Information Sciences 462 (2018) 262–277

Contents lists available at ScienceDirect

Information Sciences

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

Blockchain based efficient and robust fair payment for outsourcing services in cloud computing

Yinghui Zhang[a,b,c,d,*], Robert H. Deng[b], Ximeng Liu[b], Dong Zheng[a,d]

[a] National Engineering Laboratory for Wireless Security, Xi'an University of Posts and Telecommunications, Xi'an 710121, PR China
[b] School of Information Systems, Singapore Management University, Singapore
[c] State Key Laboratory of Cryptology, P.O. Box 5159, Beijing 100878, PR China
[d] Westone Cryptologic Research Center, Beijing 100070, PR China

ARTICLE INFO

Article history:
Received 30 March 2018
Revised 2 June 2018
Accepted 7 June 2018
Available online 15 June 2018

Keywords:
Blockchain
Cloud security
Fair payment
Provable data possession
Outsourcing computation
Authentication

ABSTRACT

As an attractive business model of cloud computing, outsourcing services usually involve online payment and security issues. The mutual distrust between users and outsourcing service providers may severely impede the wide adoption of cloud computing. Nevertheless, most existing payment solutions only consider a specific type of outsourcing service and rely on a trusted third-party to realize fairness.

In this paper, in order to realize secure and fair payment of outsourcing services in general without relying on any third-party, trusted or not, we introduce BCPay, a blockchain based fair payment framework for outsourcing services in cloud computing. We first present the system architecture, specifications and adversary model of BCPay, then describe in detail its design. Our security analysis indicates that BCPay achieves *Soundness* and what we call *Robust Fairness*, where the fairness is resilient to eavesdropping and malleability attacks. Furthermore, our performance evaluation shows that BCPay is very efficient in terms of the number of transactions and computation cost. As illustrative applications of BCPay, we further construct a blockchain-based provable data possession scheme in cloud computing and a blockchain-based outsourcing computation protocol in fog computing.

1. Introduction

As a promising computing paradigm, cloud computing has many attractive benefits, such as flexibility, high efficiency and high availability. It can provide a diversity of outsourcing services including storage and computations [3]. With the rapid development of cloud computing technologies, an increasing number of individuals and enterprises have uploaded their various data onto third-party cloud platforms either for ease of sharing or for cost savings. The cloud storage service of Dropbox currently has approximately 500 million registered users and 500 petabytes of user data [27]. Users can also subscribe to flexible computation resources from cloud service providers such as Google and Amazon. In order to facilitate the operation of computation, storage and networking services between end users and cloud computing data centers, fog computing further extends cloud computing to the edge of the network [9]. In fog computing, the outsourcing computation

[*] Corresponding author at: National Engineering Laboratory for Wireless Security, Xi'an University of Posts and Telecommunications, Xi'an 710121, PR China.
E-mail addresses: yhzhaang@163.com (Y. Zhang), robertdeng@smu.edu.sg (R.H. Deng), snbnix@gmail.com (X. Liu), zhengdong@xupt.edu.cn (D. Zheng).

https://doi.org/10.1016/j.ins.2018.06.018
0020-0255/© 2018 Elsevier Inc. All rights reserved.

service is required because end users usually are resource-constrained. Obviously, outsourcing services play an important role in the development of cloud and fog computing.

Although cloud computing allows users to customize outsourcing services, its unique aspects also raise various security and privacy concerns [29,32,34,35,41,49–51]. In cloud storage, for instance, users usually require assurance of data possession besides confidentiality of outsourced data. As for computation, users expect to get valid and correct computation results from the outsourcing service provider once the service fee is paid. Recently, great efforts have been made to realize provable data possession (PDP) [4,6] and verifiable outsourcing computation [13,15,25,33,42]. However, most of the existing schemes do not consider the payment issues in outsourcing services. Take PDP as an example. In a challenge proof of PDP, if the server is malicious, a user's data may be lost without any compensation even if he/she has paid for the service. On the other hand, in the case of a malicious user, the server cannot earn the service fee from the user even if it enforces a valid and correct PDP service. Because of the distrust between the user and the server [18,24,37,47,48], the payment issues are sufficiently challenging for outsourcing services considering fairness.

