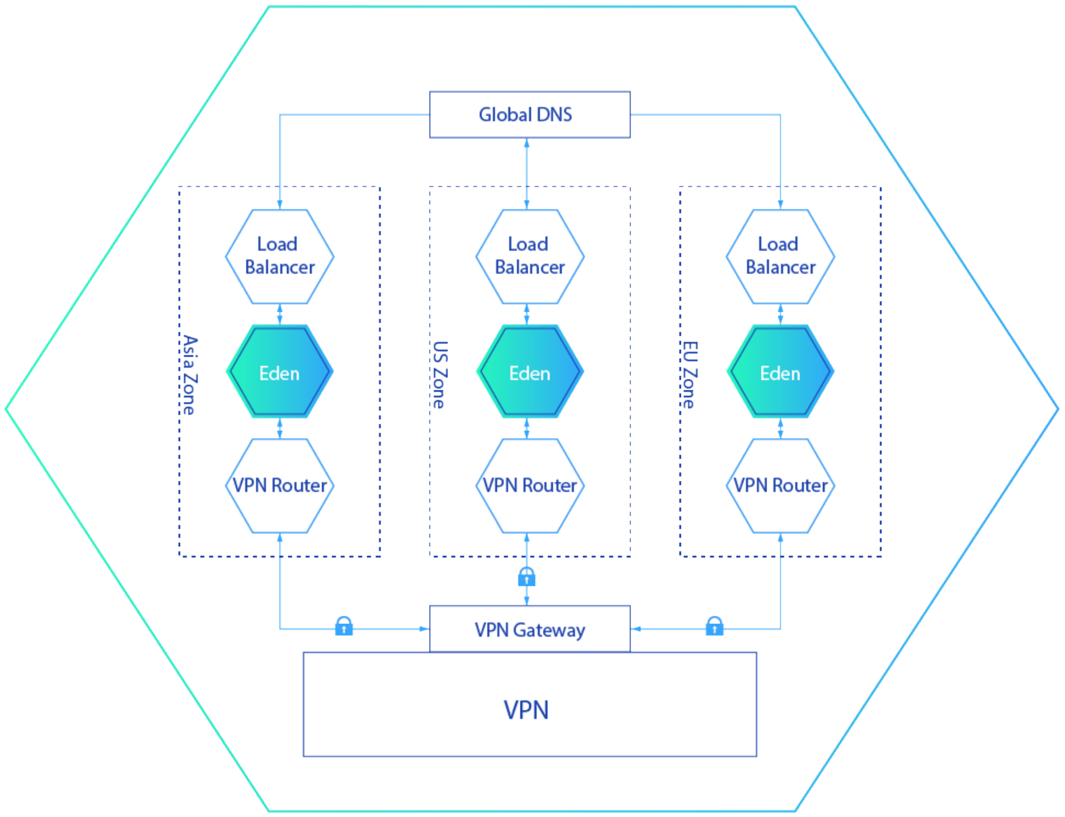


Figure 8 Eden System Operation for High Availability

Figure 8 shows a configuration of an operating environment of Eden to which a multi data center pattern and a VPN are applied. The operating environment receives a data request from outside a global DNS, plays the role of being connected to an appropriate service zone, and secures availability by operating multiple global DNS servers. Endpoints of all services are designed and operated so as to be the global DNS. A load balancer delivers requests forwarded from the global DNS to Eden servers in order to be processed. The load balancer not only requests routing but also collects status information from each of the servers. This helps perform a more intelligent service operation than a round-robin service operation, which in turn allows the system to pinpoint servers that encounter a problem and to monitor the workload on each server, thereby aiding in capacity planning.

Servers running Eden are protected by an operational firewall. The operational firewall is a way to organize the Eden servers into functional groups and to apply a firewall policy to each of the organized functional groups. The operational firewall can functionally apply a well-abstracted security policy to a server so that a security policy can be flexibly designed, applied to each of the groups, and managed internally. This allows the Eden architecture to minimize any form of potential mistake in setting work by users.

If a VPN in full mesh topology is built between service zones, performance and management problems will arise because each VPN configuration becomes more complicated as the range of the service zone increases. The Eden operating system can configure a VPN in a star topology so that a VPN router in a service zone can be connected with a VPN gateway without connecting to all of the service zones and enable VPN networking with the other service zones.

3.5 Proof-of-Elapsed-Time (PoET)

The consensus algorithm plays an important role in a blockchain technique. There are two approaches. The first is "Nakamoto Consensus," which is a way to conduct a leader selection through a lottery process. When selected as a leader, one has the right to authenticate a previous block and to create a new block. In case of Bitcoin, a node that solves a hash puzzle first is selected as the leader. The second method uses "BFT (Byzantine Fault Tolerance)." This method does not select a leader and a final agreement is reached through several stages of voting.

Eden uses Proof-of-Elapsed-Time (PoET) as a consensus algorithm. PoET is a "Nakamoto Consensus" method, which uses a CPU command to select a leader randomly without using enormous levels of energy to solve a hash problem like Bitcoin currently does. PoET provides an opportunity to become a leader with block generation authority for all nodes participating in a blockchain network with a probability similar to of other leader selection algorithms (Foundation, 2017). PoET is implemented in an SGX enclave so as to defend against hacker attacks and to allow the leader selection process to proceed safely. At each node, PoET uses a CPU command in the SGX enclave to obtain a wait time that follows an exponential distribution as a random number and selects the node that has the smallest wait time as the leader.

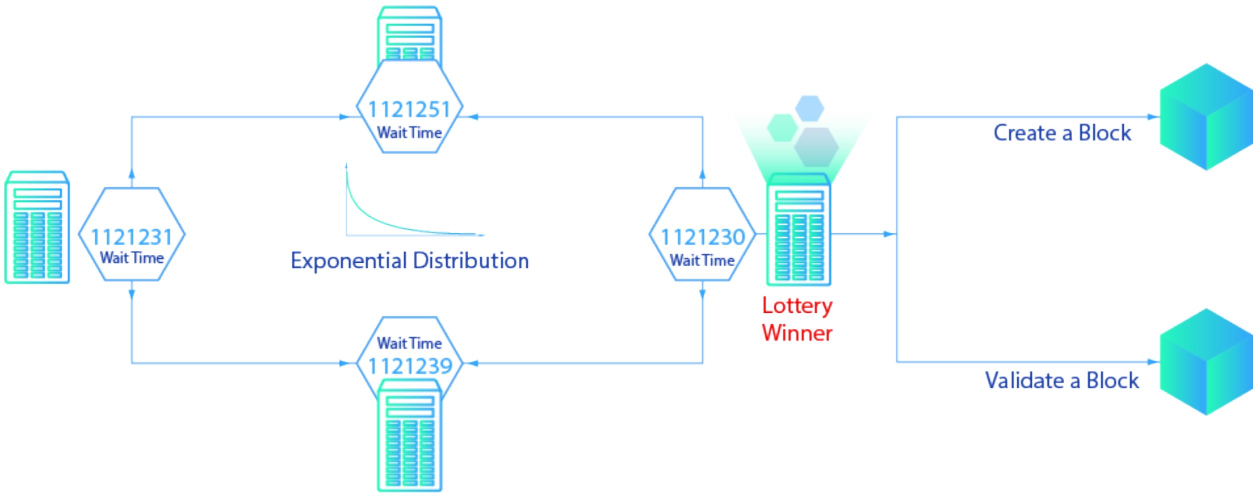


Figure 8 PoET

PoET is designed to follow the Poisson distribution, which is a form of discrete probability distribution that follows the exponential distribution shown below and expresses how many times a certain number of events occur within a unit time if the event is independent.

$$f(t) = \lambda e^{-\lambda t}, t > 0$$

3.6 Distributed Ledger Layer (DLL)

The distributed ledger layer provides decentralized database functionality to Eden and is used on the basis of the Linux Foundation's open source project, Hyperledger. The DLL stores all data generated by Eden in a block. The data cannot be modified and can be only added. The DLL stores the data on a disk device and assumes that all of the stored data originated from a legitimate transaction.

