Journal of Cloud Computing:
Advances, Systems and Applications

RESEARCH

Open Access

Timed-release encryption anonymous interaction protocol based on smart contract

Ke Yuan[1,2], Zilin Wang[1], Keyan Chen[1], Bingcai Zhou[1], Zheng Li[1*] and Chunfu Jia[1,3]

Abstract

Timed-release encryption (TRE) is a cryptographic primitive that can control the decryption time and has significant application value in time-sensitive scenarios. To solve the reliability issue of nodes in existing TRE anonymous interaction schemes, we propose a blockchain-based TRE protocol for anonymous query time trapdoors. In our protocol, the recipient divides the encrypted trapdoor request information into n ciphertext fragments using secret sharing technology near the decryption time, and employs the idea of onion routing to perform layer-by-layer encryption, creating onion-type data transmitted through middlemen selected from the smart contract. After receiving the ciphertext fragments, the time server integrates them to obtain the trapdoor request information and returns the corresponding time trapdoor to the recipient. This allows the recipient to query any time trapdoor anonymously. Our protocol provides a normative design for the smart contract and specific constraints on the participants' behavior. Compared with the related anonymous query trapdoor schemes, our protocol improves the probability of successful queries. Security analysis shows that our protocol can resist release-ahead attack, interruption attack, eavesdropping attack, and replacement attack. Performance analysis shows that our protocol outperforms related protocols regarding anonymity, efficiency, and flexibility, achieving highly efficient anonymous interactions. Finally, we conducted an experiment in the Ethereum *Rinkeby* test network. For the settings of ciphertext fragment number $n = 3$ and ciphertext fragment threshold $t = 2$, the gas consumption for a user to execute the contract was \$5.66, which was higher than the contract cost of related schemes, but the contract execution cost was within an acceptable range.

Keywords Timed-release encryption, Anonymous interaction, Blockchain, Smart contract, Onion routing

Introduction

Timed-release encryption (TRE) [1] is essential for releasing sensitive data at a predetermined time. It has many application scenarios in real life, such as timing the release of online exam papers and timing the acquisition of confidential data in the cloud. The original TRE scheme relied on time-lock puzzles (TLP) [2, 3], which linked information decryption with a computation puzzle that could only be solved at a specific time. However, new methods have replaced this method because it is difficult to precisely control the decryption time due to the varying computational capabilities of different devices. Methods based on interactive time server [4] involve the recipient interacting with the server to obtain the trapdoor and decrypt the message. Methods based on non-interactive time server [5–9] involve the time server periodically broadcasting time trapdoors. As the current demand shifts towards the anonymous query of time trapdoors at any time, researchers aim to construct a secure and efficient time trapdoor query scheme. Reliability issues with intermediate transmission nodes may arise when querying time trapdoors in traditional wired networks [10]. Such intermediate nodes may

*Correspondence:
Zheng Li
lizheng@henu.edu.cn
[1] School of Computer and Information Engineering, Henan University, Kaifeng 475004, Henan, China
[2] Henan Province Engineering Research Center of Spatial Information Processing, Henan University, Kaifeng 475004, Henan, China
[3] College of Cybersecurity, Nankai University, Tianjing 300350, China

drop information transmission due to bribery by attackers or may be unable to transmit information for their own reasons. In addition, the time server may generate dishonest time trapdoors. This can result in users losing benefits due to the inability to transmit interactive information on time. Therefore, if a protocol restricts participants' behavior and rewards honest behavior while punishing dishonest behavior, it can greatly improve the stability and security of information transmission.

We consider using blockchain-based smart contract technology to solve the problems mentioned above. Applying blockchain technology [11, 12] can ensure the integrity of trapdoor requests and information during transmission. Furthermore, since all transactions in the blockchain are verified and backed up by all nodes in the network, this can increase the probability of successful trapdoor queries. Smart contracts [13, 14] was first proposed by Nick Szabo in 1994, and it is a blockchain-based, tamper-proof digital contract. Smart contracts automatically execute contract terms on the blockchain, greatly reducing disputes and controversies that may arise during contract execution due to its decentralized nature. These contracts are transparent to all participants, regulate their behavior, and provide strong support for the implementation of reward and punishment systems. In the scenario described in this article, users need to pay a certain amount to initiate the smart contract, which will allocate a private contract to each participant. During the execution process, the contract will determine whether each participant has committed a breach of contract and reward or punish them accordingly. Through contract constraints, the honesty and reliability of contract participants are guaranteed, reducing the probability of query failure. (t, n) secret sharing refers to the process of breaking down a piece of information into n shares, with only t of the n shares required to recover the information, but if there are only $t-1$ or fewer shares, the information cannot be recovered. We can use this property to solve the problem of unreliable intermediate transmission nodes. In the scenario of this article, the sender can divide the trapdoor request information into n paths for transmission. As long as t shares are successfully transmitted, the trapdoor request information can be recovered, improving the practicality of the scheme.

We have proposed a protocol for TRE called TAISC (TRE anonymous interaction based on smart contract), which utilizes smart contracts in a blockchain network to implement anonymous query time trapdoors. Our protocol operates in a wired environment. Users can hide their identities through our protocol when performing query time trapdoors. Under the constraints of the contract, participants cannot obtain the user's identity information and must honestly execute the protocol until the query is completed.

Our contribution

In this paper, we propose a scheme to solve the issues of low node reliability and user identity privacy during query time trapdoors. This paper introduces the TAISC protocol and its smart contract implementation, accompanied by security and performance analyses.

We propose a smart contract on the blockchain network that satisfies users' need for anonymous query time trapdoors using onion routing technology [15] and secret sharing technology [16]. The main contributions of our scheme are as follows:

- We use the idea of onion routing for information transmission to achieve anonymous query any time trapdoor.
- We use (t, n) secret sharing to transmit information through multiple paths, resolving the issue of path failure caused by node failure.
- We leverage the contract's constraining effect to ensure each participant's honesty and reliability, thereby reducing the probability of query failure.