In order to simultaneously address the payment and security issues, most of the existing schemes adopt the (default) traditional payment mechanism and rely on a trusted third-party such as a bank. For example, the Google cloud platform provides a series of cloud services including computing and data storage, and the registration requires a bank account [20]. In cloud computing, however, the traditional payment solution suffers several drawbacks. First, it is assumed that the bank is trusted by all the users and the server and it deals with all procedures in a fair manner. Second, the payment mechanism needs to be adapted to multiple banks used by different participants and has to be updated whenever they change, which will become a bottleneck of the payment system. Last but not least, users' privacy associated with bank accounts may be violated.

Recently, blockchain technologies have gained prominent popularity mostly due to its distributed nature and the lack of a central authority. In blockchain-based outsourcing services, the service fee is transferred directly between the user and the server and they do not have to trust any third-party. However, to the best of our knowledge, blockchain technologies have seldom been used in general for fair payment of outsourcing services in cloud and fog computing.

1.1. Our contributions

To eliminate the third-party, trusted or not, while ensuring the fairness of payment against malicious users and outsourcing service providers, we introduce BCPay, a blockchain based fair payment framework for outsourcing services in cloud and fog computing. Our contributions are three-folds:

1. We first propose the system architecture, specifications and adversary model of BCPay, then describe its design details. We prove that BCPay enjoys *Soundness* and *Robust Fairness* where the latter implies that fairness is resilient to any attacks including eavesdropping and malleability attacks without relying on any third-party.
2. In BCPay, soundness and robust fairness are achieved by an *all-or-nothing checking-proof* protocol. In the protocol, it is ensured that the outsourcing service provider either earns the service fee and gets his/her guaranty back simultaneously or pays a penalty in the form of deposit to the user. Besides, our performance evaluation shows that BCPay is very efficient in terms of the number of involved transactions and computation cost.
3. To illustrate the applications of BCPay, we propose a blockchain-based PDP scheme in cloud computing and an outsourcing computation protocol suitable for fog computing.

1.2. Related work

As an earlier and important application of blockchain technologies, Bitcoin was announced under the pseudonym Satoshi Nakamoto [39]. To facilitate the wide use of blockchain technologies, Buterin [10] proposed Ethereum, a next-generation smart contract and decentralized application platform. Later, Andrychowicz et al. [2] proposed a bitcoin-based timed commitment scheme, in which the committer has to reveal his/her secret before a specific time, or to pay a fine. With bitcoin-based timed commitments in place, they further constructed protocols for secure multiparty lotteries. In order to realize more general computation, Andrychowicz et al. [1] proposed a simultaneous Bitcoin-based timed commitment scheme. Subsequently, they presented a two-party computation protocol, which modifies the Bitcoin specifications to resist malleability attacks. Similar ideas were developed independently by Bentov et al. [8]. Note that all these bitcoin-based schemes cannot realize what we call all-or-nothing property which is required in outsourcing services. Specifically, the all-or-nothing property ensures that the outsourcing service provider either earns the service fee and gets his/her guaranty back simultaneously or pays a penalty to the user. The line of work on outsourcing service consists of outsourcing storage and outsourcing computation.

As for outsourcing storage, based on RSA homomorphic tags, Ateniese et al. [4] proposed the first PDP scheme, which allows users to challenge the cloud server for a proof that the integrity of their data is not violated. Recently, homomorphic signature and encryption technologies have obtained many attentions [40,45]. In the same year, Juels et al. [30] defined and explored proofs of retrievability, which enables the cloud server to produce a concise proof that a user can retrieve a target file. Later, Ateniese et al. [7] presented a PDP scheme based on identification protocols supporting public verification. Data dynamics are further considered in [5,43,46]. On the other hand, outsourcing computation enables resource-limited

Table 1
Notations used in BCPay.

