Contents lists available at ScienceDirect

Computers & Security

journal homepage: www.elsevier.com/locate/cose

TrustBuilder: A non-repudiation scheme for IoT cloud applications

Fei Chen[a], Jiahao Wang[a], Jianqiang Li[a], Yang Xu[b], Cheng Zhang[b], Tao Xiang[c,*]

[a] College of Computer Science and Engineering, Shenzhen University, China
[b] College of Computer Science and Electronic Engineering, Hunan University, Changsha, China
[c] College of Computer Science, Chongqing University, Chongqing, China

A R T I C L E I N F O

Article history:
Received 13 October 2021
Revised 23 January 2022
Accepted 16 February 2022
Available online 18 February 2022

Keywords:
Non-repudiation
IoT
Cloud computing
Blockchain
Trust

A B S T R A C T

The IoT cloud computing paradigm is emerging for IoT applications. In such a paradigm, how to guarantee non-repudiation service provisioning has attracted research efforts recently. While existing solutions could work for distributed IoT cloud applications, storage, computation, and economic cost is still a practical challenging concern. In this paper, we propose a new, cost-lower non-repudiation scheme for IoT cloud applications. The proposed scheme guarantees that neither the IoT client nor the cloud could repudiate a service enjoyment and provisioning. Specifically, the proposed scheme employs a blockchain to achieve non-repudiation. First, when the cloud provides a service, it encrypts the service, stores a cryptographic hash digest of the encrypted service on the blockchain, and then sends the encrypted service to the IoT client. Second, the IoT client needs to acknowledge the hash digest on the blockchain to obtain the service. Third, the cloud sends the decryption key to the blockchain under the IoT client's public key to finish a service provisioning. We show that the proposed scheme achieves non-repudiation fairly. We prototyped, evaluated, and open-sourced the proposed scheme. Experimental results confirmed the efficiency of the proposed scheme and the speedup compared with the state-of-the-art solution.

1. Introduction

Internet of Things (IoT) has found numerous applications in modern society, ranging from smart manufacturing, smart home, smart city, etc. The number of IoT devices will dramatically increase for IoT applications in the near future due to large-scale adoption of IoT applications. Due to the limitations of local computing and storage resources, the control paradigm of IoT systems has evolved from a single vendor control to the current mainstream cloud-based control. The cloud, which has a huge computing, storage, and middle-ware pool, is an ideal choice to accommodate large-scale IoT devices. In such an IoT cloud computing paradigm, an IoT device obtains data and services from a cloud; the cloud also charges for the provision of these services.

While cloud computing brings various benefits for building IoT systems, it also brings some trust concerns (Alzubaidi et al., 2019; Elloh et al., 2021; Fan et al., 2020; Li et al., 2021; Pachala et al., 2021; Rupa et al., 2020a; Saini et al., 2021; Shi et al., 2020; Wang et al., 2020; Zhang et al., 2021; Zhao et al., 2019; Zhao et al., 2021b). On the cloud side, the cloud's service may not be dependable. For example, the cloud could have software / hardware bugs, system failures, and internal management problems. The cloud could also be attacked by attackers. Thus, for cautious users and critical systems, the IoT devices expect a trusted cloud service. On the IoT device side, due to economic incentives, it may also refuse to pay the service by claiming that it does not obtain the service from the cloud while it indeed enjoyed the provided service. We note that this claiming is reasonable in some cases. For example, in cases of cloud attacks, cloud system failures, poor network conditions, etc., the cloud service is indeed down. However, these cases should not be abused by the IoT devices.

To build trust between IoT devices and cloud service providers, researchers have been studying the non-repudiation problem for emerging IoT cloud applications by proposing an efficient non-repudiation scheme. Non-repudiation enables two folds (Feng et al., 2010; Joseph et al., 2019; Li et al., 2015; Markowitch and Roggeman, 1999; Zhou and Gollmann, 1996; Zhou and Lam, 1999). First, a cloud cannot cheat by claiming that it has provided the requested services for the IoT devices while the cloud does not. Second, an IoT device also cannot deny that it has not obtained a service from the cloud while it does enjoy the service. Thus, a non-repudiation scheme can add a trust layer for the cloud and the IoT devices.

We use an example in the healthcare area to illustrate the importance of the non-repudiation service provisioning. With the

* Corresponding author.
E-mail addresses: fchen@szu.edu.cn (F. Chen), 1910273023@email.szu.edu.cn (J. Wang), lijq@szu.edu.cn (J. Li), xuyangcs@hnu.edu.cn (Y. Xu), txiang@cqu.edu.cn (T. Xiang).

https://doi.org/10.1016/j.cose.2022.102664
0167-4048/© 2022 Elsevier Ltd. All rights reserved.

digital transformation of the healthcare industry, IoT technology and cloud services have been introduced into the hospital to help physicians better monitor and manage clinical care. For the treatment of diabetic foot ulcer (DFU) patients, an effective solution is to use IoT devices such as therapeutic shoes to jointly upload the plantar pressure data to the cloud, which provides plantar pressure analysis services. This provides physicians with treatment decisions, enables clinicians to monitor patients remotely after they leave the clinic, allows for better patient tracking, and improves care planning. However, in the event of a miscalculation resulting in amputation of a patient's limb or worse, the cloud could maliciously tamper with the data and deny the service/data provisioned. In this case, a non-repudiation service protocol (e.g., the one in this work) is both important for the client and the service provider.

1.1. Limitation of prior art

Traditional non-repudiation researches could be divided into two categories based on whether using a trusted third party (TTP). The first category is TTP-based schemes, e.g., Asokan et al. (1997); Coffey and Saidha (1996); Zhou and Gollman (1996). In this research line, TTP plays the role of the evidence generator and the evidence verifier. Once disputes occur, TTP provides evidence for arbitration.

Another category is probabilistic schemes, e.g., Markowitch and Roggeman (1999); Mitsianis (2001). This category divides the evidence into several parts and sends them to the receiver part by part. The key point is that once the receiver receives a part, the sender requires that the receiver returns an acknowledgement before sending the next part. The acknowledgement is used as a non-repudiation evidence.

In decentralized IoT environments, the traditional non-repudiation approaches are not applicable due to usability issues. The TTP-based approach relies on the fairness and justice of a TTP. However, in reality, a trustable TTP is not always available. The TTP approach is also not efficient. This is because all the communication needs to be relayed by the TTP. Furthermore, the centralized TTP has a risk of single-point-of failure and performance bottlenecks, which renders that it is not suitable for distributed IoT environments. The probabilistic approach requires multiple rounds of communication between the parties. In case of poor network conditions, the required communication is even larger. Its efficiency is a critical concern for IoT systems.

More recently, researchers have proposed to leverage blockchain to provide non-repudiation service, e.g., Aniello et al. (2016); Xu et al. (2019); Zou et al. (2016, 2018). The proposed schemes employ the blockchain as a distributed trusted third party to record the interactions between the sender and receiver. Because blockchain has the advantage of the decentralization and tamper-resistance, no single entity is able to cheat due to the consensus mechanism of the blockchain. Non-repudiation is then supported by the consensus mechanisms and smart contracts of the blockchain. It is worth noting that these solutions either store large data on the blockchain or require complicated cryptographic computations. When deployed in practice, the cost of large data storage and intensive computations are high during block mining, which is a considerable concern for practical adoption.

1.2. Proposed approach

This work aims to propose a new blockchain-based non-repudiation service-provisioning scheme for IoT cloud applications. The proposed scheme features small on-chain storage. It only stores few small hash values on the blockchain. The proposed scheme also features little computation. It does not require complicated large integer computations as in previous solutions that employ public key cryptography. Combing the two features, the proposed scheme incurs smaller cost compared with state-of-the-art works.

In a high level, the proposed scheme works in two rounds as follows. Suppose an IoT device wants to use a cloud service. In the first round, the IoT device sends a request to the cloud. To provisioning the IoT device and enabling non-repudiation, the cloud sends a "locked" version of the service to the IoT device. The cloud also sends a digest of the locked service to the blockchain. This digest serves to prevent potential cheating of the IoT device and the cloud. After the IoT device receives the locked service, it computes and checks the digest again. If matched, it sends the digest to the blockchain as an acknowledge of receiving the locked version. In the second round, the cloud sends the key that is able to unlock the service to the blockchain. To obtain the real service, the IoT device fetches the key on the blockchain to unlock the data.

