Scientific Research Publishing

Journal of Computer and Communications, 2021, 9, 95-109
https://www.scirp.org/journal/jcc
ISSN Online: 2327-5227
ISSN Print: 2327-5219

Trusted Blockchain Oracle Scheme Based on Aggregate Signature

Xiaodong Liu[1], Jun Feng[2]

[1]College of Information Science and Technology, Jinan University, Guangzhou, China
[2]Hangzhou Institute of Quality and Technology Supervision and Testing, Hangzhou, China
Email: cryptdong@163.com

How to cite this paper: Liu, X.D. and Feng, J. (2021) Trusted Blockchain Oracle Scheme Based on Aggregate Signature. *Journal of Computer and Communications*, **9**, 95-109.
https://doi.org/10.4236/jcc.2021.93007

Received: February 8, 2021
Accepted: March 27, 2021
Published: March 30, 2021

Abstract

With the development of blockchain technology, more and more applications need out-of-chain data. Thus, blockchain oracles have become an important bridge for transferring data on and off the chain. This paper studies the mainstream blockchain oracles scheme, summarizes the shortcomings of the existing schemes and proposes a new blockchain oracle scheme based on BLS (Bohen-Lynn-Shacham) aggregation signature to ensure that off-chain data can be transferred into the blockchain in a trusted and reliable way. Specifically, the scheme uses multiple blockchain oracles to avoid the single point of failure or even a small number of malicious oracles, and improve the credibility of data. At the same time, it not only uses BLS aggregate signature to reduce the storage cost and communication overhead, but also uses commitment mechanisms to ensure the reliability and authenticity of the data. Besides, the simulation results show that the scheme can meet the practical application requirements.

Keywords

Blockchain, Blockchain Oracle, Aggregate Signature, BLS Signature

1. Introduction

Blockchain is a decentralized, unchangeable data ledger [1]. It stores and verifies data through network nodes, which can solve the trust problem between unrelated nodes. Therefore, the blockchain will break the existing business model and infrastructure in many fields. And its distributed, immutable, persistent and other characteristics [2] make it have broad application prospects in the fields of finance, logistics, digital copyright, and public services.

As a special form of data, smart contracts [3] can also be written into the

blockchain. When the pre-set conditions are met, smart contracts can be automatically executed [4], ensuring that the entire process of data storage, reading, and execution is transparent, traceable, and immutable, thus creating a new and powerful way to establish a trust relationship between objects.

It is worth noting that the blockchain is a deterministic and closed system environment, and no uncertain factors are allowed in smart contracts. Therefore, the blockchain cannot actively obtain off-chain data. In order to overcome the limitations of smart contracts effectively and enable off-chain data to communicate with smart contracts and transfer data, it is necessary to deploy a bridge connecting the real world and smart contracts on the blockchain. The bridge is called blockchain oracle [5]. The user first informs the oracle on the chain of the external data that needs to be obtained through the network request, and then the oracle node obtains the external data through the off-chain API, and finally, the off-chain data will be returned to the user through the oracle. The off-chain data acquisition process [6] of the oracle is shown in **Figure 1**.

The oracle can be regarded as the only interface for data interaction between the blockchain and the external world, which is particularly important in the construction of the blockchain ecosystem. Take the network transaction system based on blockchain technology as an example. In order to be scalable and reliable, the system not only needs to obtain data on the blockchain efficiently and quickly, but also needs to obtain real data from off-chain or cross-chain data.

Due to the consensus algorithm of the blockchain and the use of a hash algorithm based on the Merkle tree structure, the integrity of the data on the chain is guaranteed. However, because the off-chain data is independent of the blockchain, a specific solution is needed to ensure the reliability and consistency of the off-chain data. In order to broaden the application scenarios of blockchain and ensure the reliability of data on the chain, this article summarizes the advantages and disadvantages of the current mainstream blockchain oracle scheme and proposes a new blockchain oracle based on BLS (Bohen-Lynn-Shacham) aggregation signature. The scheme reduces the storage space of the blockchain while ensuring the credibility of the on-chain data, and proves the practical value of the scheme through simulation experiments.

The organization of this paper is as follows: Chapter 2 introduces the current mainstream blockchain oracle scheme, and summarizes its advantages and disadvantages; Chapter 3 introduces the theoretical basis of BLS aggregation signature; Chapter 4 proposes a new blockchain oracle scheme based on BLS aggregation signature, and discusses the security and correctness of the scheme; Chapter 5

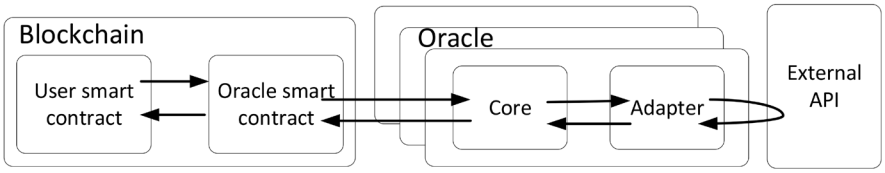

Figure 1. Blockchain oracles framework.

conducted experiments on the program, and verified the feasibility of the program through comparison; Chapter 6 summarizes the work of this article and proposes the next research plan.