$\mathcal{C}$	The client	$I_{i,j}$	The hash value of the jth
$\mathcal{S}$	The server		node with height i in $\mathcal{T}_\ell$
H	A hash function	σ_{root}	The ECDSA signature of
r_S	The secret of $\mathcal{S}$		the root of $\mathcal{T}_\ell$
h_S	The hash value $H(r_S)$	chal	A challenge-related hash value
t	A time-lock		set used in the service checking
$\mathcal{T}_\ell$	A service data tree	chal_0	A challenge-related variable set
ℓ	The height of $\mathcal{T}_\ell$		used in the service checking
(pk_A, sk_A)	An ECDSA key pair of A	$\mathsf{ChalIndex}$	A challenge-related index
data_0	Service-related local data		set used in the service checking
data_1	The outsourcing data	$\max_B$	The maximal delay between
$\mathsf{chaldata}$	A challenge (data indexes)		broadcasting a transaction and
D_i	The ith data block in data_1		including it on the blockchain

end users in fog computing to complete computationally expensive tasks with the help of fog nodes (a.k.a. workers). This introduces the potential of cheating by untrusted participants in a commercial setting. To protect the rights and interests of users, the concept of ringer [26] is introduced to verify the validity of outsourcing computation results. In order to improve efficiency, Du et al. [22] presented a commitment-based scheme to prevent workers from cheating. Gennaro et al. [25] proposed a verifiable outsourcing computation scheme while protecting the input and output privacy. Carbunar et al. [12] proposed several outsourcing computation solutions that simultaneously ensure correct remuneration for computation tasks completed on time and prevent workers' laziness. Chen et al. [16] considered outsourcing computation with such workers that may not send the computation results on time. Chen et al. [14] further proposed a conditional e-payment system based on a restrictive partially blind signature scheme. Song et al. [42] proposed a solution to verifiable outsourcing of polynomial evaluation. Additionally, verifiable computation over large database is studied in [17].

In the above schemes, however, either the payment issue is not taken into account or the traditional payment framework is adopted, which needs a trusted third-party to realize fair payment. To solve these problems, blockchain technologies have been introduced to outsourcing services. Compared to traditional payment technologies, the independence from central authorities is the key advantage of blockchain-based solutions. Ateniese et al. [6] introduced accountable storage based on an extension of invertible Bloom filters, and showed how to combine it with Bitcoin based zero-knowledge proofs. However, the combination involves a trusted third-party called Bitcoin arbitrator. Huang et al. [28] proposed a blockchain-based outsourcing computation scheme, in which a trusted third-party is still required. Obviously, all these schemes [6,28] fail to truly realize blockchain-based decentralized outsourcing services. Campanelli et al. [11] defined the notion of zero-knowledge contingent service payment to realize service payment based on blockchains. They constructed two high-level protocols and presented a concrete realization based on the proof of retrievability service. However, the proposed protocols are only conceptual and lack design details, of which the efficiency remains to be improved because a witness indistinguishable protocol [23] is used as a building block. Based on game theory and Ethereum smart contracts, Dong et al. [21] proposed a protocol for checking the correctness of computation in cloud computing. However, it is assumed that users are honest and two clouds cannot collude. On the other hand, in order to improve the transaction throughput and latency in blockchains, current efforts focus on off-chain payment channels which can be combined in a payment-channel network to enable a number of payments without accessing the blockchain. Khalil et al. [31] presented a solution which allows an arbitrary set of users in the payment-channel network to securely rebalance their channels. Malavolta et al. [36] formalized the security and privacy notions in a payment-channel network including balance security and value privacy. In this paper, we propose a general blockchain-based payment solution for outsourcing services, which can efficiently address the threat of cheating from malicious participants and offer guarantees that the service has been correctly enforced.

1.3. Organization

The rest of the paper is organized as follows. Some preliminaries are given in Section 2. We then present the system architecture, specifications and adversary model in Section 3. The proposed framework BCPay together with its security analysis are presented in Section 4. Section 5 shows the performance evaluation of BCPay. In Section 6, we present several applications of BCPay. Finally, concluding remarks are made in Section 7.

2. Preliminaries

In this section, we first list some notations and then briefly review blockchains and Bitcoin-based timed commitments.

2.1. Notations

In Table 1, we present notations mainly used in BCPay.