In case of repudiation incidents, the proposed scheme solves repudiation as follows. First, suppose the cloud cheats by sending a wrong service to the IoT device. To solve this, the IoT device sends the arbitrator the "locked" service and the key to unlock the data. By comparing the digest of the service the cloud stored on the blockchain, the arbitrator is able to judge whether the cloud cheats. Second, suppose the IoT device cheats by denying the receipt of the service. In this case, the cloud could sends the "locked" service and its companion key to the arbitrator. By checking the service and the acknowledgement from the IoT device, the arbitrator is able to detect the cheating.

1.3. Technical challenges and solutions

From the technical perspective, the proposed scheme needs to solve two challenges. The first challenge is to enable non-repudiation and achieve fairness. The second challenge is to ensure efficiency and privacy.

The first challenge is solved by scheme design and the inherent blockchain property. First, the proposed scheme requires both the cloud and the user to "prove themselves" on the blockchain. In the first round of the proposed scheme, the cloud is required to send a digest of the "locked" service data. The IoT device is also required to reproduce the digest after receiving the data. *Because the data is locked somehow, the IoT device is not able to consume the real service.* In order to obtain the real service, the IoT device's only choice is to confirm the digest if the digest is chosen to be *collision resistant*. In the proposed scheme, we employ a cryptographic hash function to compute the digest to enable collision resistance. For the cloud, because it also needs to store the digest that will be reproduced by the IoT device, the cloud's only choice is to send the true service. Otherwise, the arbitrator is able to detect the wrong service using the service data provided by the IoT device and the on-chain digest. Thus, the two proofs bind both the cloud and the IoT device on the blockchain in the sense that they cannot deny their services. Technically, we use an encryption scheme to "lock" the service data and store the proofs in a smart contract provided by the arbitrator. In the second round, the cloud sends the key to the IoT client to unlock the service.

Besides the on-chain proof, the proposed scheme relies on tamper-resistance of the blockchain. This is because the digest is critical to arbitrate potential repudiation incidents. Blockchain is maintained distributively by many miners using a consensus mechanism. No single entity, either the IoT device or the cloud, is able to modify the digests on the blockchain. This ensures the dependability of the digests. Combined together with the two round scheme design, the proposed scheme achieves non-repudiation.

Table 1
Comparison with related work.

Protocol	Computation cost	Storage cost	Economic cost	Data privacy	Blockchain platform	Opensource evaluation	Publication year
This work	light	light	light	yes	any	yes	/
Wang et al. (2021)	median	light	light	yes	customized	no	2021
Ersoy et al. (2021)	light	light	light	no	any	yes	2021
Delgado-Segura et al. (2020)	median	heavy	median	yes	customized	no	2020
Xu et al. (2019)	median	light	median	no	any	no	2019
Aniello et al. (2016)	light	heavy	heavy	yes	any	no	2016
Zou et al. (2016)	light	heavy	heavy	no	customized	no	2016

The second challenge is solved by data separation and encryption. First, considering the storage characteristic and performance of the blockchain, saving the service data on-chain directly results in low efficiency and cause privacy concerns. This is because all miners needs to compute and store the data to achieve consensus. Large data size degrades blockchain performance and increases mining cost. The proposed scheme sends the requested service off-chain and only requires to record the hash digest on-chain. Second, storing the decryption key on a public blockchain may leak data privacy. The proposed scheme uses public key encryption to protect the decryption key. In the second round, the cloud uses the public key of the IoT device to encrypt the decryption key. The IoT device fetches the ciphertext of the key, decrypts it, and then unlocks the real service.

1.4. Summary of experimental results

We prototyped and opensourced the proposed scheme (Wang, 2021). We deployed the smart contracts on Ethereum test net using a PoA consensus mechanism and measured the performance. In a setting of 2000 transactions per minute, the blockchain grows 2.86 megabytes per hour. For a service provisioning process, it cost 148,161 units of gas in total. For a 10 megabyte service program, the IoT client takes 8919.05 ms to obtain the service. The experimental results show that the proposed scheme is efficient and promising for practical use.

1.5. Contribution

Table 1 shows the comparison of the proposed scheme with the state-of-the-art schemes. The main advantage of the proposed scheme lies in its efficiency due to simpler design. The proposed scheme incurs light-weight cost for computation, storage, and economic gas consumption. It also features data privacy protection, blockchain platform independence, and opensource evaluation. Section 5 will discuss detailed comparison later.

In summary, the contribution of the paper is as follow:

- Efficient construction. We propose an efficient blockchain-based non-repudiation service-provisioning scheme for IoT cloud applications. It only requires small on-chain storage cost and little computation cost. It also protects user data privacy.
- Solid analysis and evaluation. We analyzed all potential repudiations and showed the corresponding resolution handling mechanisms. We also prototyped and opensourced the proposed scheme. Theoretical and experimental results also validated the performance of the proposed scheme.

1.6. Organization of the paper

The remainder of this paper is structured as follows. Section 2 reviews related works on non-repudiation schemes. Section 3 presents a model of non-repudiation service provisioning for IoT cloud applications. Section 4 presents the proposed non-repudiation service-provisioning scheme and the dispute resolution mechanism. Sections 5 and 6 analyze the proposed scheme both theoretically and experimentally. Finally, Section 7 concludes the paper.

2. Related work

We categorize existing researches on non-repudiation schemes into three groups, i.e., trusted third party (TTP) based approach, probabilistic approach, and blockchain based approach. The first two approaches are traditional ones and the last approach is studied more recently. We review them respectively.

2.1. Trusted third party based approaches

This line of research uses a trusted third party to oversee communication processes and support later arbitration. Coffey and Saidha (1996) proposed a TTP-based scheme for the general non-repudiation problem. In their proposed scheme, the TTP is used as the evidence collector and provider between the service provider and client. The fundamental feature of this scheme is that both the provider and the client have to submit their digitally signed evidence to the TTP. In such a manner, neither the service provider nor the client can deny their participation in the exchange.

To reduce the overhead of TTP, Zhou and Gollman (1996) proposed an online-TTP-based scheme. This scheme also requires a TTP but attempts to minimize its involvement in the execution of the protocol. An online TTP does not need to participate in every step of the data transmission; participants can exchange information directly. However, the TTP does need to involve in every round of information exchange. To further reduce the workload of TTP, Asokan et al. (1997) proposed a general scheme based on an off-line TTP. In this scheme, the TTP is activated only when one dishonest entity makes inappropriate behavior or network errors happen.

2.2. Probabilistic approaches

This line of research relies on a lot of communication between service provider and client. For example, Markowitch and Roggeman (1999) proposed a probabilistic scheme to solve the general non-repudiation problem. This scheme is based on the secret knowledge of the number of steps for the protocol. The service provider sends the encryption data to the client and then sends fake keys for multiple iterations. The true key is at the nth iteration, where n is determined by the service provider randomly. For each iteration, the client does not have enough time to verify whether the received key is authentic. The service provider can terminate the protocol if it does not receive the client's evidential receipt immediately. This protocol requires n to be a sufficiently large number to ensure fairness with a certain probability.

A similar protocol was proposed by Mitsianis (2001). The service provider splits the ciphered key in n parts of different random lengths and then sends them for n iterations. The major difference between the protocol in Markowitch and Roggeman (1999) and the described protocol in Mitsianis (2001) is whether the number of protocol's rounds is fixed. Analogously, the number of transmissions is still significant to provide fairness with an adequate probability.

2.3. Blockchain based approach

This line of research is more recent. It uses blockchain to record the non-repudiation evidence between communication processes. Blockchain is essentially a distributed database of records (Crosby et al., 2016; Manski, 2017; Saini et al., 2021; Zhao et al., 2021a), which has many attracting feature, e.g., tamper-resistant, undeniable, and globally unique. Especially, each transaction in the public ledger can never be changed by a small group of users, which is suitable to solve non-repudiation problems in service-provisioning scenarios.