2. Related Work

The current mainstream blockchain oracle on-chain mechanism [7] is divided into three categories, namely software-based, hardware-based and voting-based oracles. Among them, software-based oracles include Provable [5], TLS-N [8], etc.; hardware-based oracles include Town Crier [9], etc.; voting-based oracles include ChainLink [5], Augur [10], ASTRAEA [11], etc.

Provable [5] is an oracle scheme that provides centralized data transmission services for Ethereum, which provides a security guarantee for the data transmission layer of smart contracts. Provable obtains external data from the API or data source specified by the user's smart contract and can prove that the data provided to the smart contract is the correct data obtained from the specified API or data source at a specific point in time, ensuring verifiability and availability of out-of-chain data. However, it is only applicable to unsupported TLS versions (1.1 or lower), and can only be used in Ethereum, which is costly and relies on a single data source.

TLS-N [8] is the first oracle scheme based on content extraction signatures. It uses the privacy protection function of Transport Layer Security (TLS) to generate non-interactive session proof based on blockchain smart contracts that can be effectively verified by third parties. TLS-N provides a practical and decentralized blockchain oracle, which enhances the auditability and reliability of web content. When generating a certificate, part of the TLS session (such as passwords, cookies) can be hidden to protect privacy, while verifying the rest of the content. TLS-N is compatible with TLS 1.3, but the TLS-N scheme increases communication overhead and the deployability is poor, which requires some improvements to TLS. It can be understood from previous deployments that the standardization and adoption process of TLS is very slow, so it will take longer time to improve the TLS standard.

Town Crier [9] is a verifiable data supply oracle system based on the Trusted Execution Environment (TEE), using trusted hardware (Intel SGX, Intel Software Guard Extensions) and HTTPS designed a trusted hardware hybrid protocol based on blockchain to solve the limitation of HTTPS lack of digital signature. Among them, Intel SGX's Enclave can encrypt and decrypt data requests from smart contracts or external data from data sources, and securely manage sensitive information. However, Town Crier requires hardware support, highly depends on the trusted execution environment, which is subject to security vulnerabilities attacks against Intel CPU and SGX, such as Foreshadow [12] and Spectre [12].

ChainLink [5] is the first decentralized oracle solution on Ethereum. It combines smart contracts on the chain with distributed data sources off the chain

and securely pushes data between smart contracts and APIs through a reputation mechanism and an aggregation model. However, the aggregation cost of distributed data sources is high and the scalability is poor.

Augur [10] is a decentralized and low-cost prediction market platform oracle built on Ethereum. The prediction market is open to the general public, where users vote on information from the outside world. And their voting rights are allocated to different token holders, and the prediction results need to be agreed by a majority of users. Augur's incentive structure is designed to encourage users to remain honest and report accurate results to maximize their profits. However, Augur's consensus mechanism design leads to low prediction efficiency, prediction accuracy is restricted by the scale of the platform, and its uneven distribution of tokens damages the credibility of prediction results.

In general, the current mainstream oracle implementation schemes have the following problems: 1) Single data source. Users need to specify a single data source to obtain external data. If the data source itself is fake or maliciously tampered with, the data returned by the oracle is also wrong; 2) Poor deployment. The solution depends on hardware security or needs to make substantial changes to existing mainstream protocols; 3) High complexity of the scheme. It leads to uneven distribution of incentive mechanism or affects the final result; 4) Limited application scope. The current blockchain can guarantee the integrity of on-chain data. However, it cannot effectively guarantee the consistency of the off-chain data [13]. Besides, although the oracle can help the blockchain to obtain external data, it is still difficult to guarantee the reliability of external data sources [14], and if a single predictor is used, it will lead to centralization, leading to the risk of corruption, malicious and incorrect data generation.

In response to the above problems, this paper combines the mainstream blockchain oracle scheme and proposes an automated, lightweight and accurate blockchain oracle scheme. The solution uses multiple blockchain oracles to avoid single points of failure or even a small number of malicious oracles. The use of BLS aggregated signature reduces block space overhead and communication load. Besides, the method of using commitments to reveal ensures the authenticity of the off-chain data and improves the scalability and reliability based on the blockchain system.

3. Theoretical Basis

3.1. Bilinear Mapping

Let G_1, G_2, G_T each be a multiplicative cyclic group with a large prime number p. Let $e: G_1 \times G_2 \to G_T$ be a map with the following properties:

1) Bilinear: For all $U \in G_1, V \in G_2$ and $a, b \in Z_p^*$, there is always $e(aU, bV) = e(U, V)^{ab}$.

2) Non-degeneracy: There exists $U \in G_1, V \in G_2$, such that $e(U, V) \neq 1$, in other words, the map does not send all pairs into the identity in G_T.

3) Computability: For all $U \in G_1, V \in G_2$, the polynomial-time algorithm

calculation can be found.