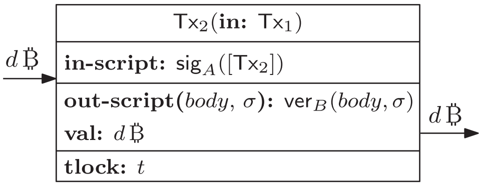

$\mathsf{Tx}_2(\mathbf{in:}\ \mathsf{Tx}_1)$
d ฿
$\mathbf{in\text{-}script:}\ \mathsf{sig}_A([\mathsf{Tx}_2])$
$\mathbf{out\text{-}script}(body, \sigma)\mathbf{:}\ \mathsf{ver}_B(body, \sigma)$
$\mathbf{val:}\ d$ ฿
d ฿
$\mathbf{tlock:}\ t$

Fig. 1. An example of transaction.

2.2. Blockchain

The blockchain is an essential technology behind many cryptocurrencies, with Bitcoin and Ethereum as the two most widely used ones. The idea of the blockchain is that the longest chain is accepted as the proper one. In the following, we describe blockchain in terms of the Bitcoin currency system, including addresses and transactions.

As an important ingredient of the Bitcoin system, the ECDSA signature is associated with a public-secret key pair (pk, sk). Technically, an address is a hash of a public key pk. To keep the exposition as simple as possible, we use pk to represent an address. Suppose a user A has a key pair (pk_A, sk_A), then $\mathsf{sig}_A(m)$ denotes the ECDSA signature on a message m associated with sk_A, and $\mathsf{vec}_A(m, \sigma)$ denotes the result of the verification of the ECDSA signature σ on the message m with regard to pk_A. The most general form of a Bitcoin transaction Tx_x is

$$((y_1, a_1, \sigma_1), \ldots, (y_n, a_n, \sigma_n), (v_1, \pi_1), \ldots, (v_m, \pi_m), t).$$

The inputs of Tx_x are triples $(y_1, a_1, \sigma_1), \ldots, (y_n, a_n, \sigma_n)$, where y_i is the hash of some previous transaction Tx_{y_i}, a_i is an index of the output of Tx_{y_i} and σ_i is called an input script. The outputs of Tx_x are a list of pairs $(v_1, \pi_1), \ldots, (v_m, \pi_m)$, where v_i is the value of the ith output of Tx_x and π_i is an output script. In particular, t is a time-lock, which means that Tx_x is valid only if time t is reached. In Ethereum, similar mechanisms can be realized based on the Ethereum Alarm Clock [38]. Furthermore, the body of Tx_x is denoted as

$$[\mathsf{Tx}_x] = ((y_1, a_1), \ldots, (y_n, a_n), (v_1, \pi_1), \ldots, (v_m, \pi_m), t),$$

which is equal to Tx_x without the input script. The transaction Tx_x is valid if $\pi_i'([\mathsf{Tx}_x], \sigma_i)$ evaluates to true for $1 \le i \le n$, where π_i' is the output script of the a_ith output of Tx_{y_i}. The scripts are written in the Bitcoin scripting language, which is a stack based, not Turing-complete language. In Fig. 1, as an example of transactions, the user A aims to transfer d ฿ from Tx_1 to the user B after time t based on Tx_2, where the output script is an ECDSA signature verification. Similar to [1,2], to keep the exposition simple we present our results assuming that the transaction fees are zero.

2.3. Bitcoin-based timed commitment

In BCPay, the bitcoin-based timed commitment scheme [2] is used, which is also adopted by Ateniese et al. [6] and Huang et al. [28]. The commitment scheme is denoted by $\mathsf{CS}(\mathcal{S}, \mathcal{C}, d, t, s)$ and is executed between $\mathcal{S}$ and $\mathcal{C}$, where the outsourcing service provider $\mathcal{S}$ acts as a committer and the outsourcing service client $\mathcal{C}$ acts as a receipt. Concretely, $\mathcal{S}$ commits to a secret s and has to open the commitment before a specific time t to get his/her deposit of value d ฿ back. Otherwise, the deposit will be given to $\mathcal{C}$. The commitment scheme consists of three phases: the commitment phase $\mathsf{CS.Commit}(\mathcal{S}, \mathcal{C}, d, t, s)$, the opening phase $\mathsf{CS.Open}(\mathcal{S}, \mathcal{C}, d, t, s)$ and the punishment phase $\mathsf{CS.Fine}(\mathcal{S}, \mathcal{C}, d, t, s)$. Note that the punishment phase is performed only if the opening phase is not correctly performed. Three transactions $\mathsf{TxCommit}$, TxOpen and TxFine, as shown in Fig. 2, are involved in the commitment phase, the opening phase and the punishment phase, respectively. In Fig. 2, the omitted arguments of scripts are denoted by $\perp$ and H is a hash function. Please refer to Andrychowicz et al. [2] for more details.