Zou et al. (2016) proposed a peer-to-peer non-reputable service contract management scheme (SCMS) based on blockchain. This scheme requires service provider to publish service contracts on the blockchain. It then requires clients to submit receipt evidence for the corresponding services to prevent repudiation. To solve the limitations of existing consensus protocols in public or consortium/private blockchain, Zou et al. (2018) developed a new consensus protocol based on SCMS, called Proof-of-Trust consensus, which utilizes a widely used trust mechanism in the online service industry. However, these schemes (Zou et al., 2016; 2018) only guarantee a weak fairness because it does not consider the existence of a malicious client. For example, a dishonest client may refuse to publish evidence or publish incorrect proof on the blockchain.

To ensure true fairness, Aniello et al. (2016) proposed a log-based dispute resolution solution for multi-party transactions. This scheme records all signed service events on the blockchain as non-repudiation evidence. Xu et al. (2019) proposed a blockchain-based non-repudiation scheme for untrusted and distributed IIoT scenarios. The proposed scheme divides a requested service into two segments, i.e., a small fragment S_1 for on-chain evidence and a major part S_2 for off-chain delivery. Moreover, the scheme designs a homomorphic-hash-based service verification method to enable non-repudiation using on-chain evidence only. This scheme assumes that the initial hash value of the service stored on the blockchain is trusted. In case of a cloud's non-repudiation, the scheme does not work. When an IoT client receives a large portion of the data and refuses to proceed, data privacy is leaked. The computational overhead of the homomorphic hash algorithm is also a practical concern.

More recently, Wang et al. (2021) proposed a blockchain-based two-stage protocol for non-repudiation data delivery using a customized consensus mechanism. In this scheme, the data is also divided into two chunks. The difference is that the receiver is the last to obtain the second chunk that is to be verified by other blockchain nodes. Ersoy et al. (2021) proposed a non-repudiation scheme for digital goods exchange. This scheme divides the data into multiple blocks and encrypts each block using a different key. The encryption keys are derived using a master key in a controlled way. After the receiver checking selected data blocks, the sender publishes the master key on the blockchain to finish data exchange. Delgado-Segura et al. (2020) also used the idea of dividing the data into multiple pieces to support non-repudiable data trading. This scheme encrypted each block using a common public key. Later, the scheme used the k-value repeat vulnerability of the ECDSA signature scheme to enable the receiver to deduce the key. However, this scheme relies on characteristics of the Bitcoin blockchain system.

To sum up, although a variety of non-repudiation schemes have been proposed, they face different practical concerns. For distributed IoT cloud applications, it is difficult to set up a trusted third party that is neutral to every entity. The trusted third party is also a single point of failure. Probabilistic approaches require multiple rounds of communication. Efficiency is an issue; fairness could also be improved. Blockchain based approach is recent research effort. We aim to improve this approach in this work.

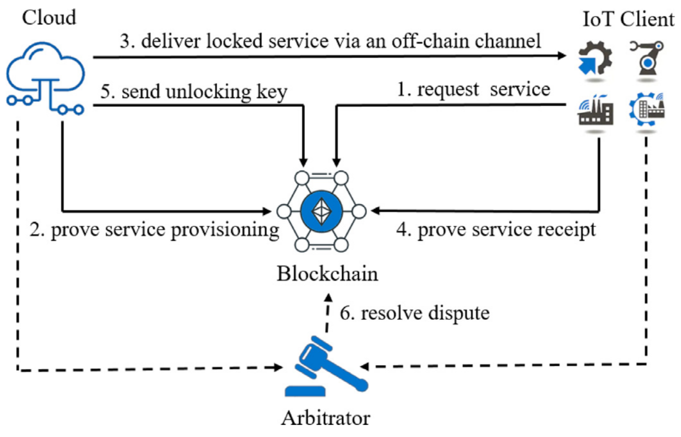


Fig. 1. System model.

It is also worth noting that researchers are leveraging blockchain to support other IoT/cloud related applications, besides the non-repudiable data trading in this work. For example, the scheme in Rupa et al. (2020c) proposed to use blockchain to address the issues of data privacy and data preservation in IoT Devices, UAVs, and Drone applications. The scheme in Rupa et al. (2020b) leveraged the immutable property of the blockchain to improve data privacy and portability for the cloud integrated IoT architecture. The scheme in Rupa et al. (2021) used blockchain to manage medical certificates. Interested readers may also refer to recent surveys (e.g. Chen et al., 2020) for more comprehensive blockchain applications.

3. Proposed model

3.1. System model

Fig. 1 shows the proposed system model for non-repudiable service-provisioning in IoT cloud applications. The model has four entities, namely, the cloud service provider, the IoT client, the blockchain, the arbitrator. The system works as follows. During setup, the cloud service provider and the IoT client connects to the blockchain by running a node in the blockchain network. They also agree on a service provisioning smart contract on the blockchain, which is provided by the arbitrator. The smart contract records the interactions between the cloud and the IoT client during service provisioning. When the cloud and the IoT client have disputes, the arbitrator employs the smart contract to judge which entity cheats.

When the system setup is ready, the IoT client could request service from the cloud. The service is modeled as a data object; for example, it could be an executable program. To request the service, the IoT client sends a service request transaction to the smart contract in the blockchain. The cloud that is listening to the blockchain observes the request and then prepares to send the service to the IoT client. The cloud first sends some data to the smart contract as a promise of service provisioning. Then the cloud sends some version of the data to the IoT client off-line, which contains sufficient information of the service but also makes the user unable to obtain the real data before the IoT client's acknowledgement. After receiving the data from the cloud, the IoT client sends an acknowledgement to the smart contract to indicate that it does receive the service. Finally, the cloud sends a short information to the smart contract, using which the IoT client is able to recover the real service. All these interactions are stored on the blockchain using the blockchain's consensus mechanism. The arbitrator also uses the smart contract to resolve disputes between the cloud and the IoT client.

Formally, we abstract a non-repudiation scheme as six phases as follows.

1. `request_service`. The IoT client sends a service request to the service provider using the blockchain.
2. `promise_service`. The cloud sends a service promise to the blockchain.
3. `send_masked_service`. The cloud sends the requested service to the IoT client via an off-chain channel.
4. `confirm_receipt`. After receiving the masked service via the off-chain channel, the IoT client sends an acknowledgement to the blockchain to indicate that it received the masked service.
5. `send_key`. The cloud sends the key that is able to unmask the service to the blockchain. The IoT client fetches this key to obtain the real service.
6. `resolve_disputes`. The arbitrator uses the interaction records in the smart contract to resolve any dispute for the cloud and the IoT client.

3.2. Design goals

To achieve non-repudiation, the proposed scheme has two design goals. First, the proposed scheme should be correct. If the cloud and the IoT client are both honest, the IoT client should indeed obtain the requested service. No disputes should occur. Second, the proposed scheme should be secure. If the cloud denies service provisioning, the arbitrator should resolve this cheating after the IoT devices initiates an arbitration request. Similarly, if the IoT device denies receiving a service, the arbitrator should also find out this cheating on the cloud's arbitration request.

3.3. Threat model & assumptions

We model both the service provider and the IoT client as untrusted. The cloud service provider may cheat. It may deny sending the requested service. The cheating may be caused by management issues, internal technical failures, and external system attacks. Analogously, the client may also cheat for its own benefit, denying the correctness of the service or denying having received the program maliciously.

To make the presentation compact and concise, we make some assumptions that sounds in practical systems. These assumptions help to keep our focus on the non-repudiation protection. We assume that a secure blockchain infrastructure is available. No single entity is able to control, modify the blockchain.

One important assumption We assume that the cloud is not able to send a nonsense data to the IoT device. This could be achieved using other methods. For example, one may use a signature scheme. When the IoT system administrator deploys the service on the cloud, the administrator signs each service uniquely. The signature binds each service ID and the corresponding service data as a whole. In this way, the IoT client and the arbitrator is able to detect it if the cloud indeed sends a nonsense data. For another example, one may rely on an arbitrator who has background knowledge. The arbitrator understands whether a service is nonsense. In the following, we use a signature scheme as an example to present the proposed scheme.