The DLL can find the necessary data from the outside by using a block ID or a transaction ID, and also provides a function of accessing information about all transactions included in the block. The DLL has a BlockCache module to minimize disk access.

BlockCache mainly stores currently used blocks in memory, and when a requested block cannot be found, it reads the block from a disk and loads it into the memory. Depending on frequency of use, it is possible to determine blocks that should be kept in the memory and those that should not, and thus a cache effect with optimal use of memory can be obtained.

3.7 Software Connector

Software systems researcher Xiwei Xu proposed a method of using blockchain technology as a form of software connector (Xu, 2016). This view can improve the quality of software architecture and enhance information transparency and traceability. In addition, it is also an important architectural design in determining which information is stored in the blockchain (on-chain) and which are stored off-chain as the data size stored in the blockchain is limited.

Eden designed the entire architecture by considering the distributed ledger layer, which stores information about various transactions that occur, as a software connector. Any module can interact with the distributed ledger layer only when interfaces defined in the software connector are followed. Software connectors are a form of middleware component for interoperability among software modules, and they are essential elements in distributed environments that affect performance, reliability, and security.

Hyperledger adopts a modular design, which makes it easy to use a distributed ledger function as a software connector in the entire architectural design because it can easily integrate functions into a system according to a designer's purpose and needs.

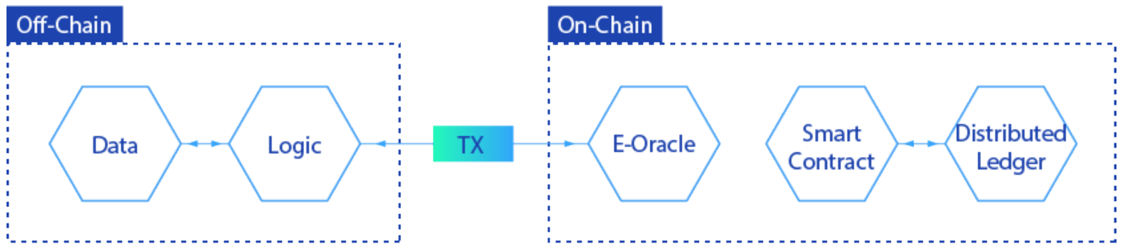


Figure 10 Blockchain as a Software Connector

3.8 Execution Layer (EL)

The Execution Layer (EL) executes, processes, and verifies transactions, providing important functions for the smart contracts running on the Eden platform.

Transactions must be used in order to update the data in Eden's distributed ledger. The EL provides a Trusted Execution Environment (TEE) in which the transaction can be safely protected from data extortion or data forgery by hacker attacks. All transactions and key logic including smart contracts are executed in the TEE.

The EL has Transaction Execution Scheduling (TES) which specifies how transactions are to be executed and on which node. TES is an important technical factor that affects performance and can show different processing performances with the same computing resources depending on how transactions are managed and executed. TES also includes an Ethereum Virtual Machine (EVM) to run smart contracts written in Solidity. If a transaction being executed is a smart contract running on Ethereum, the EL executes the EVM.

3.9 Ethereum Virtual Machine (EVM)

Currently, Ethereum's smart contracts have many users and are used in several blockchain projects to demonstrate their stability and functionality. Instead of developing a new language or system for smart contracts, Eden leverages the EVM in an Eden permissioned blockchain to increase accessibility to Eden smart contracts and to create a variety of smart contracts.

3.10 Bridge Layer (BL)

The bridge layer (BL) is an important technical element that differentiates Eden from other blockchains and guarantees zero-knowledge trusted connectivity between on-chain and off-chain.

In Ethereum, smart contracts can interact with external systems using Oracle, but this does not guarantee trustworthiness of data from external sources. To realize a smart contract-based programmable economy it is necessary to secure trusted connectivity that enables the blockchain and external systems to securely function despite threats from hackers. Once trusted connectivity is secured, it is possible to expand functions by interacting with external systems and by automating various services.

The BL is composed of an on-chain module for interacting with a smart contract, an off-chain module for interacting with an external system, and modules for networking between the on-chain and off-chain modules. The on-chain module serves as a gateway for responding to external data requests required by smart contracts. The off-chain module fetches data requested with an on-chain module by accessing the actual external system; it then verifies the data, selects a specific value, and transmits it to the on-chain module. Since the on-chain and off-chain modules are located in different networks, they configure a network on which they can securely communicate and exchange data.

3.11 Transaction Execution Scheduling (TES)

An Eden client can update data on the DLL through transactions. Smart contract execution is also possible through the transactions. Eden uses a batch and a block as units for grouping transactions. The transactions are gathered to form a batch and the batches form a block. In the EL, transactions are processed on a block-by-block basis.

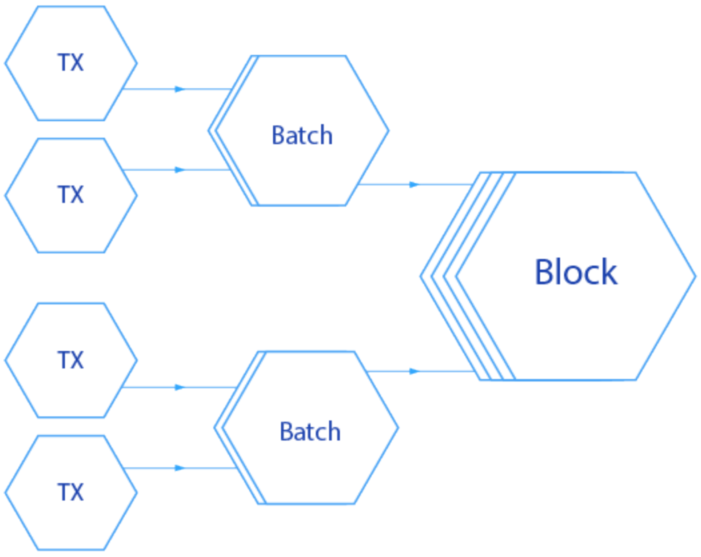


Figure 11 Transaction, Batch, and Block

By grouping transactions into batches, dependency between transactions that can occur in a smart contract can be easily observed. Transactions in the same batch are assumed to be related and if there is a problem in executing certain transactions, other transactions can be invalidated. For example, if hypothetical transaction A generated a transaction B and a transaction C, all of the transactions become related, and the process is considered to be normally completed only when all of the transactions are executed successfully. If the transactions are grouped into batches and an error occurs in the transactions, one or all of the transactions included in the batch can be invalidated without considering the execution order or correlation between the transactions. As a result, dependency problems can be easily solved.