3.2. BLS Signature Algorithm and Aggregate Signature Algorithm

BLS signature algorithm [15] [16] is a digital signature algorithm constructed on the basis of bilinear mapping, which is mainly divided into four parts: initialization, key generation, signature and verification. BLS aggregate signature algorithm [17] also mainly consists of four parts: initialization, key generation, aggregate signature and aggregate signature verification. Its algorithm initialization and key generation are the same as BLS signature algorithm.

1) Initialization

Let G_1, G_2, G_T each be a multiplicative cyclic group with a large prime number *p*, the generators are g_1 and g_2 respectively. The bilinear pairing is given by $e: G_1 \times G_2 \to G_T$. Define cryptographic hash functions $H: \{0,1\}^* \to G_2$. The publish system parameters are params = $\left(G_1, G_2, G_T, e, g_1, g_2, p, H\right)$.

2) Key generation

Choose a random number $x \in Z_p$ and calculate $v = g_1^x \in G_1$, the private key is marked as $sk = x$, and the public key is marked as $pk = v$.

3) Signature

A given private key *sk* is used to generate signatures on the received message *M*, the signature $\sigma = H(M)^{sk} \in G_2$. $H(M)$ represents the process of hashing the message *M*. In order to fight against forgery of signatures, the hash function input data contains the public key *pk*.

4) Verification

Given the public key *pk*, enter the message *M* and the signature σ to determine $e(g_1, \sigma) = e(pk, H(M))$ whether it is true.

Proof: The proof process is shown in formula (1).

$$e(g_1, \sigma) = e\left(g_1, H(M)^{sk}\right) = e\left(g_1^{sk}, H(M)\right) = e\left(pk, H(M)\right) \tag{1}$$

5) Aggregate signature

For the aggregated set of users, each user is given an index *i*, from 1 to $k = |U|$, and each user $u_i \in U$ generates a signature σ_i for different message $M_i \in \{0,1\}^*$. Calculate $\sigma = \prod_{i=1}^{k} \sigma_i$ and the aggregate signature is σ.

6) Aggregate signature verification

Give the aggregate signature $\sigma \in G_2$ of the aggregate users' set *U*, the original message $M_i \in \{0,1\}^*$ and the public key pk_i of all users $u_i \in U$. Verify the aggregate signature σ to make the formula $e(g_1, \sigma) = \prod_{i=1}^{k} e\left(pk_i, H(M_i)\right)$ holds.

Proof: The proof process is shown in formula (2).

$$\begin{aligned} e(g_1, \sigma) &= \prod_{i=1}^{k} e(g_1, \sigma_i) = \prod_{i=1}^{k} e\left(g_1, H(M_i)^{sk_i}\right) \\ &= \prod_{i=1}^{k} e\left(g_1^{sk_i}, H(M_i)\right) = \prod_{i=1}^{k} e\left(pk_i, H(M_i)\right) \end{aligned} \tag{2}$$

4. Scheme Design

To improve the reliability of external data on the chain and prevent a single point of failure, the scheme uses multiple oracles. The basic framework of the blockchain oracle scheme is shown in Figure 2. When smart contracts need to obtain important external data, such as asset prices in financial derivatives, currency exchange rates for real-time transactions, the flight arrival time for flight delay insurance, and important game results, obtaining data from multiple data sources can improve the accuracy and credibility of the data, but some data sources may be malicious. The existing aggregate signature algorithm cannot exclude malicious signatures. The program will filter the requested data through the commit-reveal method, exclude malicious data sources, and improve the verification efficiency. Finally, the data on the chain is aggregated and signed, which reduces the amount of data on the chain and saves costs. Each oracle can obtain the external data required by the user's smart contract from each external API. The oracle can take the form of subscription or split reward to encourage the external API to return the correct message.

This paper proposes a blockchain oracle scheme based on BLS aggregated signatures. The scheme is a completely deterministic signature algorithm, which is divided into five stages. The first stage is the initialization stage. After the oracle receives the data request, it initializes the parameters of the BLS aggregation signature. The second stage is the signature collection stage, where the oracle smart contract obtains the signature of each oracle's requesting data. The third stage is the commit-reveal stage. After the oracle smart contract obtains the commitment of each oracle for the requested data, the oracle smart contract sends a request to each oracle to obtain the plaintext of the requested data. The fourth stage is the verification stage. After the oracle smart contract verifies the validity of the signature, it screens the request data obtained from multiple oracles, and finally obtains consistent request data. The fifth stage is the aggregation signature stage. The oracle smart contract performs aggregation signature on the consistency request data. The sixth stage is the incentive stage. The oracle that submits the consistency request data in the fifth stage is rewarded. The program parameters are shown in Table 1.