4. Detailed scheme

We now present our approach to solve the non-repudiation service-provisioning problem for IoT cloud applications using blockchain. We first introduce our high-level idea. Then, we describe all the details of the proposed scheme.

4.1. High-level idea

To enable non-repudiation, the main idea is to require the cloud and the IoT client make public commitments on the data they send and receive. These public commitments are stored in the blockchain, which are trusted by all parties. In case of disputes, the arbitrator uses the public commitments to judge the cloud and the IoT client.

Specifically, to request a service, the IoT client sends the request to a smart contract that is run by the arbitrator. The cloud uses two steps to serve the IoT client. First, the cloud encrypts the service and computes a collision-resistant cryptographic hash digest of the encrypted service. The cloud also sends the cryptographic digest to the smart contract. Then, the cloud sends the encrypted data to the IoT client. The encrypted data is a kind of "locked" version of the real service. The cloud requires the IoT client to acknowledge the receipt of the encrypted service. The cryptographic digest of the encrypted service on the blockchain serves as a promise of the cloud that the cloud will send the data to the IoT client.

After the user receives the encrypted data, the user computes the cryptographic hash digest of the received data to validate whether the cloud sends the same data as promised on the blockchain. If the two digests match, the user needs to send a confirmation that the encrypted data is received in order to obtain the decryption key of the encrypted service. If they do not match, the IoT client aborts the running. Only after the cloud finding the user's acknowledgement on the smart contract, the cloud enters the second step.

In the second step, the cloud sends the decryption key to the smart contract. Because the smart contract on the blockchain is public information, the IoT client also obtains the decryption key. To further protect the privacy of the decryption key, the cloud encrypts the decryption key using the IoT client's public key before sending it to the blockchain. Then only the IoT client is able to decrypt the service.

We explain the intuition why the cloud and the IoT client cannot cheat. Suppose the cloud cheats by sending a wrong data randomly. The IoT client could decrypt the received key and gives both this key and the encrypted data to the arbitrator. The arbitrator first checks whether the encrypted data is the same as promised by the cloud using the on-chain cryptographic hash value. Once this step passes, the arbitrator further checks whether the decrypted service is correct. As we have assumed in Section 3.3, the ID and the content of the service is well protected using a public signature scheme of the IoT cloud application administrator. By checking the signature, the arbitrator is able to detect the cloud's cheating.

Suppose the IoT client cheats. After receiving the decryption key, the IoT client refuses service payment and says that the cloud sends a wrong data. Note that the IoT client has acknowledged the receipt by sending a confirmation digest in the blockchain. The cloud could show the encrypted data and the encryption key to the arbitrator. By checking the cryptographic digest of the encrypted data and the signature of the service, the arbitrator is able to detect the IoT client's cheating.

4.2. Notations

Before discussing the details, we present some notations that help to explain the proposed scheme. Let S denote the service program/data. Denote the IoT client as C and the service provider as SP. The proposed scheme uses both symmetric encryption and asymmetric encryption. Denote the symmetric encryp-

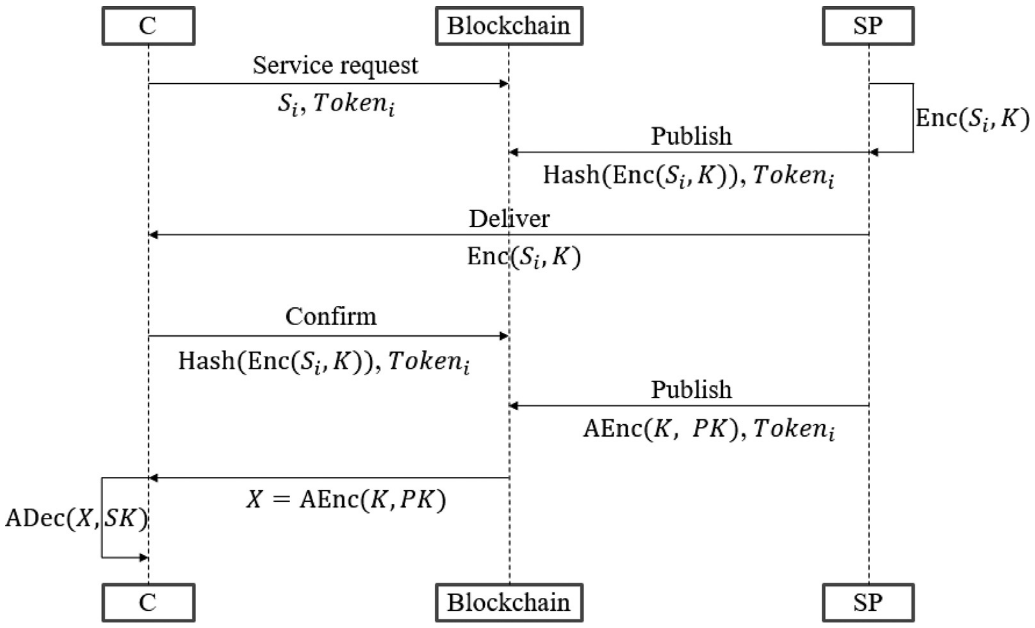


Fig. 2. Scheme summary.

Table 2
Notations.

Symbol	Meaning
S	the service program/data
C	the IoT client
SP	the service provider
PK, SK	IoT client public/private key pair
X	a message
$\text{Enc}(X, K)$	symmetric encryption of X using K
$\text{Dec}(X, K)$	symmetric decryption of X using K
$\text{AEnc}(X, PK)$	asymmetric encryption of X using PK
$\text{ADec}(X, SK)$	asymmetric decryption of X using SK
$\text{Hash}(X)$	cryptographic digest of X using Hash

tion algorithm as $\text{Enc}(\cdot), \text{Dec}(\cdot)$ and asymmetric encryption algorithm as $\text{AEnc}(\cdot), \text{ADec}(\cdot)$. Denote the IoT client's public encryption/decryption key pair as (PK, SK). Denote a collision resistant cryptographic hash function as $\text{Hash}(\cdot)$. For quick reference, we list all of the used notations in Table 2.

4.3. Detailed scheme

Fig. 2 gives an overview of the proposed scheme. In the following, we present all the details of the proposed non-repudiation scheme.

Our scheme is built upon a blockchain with a consensus mechanism. Either a public blockchain or a consortium blockchain is suitable for the proposed scheme conditioned that the blockchain is tamper-proof. When the proposed system initializes, each entity (i.e., the cloud, the IoT client, and the arbitrator) has an account on the blockchain. The arbitrator initializes a non-repudiation-supported smart contract to record all future service interactions between the IoT client and the cloud. To request/send service, the IoT client/cloud will trigger smart contract invocations in the form of blockchain transactions. After consensus, the transaction is eventually packaged in blocks and published on the blockchain as immutable evidence.

Phase 1 `request_service`. The client C triggers the smart contract and calls the service request method of the contract. The data field of the transaction mainly contains the service identity S_i and a request identity $Token_i$. The service identity S_i uniquely defined a service on the cloud. The $Token_i$ is initiated by C to uniquely identifies a service request on the blockchain and will be used by both the IoT client C_i and the cloud SP to identify the subsequent service steps.

Phase 2 `promise_service`. On a service request from the IoT client, the cloud SP runs Algorithm 1 to process the request.

Algorithm 1 Promise_service.

```
Input: $S_i$, i.e., the requested service
Output: the digest of the encrypted $S_i$
1: choose a secret key $K$
2: encrypt the service to get $\text{Enc}(S_i, K)$
3: compute a collision resistant hash digest of the encrypted service $\text{Hash}(\text{Enc}(S_i, K))$
4: send the digest to the smart contract as a promise for the requested service
5: return $\text{Hash}(\text{Enc}(S_i, K))$
```

The cloud takes two steps, i.e., encryption and hashing. First, the cloud chooses a secret key K of a symmetric encryption algorithm (e.g., AES). The cloud then encrypts the requested service S_i to get $\text{Enc}(S_i, K)$. Second, the cloud uses a collision-resistant hash function Hash (e.g., SHA256) to compute a cryptographic digest of the encrypted data, i.e., $\text{Hash}(\text{Enc}(\text{S}_i, \text{K}))$.