3.12 Transaction Data Structure

A transaction consists a header and body, which contain the following data fields:

Field	Description
Public Key	Client’s Public key
Dependency	Transaction Data with Dependency

Field	Description
Namespace	A Namespace for Differentiating between Transactions
Nonce	Random Data for Differentiating between Transactions

Figure 9 Transaction Header

Field	Description
Header	Transaction header Data
Header_Signature	Signature Data
Payload	Payload Data

Figure 10 Transaction Body

Similar to other blockchain technologies, Eden is designed to sign the transaction using a client's public key. In addition, the transaction header and body are configured as a pair. The dependency field of the transaction header specifies the transaction that have a corresponding transaction and dependency. The dependency field indicates which transactions are involved when transactions are run in parallel, so transaction scheduling can be performed according to relationship and order.

A namespace plays the role of a delimiter that indicates the nature of the transaction. In Eden, anyone can create their own coin or blockchain system using smart contracts. For example, if Alice wants to create Alice Coin, she can issue Alice Coin using a supplied smart contract template. In order to trade Alice Coin issued in this way, a transaction must be generated. At this time, "Alice" is assigned to a namespace to indicate that the transaction is related to Alice Coin.

Eden can collect and process Alice Coin related transactions only by referencing the transaction header. Using this approach allows Eden to greatly improve performance and scalability by parallelizing transactions according to the namespace.

3.13 Merkle Tree

Merkle Tree was presented by Ralph Merkle in 1998 and is a tree data structure for authentication (Merkle, 1998). All of the leaf nodes have hash values of child nodes and data is stored in non-leaf nodes. Merkle Tree is currently used in a variety of projects such as IPFS, ZFS, Bitcoin, Ethereum, and Apache Cassandra because it can efficiently store data and verify data Integrity with less data.

Since a root node of the Merkle Tree is updated every time a new node is added, and a hash function is recursively applied to a hash value of the leaf nodes, a hash value of the root node is different even when only one value of a specific node is changed. This property enables data integrity to be verified only by knowing a root node value, its own node value, and a counterparty node value.

3.14 Namespace

Eden uses a Radix Merkle Tree to store a current state of the blockchain. Validator nodes that check conformity of blocks all contain Radix Merkle Tree. Radix Merkle Tree displays some data with optimal space. If there is only one child node, it unites the nodes into one, so it can effectively use memory.

In a leaf node of the Radix Merkle Tree, a node address is included, and thus it is possible to identify a sibling or a parent of the node by the node address value. A validator node examines a node address included in a transaction within a block and a batch to verify the transaction.

Node Address = Namespace + Node Path

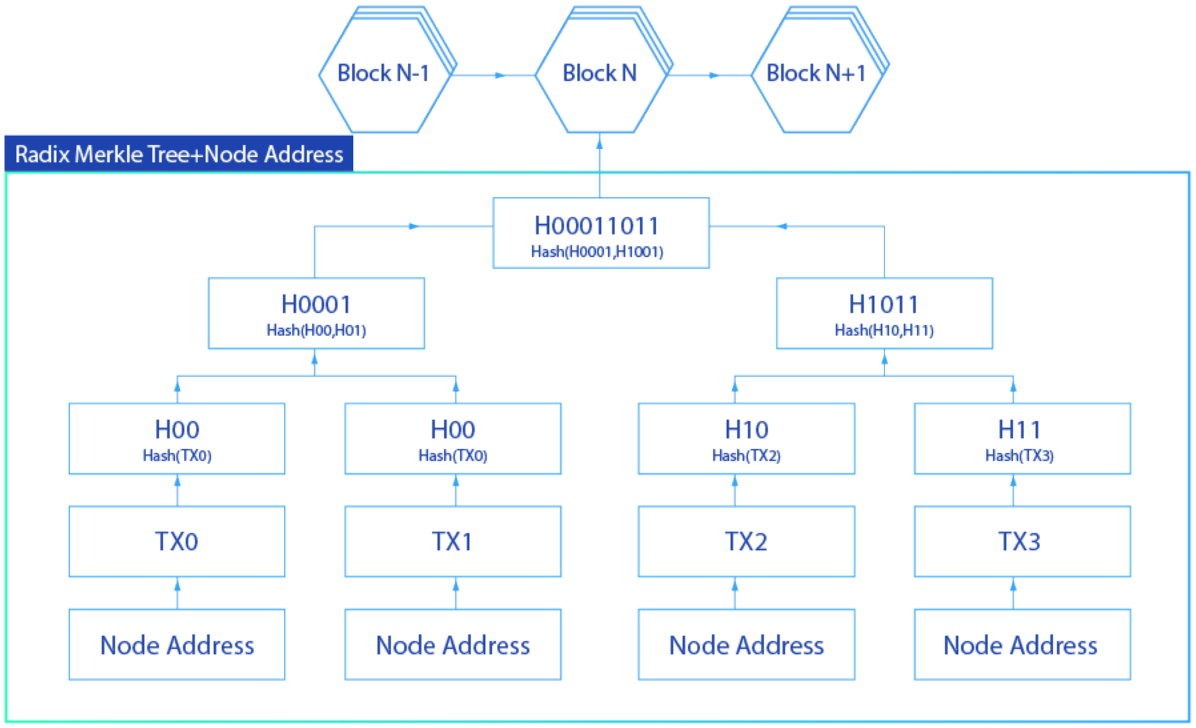


Figure 11 Radix Merkle Tree with Node Address

A namespace is a form of identification value for ascertaining the type of transaction and all transactions in Eden must contain namespace information. Validator nodes can use the namespace information to group transactions into blocks of related transactions. For example, for a transaction that contains simple transactional information, the namespace "EDN" is used, and for smart contract XYZ, a namespace "XYZ" is used. The validator node can distinguish XYZ-related transactions from EDN-related transactions by simply checking a namespace contained in the transaction. Since EDN and XYZ are different types of transactions there is no data consistency problem and both transactions can be executed in parallel. As a result, it is no longer necessary to execute one transaction at a time due to data consistency issues as is the case for many existing solutions in the blockchain space.

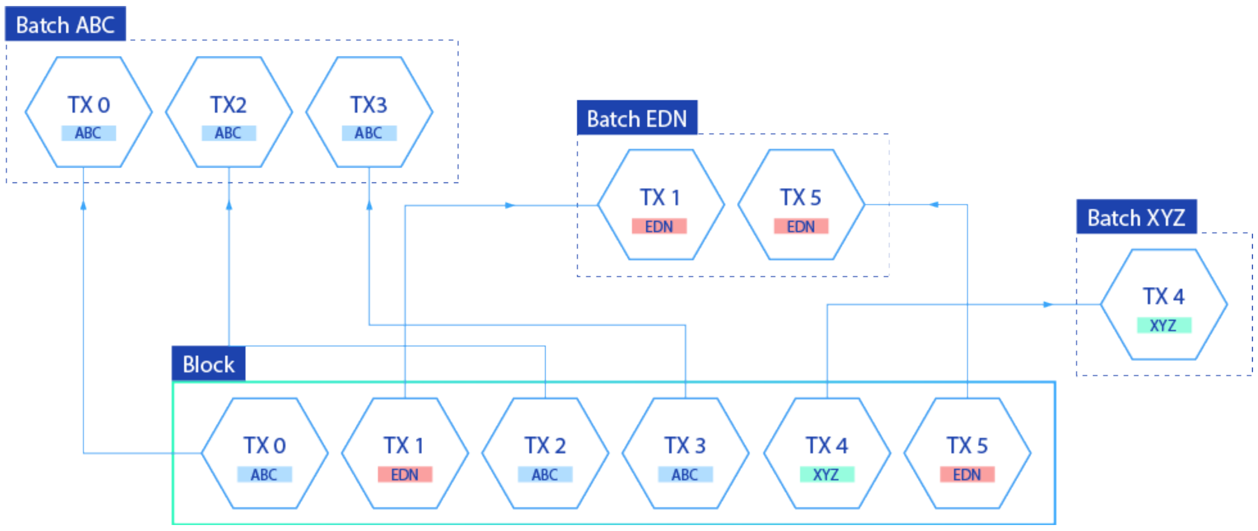


Figure 12 Transaction based on Namespace

3.15 Transaction Routing

Eden runs in a TEE prepared from a namespace by using transaction routing. When a router receives an execution request for transactions submitted from a validator node, it searches a resource registry for an execution node in order to render batches separated by the namespace and forwards the transactions to be processed by the execution node.