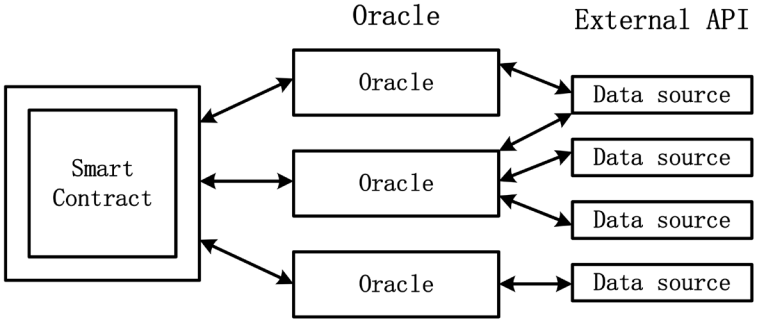

Figure 2. Distributed oracles and data sources.

Table 1. Parameters of scheme.

Parameter	Definition
O_i	The i-th oracle
pk_i	The public key of the i-th oracle
sk_i	The private key of the i-th oracle
m_i	The query result returned by the i-th oracle
σ_i	The signature of the i-th oracle
ID_i	The identification of the i-th oracle
R	The set of $2f+1$ messages received from the oracle smart contract
m	The request returned to the user's smart contract
D_k	The set of identical results that is greater than or equal to $f+1$
σ	The aggregate signature of all σ_i in D_k
I	The set of all σ_i in D_k

4.1. The Specific Scheme

There are a total of n ($n \geq 3f+1$) oracle nodes in this scheme, and only f oracle nodes are allowed to be malicious. The malicious nodes may have dishonest behaviors, such as invalid signatures or submitting wrong request data.

Step 1. Initialization

After the oracle smart contract receives the data request from the user's smart contract, it sends the data request to each oracle, where G_1, G_2, G_T are the multiplicative cyclic groups with the order of a large prime number p, the generators are g_1 and g_2 respectively, and each oracle set the private key $sk_i \in \mathbb{Z}_p$, calculate the public key $pk_i = g_1^{sk_i} \in G_1$, and send the public key back to the oracle smart contract.

Step 2. Signature collection

The oracle smart contract is sent to each oracle for query requirements. The oracle requests data from the external API and the external API returns the query data mi to the oracle. After the oracle splices m_i and pk_i, the oracle generates a signature $\sigma_i = H\left(m_i \| pk_i\right)^{sk_i} \in G_2$ and sends $c_i = \{ID_i, \sigma_i, pk_i\}$ to the oracle smart contract.

Step 3. Commit—Reveal

Assume that C is the set of information $c_i = \{ID_i, \sigma_i, pk_i\}$ collected by the oracle smart contract. When C contains $2f+1$ pieces of information $c_i = \{ID_i, \sigma_i, pk_i\}$, the smart contract generates a set $R = \{ID_i \in C\}$ broadcasts the oracles in R to return the plaintext of the query data, and the oracles that receive the broadcast R return $d_i = \{ID_i, \sigma_i, m_i\}$.

Step 4. Validation

The oracle smart contract verifies the validity of the signature σ_i returned by each oracle O_i. If the signature is valid, the oracle smart contract will be classi-

fied into different sets D_i according to the category of m_i, until the set D_k has $f+1$ or more than $f+1$ identical external data m, and the consistent data m is obtained.

Step 5. Aggregate signature

The oracle smart contract performs aggregation signature $\sigma = \sum_{i=1}^{k} \sigma_i$ on D_k with $f+1$ or more of the same external data m, sorts out the set $I = \{ID_i \in D_k\}$ of ID_i in D_k, and returns the result $res = \{m, \sigma, I\}$ to the user smart contract.

Step 6. Incentive

The smart contract rewards oracles in the I with valid signatures and correct request data.

The main flow diagram of the scheme is shown in Figure 3.

4.2. Security Analysis

In order to analyze and define security [18], suppose there is an adversary $\mathcal{A}$ who wants to forge a BLS aggregate signature. The security of the BLS aggregated signature scheme is equivalent to that there is no opponent $\mathcal{A}$ who can forge aggregated signatures within a certain game range. The existence of forgery means that attacker $\mathcal{A}$ attempts to forge an aggregated signature on a message of his choice through a group of users. In the aggregated key selection security model, adversary $\mathcal{A}$ is given the ability to challenge the public key and select other public keys. His goal is to forge the existence of the collective signature. The adversary also gains access to the signing oracle of the challenge key. The specific process is shown in Table 2. His advantage Adv AggSig $\mathcal{A}$ is defined as his probability of winning in the game.

Note that the scheme does not impact the security of the signature scheme. In other words, the security of the signature is equal to the security of the used signature scheme [19].

Adversary $\mathcal{A}$ has the ability to generate keys in the key selection model, and there is a potential multi-signature attack [20]. If the messages in the scheme are all M, the hash values are all $h = H(M) \in G_2$, and the aggregated signature is vulnerable to rogue key [21] attacks. $\mathcal{A}$ rogue key attack is an attack that uses special parameters to make the aggregated signature offset valid parameters

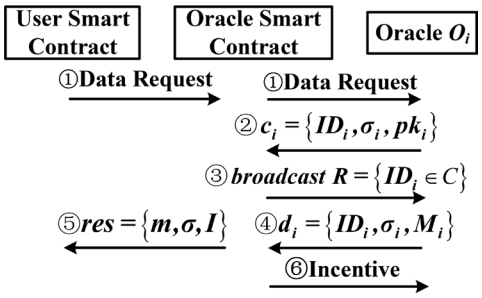

Figure 3. Schematic diagram of the main flow.