After the digest computation, the cloud invokes the smart contract. This encapsulates the cryptographic digest with $Token_i$ used in the smart contract invocation transaction. Note that the proposed scheme only records a small hash value of the data on the blockchain. Compared with recording the whole service data, this is a lightweight evidence.

Phase 3 `send_masked_service`. After publishing the collision resistant cryptographic hash digest of the encrypted service on the blockchain, the cloud also sends the encrypted service $\text{Enc}(S_i, K)$ to the IoT client via an off-chain channel. To obtain the real service program/data, the cloud needs to decrypt the encrypted data. However, the secret key K is currently kept by the cloud. This arrangement will force the IoT client to publish an acknowledge of receiving the encrypted data. This provides fairness because the IoT client cannot decrypt the data without explicitly publishing an acknowledgement. Additionally, the off-chain transmission reduces the storage burden of the blockchain and improves the practical feasibility of the proposed scheme. The encryption also ensures the privacy of the service program/data.

Phase 4 `confirm_receipt`. To obtain the secret key, the IoT client has to confirm the received encrypted data. Algorithm 2 lists

Algorithm 2 Confirm_receipt.

```
Input: the received encrypted service $\mathrm{Enc}(S_i, K)$, request ID $Token_i$,
    and the cloud's on-chain promised digest $d$
Output: a confirmation on the blockchain or abort
1: compute $d' \leftarrow \mathrm{Hash}(\mathrm{Enc}(S_i, K))$
2: if $d' != d$ then
3:     the client aborts the service request
4: else
5:     the client confirms $(d', Token_i)$ on the smart contract
6: end if
```

this phase. The IoT client runs as follows. After receiving the encrypted $\mathrm{Enc}(S_i, K)$ via the off-chain channel, it first uses the same collision-resistant cryptographic hash function to calculate its digest. It compares the recomputed digest with the one the cloud promises on the blockchain. If the two digests do not match, the IoT client aborts and reports this incident. Otherwise, the IoT client invokes the smart contract to acknowledge the digest of the received encrypted data. Specifically, the invocation encapsulates the recomputed digest $\mathrm{Enc}(S_i, K)$ and the $Token_i$ in a transaction.

We discuss more on a mismatch for completeness. Many factors could cause a mismatch between two hash digests. For example, this could be due to incorrect data delivered by a dishonest service provider. This could also be due to network transmission error. The cloud could also intentionally send a wrong digest to the blockchain to deny a service. In any case, the IoT client and the smart contract need to check the two digests. When mismatch occurs, the IoT client and the smart contract could terminate the current service using ID $Token_i$. This avoids the possibility of subsequent disputes and further losses. Additional, the cloud can also terminate the service by invoking the smart contract if there is no response from the IoT client within a maximal time.

Phase 5 `send_key`. Once the cloud finds that the IoT client commits the same digest, the cloud proceeds to this phase. Otherwise, the cloud aborts this round of service provisioning. The cloud uses an asymmetrically encryption algorithm to encrypt the secret key K. Note that this K is used in phase 2 to encrypt the service that is later transferred to the IoT client off-chain. The encryption result is $\mathrm{AEnc}(K, PK)$, where PK represents the public key of the IoT client. Then the cloud sends this decryption key as well as the service request ID $Token_i$ to the blockchain by invoking the smart contract.

Note that the *encrypted* secret key is published on the blockchain as lightweight evidence for possible disputes in the future. Only the client who has the corresponding private key for decryption is able to obtain the secret key K. Thus, service privacy and access control are also assured.

4.4. Dispute resolution

Phase 6 resolves potential repudiation incidents. We first address repudiation from the cloud; we then handle the client's repudiation. As we have assumed, the cloud and the IoT client both follow the running of the proposed scheme. They finish all the 5 phases running. Otherwise, using a timing mechanism, this round of service request is terminated. Thus, we consider potential repudiation after finishing all the 5 phases.

For the cloud, cheating occurs when it sends a wrong service (e.g., a different service or a non-sense random data) that is not requested by the IoT client. Following the running flow of the proposed scheme as in Fig. 2, the cloud's repudiations much lie in the followings.

1. The cloud denies having send $\mathrm{Enc}(S, K)$ to IoT client.
2. The cloud denies having published $\mathrm{AEnc}(K, PK)$ on the blockchain.
3. C claims that it fails to use SK to decrypt $\mathrm{AEnc}(K, PK)$ while the cloud denies that.
4. C claims that it fails to use K to decrypt $\mathrm{Enc}(S, K)$ while the cloud denies that.
5. C claims that it has obtained an un-executable service program S while the cloud denies that.

When the cloud repudiates, the IoT client collects evidence and asks the arbitrator to judge for this repudiation incident. The arbitrator could also ask the cloud to assist in the judging process occasionally. For case 1, the client gives the received $\mathrm{Enc}(S, K)$ to the arbitrator. The arbitrator uses the public cryptographic hashing function to compute the hash digest. If the hash digest is consistent with the one the cloud promised in the smart contract, the cloud did cheat. This is because the cryptographic hashing function is collision resistant and the IoT client is not able to find a preimage that has the same digest as the cloud promises. For case 2, the arbitrator just checks the existence of $\mathrm{AEnc}(K, PK)$ in the smart contract. Because blockchain is tamper-proof and the five phases of running are finished, the arbitrator will detect this.

For case 3, two methods exist. First, the IoT client could present the secret key to the arbitrator. The arbitrator validates it using the corresponding public key. Then the arbitrator decrypts $\mathrm{AEnc}(K, PK)$ to check whether the cloud cheats. Second, the arbitrator could ask the cloud to present K and then checks whether $\mathrm{AEnc}(K, PK)$ is correct. For case 4, similar to case 3, the IoT client gives its secret key and $\mathrm{Enc}(S, K)$ to the arbitrator. The arbitrator checks whether $\mathrm{Enc}(S, K)$ has the same digest as the cloud promises in the smart contract. If yes, the arbitrator runs the decryption process to check whether K can decrypt $\mathrm{Enc}(S, K)$. The arbitrator could also ask the cloud to provide the secret key K make the checking. For case 5, the IoT client gives the decryption key K and $\mathrm{Enc}(S, K)$ to the arbitrator. The arbitrator checks whether K is correct and $\mathrm{Enc}(S, K)$ is correct by comparing its hash digest with the on-chain one. If they are all correct, the arbitrator decrypts $\mathrm{Enc}(S, K)$ using the K. By checking whether the decrypted service is authorized by the system administrator using a signature scheme, the arbitrator is able to check whether the cloud cheats.

Similarly, the IoT client cheats when denying an obtained service while it did enjoy the requested service. According to the running flow, the IoT client 's repudiations much lie in the followings.

1. SP The IoT client denies receiving $\mathrm{Enc}(S)$.
2. SP The IoT client denies receiving $\mathrm{AEnc}(K, PK)$.
3. C The IoT client claims that it fails to use SK to decrypt $\mathrm{AEnc}(K, PK)$ while the cloud did encrypt the key using the IoT client's public key.
4. C The IoT client claims that it fails to use K to decrypt $\mathrm{Enc}(S, K)$ while the cloud did use K to encrypt the service.
5. C The IoT claims that it has obtained an un-executable service program S while the cloud did send the correct service.

The arbitrator handles the IoT client's repudiation similarly as above. For case 1, the cloud gives the arbitrator $\mathrm{Enc}(S)$. The arbitrator computes its cryptographic hash digest and compares it with the one that the IoT client confirmed in the smart contract. Because all the five phases are run, the IoT client must have already confirmed the same digest. Then, the arbitrator can determine the IoT client's cheating. For case 2, the arbitrator checks it on the tamper-proof blockchain. If it is there, the IoT client cheats. For case 3, the cloud gives the key K and encryption randomness to the arbitrator. Using the public key of the IoT client, the arbitrator checks whether the recomputed $\mathrm{AEnc}(K, PK)$ is consistent with the one stored on the blockchain. For case 4, the cloud gives the key K and the encrypted service $\mathrm{Enc}(S, K)$ to the arbitrator. After checking the validness of $\mathrm{Enc}(S, K)$ using the on-chain hash digest, the arbitrator decrypts it using key K. If it decrypts successfully, the client cheats. For case 5, similar to case 4, the arbitrator decrypts the service. The arbitrator checks whether the decrypted service is authorized by the system administrator.