In the resource registry, computing nodes for processing each namespace are recorded. If the router does not find the namespace in the resource registry, it transfers the transaction to a default execution node and carries out the transaction. TEE allocated by namespaces can be run independently and simultaneously without a global lock on the current state because there does not exist data consistency issues. By monitoring a workload allocated to the TEE in real time and by constructing an appropriate number of execution nodes, it is possible to operate the blockchain with optimal computing resources according to the required computing power.

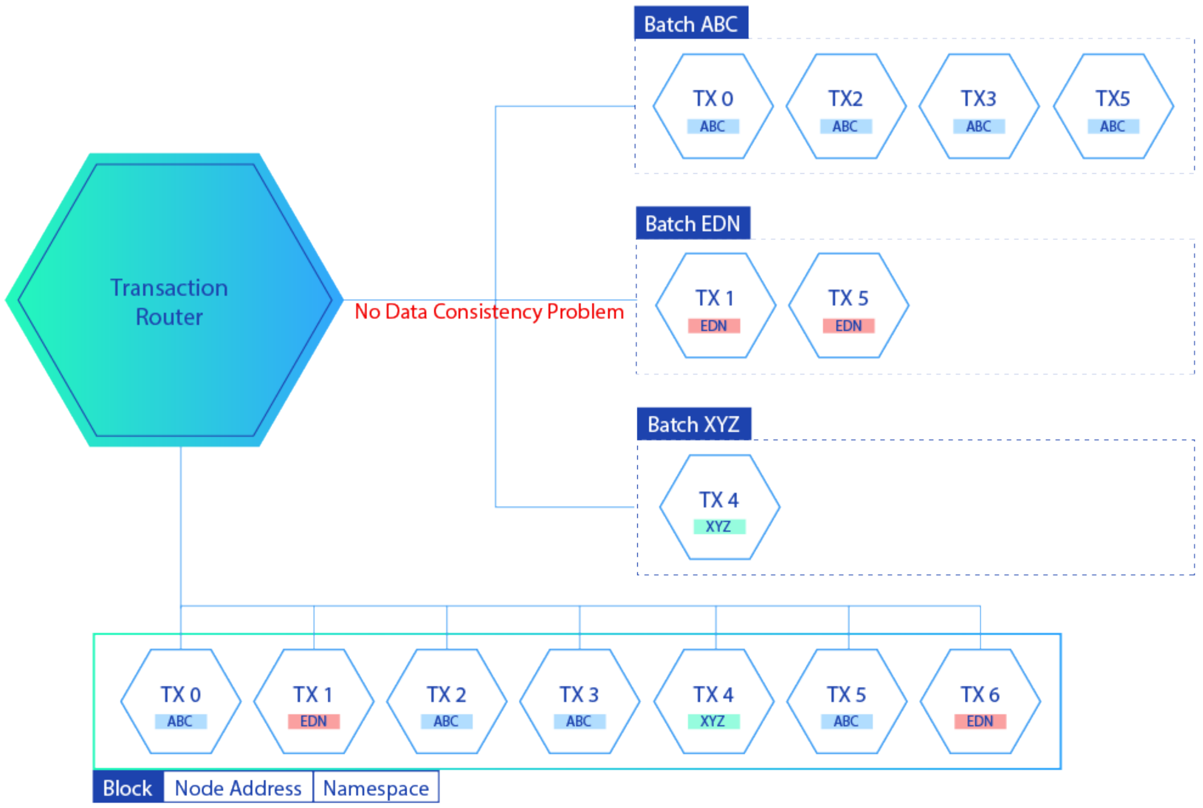


Figure 13 Transaction Routing by Namespace

3.16 E-Bridge

E-bridge is a core technology of the BL and connects Eden with an external system, which essentially acts as a software bridge connecting the on-chain with the off-chain.

Data moving within a blockchain can ensure security by implementing technologies such as a consensus algorithm and encrypted messaging. However, in the case of an external system the data cannot guarantee security because it can be attacked by hackers in various places. Therefore, when the on-chain module and the off-chain module interact, it is necessary to trust the external system and to assure that the data transmitted from the external system is securely transmitted without any threat when it is linked with the on-chain module. A trust problem of an off-chain data source can be resolved if the external system provides public services and has a reputation. For example, if Apple stock is needed in a smart contract and data is received from Nasdaq, the provided data can be considered reliable.

A trust problem that can occur in the off-chain module when communicating with an external system can be largely classified into three issues:

1. Data source reliability: the reliability problem of whether the data provided by the external system is correct
2. Reliable data transfer from a data source: correct delivery of data means that data from a reliable data source is simply transmitted fully without being subjected to hacking, data corruption or improper data fetching
3. Reliable data transfer from the off-chain module to the on-chain module: reliable data transmitted from an external system should be transmitted to the on-chain module without being attacked

E-Bridge is a technology that solves these technical problems and provides trusted connectivity between the on-chain and off-chain modules. The E-Bridge is designed in such a way that there exist software modules such as an Oracle server and enclave located off-chain, and an executor located on-chain. The Oracle clients are able to interact with one another to secure connectivity.

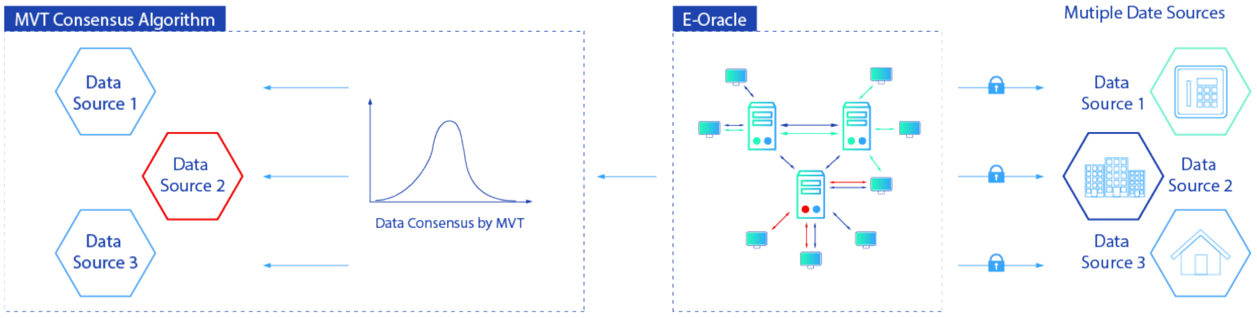


Figure 14 E-Bridge Layer

3.16.1 E-Bridge Architecture

E-Bridge consists of an executor that executes transactions, an E-Oracle client forwarding external data access requests from the smart contract, an E-Oracle server processing requests from clients, and an SGX Enclave which acts as a TEE running E-Oracle Server programs.