Table 2. Aggregation signature forgery attack model.

Aggregation signature forgery attack
1. Initialization. The aggregate signature forger $\mathcal{A}$ obtains a randomly generated public key pk_1.
2. Queries. Proceeding adaptively, $\mathcal{A}$ requests signatures with pk_1 on messages of his choice.
3. Response. Finally, $\mathcal{A}$ outputs $k-1$ additional public keys pk_2, …, pk_k. Here k is a game parameter, at most N. These keys and the initial key pk_1 will be included in the aggregate signature forged by $\mathcal{A}$. $\mathcal{A}$ also outputs messages m_1, …, m_k, and finally, $\mathcal{A}$ generates an aggregate signature σ that is signed by k users on their corresponding messages.
4. If the aggregate signature σ is an effective aggregation of messages m_1, …, m_k under the keys pk_1, …, pk_k, and σ is nontrivial, *i.e.*, that is, $\mathcal{A}$ did not request a signature on M_1 under pk_1, the forger wins. The probability is over the coin tosses of the key-generation algorithm and of $\mathcal{A}$.

during the aggregation process. Assuming that the public key of the honest user I is $pk_1 = g_1^{x_1}$, the malicious user II chooses $x_2 \in \mathbb{Z}_p$ and constructs the public key $pk_2 = g_1^{x_2} g_1^{-x_1}$. For any M, the malicious user II can calculate the aggregate signature $\sigma_A = H(M)^{x_2}$ and declare that it is the valid aggregate signature of users I and II.

Proof The proof process is shown in formula (3) [21].

$$\begin{aligned} & e\left(g_1^{x_1}, H(M)\right) \cdot e\left(g_1^{x_2} g_1^{-x_1}, H(M)\right) \\ &= e\left(g_1^{x_1} g_1^{x_2} g_1^{-x_1}, H(M)\right) \\ &= e\left(g_1^{x_2}, H(M)\right) \\ &= e\left(g_1, \sigma_A\right) \end{aligned} \tag{3}$$

In the scheme proposed in this paper, each oracle sets its own private key, and no private key is transmitted during the interaction. Due to the difficulty of CDH, adversary $\mathcal{A}$ cannot derive the corresponding private key from the public key. In addition, since the hash value calculated by each oracle contains the public key pk_i, each hash value is different, so rogue key attacks cannot be carried out during the aggregation of signatures.

4.3. Scheme Discussion

4.3.1. Reliability of Requested Data

The request data returned to the user's smart contract is always reliable. There are a total of $n(n \geq 3f+1)$ oracles in the scheme, allowing f oracles to be dishonest, so at least $2f+1$ oracles are honest. Among the $2f+1$ oracles received in the commit phase, at least $f+1$ oracles are honest, which is larger than f dishonest oracles, and the number of honest oracles is always greater than the number of dishonest oracles. The reveal stage can always ensure that there are greater than or equal to $f+1$ oracles that return the same correct answer. Therefore, the reliability of the data requested on the chain is ensured.

4.3.2. Validity of Aggregate Signature

The aggregated signature is aggregated from the signatures of the oracles that are honest and return the correct data. When the aggregated signature is to be veri-

fied, the public key of each oracle can be found according to the identity set $I=\{ID_i \in D_k\}$, calculated $h=H(m_i \| pk_i)$, and substituted into the formula $e(g_1,\sigma)=\prod_{i=1}^{k} e(pk_i, H(M))$ to judge whether the aggregated signature σ is valid. The public keys in all identity sets are needed for verification. Assuming there are n oracles in identity set I, $n+1$ pairing operations are required, but the aggregate signature only occupies 160 bits in the block, saving the block space.

4.3.3. Correctness of Motivation

The scheme effectively prevents free-riding attacks and ensures that dishonest oracles cannot get rewards. The commit-reveal stage effectively prevents free-riding attacks. A free-riding attack [5] refers to an attack in which a malicious oracle machine directly copies request data obtained by other oracles without spending additional costs to obtain request data and obtain rewards. In this scheme, the oracle does not summit mi in the commitment phase, but submits $c_i=\{ID_i,\sigma_i,pk_i\}$, ensuring the confidentiality of mi. After the oracle smart contract needs to verify the validity of the signature in the verification phase, the oracle will be classified according to the difference of mi in the revealing phase. If the signature is invalid, the oracle is considered dishonest, and the dishonest oracle will not appear in the reward set D_k. The commit-reveal stage ensures that the dishonest oracle is not rewarded and the accuracy of the incentive is guaranteed.