5. Theoretical analysis

Correctness Correctness of the proposed scheme is relatively straight-forward. Suppose both the cloud and the IoT client are honest. The user obtains the encrypted service in phase 3. The user also gets the decryption key in phase 5. After decrypting the service, the IoT client is able to enjoy the service.

Security Security means that neither entity is able to repudiate the service. We have exhausted all possible cases of the cloud and the client in Section 4.4. For detailed analysis, one may refer to Section 4.4. The idea is as follows. Suppose the cloud sends a nonsense service and finishes the five phases of the proposed scheme. The arbitrator could detect this using the signature scheme of the system administrator. Suppose the cloud sends the a correct version of the encrypted service, but sends a wrong decryption key. The arbitrator is able to detect this by checking the decryption. Suppose the IoT client denies a receipt of a service. The client needs to find some data that has the same hash digest with $\mathrm{Enc}(S, K)$. However, this is impossible because the employed cryptographic hash function is collision-resistant.

Theoretical performance Suppose the size of the request service is n. The proposed scheme only stores the hash digests of the data in the smart contract. Thus, the storage cost of the smart contract is $O(1)$. The smart contract provides read and write services to the cloud and the IoT client. This is only related to the size of the hash digests. The computation cost is also $O(1)$. For repudiation handling, the arbitrator needs to compute the hash digest and/or encryption/decryption operations. The cost is $O(n)$.

Performance comparison Table 1 shows the comparison of the proposed scheme with recent works. Because different schemes differ substantially, we give a high-level theoretical comparison and a detailed experimental comparison in the next section. The schemes (Delgado-Segura et al., 2020; Wang et al., 2021; Zou et al., 2016) require a special blockchain platform to accommodate the proposed schemes. The proposed scheme requires no computation in the arbitrator's smart contract. Thus, its computation cost is smaller than (Xu et al., 2019). However, the tradeoff is that the proposed scheme needs an off-line arbitration for disputes. Because disputes should occur not frequently, this choice is practically reasonable. The proposed scheme only stores tiny hash digests on the blockchain. Thus, its storage cost is smaller than (Aniello et al., 2016; Zou et al., 2016). Because blockchain storage and computation need gas consumption, the proposed scheme also has smaller economic cost than (Aniello et al., 2016; Xu et al., 2019; Zou et al., 2016). The proposed scheme encrypts the service. Before sending the decryption key, the privacy is well protect. In contrast, some schemes send partial data to the IoT client before the IoT client's acknowledgement. Once the client aborts the protocol, the partial data leaks data privacy (Xu et al., 2019; Zou et al., 2016). The proposed scheme also has no special requirement on the underlying blockchain; existing blockchain platforms are well suited. For data privacy, the schemes (Ersoy et al., 2021; Xu et al., 2019; Zou et al., 2016) have a risk to leak user data. Both the scheme in Ersoy et al. (2021) and the proposed scheme opensourced the experimental evaluation online.

More discussion The proposed scheme uses encryption to enable non-repudiation of the IoT client. Encryption enforces the IoT client to follow the protocol because the IoT client cannot obtain useful service from an encrypted one. This differs from some traditional schemes by sending portion of the service to the client. Such schemes leak information of the requested service; the IoT client may quit the running once it could derive what it wants from this partial data.

However, we note that encryption also brings performance tradeoff. In this case, arbitration needs to recompute the encryption to handle some repudiations. This incurs cost for the arbitrator. We believe this tradeoff rewards in practice. This is because a system normally runs honestly; only in rare cases, it behaves abnormally. Thus, arbitration should not always happen. Once it does happen, the cost of recomputing the encryption is also acceptable.

The arbitration process could also be automated in the smart contract. However, this makes the smart contract much larger and the cost higher. Because arbitration should not always happen in practice, we leave it as a single process and run it timely off-line as needed.

6. Experimental evaluation

In this section, we evaluate the performance of the proposed scheme on the Ethereum test environment. We first introduce the setup, and then present detailed experimental results. We also compare the performance of the proposed scheme with the state-of-the-art scheme (Xu et al., 2019).

6.1. Methodology

We prototyped a proof-of-concept of the proposed scheme. We built a Ethereum test environment to evaluate the prototype. Our prototype and detailed evaluation steps could be found at (Wang, 2021).

Specifically, we first built a blockchain network using Geth 1.9.23[1] with proof-of-authority (PoA) consensus mechanism. The test net supports both proof-of-authority (PoA) and proof-of-work (PoW) consensus. We chose PoA for better performance, which is also a good choice for a consortium blockchain. PoA is a family of Byzantine fault-tolerant (BFT) consensus algorithms largely used in practice to ensure better performance (De Angelis et al., 2018). We deployed the proposed non-repudiation smart contract on the blockchain.

We conducted the experiments on a server device with 2400 MHz CPU and 256 GB memory. We simulated 50 IoT client nodes and 6 service provider notes in the server. For each node, we assigned a different port number to differentiate them. Additionally, we used 5 nodes for voting in the PoA consensus mechanism. We used SHA-256 algorithm to calculate the hash value, AES-192 algorithm to encrypt the service program, and ECC-256 algorithm to encrypt the secret key. To set up the blockchain, we used `puppeth`, a tool provided by Ethereum client, to quickly create a genesis file. This was used to initialize each node and the creation block of the blockchain. During the process of building the genesis file, we also set the listening port (i.e., from 30,311 to 30,366) and RPC listening port (i.e., from 8501 to 8556) for each node. The blockchain system was run after we started all the nodes and the mining process. We installed the `solc` module to compile the smart contract, which was written in the `solidity` language. To interact with the blockchain, we installed the `web3`, `async`,

[1] https://geth.ethereum.org/.

Table 3
Storage overhead.

Transaction rate	Per minute	Per hour
500 transactions/min	15.07 KB	952.42 KB
1000 transactions/min	32.12 KB	1.76 MB
2000 transactions/min	53.38 KB	2.86 MB

Table 4
Average gas consumption and economic cost.

Event	Average gas consumption	Economic cost
`request_service`	45,620	8.17 USD
`promise_service`	44,718	8.01 USD
`confirm_service`	25,607	4.59 USD
`send_key`	32,216	5.77 USD

Table 5
Average transaction latency in milliseconds.

Test ID	1	2	3	4	5	6	7
Transaction Latency	2.969	1.771	2.053	2.926	1.662	2.382	2.458

Table 6
Computation cost of the IoT client.

Service size	Average hashing time	Average decrypting time
16 KB	0.092 ms	0.144 ms
160 KB	0.096 ms	1.245 ms
1 M	6.822 ms	8.907 ms
10 M	111.626 ms	89.433 ms

`js-sha256` and `node-rsa` modules to run the `Javascript` code to deploy and call the smart contract.

For performance indicators, we measured the storage cost, gas consumption, transaction latency, and computation cost. The storage cost impacts both scalability and performance of the underlying blockchain. The gas consumption is critical to the practicability of the smart contracts. Transaction latency influences throughput of the proposed scheme. The computation cost of the IoT client also matters because an IoT client is often resource constrained. For each indicator, a smaller cost is better.

6.2. Results

Storage overhead Table 3 shows the storage cost of the proposed scheme. Storage cost represents the growth of data storage of the blockchain system. In the evaluation, we use a separate Javascript programme to invoke the smart contract under different frequencies. We invoked the smart contract with three settings, i.e., 500, 1000, 2000 transactions per minute. We continued the test in different minutes and different hours. After the invoking, we checked the blockchain log generated by Geth and computed the size growth of the blockchain.

As in Table 3, when there are 2000 transactions appended to the blockchain system every minute, the storage grows 2.86 MB per hour. The storage cost is small. Comparing the different settings, the storage also grows linearly with the number of transactions. Linear growth also ensures the scalability.