Related works

The existing TRE schemes based on blockchain smart contracts [17–20] mainly utilize the blockchain to control the time of obtaining the decryption key. In 2018, Li et al. [18] proposed using a smart contract in the Ethereum blockchain to achieve timed decrypt confidential data. This approach designs an executable smart contract that uses a set of peers to transmit the decryption key to achieve timed decryption of confidential data without needing a third party time server to provide time trapdoors. In the same year, Ning et al. [19] proposed an incentive-based approach that combines threshold secret sharing with the blockchain-based smart contract to pre-distribute the secret to participants and reconstruct the secret at a designated time to achieve time-released cryptographic protocol. The protocol is based on the time-sequenced block generation of the blockchain to achieve timing control and uses the smart contract to enforce the contractual obligations of relevant parties. The above scheme does not involve trusted third parties such as time servers. However, the above scheme requires users to directly select middlemen and interact directly with them, which cannot protect the identity privacy of both parties. In addition, the scheme does not fully consider the additional query overhead caused by node failure, resulting in a certain impact on its efficiency. Secondly, the scheme requires the use of block generation time trapdoor, which is relatively less flexible. In 2022, Yuan Ke et al. proposed a scheme for anonymous querying of time trapdoors based on onion routing. The scheme is based on onion routing technology and broadcast encryption technology [21] and improves the original TRE scheme by ensuring user identity privacy when

querying time trapdoors. However, the onion routing network is currently composed of voluntary Internet users and has shortcomings in reliability.

TAISC protocol overview

This section will present the TAISC protocol model and briefly describe the involved participants, the objective, workflow, and potential attacks.

Participants

The TAISC protocol involves three participating entities: a sender, a middleman cluster, and a time server. Middleman refers to the blockchain peer(node) that provides services and registers to the contract.

Sender(*Bob*). *Bob* is the data recipient in the TRE protocol but is referred to as the sender in the following text for ease of exposition. To initiate the contract, *Bob* submits middleman selection criteria to the smart contract and pays the remuneration. *Bob* uses the public key of the time server *TS* to encrypt the trapdoor request information *T* and obtains the ciphertext *C*. Then, *Bob* divides *C* into *n* fragments and sends them simultaneously through *n* different paths.

Middleman cluster(*Peers*). A new peer(node) can register as a middleman by paying a security deposit to the smart contract to be added into the registration list maintained by the contract. The middleman cluster *Peers* is a collection of peers selected by the smart contract from the registration list according to *Bob*'s selection criteria. After *Bob* initiates the smart contract, they will receive tasks and obtain the corresponding rewards after completing tasks.

Time server(*TS*). The time server *TS* is the TAISC protocol's recipient. After receiving the ciphertext fragments of the trapdoor request, *TS* combines them to obtain the complete trapdoor request ciphertext *C*, decrypts it to obtain the time *T*, and generates the corresponding time trapdoor. The time trapdoor is encrypted using its private key and returned to *Bob*.

Protocol objective and workflow

The objective of the TAISC protocol model is to enable users to anonymously query time trapdoors. Namely, authorized users can anonymously request time trapdoors from the time server through the protocol. During the protocol execution, the time server cannot obtain the user's identity information, and third parties cannot obtain the user's identity information or transmitted content. We propose implementing the protocol by designing an anonymous query time trapdoor smart contract C_{TAISC}.

First, a blockchain peer(node) must register as a middleman by submitting its public key and other information to the contract C_{TAISC}. This information is maintained in the registration list maintained by C_{TAISC}. Close to the designated decryption time, the sender *Bob* pays the remuneration to the contract C_{TAISC} to initiate C_{TAISC} and sends a trapdoor request to the time server through the middlemen. After the participants in the C_{TAISC} contract execute the contract, *Bob* obtains the requested time trapdoor S_T. The process is as follows: *Bob* submits the middleman selection criteria to the contract C_{TAISC} to obtain the required middlemen information. Then, *Bob* divides the trapdoor request ciphertext $C = E_{TS\_pub}(T)$ into n ciphertext fragments $< C_1, C_2, ..., C_n >$ and divides the middlemen into n groups. Finally, *Bob* based on the middlemen's information to construct n onion-type data:

$$O_i = E_{OR_{pub}^{(i_1)}}(E_{OR_{pub}^{(i_2)}}(E_{OR_{pub}^{(i_3)}}(C_i, IP_{TS}), IP_{OR_{i_3}}), IP_{OR_{i_2}}), i = 1, 2, ..., n \quad (1)$$

Here, $E_{OR_{pub}^{(i_1)}}$, $E_{OR_{pub}^{(i_2)}}$, and $E_{OR_{pub}^{(i_3)}}$ represent the cipher transformations encrypted using the public keys of the middlemen at each layer; $IP_{OR_{i_2}}$, $IP_{OR_{i_3}}$, and IP_{TS} represent the addresses of the next layer middlemen; C_i represents the ciphertext fragment. After *Bob* pays the remuneration to the contract C_{TAISC}, C_{TAISC} allocates its private contract modules to the selected middlemen. After receiving the onion message, each middleman executes its private contract module. The time server combines the ciphertext fragments into the original ciphertext, then decrypts the ciphertext to obtain the decryption time *T*, generates the corresponding time trapdoor, and returns to *Bob*. This protocol for anonymous interaction between a user and a time server based on the smart contract is referred to as the TAISC protocol in this paper, as shown in Fig. 1.

The TAISC protocol involves four stages: peer registration, service setup, middleman private contract allocation, and service execution. In the following, we will briefly introduce each stage.

Peer registration. Any peer on the blockchain network can register as a middleman by paying a security deposit to the contract C_{TAISC} at any time. By submitting a registration request, the peer indicates that it can provide services to execute the contract and submits its public key, *IP* address and work window. After successful registration, C_{TAISC} adds the peer to the registration list and can be selected to execute the contract C_{TAISC}. Upon completion, the peer can get the reward.