5. Experimental Analysis

The experimental test runs in the Ubuntu 18.04 environment, and the specific configuration is Intel(R) Core(TM) i5-4210M CPU @ 2.60 GHz, 4 GB memory.

In order to verify the practical value of the scheme in this paper, the experiment mainly tests the performance of BLS aggregated signatures and the performance of multiple oracles. The experiment compares different signature schemes and tests the gas consumed by different signature schemes. After the smart contract is compiled on Remix, it is deployed on the Geth node of the Ethereum private chain for testing. Besides, gas is the count of the internal workload of the Ethereum virtual machine. All transactions, execution of smart contracts, or data storage all need to consume gas. The currency used by Ethereum is ETH, and the unit of gas is wei, 1 ETH = 10^{18} wei.

Figure 4 shows the average time consumption of each phase of the BLS signature. It can be concluded that the verification phase of the BLS signature consumes the most time, followed by the signature phase, and the least time-consuming phase is the aggregation signature. The signature phase includes the initialization of the signature algorithm and the completion of the signing of the message.

The total time of a BLS signature corresponds to the main time of actual operation of a blockchain oracle. Since in practical applications, the number of oracles is directly proportional to the amount of incentives, in order to balance

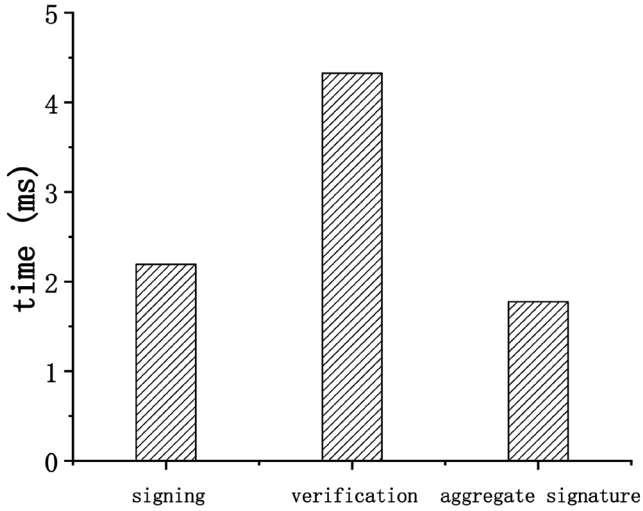

Figure 4. BLS signature average time for each phase.

Table 3. The elapsed time for each phase of BLS aggregate signature.

Number of oracles	10	20	30	40	50
Signing time (ms)	18.4	48.4	61.8	76.8	90
Validating signature time (ms)	42.1	85	131.2	176	219.2
Aggregate signatures time (ms)	14	29.8	49.5	70.4	84.5
Total signature time (ms)	74.5	163.2	242.5	323.2	393.7

costs and ensure data reliability, the scheme needs to select an appropriate number of oracles according to the actual situation. When the number of oracles is equal to fifty times, the total time is 393.7 milliseconds, which proves that the scheme has relatively high feasibility. Refer to Table 3 for detailed data on the time-consuming of each stage and total time-consuming of the BLS aggregation signature.

Figure 5 shows that the gas consumed by each scheme increases with the increase of the number of oracles, and the two show a certain linear relationship. Among them, BGLS is a BLS aggregation signature scheme with the lowest cost, but it cannot detect malicious oracles. BGR is a trapdoor permutation-based aggregate signature scheme proposed by Brogle *et al.* [22], which can be regarded as an aggregation variant of RSA. The disadvantage of BGR is that the aggregated signatures are created in sequence, which makes this scheme not suitable for application scenarios with multiple oracles. The solution in this paper is a fault-tolerant BLS aggregation signature. It is necessary to verify each BLS signature one by one before aggregation. The increase in the number of oracles will increase the number of finite field multiplications and pairing operations of two points, so the consumption of gas will be more than the BLS aggregation signature scheme. TLS-N can generate non-interactive session proofs based on blockchain smart contracts that can be effectively verified by a third party, effectively ensuring the correctness of the data on the chain, which consumes the

most gas. The data measured by the smart contract is platform-independent. It can be compiled once and can be executed in various systems. Due to the computer's scheduling, a certain degree of error will be caused, but the overall gas consumption is basically the same. The gas consumed by the scheme in this paper is between BGR and TLS-N, but BGR needs to create signatures in order [23], which is not suitable for application in multi-oracles scenarios. Although TLS-N can provide verifiable proof of the data on the chain, it consumes much more gas than the scheme in this article. Therefore, according to the gas consumed by each scheme, the fault-tolerant BLS aggregation signature blockchain oracle scheme proposed in this paper has practical value.