Gas consumption We measured the service-provisioning processes for 500 times and calculated the average gas consumption. Similarly, we used a Javascript program to invoke different functions of the smart contract. After that, we checked the newly generated block information to obtain the gas consumption.

Table 4 shows the average gas consumption and economic cost of each phase in the proposed scheme. In Ethereum, the amount of gas reflects the execution consumption for handling a blockchain transaction. The fee for a transaction is determined by gas quantity times gas price in Ethereum, where the gas quantity is the amount of gas consumed during the execution of a transaction and the gas price is the price per unit of gas. As on 2021-10-12, we set the default gas price to 52 Gwei according to the Ethereum society; also note that 1 ether equals 10^9 Gwei and 1 ether equals 3444 USD.

As is in Table 4, the `request_service` phase consumes the largest gas. In this phase, we wrote some address data and service ID data in the smart contract. The `promise_service` phase also writes a hash value in the smart contract and consumes similar gas. For the `confirm_service` phase, we just checked whether two hash values are consistent. We note that we only need to modify a smart contract variable. Thus, its gas consumption is much smaller than `request_service`. For the `send_key` phase, we also need to write a decryption key value in the smart contract. However, its size is smaller. Thus, the gas consumption is also smaller. We note that cryptocurrency is extremely expensive and volatile; we give the monetary cost here for demonstration only. In practice, cheap and stable blockchain platforms could be used.

In total, the average service-provisioning process needs to consume 148,161 units of gas by summing up all the consumption for the four online phases. The average consumption of each single phase is 37,045 units of gas. This gas consumption is reasonably small. The proposed scheme only saves light-weight service evidence on the blockchain. Thus its cost during blockchain mining is also small.

Transaction latency The transaction latency reflects the response delay of the blockchain system. We initiated 100,000 transactions and recorded the transaction latency. The data were obtained similar to the storage cost evaluation, i.e., we used the Geth log to derive the latency data. We also averaged the transaction latency to obtain a stable value.

Table 5 shows the transaction latency under seven tests. We found that the delay is around 2.315 ms. The result shows that the proposed scheme can handle transactions very quickly.

Computation cost of IoT clients Table 6 lists the computation cost of an IoT client of the proposed scheme. We employed a Javascript program to obtain the performance result. The main computation cost of an IoT client contains two parts. One is the overhead of computing the cryptographic hash function to obtain the hash digest of the encrypted service. The other is the cost for decrypting the received service. We report an average measure of 1000 tests for services with different size. As in Table 6, experimental evaluations show that the two costs are small. For a 10 megabytes service, hashing takes roughly 112 ms and decrypting takes 89 ms.

6.3. Comparison with the state-of-the-art scheme

We also compared the performance of the proposed scheme with the state-of-the-art, general-purpose scheme (Xu et al., 2019). The performance result is from Xu et al. (2019) and our evaluation. Note that the main difference of the proposed scheme with (Xu et al., 2019) lies in the design. The proposed scheme uses an off-line arbitrator and protects service privacy. In contrast, the scheme (Xu et al., 2019) uses online arbitrator. It is not able to protect service privacy if IoT client aborts once it gets some portion of data. This main difference implies that the gas consumption and IoT client cost should differ significantly.

For gas consumption, the scheme (Xu et al., 2019) consumes around 200,000 units of gas for a service provisioning. The proposed scheme costs is less than 150,000, which is more efficient. The reason is that the scheme (Xu et al., 2019) needs to publish portion of the data to the blockchain. This portion of data needs to be mined and then packed into blocks. It takes more gas consumption.

For IoT client computation, Xu et al. (2019) costs 8919.05 ms to calculate the hash value of a 10 MB program. In contrast, our scheme takes 111.626 ms for hashing, 0.245 ms decrypting the key, and another 89.433 ms to decrypt the service. In total, it takes 201.304 ms. The cost is considerably smaller.

7. Conclusion

We proposed a non-repudiation scheme for IoT cloud applications in this work. The scheme employs a blockchain to record tiny values about a requested service. Specifically, the blockchain first stores a hash digest of the encrypted service from the cloud, then stores an acknowledgement of the IoT client, and finally stores the encryption key under the IoT client's public key. Combing the three, the scheme guarantees that both the IoT client and the cloud cannot deny a finished service between them.

With IoT cloud computing paradigm getting widely used, the proposed scheme could help to build a trust between the IoT client and the cloud when enjoying and provisioning services. The added trust layer is useful for those critical applications, cautious IoT cloud users, and cloud platforms. Essentially, non-repudiation service is a special case of data trading. In future, it is an interesting work to investigate blockchain-based general data trading schemes that support more versatile real-world data trade needs.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

CRediT authorship contribution statement

Fei Chen: Conceptualization, Methodology, Writing – original draft. **Jiahao Wang:** Methodology, Software, Validation, Writing – original draft. **Jianqiang Li:** Conceptualization, Methodology, Writing – original draft. **Yang Xu:** Software, Validation, Writing – review & editing. **Cheng Zhang:** Software, Validation, Writing – review & editing. **Tao Xiang:** Conceptualization, Methodology, Writing – original draft.

Acknowledgments

This work was partially supported by the National Natural Science Foundation of China under Grant no. (61872243, 62072062, U20A20176, 62002113, 62073225), Guangdong Basic and Applied Basic Research Foundation (2020A1515011489), Natural Science Foundation of Hunan Province (no. 2021JJ40122), the National Key RD Program of China under Grant (2020YFA0908700), the Natural Science Foundation of Guangdong Province-Outstanding Youth Program under Grant (2019B151502018), and Shenzhen Science and Technology Innovation Commission (R2020A045).