Service setup. The sender *Bob* specifies the expected service execution time interval $[T_f, T_e]$, the required security deposit for each middleman, and the reward each middleman can get upon completing the contract C_{TAISC}. C_{TAISC} selects suitable middlemen from the registration list based on their work window and security deposit. Then, C_{TAISC} sends *Bob* the middleman cluster

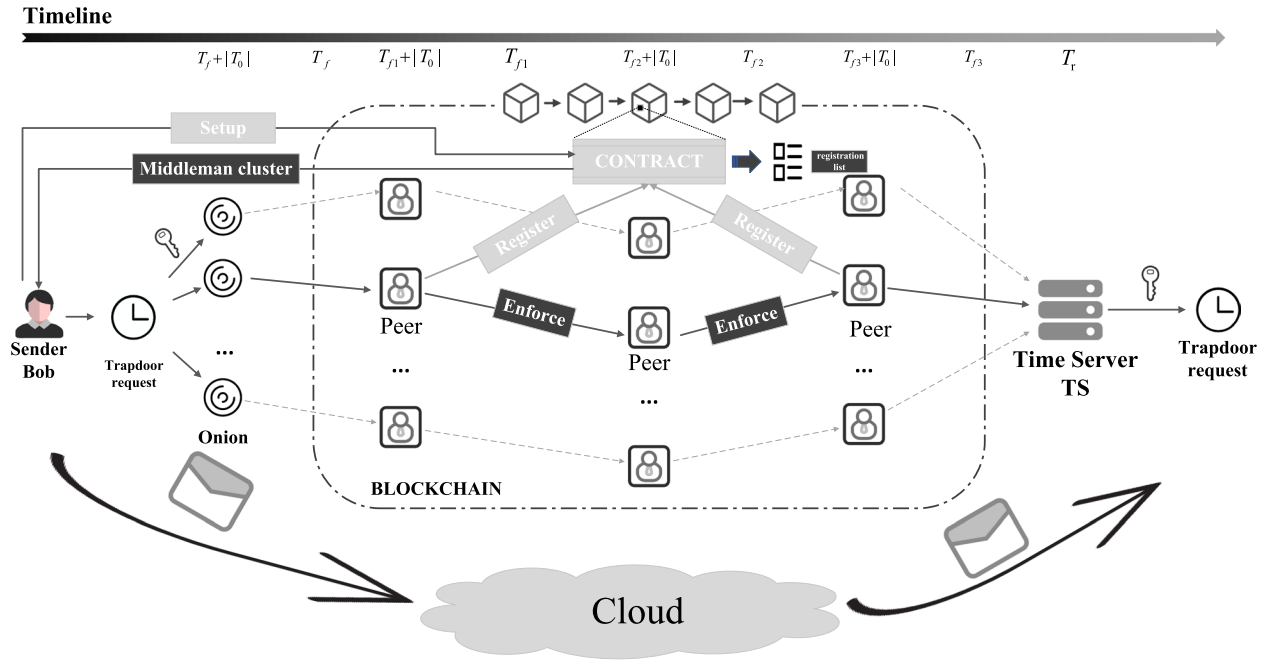

Fig. 1 The TAISC Protocol for anonymous query time trapdoors based on smart contract

$Peers = \left\{IP_i, OR_{pub}^{(i)}\right\}, i = 1, 2, \ldots, n$. IP_i represents the IP address and $OR_{pub}^{(i)}$ represents the public key of the ith middleman in the cluster.

Middleman private contract allocation. After the sender *Bob* submits the middleman selection criteria to the contract C_{TAISC}, C_{TAISC} assigns suitable middlemen and allocates each middleman with a private contract P_i. The middleman executes the contract based on P_i.

Service execution. Before the expected service execution time interval $[T_f, T_e]$, namely before time T_f, the sender *Bob* needs to divide the trapdoor request ciphertext and encrypt ciphertext fragments layer by layer using the public keys of the selected middlemen, generating n onions. After executing their respective private contracts, each middleman receives a reward distributed according to its private contract. These rewards are allocated in advance for each private contract by the contract C_{TAISC}. If a middleman fails to complete its private contract, its deposit will be confiscated.

Figure 2 shows the contract C_{TAISC} designed for implementing the TAISC protocol. In this figure, P_{x_1}, P_{y_1}, P_{z_1}, and so on are the private contracts of the middlemen.

Users can initiate the smart contract C_{TAISC} with legal authorization. When the sender *Bob* initiates C_{TAISC}, a specific remuneration must be paid.

Protocol assumptions

In the TAISC protocol, we make the following assumptions regarding the participants and the protocol execution process:

- We assume that the sender *Bob* wants his trapdoor request information transmitted quickly and on time and is willing to pay remuneration. Meanwhile, we assume that the reward that *Bob* may obtain upon successful information transmission is R.
- We assume that there are enough peers registered as middlemen.
- We assume that any adversary among the contracting parties who engage in dishonest behavior by accepting bribes is rational, and there are no costless attacks.

Attack models

In our work, we have considered four possible attacks assuming the attacker is rational: release-ahead attack, interruption attack, eavesdropping attack, and replacement attack. We will provide a detailed account of whether the TAISC protocol can withstand these attacks in Security analysis section.

Release-ahead attack: The release-ahead attack means the middleman transmits the message before the time specified by the sender *Bob*. The attacker aims to complete