3. System architecture, specifications and adversary model

In this section, we first present the system architecture and specifications of BCPay. Then, the adversary model and design goals of BCPay are described in detail.

3.1. System architecture of BCPay

The system architecture of BCPay is illustrated in Fig. 3, and it involves clients (i.e., users), servers (i.e., outsourcing service providers) and a blockchain. In the rest of this paper, we use $\mathcal{C}$ and $\mathcal{S}$ to denote a client and a server, respectively. Suppose $\mathcal{C}$ plans to subscribe to an outsourcing service sv from $\mathcal{S}$. To keep the presentation compact, we only show the main procedures of BCPay in Fig. 3. The procedures (1), (2), (3.1) and (3.2) are used to implement sv. The procedures (4), (5), (6.1) and (6.2) are used to check the sv implementation and the checking result is reflected in the service payment

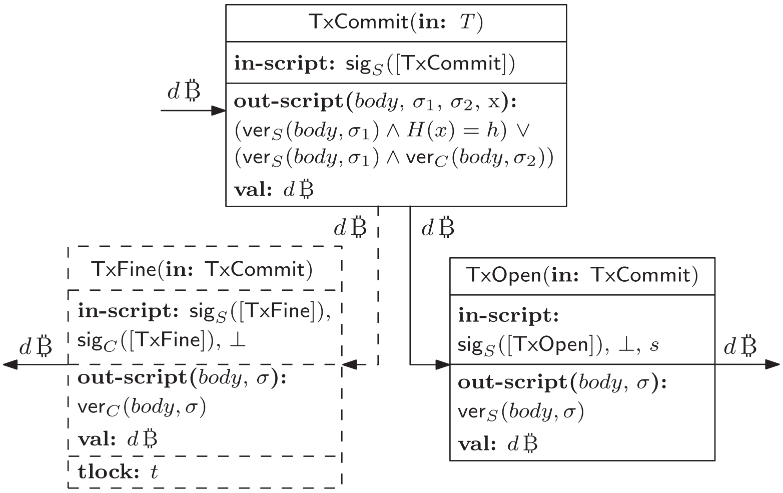

TxCommit(in: T)
in-script: $\mathsf{sig}_S([\mathsf{TxCommit}])$
out-script($body$, σ_1, σ_2, x):
$(\mathsf{ver}_S(body, \sigma_1) \wedge H(x) = h) \vee$
$(\mathsf{ver}_S(body, \sigma_1) \wedge \mathsf{ver}_C(body, \sigma_2))$
val: d ฿
d ฿
TxFine(in: TxCommit)
in-script: $\mathsf{sig}_S([\mathsf{TxFine}])$, $\mathsf{sig}_C([\mathsf{TxFine}])$, $\perp$
out-script($body$, σ):
$\mathsf{ver}_C(body, \sigma)$
val: d ฿
tlock: t
TxOpen(in: TxCommit)
in-script:
$\mathsf{sig}_S([\mathsf{TxOpen}])$, $\perp$, s
out-script($body$, σ):
$\mathsf{ver}_S(body, \sigma)$
val: d ฿

Fig. 2. The transactions involved in bitcoin-based timed commitments.