References

Alzubaidi, A., Solaiman, E., Patel, P., Mitra, K., 2019. Blockchain-based SLA management in the context of IoT. IT Prof. 21 (4), 33–40. doi:10.1109/MITP.2019.2909216.
Aniello, L., Baldoni, R., Lombardi, F., 2016. A blockchain-based solution for enabling log-based resolution of disputes in multi-party transactions. In: International Conference in Software Engineering for Defence Applications. Springer, pp. 53–58.
Asokan, N., Schunter, M., Waidner, M., 1997. Optimistic protocols for fair exchange. In: Proceedings of the 4th ACM Conference on Computer and Communications Security, pp. 7–17.
Chen, F., Xiao, Z., Cui, L., Lin, Q., Li, J., Yu, S., 2020. Blockchain for internet of things applications: areview and open issues. J. Netw. Comput. Appl. 102839.
Coffey, T., Saidha, P., 1996. Non-repudiation with mandatory proof of receipt. ACM SIGCOMM Comput. Commun. Rev. 26 (1), 6–17.
Crosby, M., Pattanayak, P., Verma, S., Kalyanaraman, V., et al., 2016. Blockchain technology: beyond Bitcoin. Appl. Innov. 2 (6–10), 71.
De Angelis, S., Aniello, L., Baldoni, R., Lombardi, F., Margheri, A., Sassone, V., 2018. PBFT vs. proof-of-authority: applying the CAP theorem to permissioned blockchain. In: Italian Conference on Cyber Security, Milan, Italy.
Delgado-Segura, S., Pérez-Solà, C., Navarro-Arribas, G., Herrera-Joancomartí, J., 2020. A fair protocol for data trading based on Bitcoin transactions. Future Gener. Comput. Syst. 107, 832–840.
Elloh, Y.A., Hammi, B., Serhrouchni, A., Zeadally, S., 2021. A blockchain-based certificate revocation management and status verification system. Comput. Secur. 104, 102209.
Ersoy, O., Genç, Z.A., Erkin, Z., Conti, M., 2021. Practical exchange for unique digital goods. In: 2021 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPS). IEEE, pp. 49–58.
Fan, K., Bao, Z., Liu, M., Vasilakos, A.V., Shi, W., 2020. Dredas: decentralized, reliable and efficient remote outsourced data auditing scheme with blockchain smart contract for industrial IoT. Future Gener. Comput. Syst. 110, 665–674.
Feng, J., Chen, Y., Ku, W., Liu, P., 2010. Analysis of integrity vulnerabilities and a non-repudiation protocol for cloud data storage platforms. In: International Conference on Parallel Processing Workshops, pp. 251–258.
Joseph, A.O., Raffety, J., Morrow, P., Zhiwei, L., Nugent, C., McClean, S., Ducatel, G., 2019. Securing self-organizing IoT ecosystem: adistributed ledger technology approach. In: 2019 IEEE 5th World Forum on Internet of Things (WF-IoT), pp. 809–814.
Li, J., Lu, H., Guizani, M., 2015. ACPN: a novel authentication framework with conditional privacy-preservation and non-repudiation for VANETs. IEEE Trans. Parallel Distrib. Syst. 26 (4), 938–948.
Li, L., Liu, J., Jia, P., 2021. Sectep: enabling secure tender evaluation with sealed prices and quality evaluation in procurement bidding systems over blockchain. Comput. Secur. 103, 102188.
Manski, S., 2017. Building the blockchain world: technological commonwealth or just more of the same? Strateg. Change 26 (5), 511–522.
Markowitch, O., Roggeman, Y., 1999. Probabilistic non-repudiation without trusted third party. In: Second Conference on Security in Communication Networks, vol. 99, pp. 25–36.
Mitsianis, J., 2001. A new approach to enforcing non-repudiation of receipt. Manuscript 1–8.
Pachala, S., Rupa, C., Sumalatha, L., 2021. An improved security and privacy management system for data in multi-cloud environments using a hybrid approach. Evol. Intell. 14 (2), 1117–1133.
Rupa, C., Midhunchakkaravarthy, D., Hasan, M.K., Alhumyani, H., Saeed, R.A., 2021. Industry 5.0: ethereum blockchain technology based DApp smart contract. Math. Biosci. Eng. 18 (5), 7010–7027.
Rupa, C., Patan, R., Al-Turjman, F., Mostarda, L., 2020. Enhancing the access privacy of IDaaS system using SAML protocol in fog computing. IEEE Access 8, 168793–168801.
Rupa, C., Srivastava, G., Gadekallu, T.R., Maddikunta, P.K.R., Bhattacharya, S., 2020. A blockchain based cloud integrated IoT architecture using a hybrid design. In: International Conference on Collaborative Computing: Networking, Applications and Worksharing. Springer, pp. 550–559.
Rupa, C., Srivastava, G., Gadekallu, T.R., Maddikunta, P.K.R., Bhattacharya, S., 2020. Security and privacy of UAV data using blockchain technology. J. Inf. Secur. Appl. 55, 102670.
Saini, A., Zhu, Q., Singh, N., Xiang, Y., Gao, L., Zhang, Y., 2021. A smart-contract-based access control framework for cloud smart healthcare system. IEEE Internet Things J. 8 (7), 5914–5925.
Shi, S., He, D., Li, L., Kumar, N., Khan, M.K., Choo, K.R., 2020. Applications of blockchain in ensuring the security and privacy of electronic health record systems: a survey. Comput. Secur. 97, 101966.
Wang, J., 2021. Source code for experimental evaluation. https://github.com/szu-security-group/trust-builder.
Wang, L., Liu, J., Liu, W., 2021. Staged data delivery protocol: a blockchain-based two-stage protocol for non-repudiation data delivery. Concurr. Comput. 33 (13), e6240.
Wang, T., Wang, P., Cai, S., Ma, Y., Liu, A., Xie, M., 2020. A unified trustworthy environment establishment based on edge computing in industrial IoT. IEEE Trans. Ind. Inf. 16 (9), 6083–6091.
Xu, Y., Ren, J., Wang, G., Zhang, C., Yang, J., Zhang, Y., 2019. A blockchain-based non-repudiation network computing service scheme for industrial IoT. IEEE Trans. Ind. Inf. 15 (6), 3632–3641.
Zhang, L., Zou, Y., Wang, W., Jin, Z., Su, Y., Chen, H., 2021. Resource allocation and trust computing for blockchain-enabled edge computing system. Comput. Secur. 105, 102249.
Zhao, W., Jiang, C., Gao, H., Yang, S., Luo, X., 2021. Blockchain-enabled cyber-physical systems: areview. IEEE Internet Things J. 8 (6), 4023–4034.
Zhao, Y., Yang, L.T., Sun, J., 2019. Privacy-preserving tensor-based multiple clusterings on cloud for industrial IoT. IEEE Trans. Ind. Inf. 15 (4), 2372–2381.
Zhao, Y., Zhao, J., Jiang, L., Tan, R., Niyato, D., Li, Z., Lyu, L., Liu, Y., 2021. Privacy-preserving blockchain-based federated learning for IoT devices. IEEE Internet Things J. 8 (3), 1817–1829.
Zhou, J., Gollman, D., 1996. A fair non-repudiation protocol. In: Proc. of IEEE Symposium on Security and Privacy. IEEE, pp. 55–61.
Zhou, J., Gollmann, D., 1996. Observations on non-repudiation. In: International Conference on the Theory and Application of Cryptology and Information Security, pp. 133–144.

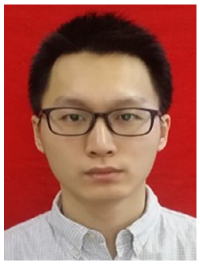
Zhou, J., Lam, K., 1999. Securing digital signatures for non-repudiation. Comput. Commun. 22 (8), 710–716.
Zou, J., Wang, Y., Orgun, M.A., 2016. A dispute arbitration protocol based on a peer–to-peer service contract management scheme. In: IEEE International Conference on Web Services. IEEE, pp. 41–48.
Zou, J., Wang, Y., Orgun, M.A., 2016. A dispute arbitration protocol based on a peer–to-peer service contract management scheme. In: 2016 IEEE International Conference on Web Services (ICWS), pp. 41–48.
Zou, J., Ye, B., Qu, L., Wang, Y., Orgun, M.A., Li, L., 2018. A proof-of-trust consensus protocol for enhancing accountability in crowdsourcing services. IEEE Trans. Serv. Comput. 12 (3), 429–445.

Fei Chen received the Ph.D. degree in computer science and engineering from The Chinese University of Hong Kong in 2014. He is currently an Associate Professor with College of Computer Science and Software Engineering, Shenzhen University, China. His research interests include data protection and privacy, and distributed systems and applications.

Jiahao Wang is currently a master student at College of Computer Science and Software Engineering, Shenzhen University, China. His research interests include data protection and privacy, and distributed systems and applications.

Jianqiang Li received the B.S. and Ph.D. degrees in automation from the South China University of Technology, Guangzhou, China, in 2003 and 2008, respectively. He is a Professor with the College of Computer and Software Engineering, Shenzhen University, Shenzhen, China. He led three projects of the National Natural Science Foundation and three projects of the Natural Science Foundation of Guangdong Province, China. His current research interests include data analysis, embedded systems, and the Internet of Things.

Yang Xu received the Ph.D. degree in Computer Science and Technology from Central South University, China. From 2015 to 2017, he was a visiting scholar in the Department of Computer Science and Engineering at Texas A&M University, USA. He is currently an Assistant Professor at the College of Computer Science and Electronic Engineering, Hunan University, China. He is working in the areas of cloud computing, blockchain, and operating system. He has published over 40 articles in international journals and conferences, including IEEE TSC, TII, TETC, TCBB, TNSE etc. He was the awardee of the Best Paper Award of IEEE International Conference on Internet of People (IoP 2018). He is a member of IEEE, and a member of CCF Technical Committee on Blockchain. He serves as the Program Committee Chair for IWCSS 2020, the Publicity Chair for CPSCom 2020, and as a reviewer of over 10 international journals.



Cheng Zhang received his B.Sc. degree in Shenyang University of Technology. He is currently pursuing the Ph.D. degree in the College of Computer Science and Electronic Engineering, Hunan University, China. His research interests mainly focus on the network security and blockchain.

Tao Xiang received the BEng, MS and Ph.D. degrees in computer science from Chongqing University, China, in 2003, 2005, and 2008, respectively. He is currently a Professor of the College of Computer Science at Chongqing University. Dr. Xiang's research interests include multimedia security, cloud security, data privacy and cryptography. He has published over 100 papers on international journals and conferences. He also served as a referee for numerous international journals and conferences