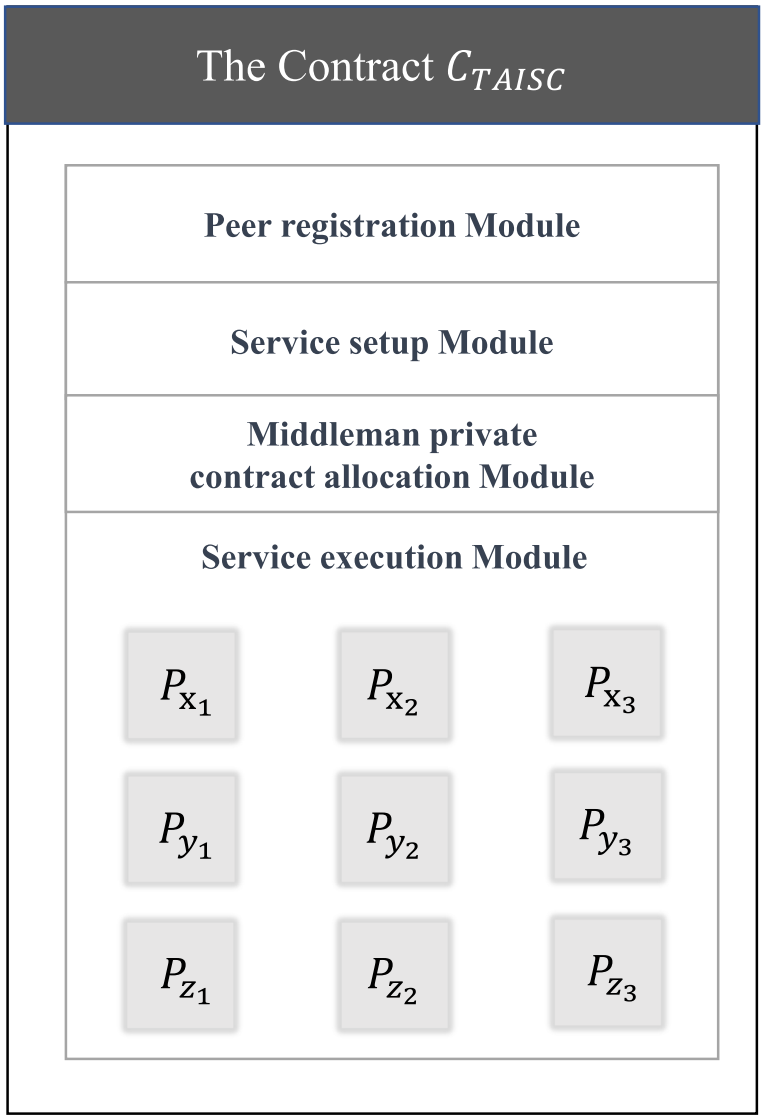

Fig. 2 The contract C_{TAISC}

the information transmission and obtain the time trapdoor before *Bob* receives the response from the time server *TS*.

Interruption attack: The interruption attack means the middleman fails to transmit the message successfully. Interruption attacks typically occur when the middleman is bribed and refuses to transmit the obtained message. Since the middleman provides a security deposit at registration, one reason for the middleman to launch an interruption attack is when the bribe offered by the attacker exceeds its security deposit.

Eavesdropping attack: The eavesdropping attack means the attacker attempts to obtain the information's content or source through eavesdropping.

Replacement attack: The replacement attack means the attacker bribes the middlemen to tamper with the content of the information and cause the information to be incorrectly transmitted, thereby disrupting the transmission of the information.

Detail of the smart contract C_{TAISC}

We utilize the smart contract C_{TAISC} to implement the anonymous query time trapdoors TAISC protocol. This section describes the four modules of the proposed contract C_{TAISC} in detail.

Peer Registration Module

The Peer Registration Module is designed for any blockchain peer that can provide services. With this module, any peer on the blockchain network can register as a middleman to the contract C_{TAISC}.

When applying to register as a middleman, each peer is required to provide the following information to the contract C_{TAISC}.

Peer Registration Provides Information

a) The peer's public key OR_{pub} and IP address.
b) The peer's circulating funds, which can serve as deposits, amount to d_c.
c) The peer's working window $[T_x, T_y]$, namely the time interval during which it can provide services. Each peer can modify its working window or give up providing services to the contract before its security deposit is frozen.

After verifying the submitted information, C_{TAISC} adds the qualified peers to the middleman registration list. A middleman's information in the registration list includes its public key, *IP*, and working window.

Service Setup module

The Service Setup module defines the tasks the sender *Bob* must complete before executing the contract C_{TAISC}.

Service Setup Module Algorithm

a) Before time T_f, the sender *Bob* calculates the required remuneration r for each middleman, the deposit d_p each middleman needs to pay, $3n$ middlemen required, and the expected service time interval $[T_f, T_e]$. Bob then submits $3n$, d_p and $[T_f, T_e]$ to the contract C_{TAISC}. The contract filters qualified middlemen information from the registration list and returns it to Bob by determining whether the registered middleman's circulating fund d_c is greater than the deposit d_p specified by Bob, and whether the time interval $[T_x, T_y]$ during which the registered middleman can provide services is included in the service time range $[T_f, T_e]$ required by Bob. Qualified middlemen are required to pay a deposit d_p to the contract.
b) At time T_f, *Bob* pays the contract C_{TAISC} remuneration $3nr$ to initiate C_{TAISC}, and C_{TAISC} allocates private contract to each middleman.

Before service setup, *Bob* uses the time server's public key to encrypt the trapdoor request information and decompose the ciphertext C obtained into n ciphertext fragments $< C_1, C_2, ..., C_n >$. After service setup, *Bob* divides the middlemen into three groups and uses their public keys to encrypt n ciphertext fragments layer-by-layer, creating n onions. As an example, the onion constructed for the first path is as follows:

$$O_1 = E_{OR_{pub}^{(1_1)}}(E_{OR_{pub}^{(1_2)}}(E_{OR_{pub}^{(1_3)}}(C_1, IP_{TS}), IP_{OR_{1_3}}), IP_{OR_{1_2}}) \quad (2)$$

The TAISC protocol uses the (t, n) secret sharing method to decompose the ciphertext into n fragments. The original ciphertext can be recovered by transmitting t or more of these fragments. The protocol transmits the trapdoor request information through n paths, and the time server *TS* can reconstruct the trapdoor request when t or more of these paths have been successfully transmitted.

Assuming the sender's ciphertext value is v, the reward generated upon successful transmission is R, and the remuneration paid to each middleman is r. The attacker's bribe to each middleman is $b_r > 0$. Given n transmission paths, the attacker should pay a total bribe amount of $b_{r\_sum} < R$. If a middleman accepts the bribe, its deposit d_p will be confiscated. Bribery can only be successful if the bribe amount exceeds the middleman's deposit d_p, namely $d_p > b_r$.

When an attacker wants to replace the sender to obtain the reward through an attack, the attacker should bribe all middlemen on at least $n - t + 1$ paths, and the bribe that the attacker should pay is $3 \times b_r \times (n - t + 1) < R$. Therefore, the deposit of the middleman only needs to meet $d_p > \frac{R}{3\times(n-t+1)}$ to ensure $d_p > b_r$. It is known that the smaller t is, the less deposit is required. However, if t is too small, it will make it easier for the attacker to launch a release-ahead attack. At this time, the attacker needs to bribe all middlemen on at least t paths, and the bribery that the attacker needs to pay is $3 \times b_r \times t < R$. Therefore, the deposit of each middleman should meet $d_p > \frac{R}{3t}$. In summary, t should satisfy $t = \frac{n+1}{2}$, and the deposit d_p should satisfy $d_p = \frac{R}{3t}$.

Middleman private contract allocation module

The TAISC protocol designs a private contract for each middleman's behavior, with the public contract C_{TAISC} ensuring the proper execution of each private contract. After receiving the message from the previous hop, the middleman will execute its private contract P_i, which consists of both forward and backward algorithms.

Middleman Private Contract Algorithm

Forward algorithm: Decrypt the received onion to obtain the next hop address and the inner onion, and then send the inner onion to the next middleman. While decrypting and sending the message, submit the certificate signed with its private key and the hash value of the sent content to the contract respectively.