It can be seen from **Table 4** that the size of the BLS aggregation signature BGLS is fixed, and the size of the aggregation signature is equal to 160 bits, which has nothing to do with the number of oracles. Compared with Elliptic Curve Digital Signature Algorithm (ECDSA) signatures that cannot aggregate the signatures of the same message, BLS signature aggregation can aggregate the signatures of the same message and occupies a small space, and it is a completely deterministic signature algorithm. Because the BLS signature space is small, the gas consumed for storing BLS signatures is less than the gas consumed for storing

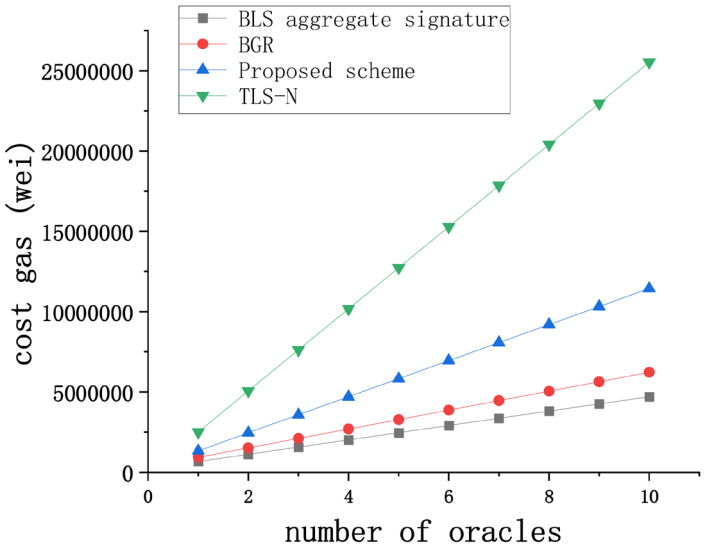

Figure 5. Gas consumption of different schemes.

Table 4. Comparison of BLS with other signatures.

	RSA	BLS	ECDSA
Aggregation feature	Yes	Yes	No
Aggregate signature	BGR	BGLS	/
Aggregate signature space (bit)	2048	160	/
Transaction cost (wei)	63,122	50,022	50,023
Storage cost (wei)	512,311	244,317	245,123
Verification cost (wei)	100,075	260,000	10,085

RSA signatures. But in terms of signature verification, the gas consumed to verify RSA signatures is less than that of BLS signatures, and the gas consumed to verify ECDSA signatures is lower. This is mainly because the cost of restoring the ECDSA public key is lower and related operations have been encapsulated into functions on Ethereum. However, since Ethereum recently accepted the improvement proposal (EIP 1108 [23]), this may significantly reduce the operating costs of BLS and BGLS. In summary, the blockchain oracle scheme that uses BLS signatures to aggregate signatures has certain advantages. Its aggregate signature space is small, which can save block space and reduce communication between blocks. In the future, the gas consumption of BLS and BGLS operations on Ethereum will be reduced.

6. Conclusions

In order to realize a large-scale blockchain-based network transaction system with scalability and reliability, the blockchain must be inseparable from off-chain data in specific application scenarios. The paper proposes a novel blockchain oracle scheme, which reduces the block space overhead and blockchain network load under the condition that the aggregated signature takes up a small space and less interaction, and ensures the consistency of the data on the chain and off the chain. In addition, the scheme can avoid single points of failure or even a small number of malicious oracles, ensuring the consistency of data on and off the chain. The experiment results show that the scheme has high practical value.

The blockchain oracle can solve the problem of blockchain access to external data. It does not destroy the decentralization of the main chain. Therefore, under the premise of privacy security, it guarantees the strong correlation of the data on the chain and off the chain, and ensures the credibility and authenticity of external data sources.

Future research work mainly includes the following three aspects:

1) We will improve the BLS aggregated signature scheme or find other aggregated signature schemes to improve the security of aggregated signatures while reducing the complexity of operations.

2) We will combine the threshold scheme to optimize the BLS signature, to further simplify the interaction between the oracle and the smart contract.

3) According to the difference and importance of the data source, we will study the blockchain oracle schemes with different fine-grained trust mechanisms.

Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

References

[1] Bonneau, J., Miller, A., Clark, J., Narayanan, A., Kroll, J.A. and Felten, E.W. (2015) SoK: Research Perspectives and Challenges for Bitcoin and Cryptocurrencies. 2015 *IEEE Symposium on Security and Privacy*, San Jose, 17-21 May 2015, 104-121.

https://doi.org/10.1109/SP.2015.14

[2] Monrat, A.A., Schelén, O. and Andersson, K. (2019) A Survey of Blockchain from the Perspectives of Applications, Challenges, and Opportunities. *IEEE Access*, **7**, 117134-117151. https://doi.org/10.1109/ACCESS.2019.2936094

[3] Zheng, Z., Xie, S., Dai, H., Chen, X. and Wang, H. (2017) An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends. 2017 *IEEE International Congress on Big Data* (*BigData Congress*), Honolulu, 25-30 June 2017, 557-564. https://doi.org/10.1109/BigDataCongress.2017.85

[4] Lin, I.C. and Liao, T.C. (2017) A Survey of Blockchain Security Issues and Challenges. *IJ Network Security*, **19**, 653-659.

[5] Beniiche, A. (2020) A Study of Blockchain Oracles. arXiv: 2004.07140.