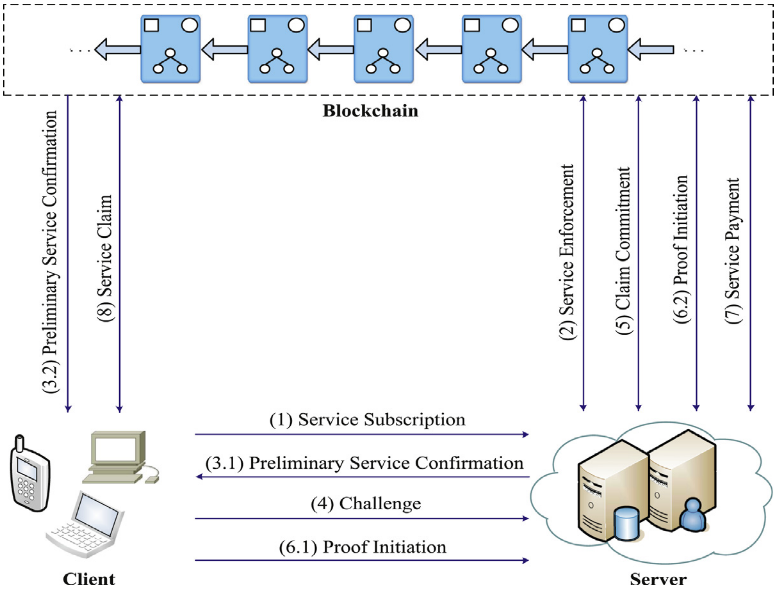


Fig. 3. The system architecture of BCPay.

(7) or the service claim (8). In BCPay, a public blockchain is considered, such as the Bitcoin blockchain and the Ethereum blockchain. The entities $\mathcal{C}$ and $\mathcal{S}$ are detailed as follows:

- *Client* $\mathcal{C}$: As a user, $\mathcal{C}$ subscribes to an outsourcing service sv from $\mathcal{S}$. After sv is enforced by $\mathcal{S}$, $\mathcal{C}$ can get a preliminary service confirmation from $\mathcal{S}$ based on the blockchain. In order to check the implementation of sv before the payment, $\mathcal{C}$ sends a challenge to $\mathcal{S}$. $\mathcal{S}$ first makes a claim commitment to ensure that $\mathcal{C}$ will get enough compensation in the form of deposits if $\mathcal{S}$ is malicious. Then, $\mathcal{C}$ and $\mathcal{S}$ jointly initiate the service implementation proof by specifying some requirements of sv. If $\mathcal{S}$ fails to provide a valid service proof that the service implementation meets the requirements before a specific time, $\mathcal{C}$ can claim enough deposits by himself from $\mathcal{S}$.
- *Server* $\mathcal{S}$: As an outsourcing service provider, $\mathcal{S}$ aims to earn service fees from $\mathcal{C}$ by enforcing services subscribed by $\mathcal{C}$. Upon receiving the service subscription request from $\mathcal{C}$, $\mathcal{S}$ completes the enforcement of sv based on the blockchain and sends to $\mathcal{C}$ a preliminary confirmation message. Then, $\mathcal{S}$ makes the claim commitment after receiving the challenge from $\mathcal{C}$. Once the joint proof initiation is finished, $\mathcal{S}$ provides a valid service implementation proof to get the service fee from $\mathcal{C}$ in the service payment phase before the specific time.

3.2. Specifications of BCPay

BCPay consists of five phases: the system setup phase, the service implementation phase, the service checking phase, the service payment phase and the service claim phase. The first four phases are compulsory and the service claim phase is performed by $\mathcal{C}$ only if $\mathcal{S}$ is malicious.[1] The details of the specifications of BCPay are as follows:

3.2.1. System setup phase

$\mathcal{C}$ and $\mathcal{S}$ initialize some parameters such as unredeemed transactions on the blockchain to be used in the subsequent phases.

3.2.2. Service implementation phase

The outsourcing service sv is implemented in this phase. Three procedures, service subscription, service enforcement and preliminary service confirmation, are sequentially performed as below.

- *Service subscription:* $\mathcal{C}$ subscribes to sv from $\mathcal{S}$ by sending service-related data to $\mathcal{S}$.
- *Service enforcement:* In this procedure, sv is enforced by $\mathcal{S}$. Upon receiving the subscription data from $\mathcal{C}$, $\mathcal{S}$ enforces sv. Then, $\mathcal{S}$ generates a digital signature according to the enforcement of sv and stores the signature on the blockchain. Finally, $\mathcal{S}$ sends a confirmation message to $\mathcal{C}$ that helps $\mathcal{C}$ to obtain the signature from the blockchain.
- *Preliminary service confirmation:* After obtaining the signature from the blockchain, $\mathcal{C}$ considers that sv has been preliminarily implemented, where "preliminarily" means that the sv implementation will be checked by $\mathcal{C}$ before the payment.