Backward algorithm: Encrypt the received onion and send the encrypted onion message to the previous hop. While receiving and sending the message, submit the certificate signed with its private key to the contract.

Service execution module

During the execution of the contract C_{TAISC}, after the middlemen on each path decrypt the corresponding ciphertext fragments, they need to submit the corresponding hash values to C_{TAISC}, which will use this information to screen for problem paths after the service ends. The service execution module provides constraint items that all participants must comply with.

The specific default judgment process during contract execution is as follows:

a) Each middleman or TS has an operation time T_o. While decrypting and transmitting the message, the middleman will generate two certificates and submits them to the contract. The contract compares the difference between the submission times of the two cer-

Service Execution Module Algorithm

a) Before time $T_f + |T_o|$, the sender Bob generates the ciphertext fragments $< C_1, C_2, ..., C_n >$ and uses the public keys of middlemen along each path to encrypt the ciphertext fragments, obtaining n onions $< O_1, O_2, ..., O_n >$. Bob submits the ciphertext C and the corresponding hash values of the ciphertext fragments to the contract C_{TAISC}.
b) At time T_f, Bob sends all onions to the first layer middlemen and submits a certificate signed with his private key to the contract to ensure that the information was sent at the correct time, eliminating the possibility of sender error.
c) Before time $T_{f1} + |T_o|$, the first layer middlemen receive the onions and execute their corresponding private contracts, which involves decrypting the first layer to obtain the inner onions $< O_1^{'}, O_2^{'}, ..., O_n^{'} >$. The middlemen then submit the certificates signed with their private keys and the hash values corresponding to that layer of the onions to the contract.
d) At time T_{f1} , the first layer middlemen send the inner onions to the second layer middlemen, and submit the certificates signed with their private keys to the contract.
e) Repeat steps c and d at time T_{f2}.
f) Before time $T_{f3} + |T_o|$, the third layer middlemen execute their private contracts: decrypting the onions to obtain the inner onions $< C_1, C_2, ..., C_n >$, namely the ciphertext fragments, and submitting their hash values to the contract C_{TAISC} to verify the correctness of the ciphertext fragments.
g) At time T_{f3}, the third layer middlemen send the ciphertext fragments to the time server TS and submit certificates signed with their private keys to the contract, along with the hash values corresponding to the ciphertext fragments. TS combines the fragments and submits the hash value of the ciphertext C to the contract.
h) At time T_r, the time server TS decrypts the ciphertext using its private key and obtains the trapdoor request time T. TS generates the corresponding time trapdoor S_T based on T, and encrypts it with TS's private key. Then returns S_T to the sender Bob on one of the paths.
i) When the middleman receives the returned time trapdoor information, it executes its private contract: encrypts the information using its private key and sends the ciphertext to the previous hop middleman while submitting a certificate signed with its private key to the contract. After receiving the returned data packet, Bob uses the public keys of the middlemen at each layer of the path to decrypt it layer-by-layer and obtain the time trapdoor.
j) The contract C_{TAISC} will distribute the reward pre-assigned to each middleman's private contract to the middlemen who complete their contracts and will refund their deposits d_p. For any dishonest middlemen, their deposits will be confiscated. If the information can be successfully transmitted, the contract will not stop (the contract C_{TAISC} is robust and will not cause transmission failure due to the failure of a middleman).

tificates, if the difference is greater than T_o, the middleman is judged to be in violation.

b) The contract detects replacement attacks in the path by comparing the hash values of the ciphertexts submitted by the third layer middlemen with the hash values of the ciphertext fragments submitted by *Bob*. If the hash value of a certain path differs from what *Bob* provided, the contract judges that there is a replacement attack on that path. The contract will compare the message hash values provided by the middlemen from the back to the front to find the violating middlemen and punish them.

For dishonest middlemen, the contract will add them to a blacklist, making them unable to receive any subsequent tasks from the contract.

The TAISC protocol analysis

This section first conducts a security analysis of the TAISC protocol, then analyzes the impact of participant behavior on the TAISC protocol, and finally analyzes the performance of the TAISC protocol.

Security analysis

Firstly, we present the security model underlying the TAISC protocol. The TAISC protocol assumes that the selected middlemen and the time server are "curious but rational," meaning that while executing the contract according to the predetermined requirements, they will attempt to infer the transmitted messages and their source and destination within their capabilities. However, it is assumed that they will not intentionally alter, destroy or refuse to transmit the messages, as doing so would confiscate their security deposits.

Next, we will conduct a security analysis of the attack models in Attack models section to demonstrate that the protocol's security is sufficient to ensure that *Bob* can successfully perform a time trapdoor query while maintaining anonymity. In our security analysis, we assume that all attackers are launching attacks to disrupt the TAISC protocol, that is, to break the sender's anonymity or to prevent the sender from successfully querying the time trapdoor. We also assume that m peers are registered as middlemen in the contract.

Release-ahead attack

Each middleman must authenticate its action by submitting a signed certificate generated with its private key to the contract during the decryption and transmission of the message.If a middleman transmits the information prematurely, it will be judged as a breach of the contract, and its deposit will be confiscated during the final settlement.If the attacker wants to bribe the middleman to transmit the information prematurely, he would need to bribe the third layer middlemen on at least t paths. Since the attacker does not know the locations of the middlemen in the path, he would need to bribe a total of $3t$ middlemen on at least t paths, paying each middleman a bribe greater than its security deposit. The probability of successfully bribing middlemen for a release-ahead attack is $C_n^t \times C_m^{3t}$.

Interruption attack

If a middleman, designated as A, fails to transmit the message to the next hop middleman within the time $T_n + T_o$, or if the next hop middleman reports an attack to the contract, it will be considered a failure to transmit the message properly. Whether active or passive, it is considered an interruption attack. The contract will verify whether middleman A has been bribed to refuse to transmit the message to the next hop and whether its previous middleman has launched a replacement attack to prevent it from executing its private contract and transmitting the message.

Suppose an attacker wants to bribe the middlemen to abandon transmitting the information to achieve the information transmission failure. In that case, the attacker must successfully bribe at least one middleman in each of the $n - t + 1$ paths. The success probability of an interruption attack is $C_n^{n-t+1} \times C_m^n$.