The Executor and E-Oracle Client are located on-chain. The E-Oracle Server and SGX Enclave are located off-chain, so the E-Protocol is used to enable secured connectivity between the on-chain and off-chain modules within the E-Bridge.

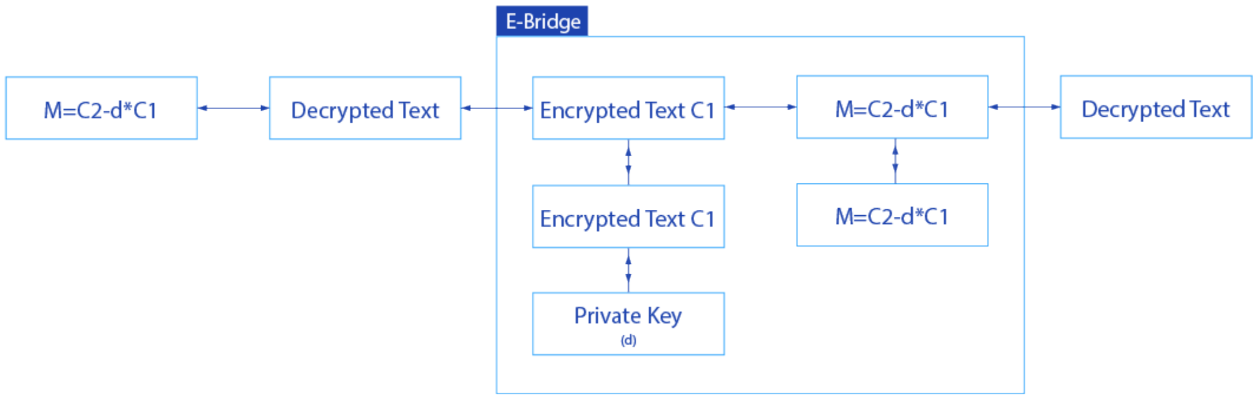


Figure 15 E-Bridge Architecture

3.17 Executor

The executor is equipped with an EVM and is responsible for running smart contracts. The executor runs in an isolated environment to ensure security and updates to the smart contract are not directly recorded on Distributed Ledger Technology (DLT). Only data verified through a validator is allowed to access the DLT. Eden is designed to run and validate smart contracts on each executor node by running N executor nodes.

3.18 E-Oracle

E-Oracle is a software module that provides functions for smart contracts to access external data. It consists of an E-Oracle client and an E-Oracle server. When the E-Oracle client requests external data, the E-Oracle server collects the data and sends it to the E-Oracle client.

Since the E-Oracle client must have the ability to be called from a smart contract, the E-Oracle client is loaded and used in the smart contract. The E-Oracle client uses an E-protocol to request data required for the E-Oracle server. The E-Oracle consists of several nodes and provides data verification methods between these nodes and a consensus algorithm for final data selection.

3.18.1 E-Oracle Client

The E-Oracle client is a special form of smart contract in which smart contracts provide an external data access functionality. When the E-Oracle client requests external data access, the E-Oracle client passes the parameters required for execution in the E-Oracle server, receives external data collected by the E-Oracle server, and delivers it to the smart contract. Therefore, the E-Oracle client acts as a gateway between the smart contract and the E-Oracle server.

3.18.2 E-Oracle Server

The E-Oracle server provides the actual functionality required to access the data requested by the E-Oracle client. An important feature of the E-Oracle server is that it can execute external data access requests from the E-Oracle client, collect external data, select appropriate values, and forward the values to the E-Oracle client.

In order to enhance security, the E-Oracle server does not run an external data access related code within the E-Oracle server itself but runs the code in a separate space called an SGX enclave. The E-Oracle server consists of N-nodes, and any external data is designed to fetch multiple pieces of data. If data from one E-Oracle server is imported, multiple E-Oracle servers may be run because data reliability or systemic problems may prevent proper importing of the data.

3.18.3 SGX Enclave

An SGX enclave is Intel's Software Guard Extension (SGX) enclave, which runs programs in a trusted environment to ensure security. The SGX enclave protects data and programs running in the enclave from hacker attacks by placing the programs in separate spaces, encrypting the data, and making them inaccessible by external processes.

Encrypted data and code are safe because they are able to be decrypted only within the enclave. Even if a hacker finds the data and code by directly accessing a memory with a side channel attack it is impossible to decrypt them because there is no way to determine the private key within the enclave.

According to senior Wipro software engineer Surenthar Selvaraj, the SGX Enclave provides security at the CPU level to defend against hardware and software attacks on Intel (Selvaraj, 2016). The memory used by the Enclave cannot be read or written regardless of the Privilege Level and CPU Mode, and cannot enter the Enclave environment with traditional calls, jumps, register manipulation, or stack manipulation. In addition, the keys used for encryption in the Enclave are randomly changed and the key values are stored in the CPU, not in memory, which cannot be accessed externally.

Eden runs the E-Oracle Server code on the SGX Enclave in order to provide robust data and system security and to defend against any form of hacker attack.

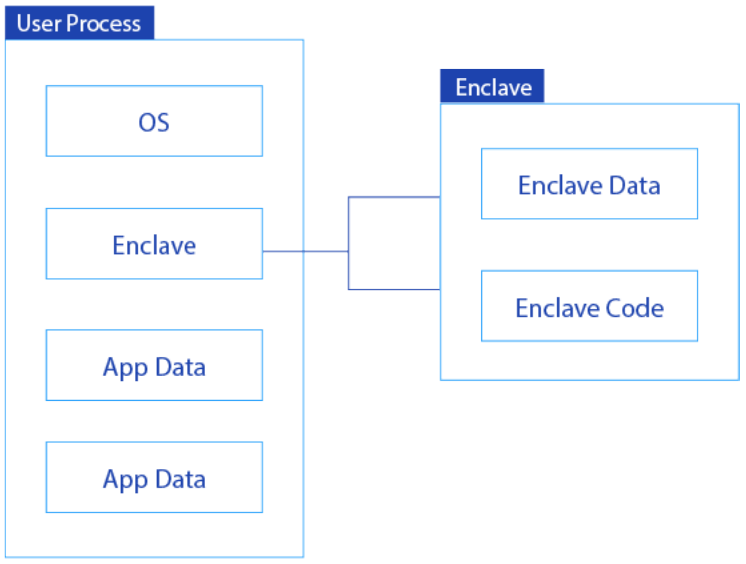


Figure 16 SGX Enclave

3.18.4 E-Oracle Consensus

Eden runs multiple E-Oracle servers to secure trusted connectivity. Though external data may have been imported while running an E-Oracle server, complete correctness of the data cannot always be ensured because there may be a system error or issue even when the data is derived from a trusted data source.

To avoid this problem, the E-Oracle operates with 2n-1 called an E-Oracle pool. If multiple E-Oracle pools are used, it will be difficult to determine which of the several values provided by each E-Oracle server to use.

The types of values that can be received from the E-Oracle server can be divided into two types, a discrete type value and a continuous type value. The discrete type value is a discontinuous value and refers to a value of type, such as true, false, "man", and "woman", while the continuous type value refers to successive values such as 15.34432 and 1.0213. While discrete type values can have the same value on multiple E-Oracle servers, the value of continuous type values can vary depending on the data source and the nature of an external system, even when the data is generated from a trusted data source. For example, if there are derivatives that are determined by Apple stock prices, an E-Oracle server acquires the Apple stock prices through various services such as Nasdaq and Yahoo Finance. If the value from Nasdaq is 175.01 and that from Yahoo Finance is 174.98, the E-Oracle server selects either one of the two values or combines these values to determine a final value to deliver to an E-Oracle client. E-Oracle consensus is a technology that determines which of several values will ultimately be used in this situation.