[6] Al-Breiki, H., Rehman, M.H.U., Salah, K. and Svetinovic, D. (2020). Trustworthy Blockchain Oracles: Review, Comparison, and Open Research Challenges. *IEEE Access*, **8**, 85675-85685. https://doi.org/10.1109/ACCESS.2020.2992698

[7] Heiss, J., Eberhardt, J. and Tai, S. (2019) From Oracles to Trustworthy Data On-Chaining Systems. 2019 *IEEE International Conference on Blockchain* (*Blockchain*), Atlanta, 14-17 July 2019, 496-503. https://doi.org/10.1109/Blockchain.2019.00075

[8] Ritzdorf, H., Wüst, K., Gervais, A., Felley, G. and Capkun, S. (2018) TLS-N: Non-Repudiation over TLS Enabling Ubiquitous Content Signing. https://doi.org/10.14722/ndss.2018.23272

[9] Zhang, F., Cecchetti, E., Croman, K., Juels, A. and Shi, E. (2016) Town Crier: An Authenticated Data Feed for Smart Contracts. *Proceedings of the* 2016 *ACM SIGSAC Conference on Computer and Communications Security*, Vienna, October 2016, 270-282. https://doi.org/10.1145/2976749.2978326

[10] Peterson, J. and Krug, J. (2015) Augur: A Decentralized, Open-Source Platform for Prediction Markets. arXiv:1501.01042.

[11] Adler, J., Berryhill, R., Veneris, A., Poulos, Z., Veira, N. and Kastania, A. (2018) Astraea: A Decentralized Blockchain Oracle. 2018 *IEEE International Conference on Internet of Things* (*IThings*) *and IEEE Green Computing and Communications* (*GreenCom*) *and IEEE Cyber*, *Physical and Social Computing* (*CPSCom*) *and IEEE Smart Data* (*SmartData*), Halifax, 30 July-3 August 2018, 1145-1152. https://doi.org/10.1109/Cybermatics_2018.2018.00207

[12] Van Bulck, J., Minkin, M., Weisse, O., Genkin, D., Kasikci, B., Piessens, F. and Strackx, R. (2018) Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution. 27*th USENIX Security Symposium*, Baltimore, 15-17 August 2018, 991-1008.

[13] Lo, S.K., Xu, X., Staples, M. and Yao, L. (2020) Reliability Analysis for Blockchain Oracles. *Computers & Electrical Engineering*, **83**, Article ID: 106582. https://doi.org/10.1016/j.compeleceng.2020.106582

[14] Gorbunov, S. and Wee, H. (2019) Digital Signatures for Consensus. *Cryptology ePrint Archive*, 269.

[15] Zhang, F., Safavi-Naini, R. and Susilo, W. (2004) An Efficient Signature Scheme from Bilinear Pairings and Its Applications. *International Workshop on Public Key Cryptography*, Singapore, 1-4 March 2004, 277-290. https://doi.org/10.1007/978-3-540-24632-9_20

[16] Boneh, D., Lynn, B. and Shacham, H. (2004) Short Signatures from the Weil Pairing. *Journal of Cryptology*, **17**, 297-319. https://doi.org/10.1007/s00145-004-0314-9

[17] Boneh, D., Drijvers, M. and Neven, G. (2018) Compact Multi-Signatures for Smaller Blockchains. *International Conference on the Theory and Application of Cryptology and Information Security*, Brisbane, 2-6 December 2018, 435-464. https://doi.org/10.1007/978-3-030-03329-3_15

[18] Goldwasser, S., Micali, S. and Rivest, R.L. (1988) A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks. *SIAM Journal on Computing*, **17**, 281-308. https://doi.org/10.1137/0217017

[19] Boneh, D., Gentry, C., Lynn, B. and Shacham, H. (2003) Aggregate and Verifiably Encrypted Signatures from Bilinear Maps. *International Conference on the Theory and Applications of Cryptographic Techniques*, Warsaw, 4-8 May 2003, 416-432. https://doi.org/10.1007/3-540-39200-9_26

[20] Micali, S., Ohta, K. and Reyzin, L. (2001) Accountable-Subgroup Multisignatures: Extended Abstract. *Proceedings of the 8th ACM Conference on Computer and Communications Security*, Philadelphia, November 2001, 245-254. https://doi.org/10.1145/501983.502017

[21] Lacharité, M.S. (2018) Security of BLS and BGLS Signatures in a Multi-User Setting. *Cryptography and Communications*, **10**, 41-58. https://doi.org/10.1007/s12095-017-0253-6

[22] Brogle, K., Goldberg, S. and Reyzin, L. (2014) Sequential Aggregate Signatures with Lazy Verification from Trapdoor Permutations. *Information and Computation*, **239**, 356-376. https://doi.org/10.1016/j.ic.2014.07.001

[23] van der Laan, B., Ersoy, O. and Erkin, Z. (2019) MUSCLE: Authenticated External Data Retrieval from Multiple Sources for Smart Contracts. *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, Limassol, April 2019, 382-391. https://doi.org/10.1145/3297280.3297320