Eavesdropping attack

This scheme can prevent eavesdropping attacks. In the trapdoor request stage, first use the time server public key to encrypt the trapdoor request information T to obtain the ciphertext, then divide the resulting ciphertext into ciphertext fragments, and finally encrypt the ciphertext fragments layer by layer according to the public key information of the middleman to construct the onion, that is, the data transmitted inside the network by T has been encrypted at least once. Without knowing the time server private key, the attacker cannot construct the requested time trapdoor. In the time trapdoor return stage, the time server generates the corresponding time trapdoor based on the request time T and encrypts it using the public key of the trapdoor requester, and then chooses one of the paths, and according to the public key information of the middlemen on this path, layers of encryption are constructed to construct an onion and return it. Therefore, similar to the trapdoor request stage, it is impossible to decrypt the message without the user's private key.

For attackers who want to eavesdrop on the information source, because each middleman only knows its

previous and next hop middlemen and does not know its position in the path, they need to bribe all middlemen on the same path. The probability of successful bribery is $C_m^n \times C_n^1 \times C_{n-1}^1 \times C_{n-2}^1$.

Replacement attack

Before the ciphertext fragments are transmitted through each path, they are submitted to the contract with their hash values. At the end of the path, the last layer middlemen submit their hash values to the contract for verification before transmitting to the time server. If the contract detects that the number of correctly transmitted ciphertext fragments is greater than or equal to t, it continues to execute. Failed paths are judged and processed after the information is successfully transmitted. If the contract detects that the number of correctly transmitted ciphertext fragments is less than t, it stops and confiscates the deposits of the paths and middlemen with transmission errors. For honest middlemen, they will receive deposit refunds and reward payments. Therefore, unbribed middlemen will execute their contracts rationally and correctly.

If an attacker wants to disrupt the message transmission by a replacement attack, he must successfully bribe at least one middleman in each of the $n - t + 1$ paths. The probability of success of the attack is the same as that of the interruption attack.

In summary, the closer m and n are, the higher the probability of the attacker's success, but it isn't easy to achieve in real life. Therefore, the TAISC protocol can resist the above attacks.

Participant behavior tree

The participant behavior tree for the contract C_{TAISC} is shown in Fig. 3. This behavior tree provides a detailed explanation of the different outcomes resulting from the different behaviors of middlemen and the corresponding gains for both middlemen and the attacker. This behavior tree assumes that the attacker can successfully bribe the middleman, ignoring the probability issues described in Security analysis section. Because the middleman behavior is consistent across all paths in the contract C_{TAISC}, only the behavior tree of a single path is shown. Let Y represents the middleman's honest execution of the contract, and N represents the middleman's acceptance of the attacker's bribe. $N_8 \sim N_{15}$ represent different outcomes resulting from the different choices of each middleman. At this point, the deposit $d_p > \frac{R}{3t}$ and bribe $b_r < \frac{1}{3}kR$ can be calculated using formula (2), where $k = \frac{1}{n}$. We take $t = 2$ and $n = 3$ obtain $d_p > \frac{1}{6}R$ and $b_r < \frac{1}{9}R$.

Among them, N_8 represents that all middlemen honestly execute the contract and receive the rewards. $N_9 \sim N_{14}$ represent one or two middlemen accepting bribes. In this case, the profit of the middleman is $b_r - d_p$. It can be inferred from $d_p > \frac{1}{6}R > b_r$ that the bribe the middleman accepts is less than its deposit. Therefore, a rational middleman would not accept the bribe. N_{15} represents three middlemen that all accept bribes. Suppose the deposit is set higher than the expected profit from the attack. In that case, a rational attacker will not launch an attack because the cost of the bribe is higher than the potential profit from the attack.

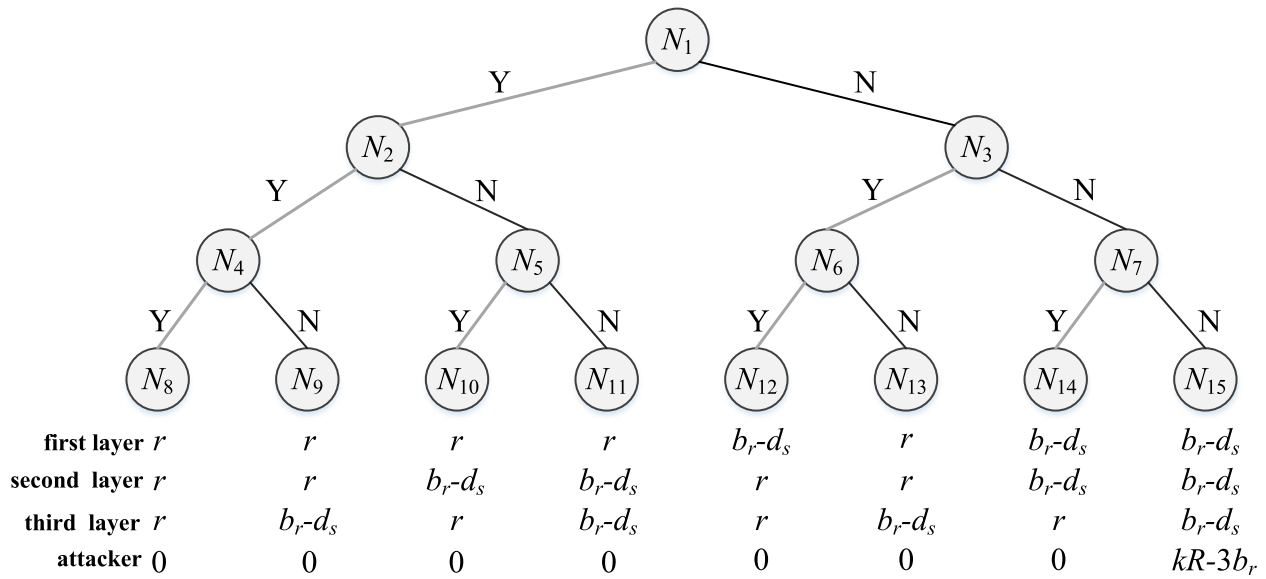

Fig. 3 The participant behavior tree for the contract C_{TAISC}

Experiment and performance analysis

The program execution environment for the contract C_{TAISC} is an Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz processor with 8GB of memory. The contract was tested in the Ethereum *Rinkeby* test network using *Solidity* language. As of June 19, 2022, the exchange rate between the test currency of Ethereum and the US dollar was 1 ETH = $949.76 [22], and the exchange rate between Ether and Gas was 1 ETH = 1×10^9 Gas [23]. Based on these exchange rates, the execution cost of the functions involved in the TAISC protocol and their conversion to US dollars are shown in Table 1.