3.18.5 Schelling Point

American Economist Thomas Schelling first devised the concept of the Schelling Point, in which people tend to behave naturally or in common sense to think for themselves rather than act randomly when doing something (Schelling, 1960). Schelling famously responded to the question of "When and where should I meet a stranger in New York if I have to meet the stranger tomorrow?" by stating “at the information booth of the Grand Central Station at noon". He explained the reason why people are likely to choose this time and place is because they are traditionally the most used time and place for appointments.

One can use the Schelling Point to create an agreement on external data within the E-Oracle. The E-Oracle server is a separate node and accesses external data using parameters passed from a smart contract and receives results. E-Oracle servers have to exchange values and select a final value by itself without a centralized system. An important basic assumption is that there is no trust in the values submitted by each of the E-Oracle servers.

Ethereum Founder Vitalik Buterin suggested the creation of Schelling Coin as a solution to this problem (Buterin, 2014). If the Schelling Coin is offered to those submitting the ultimately selected correct value as an incentive for individuals participating in the process of determining a value, then the participants will submit a value similar to the value that other individuals submit for receiving payment. As this process is repeated, values selected by the participants naturally approach an actual value.

The Schelling Point uses common facts as a focal point to make a selection. The Schelling Point has to create common knowledge among participants, continuously present incentives to distinguish between incorrect and correct values and lead the value that the majority of participants select. However, it is difficult to apply the Schelling Point if the participants are constantly changing or if common information about the value they should choose is not created.

3.18.6 E-Oracle Consensus by Median Voter Theorem (MVT)
E-Oracle servers can contain two types of data, discrete type data and continuous type data, and each data type requires a distinct consensus algorithm. E-Oracle applies majority voting to discrete type data and MVT to continuous type data.

3.18.7 Majority Voting

If the value that the E-Oracle can have is limited to "Yes"/"No" or "True"/"False," majority voting is performed to select the most common value of the E-Oracle server and confirms the value as the final value. Majority voting is a commonly used consensus algorithm that can be applied to discrete type data without difficulty.

3.18.8 Median Voter Theorem (MVT)

MVT is a method applied when there exists continuous type data. MVT is a theory in which the majority voting system will choose the result preferred by a median voter (Holcombe, 2006).

Economist Duncan Black explained that the result chosen by the median voter in the majority voting system is a "Nash Equilibrium", and as a value moves away from the value of the median voter, no benefit can be gained (Black, 1948). This implies that if a decision other than the median voter's choice is made when a large number of participants try to determine a certain value within a majority system, no benefit can be realized. In this case, median voter refers to a person who thinks that the choice selected by half of the participants is the ideal choice.

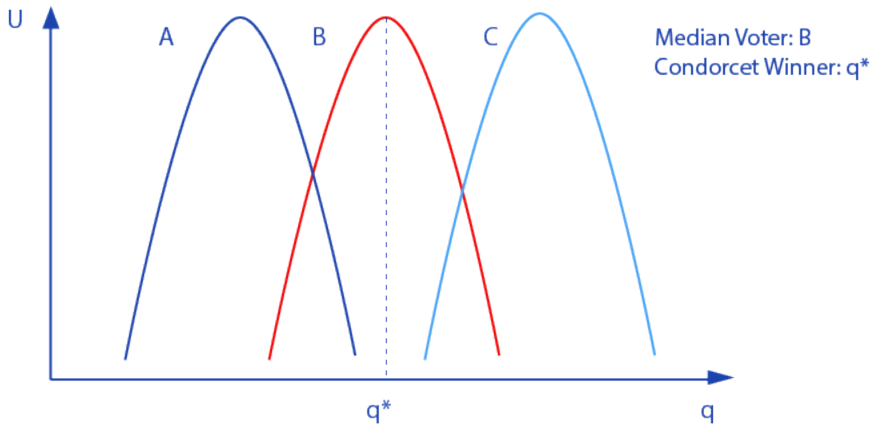


Figure 17 MVT (Puglisi, 2011)

3.19 Voting Theorem

In case of a common resource, Voting Steady-State Equilibrium (VSSE) of a resource utilization and growth rate is determined by the median voter (Borissov, 2010). An equation illustrating this principle is shown below.

$$P_t^* = (1 + \pi^*)^t P^*$$

Figure 21 Voting Steady-State Equilibrium (VSSE)

A voter strives to maximize utility. In order to increase the value of VSSE, the number of individuals who want to increase the VSSE status value must exceed 50% of the total number of individuals.

The value of VSSE is determined using the following equation. ($\boldsymbol{\beta_m}$ = Median Voter)

$$p^* = 1 - \beta_m$$

3.19.1 The equilibrium growth rate is determined as follows:

$$1 + \gamma^* = [(1+\lambda)\beta_m^{1-\alpha_1-\alpha_2}]^{\frac{1}{1-\alpha_1}}$$

Therefore, the final equilibrium state value, VSSE, is determined by the median voter, and if a majority is not secured, the VSSE value cannot be changed. Since MVT consensus algorithms are implemented in the SGX enclave, hackers must hack more than a half of the nodes in order to succeed in their attack, which requires considerable time and effort.
Since the MVT uses the median as the final selection value, it can be applied to continuous type data and thus can solve issues for cases in which majority voting is not suitable. In addition, since the MVT does not require common information unlike the Schelling Point, it can select the final value from among the values submitted by E-Oracle servers without any prior information. Finally, since the MVT does not need to introduce a separate incentive system, it can be simply and clearly implemented. Therefore, the E-Oracle server uses the MVT to select the final value when selecting continuous type data.

3.20 Threat Model

Majority voting and MVT used by the E-Oracle consensus are vulnerable to a 51% attack. If a hacker succeeds in taking over 51% of all E-Oracle servers exposed externally, the values submitted by the hijacked server will be the median, and the E-Oracle server will send the modulated values to the E-Oracle client.

In order to prevent such a risk, an E-Oracle consensus module runs programs in the SGX enclave to defend against hardware and software attacks and accesses external services using HTTPS. Data sources that do not support HTTPS do not allow data to access the E-Oracle server.

4. E-Protocol

E-Protocol is a protocol used for communication between E-Oracle servers in an E-Oracle pool. The E-Oracle servers need to interact with one another to access external data and to reach a consensus on collected data. Although the E-Oracle server cannot manipulate data or tamper with the program running on the SGX enclave, there is a possibility that a hacker could sneak into network packets, which must be prevented.

An E-protocol allows for communication using threshold cryptography and encrypts data as well. Threshold cryptography is an encryption protocol that allows K people in a group composed of N people to share special values to protect their data. (Shoup, 2000). Furthermore, threshold cryptography is a protocol with a cooperative property. Data necessary for decryption is shared among participants so that encrypted data can be decrypted only when data from one individual as well as that of other participants is present.

Threshold cryptography contains the following characteristics (Lee, Shen, and Wheatman, 2016):

1. A trusted third party may not be required depending on the algorithm used
2. Participants smaller than K cannot decrypt the message
3. An attacker cannot use the acquired information to determine the signatures of future messages
4. After signatures are created, participants do not have to create or mix signatures again

An E-protocol is applied to the process of importing the data requested by a smart contract session and an external E-Oracle client, finally selecting the value and returning it to the E-Oracle client.

4.1 Elliptic Curve Cryptography (ECC) – Threshold Cryptography (TC)

The RSA threshold method does not require a trusted third party and does not leak important information during the initial setup for secret sharing (Nguyen, 2005). The RSA threshold improves safety by solving the problems of existing threshold cryptography. The ECC–TC method provides a similar level of security with keys which are smaller in size than those of the RSA threshold, and thus it can complete operations faster while using less memory (Ibrahim & Ali, 2003). ECC - TC does not share keys because both a public key and a private key exist as points on an elliptic curve (Padmavathi & Lavanya, 2012). The ECC-TC consists of three stages: key generation, encryption, and decryption.

The ECC - TC encryption divides a message to be encrypted into N and encrypts each piece of the divided message.

The general steps for applying ECC-TC are described as follows:

1. Randomly select abase point (x, y) on the elliptic curve.
2. Place plaintext at (xm, ym).
3. Divide “m” by “w” to apply TC
4. Each node selects a private key "n" and calculates a public key.
   p = n(x,y)
   For example, Node A must generate a random number k to encrypt the message.
   Cm = {k(x,y),(xm,ym) + kpA}
5. To decrypt Cm, proceed as follows
   ((xm,ym) + kPA – nA(k(x,y)) = (xm,ym) + k(nA(x,y) – nA(k(x,y)) = (xm,ym)

The encrypted messages can be reconstructed using Shamir's method and decrypted using the private key to read the original message. The E-protocol uses ECC-TC to encrypt messages used by an E-Oracle. The following figure illustrates the process of encrypting and decrypting.

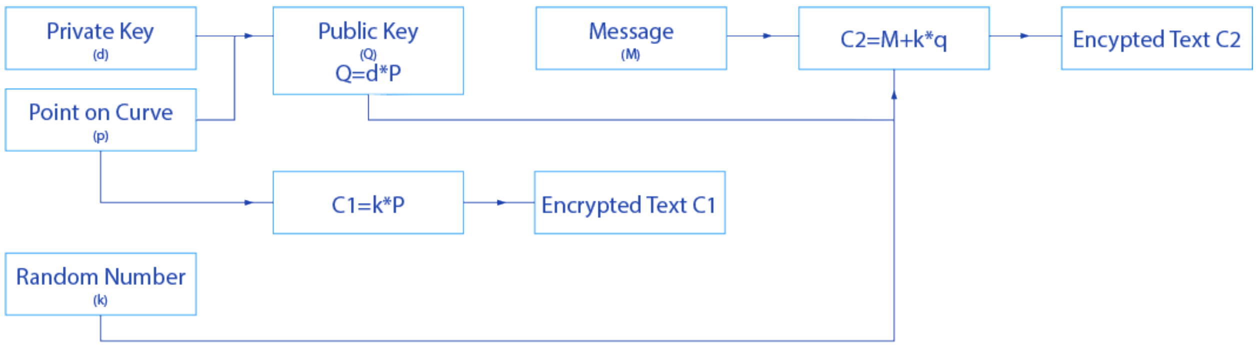


Figure 18 ECC-TC Encryption

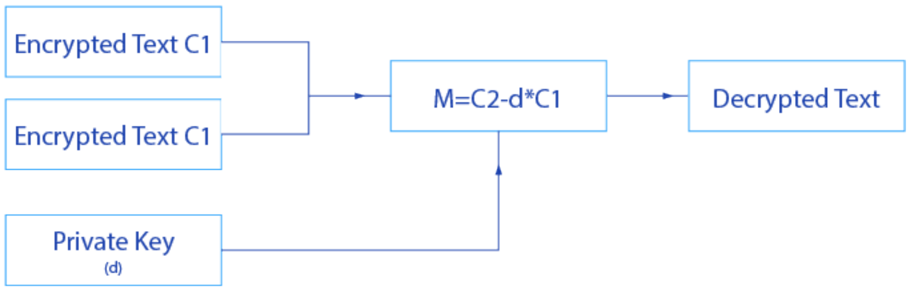


Figure 19 ECC-TC Decryption

4.2 External Attacker

If an external attacker accessed an E-Protocol message, the information that can be obtained is a public key Q and a random number K. These two pieces of information can also be obtained in a normal ECC, but an attacker cannot obtain any other information. Even if an external attacker accessed the message and knew that the E-Oracle server uses ECC-TC, the attacker can only obtain the public key Q and the random number K as before, and thus it is impossible to decrypt the message or identify the signature.

4.3 Attack from a Malicious Participant

It is possible to attempt to modulate or decrypt a message within one node participating in an E-Oracle session. However, such an attempt will also have little effect, since to decrypt a message in ECC-TC, it is necessary to know the secret keys of the nodes participating in the ECC-TC. It is impossible to decrypt the message with one individual secret key.

5. Confidentiality on Eden

Confidentiality in a blockchain can be divided into three aspects. The first is confidentiality of a smart contract, which ensures that only specific individuals are able to observe the content of the smart contract. The second is confidentiality of a transaction, which ensures that the content of the transaction is available only to the specific individuals. The third is Oracle confidentiality, which ensures that no data is exposed when requested by an external system. On account of the philosophical characteristics and the technology of the blockchain, it is difficult to secure the first and second forms of confidentiality.

In Eden, a variety of smart contracts are implemented. If an individual allows an encrypted smart contract, then that person should have full confidence in the person or organization that submitted the smart contract and execute it, believing that the initial contract is safe. This is an important assumption that can greatly affect the overall security of Eden, which cannot be guaranteed. Therefore, Eden does not allow encrypted smart contracts. Encryption for transactions is an essential element in some financial transactions such as in specialized sealed auctions. However, since Eden's smart contracts need to interact with various external systems, if Eden cannot provide records of past transactions or ownership when requested from the outside, the utility of Eden will decline.

Oracle confidentiality is an important security factor for communicating with external systems and trusted connectivity is secured using smart contracts. In order for smart contracts to be automate and secure, continuous interactions with external systems is required, and in this process, important information is able to migrate throughout the network. If a hacker is able to access this information, it may cause a margin of misuse and seriously undermine security.

5.1 Confidentiality by SGX and Encrypted Protocol

Confidentiality is ensured by using an SGX enclave and an encrypted protocol, and privacy is guaranteed so that details of the transaction and smart contract executed in Eden cannot be seen by anyone.

All processes, such as E-Oracle consensus, which access an external system from an E-Oracle server to import data and select one of multiple pieces of data, are executed in the SGX enclave. While the SGX enclave is running with limited access and data encryption, it cannot be understood by insiders such as system operators or hackers.

In the SGX enclave, external data is imported using HTTPS. Because HTTPS uses an SSL certificate and encrypted communication, it both reduces the risk of hacking and prevents an insider from learning the content of the data. Smart contracts support interactions with external systems that support HTTPS and do not allow unencrypted communication protocols. Since the SGX enclave does not store the data used for communicating with external systems in a permanent storage device such as a disk, it is deleted immediately after the execution ends.

5.2 Encrypted Protocol

Hackers have two possible attacks in terms of networking while the E-Oracle is running. SGX enclave networking with external systems and networking between E-Oracle servers. Since communication between the SGX enclave and external system requires HTTPS, the data is encrypted. In addition, communication between E-Oracle servers is encrypted using ECC-TC, and as a result it is not possible to decrypt or manipulate a message even if an attacker accesses the message.