Bob uses *setUp*() to set up the service, while a peer can register as a middleman through *newPeer*(). The contract C_{TAISC} offers incentives or penalties to middlemen through *award*(), while protocol participants submit signed certificates via *setCert*() and hash values of exchanged messages through *hash*().

In the contract C_{TAISC}, conducting one trapdoor query requires each middleman to call *setCert*() twice and *hash*() twice. Additionally, the user *Bob* needs to invoke one *setUp*(), $(n+1)$*hash*(), and one *setCert*(). The time server also calls *hash*(). Upon returning, each middleman consumes one *setCert*(). Overall, this process consumes $(6n+4) \times setCert() + (7n+2) \times hash() + setup() + award()$. In this paper, we set $n = 3$ and $t = 2$, and the gas consumption cost is $5.66.

Since no blockchain-based anonymous interactive TRE scheme exists, both DRSD [18] and TTSD [24] implement timed data encryption based on blockchain. Therefore, this paper compares the TAISC protocol with DRSD and TTSD protocols, as shown in Table 2.

Regarding the anonymity of communication, when selecting participating middlemen, users do not interact directly with them but choose from a registration list maintained by the contract. Since the contract behavior of each layer middlemen is the same, when the user distributes the trapdoor query ciphertext fragments to the first layer middlemen, the middleman cannot distinguish whether the user or other middlemen are sending the message. However, in DRSD and TTSD protocols, users cannot hide their identities because they interact directly with all middlemen. Therefore, the TAISC protocol achieves anonymous querying.

Table 1 Function execution cost

Functions	Gas cost/item	ETH cost/unit	USD ($)
newPeer	800000	0.0008	0.76
setUp	1500000	0.0015	1.42
setCert	100000	0.0001	0.09
hash	100000	0.0001	0.09
award	200000	0.0002	0.19

Table 2 Performance comparison

	TAISC	DRSD	TTSD
Anonymity	H	N	N
Efficiency	H	L	H
Flexibility	H	L	L
Gas cost	H	L	L
Communication cost	H	L	L

Regarding implementation efficiency, the TAISC protocol performs preprocessing for potential problems such as node failures during transmission. Once the behavior of trapdoor querying starts, users do not need to perform additional calculations. The trapdoor request information is constructed as a multi-layered onion, and middlemen cannot obtain specific information. In the DRSD protocol, there is only one transmission path. If a node fails, the user must reconstruct the onion for transmission, increasing the query time and possibly resulting in trapdoor information not being obtained promptly. Therefore, the TAISC protocol is more practical regarding trapdoor querying efficiency.

Regarding flexibility, different users can specify the deposit amount according to the value of the information they transmit. The computational complexity of the protocol is linearly related to the number of selected middlemen. In the TAISC protocol, the recipient can interact with the time server based on the chosen time rather than relying on block generation time, resulting in higher flexibility.

In encrypted communication, the transmitted ciphertext size can be used as a measure of communication cost, so we have counted the total ciphertext size required for transmission in the TAISC, DRSD, and TTSD protocols as communication cost. None of the TAISC, DRSD, and TTSD protocols specify specific encryption algorithms. To compare the communication costs of the three protocols more clearly, we assume that the ciphertext sizes required for transmission are all X bits. Additionally, because the main content transmitted by the TAISC protocol is the time T and time trapdoor information ciphertext, their sizes are very small, so for TAISC, DRSD, and TTSD, we uniformly ignore any changes in ciphertext size caused by encrypting or decrypting the ciphertext multiple times during transmission. For the TAISC protocol proposed in this article ($n = 3, t = 2$), in the trapdoor request stage, the trapdoor requester first encrypts the time T using the time server's public key to obtain a ciphertext of size X bits, and then divides the resulting ciphertext into three fragments of size X bits. Finally, each ciphertext fragment is transmitted to the time server through three middlemen nodes. Therefore, the

communication cost required for this stage can be represented as $12X$ bits. In the trapdoor return stage, only one path needs to be chosen to return the trapdoor information, so the communication cost is $4X$ bits. The total communication cost of TAISC is $16X$ bits. For DRSD protocol, the data sender needs to transmit a ciphertext of size X bits, and then pass it to the data receiver through three middlemen nodes successively. Therefore, the communication cost of DRSD is $4X$ bits. For TTSD protocol, the ciphertext size to be transmitted is X bits. The data sender first splits the ciphertext into fragments to obtain a total of $2X$ bits ciphertext fragments, and then sends it to the data receiver through a middleman cluster. Therefore, the communication cost of TTSD is $4X$bits. Therefore, compared to DRSD and TTSD protocols, the communication cost of TAISC is higher.

However, since the TAISC protocol uses more middlemen for trapdoor querying, it incurs higher gas costs and communication costs, which require further optimization.

Summary and future work

This paper proposes the TAISC protocol based on the smart contract for anonymous query time trapdoors, which aims to improve the reliability and stability of user query time trapdoors. Users can forcibly transmit the trapdoor request through middlemen to the time server and receive the corresponding trapdoor from the time server by executing the smart contract on Ethereum. The protocol uses onion routing technology to achieve anonymous query time trapdoors. Using secret sharing technology, the time trapdoor request information is divided into n transmission paths for transmission, increasing the probability of successful time trapdoor requests and making trap requests more stable. Using smart contracts to constrain the behavior and norms of participants improves the reliability of participants.The paper presents the protocol model, the smart contract for anonymous querying of time trapdoors, and conducts security and performance analyses for the protocol.

However, since users can anonymously interact with the time server to obtain trapdoors at any time, many users may choose to interact with the time server at their preferred times, leading to an excessive load on the time server and causing it to be unresponsive. Therefore, designing a time server enhancement scheme for anonymous query time trapdoors that can avoid denial-of-service attacks on the time server is an area for further research.

Acknowledgements
The authors express their appreciation to the National Key Research and Development Program, the National Natural Science Foundation of China,the Fundamental Research Funds for the Central Universities of China, the Natural Science Foundation of Tianjin, the Key Specialized Research and Development Program of Henan Province, the Basic Higher Educational Key Scientific Research Program of Henan Province.