5.3 Lack of Censorship

All transactions and smart contracts executed in Eden run on an EVM and the SGX enclave, and thus the data used here cannot be controlled, accessed or censored. Eden is a permissioned blockchain, but similar to any permissionless blockchain, it does not censor data, and therefore provides an environment in which all transactions and smart contracts run equally and completely.

5.Conclusion

Eden is a blockchain technology that improves speed and security, which are typical technical requirements for applying blockchain technology to programmable economics. Due to the data consistency problem, serial execution is not scalable and increasing computing power to handle more transactions cannot increase processing speeds. Eden uses namespace technology to tie transactions that contain the same Namespace with one other. Transactions belonging to different Namespaces have no data integrity issues and can be executed in parallel simultaneously. As the number of namespaces increase, individuals can increase their processing speeds by adding computing power, which can aid in achieving higher levels of scalability. If individuals require faster processing, they can create each Namespace Computing

Zone that can process transactions based on namespace in order to maintain optimal performance.

If the blockchain is used in conjunction with an external service to deal with a large number of transactions, the risk of a subsequent hacking event increases. Conventional blockchain techniques focus on the security of the blockchain itself, so external security is often not provided. Eden uses E-Oracle technology to effectively defend against hacker attacks in conjunction with external services. E-Oracle technology uses Multiple Data Source, ECC-TC, and MVT technologies. Given that the E-Oracle is implemented in an SGX Enclave, it is nearly impossible to hack. In order to succeed in hacking, 51% of the E-Oracle must attack the server. With Eden, one can easily use blockchain technology for a diverse array of enterprise use cases.

References

Alharby, M. and Moorsel, A. (2017). Blockchain-based Smart Contracts: A Systematic Mapping Study. Computer Science & Information Technology.

Black, D. (1948). On the Rationale of Group Decision-making. Journal of Political Economy.

Bontje, J. (2016). Does every node execute the contract code for each transaction? Retrieved from https://ethereum.stackexchange.com/questions/357/does-every-node-execute-the-contract-code-for-each-transaction

Buterin, V. (2014). SchellingCoin: A Minimal-Trust Universal Data Feed. Retrieved from https://blog.ethereum.org/2014/03/28/schellingcoin-a-minimal-trust-universal-data-feed/

Buterin, V. (n.d.). A next-generation smart contract and decentralized application platform. Retrieved from https://github.com/ethereum/wiki/wiki/White-Paper/

Cameron-Huff, A. (2017). Nasdaq. Retrieved from http://www.nasdaq.com/article/how-tokenization-is-putting-real-world-assets-on-blockchains-cm767952

Chamber of Digital Commerce (2016). Smart Contracts: 12 Use Cases for Business & Beyond.

Chianese, A. (2015). Smart environments and Cultural Heritage: A Novel approach to create intelligent culural spaces. Journal of Location Based Services.

Ehrsam, F. (2017, 6). Scaling Ethereum to Billions of Users. Retrieved from Medium: https://medium.com/@FEhrsam/scaling-ethereum-to-billions-of-users-f37d9f487db1

Eyal, I. (2015). The miner's dilemma. IEEE Symposium on Security and Privacy.

Tschorsch, F. (n.d.). Bitcoin and Beyond- A Technical Survey on Decentralized Digital.

Foundation, L. (2017). Sawtooth. Retrieved from https://sawtooth.hyperledger.org/docs/core/releases/latest/introduction.html#proof-of-elapsed-time-poet

Harding, D. A. (2015). Is there a maximum size of a scriptSig/scriptPubKey? Retrieved from Bitcoin Stack- Exchange: http://bitcoin.stackexchange.com/questions/35878/is- there-a-maximum-size-of-a-scriptsig-scriptpubkey.

Holcombe, R.G. (2006). Public Sector Economics, Upper Saddle River: Pearson Prentice Hall, p. 155.

Ibrahim, M.H. and Ali, I.A. (2003). A robust threshold elliptic curve digital signature providing a new verifiable secret sharing scheme. Circuits and Systems.

Jameson, H. (2017, 6). Accounts, Transactions, Gas, and Block Gas Limits in Ethereum. Retrieved from https://hudsonjameson.com: https://hudsonjameson.com/2017-06-27-accounts-transactions-gas-ethereum/

Borissov, K. (2010). Common and private ownership of exhaustible resources: theoretical implications for economic growth. Istanbul.

Konstantinos, C.. (2016). Blockchains and Smart Contracts for the Internet of Things. IEEE.

Lamela-Seijas, P. (2017). Scripting smart contracts for distributed ledger technology.

Lee, H., Shen, H., and Wheatman, B. (2016). Implementation and Discussion of Threshold RSA.

Lewis, A. (n.d.). Agentleintroductiontosmartcontracts. Retrieved from https://bitsonblocks.net/2016/02/01/a-gentle-introduction-to-smart-contracts

Li, S., Xu, L.D., and Zhao, S. (2015). The internet of things: A Survey. Information Systems Frontier.

Luu, L. (2015). On Power Splitting Games in Distributed Computation: The Case of Bitcoin Pooled Mining. IEEE.

Flood, M.D. (2017). Contract as Automaton: The Computa- tional Representation of Financial Agreements.

Merkle, R. (1998). “A digital signature based on a conventional encryption function”. Retrieved from Merkle Paper: https://people.eecs.berkeley.edu/~raluca/cs261-f15/readings/merkle.pdf

Meshenberg, R., Gopalani, N., and Kosewski, L. (2013). “Active-Active for Multi-Regional Resiliency”. Retrieved from Medium: https://medium.com/netflix-techblog/active-active-for-multi-regional-resiliency-c47719f6685b

Microsoft. (2017). Multiple Datacenter Deployment Guidance. Retrieved from Developer Network: https://msdn.microsoft.com/en-us/library/dn589779.aspx

Morabito, V. (2017). Smart contracts and licensing, in Business Innovation Through Blockchain.

Nakanishi, S. (2016, 6). Are gas limit in transaction and block gas limit different? Retrieved from StackExchange: https://ethereum.stackexchange.com/questions/7359/are-gas-limit-in-transaction-and-block-gas-limit-different

Nguyen, H. (2005). RSA Threshold Cryptography.

Padmavathi, G., and Lavanya, B. (2012). Comparison of RSA-Threshold Cryptography and ECC-Threshold Cryptography for Small Mobile Adhoc Networks. Advanced Networking and Applications.

Puglisi, R. (2011). The Political Economics Approach.

Schelling, T. (1960). The Strategy of Conflict. Harvard University Press.

Selvaraj, S. (2016). Overview of Intel Software Guard Extension Enclaves. Retrieved from Intel : https://software.intel.com/en-us/blogs/2016/06/06/overview-of-intel-software-guard-extension-enclave

Shoup, V. (2000). Pratical Threshold Signature. IBM Research Lab.

Solidity. (n.d.). Solidity. Retrieved from http://solidity.readthedocs.io/en/develop/index.html

Huckle, S. (2016). Internet of Things, Blockchain and shared Economy applictions.

Szabo, N. (1996). Smart Contracts: Building Blocks for Digital Markets.

Szabo, N. (1997). Formalizing and Securing Relationships on Public Networks.

Pureswaran, V. (2015). Device Democracy - Saving the future of the Internet of things. IBM.

Gramoli, V. (n.d.). The Blockchain as a Software Connector.

Wood, D. G. (2017). Ethereum: A secure decentralised generalized transaction ledger.

Xu, X., Pautasso, C., and Zhu, L. (2016). The Blockchain as a Software Connector. IEEE.