Authors' contributions
Ke Yuan conceived and designed the experiments, analyzed the data, authored or reviewed drafts of the article, and approved the final draft. Zilin Wang performed the experiments, analyzed the data, performed the computation work, authored or reviewed drafts of the article, and approved the final draft. Keyan Chen conceived and designed the experiments,prepared figures and tables, authored or reviewed drafts of the article, and approved the final draft. Bingcai Zhou performed the experiments, performed the computation work, prepared figures and tables. Zheng Li put forward the main idea of the schemes, and approved the final draft. Chunfu Jia analyzed the data, authored or reviewed drafts of the article, and approved the final draft.

Funding
This work was supported by the National Key Research and Development Program under Grant 2018YFA0704703, the National Natural Science Foundation of China under Grant 61972073, 61972215 and 62172238, the Fundamental Research Funds for the Central Universities of China, the Natural Science Foundation of Tianjin under Grant 20JCZDJC00640, the Key Specialized Research and Development Program of Henan Province under Grant 222102210062 and 222102210052, the Basic Higher Educational Key Scientific Research Program of Henan Province under Grant 22A413004, the Innovation Training Program for College Students of Henan province under Grant 202310475143.

Availability of data and materials
The datasets used and/or analysed during the current study available from the corresponding author on reasonable request.

Declarations

Ethics approval and consent to participate
Not applicable.

Consent for publication
The research has consent by all authors and there is no conflict.

Competing interests
The authors declare no competing interests.

Received: 11 July 2023 Accepted: 28 October 2023
Published online: 02 January 2024

References

1. Rivest RL, Shamir A, Wagner DA (2001) Time-lock puzzles and timed-release crypto. Massachusetts Institute of Technology
2. May T (1993) Timed-release crypto. http://cypherpunks.venona.com/date/1993/02/msg00129.html. Accessed date 10 Feb 1993
3. Jia L, Jager T, Kakvi SA, Warinschi B (2018) How to build time-lock encryption. Des Codes Cryptogr 86(2):1–38
4. Yang Y, Ma M (2016) Conjunctive keyword search with designated tester and timing enabled proxy re-encryption function for e-health clouds. IEEE Trans Inf Forensic Secur 11(4):746–759
5. Yuan K, Liu Z, Jia C, Yang J, and Lv S (2013) Multi-user public key timed-release searchable encryption. 2013 Fourth international conference on emerging intelligent data and web technologies. p. 363–370. https://doi.org/10.1109/EIDWT.2013.69
6. Chan A, Blake IF (2005) Scalable, server-passive, user-anonymous timed release cryptography. In: IEEE International Conference on Distributed Computing Systems. p 504-513. IEEE, Columbus
7. Paterson KG, Quaglia EA (2010) Time-specific encryption. In: Security and Cryptography for Networks, 7th International Conference, SCN 2010, p 1-16. Amalfi, Italy, September 13-15, 2010. Proceedings. Springer-Verlag, Berlin
8. Xiong J, Li F, Ma J, Liu X, Yao Z, Chen PS (2015) A full lifecycle privacy protection scheme for sensitive data in cloud computing. Peer-to-peer Netw Appl 8(8-6):1025–1037

9. Yuan K, Wang Y, Zeng Y, Ouyang W, Li Z, Jia C (2021) Provably secure security-enhanced timed-release encryption in the random oracle model. Secur Commun Netw 2021(3):1-10
10. Kuhn C, Hofheinz D, Rupp A, Strufe T (2021) Onion routing with replies. In: Tibouchi M, Wang, H (eds) Advances in Cryptology - ASIACRYPT 2021, p 573-604. Springer, Cham
11. Ayaz F, Sheng Z, Tian D, Guan YL (2022) A blockchain based federated learning for message dissemination in vehicular networks. IEEE Trans Veh Technol 71(2):1927-1940
12. Huang J, He D, Obaidat MS, Vijayakumar P, Choo K (2021) The application of the blockchain technology in voting systems: A review. ACM Comput Surv 54(3):1–28
13. Wang J, Lu N, Cheng Q, Zhou L, Shi W (2021) A secure spectrum auction scheme without the trusted party based on the smart contract. Dig Users Dig Commun 7(2):223-234
14. Ma A, Mm A, Am A, Skk B (2021) Automatic smart contract generation for internet of media things - sciencedirect. ICT Express 7(3):274-277
15. Catalano D, Fiore D, Gennaro R (2016) A certificateless approach to onion routing. Int J Inf Secur 16(3):1–17
16. Yuan J, Yang J, Wang C, Jia X, Fu FW, Xu G (2022) A new efficient hierarchical multi-secret sharing scheme based on linear homogeneous recurrence relations. Inf Sci Int J 592:36-49
17. Liu J, Garcia F, Ryan M (2015) Time-release protocol from bitcoin and witness encryption for sat. Korean Circ J 40(10):530–5
18. Chao L, Palanisamy B (2018) Decentralized release of self-emerging data using smart contracts. In: 2018 IEEE 37th Symposium on Reliable Distributed Systems (SRDS), p 213-220. IEEE, Salvador
19. Ning J, Dang H, Hou R, Chang EC (2018) Keeping time-release secrets through smart contracts. IACR Cryptol ePrint Arch 2018:1166. https://api.semanticscholar.org/CorpusID:54200316.
20. Lai WJ, Hsueh CW, Wu JL (2019) A fully decentralized time-lock encryption system on blockchain. In: 2019 IEEE International Conference on Blockchain (Blockchain), p 302-307. IEEE, Atlanta
21. Wu Q, Qin B, Zhang L, Domingo-Ferrer J, Farras O, Manjon JA (2016) Contributory broadcast encryption with efficient encryption and short ciphertexts. IEEE Trans Comput 65(2):466–479
22. Msn. currency converter. [EB/OL]. https://www.msn.cn/zh-cn/money/tools/currencyconverter/fi-brjcfr?ocid=ansMSNMoney11&duration=1D. Accessed 19 June - 01 July 2022
23. Etherscan. transaction information. [EB/OL], https://etherscan.io/block/12965263. Accessed 19 June - 01 July 2022
24. Yuan K, Cao H, Zhang S, et al (2023) A tamper-resistant timed secure data transmission protocol based on smart contract[J]. Scie Rep 13(1):11510

Publisher's Note