3.2.3. Service checking phase

This phase is used by $\mathcal{C}$ and $\mathcal{S}$ to jointly initiate the service checking. In this phase, the service requirements are specified. Three sequential sub-phases, challenge generation phase, claim commitment phase and proof initiation phase, are performed as below.

- *Challenge generation phase:* In order to check the sv implementation, $\mathcal{C}$ sends a challenge to $\mathcal{S}$ besides reaching an agreement beforehand on service-related parameters such as the compensation and penalty of $\mathcal{S}$ in the case of service failure.
- *Claim commitment phase:* This phase is used by $\mathcal{S}$ to make a commitment that once the sv implementation does not meet the requirements specified in the *Proof Initiation Phase* and the malicious $\mathcal{S}$ refuses to compensate $\mathcal{C}$ in the *Service Payment Phase* before a specific time, $\mathcal{C}$ is able to claim enough deposits of $\mathcal{S}$ by himself/herself as a penalty in the *Service Claim Phase* after the specific time.
- *Proof initiation phase:* This phase is used by $\mathcal{C}$ and $\mathcal{S}$ to realize the service checking by temporarily freezing a joint deposit consists of service fee and guaranty respectively from $\mathcal{C}$ and $\mathcal{S}$, in which the requirements of sv are agreed upon. After this phase, honest $\mathcal{C}$ can ensure that either a valid sv is achieved in the *Service Payment Phase* by paying the service fee or enough deposits are claimed in the *Service Claim Phase* no matter how $\mathcal{S}$ behaves. On the other hand, honest $\mathcal{S}$ can ensure that if the sv implementation is valid, he/she will earn the service fee no matter how $\mathcal{C}$ behaves.

3.2.4. Service payment phase

This phase is performed by $\mathcal{S}$ to earn the service fee from $\mathcal{C}$ by proving that the sv implementation meets the requirements. Certainly, $\mathcal{C}$ can ensure that the service fee is paid only if the sv implementation is what is expected.

3.2.5. Service claim phase

Only if $\mathcal{S}$ fails to prove that the sv implementation meets the requirements of $\mathcal{C}$ before a specific time, BCPay comes to the *Service Claim Phase*. This phase is used by $\mathcal{C}$ to claim enough deposits from $\mathcal{S}$ no matter how $\mathcal{S}$ behaves.

3.3. Adversary model and design goals of BCPay

In BCPay, both $\mathcal{C}$ and $\mathcal{S}$ can be malicious and they are of mutual distrust. Concretely, malicious $\mathcal{C}$ aims to enjoy the outsourcing service sv provided by $\mathcal{S}$ without paying the service fee while malicious $\mathcal{S}$ tries to get the service fee from $\mathcal{C}$ without implementing the service sv as specified in the requirements of $\mathcal{C}$. As for the blockchain, its contents are publicly available and both $\mathcal{C}$ and $\mathcal{S}$ can verify the authenticity of data in the blockchain.

In addition, no private channels are required in BCPay. Hence, eavesdropping attacks and malleability attacks should be taken into consideration. In these attacks, the adversary aims to undermine the fairness in BCPay.

- *Eavesdropping attacks:* The adversary can eavesdrop on the public channel to see the transactions sent by the honest party, before they appear on the blockchain.
- *Malleability attacks:* Based on the eavesdropping, the adversary tries to make some transactions invalid by modifying their hash values without changing the semantics.

[1] Strictly speaking, we mean $\mathcal{S}$ fails to provide a valid service implementation proof.

In BCPay, our security goals mainly include soundness and robust fairness as follows[2]:

- *Soundness:* If both $\mathcal{C}$ and $\mathcal{S}$ are honest, then $\mathcal{C}$ can obtain the required service implementation and $\mathcal{S}$ can gain the corresponding service fee.
- *Robust fairness:* The fairness means that it is infeasible for the malicious $\mathcal{C}$ to enjoy the outsourcing service sv provided by $\mathcal{S}$ without paying the service fee and it is infeasible for the malicious $\mathcal{S}$ to get the service fee paid by $\mathcal{C}$ without providing a valid sv implementation proof in terms of the requirements of $\mathcal{C}$ before a specific time. Particularly, if malicious $\mathcal{S}$ fails to provide such a proof, $\mathcal{C}$ is able to get enough compensation or penalty from $\mathcal{S}$. Robust fairness means that the fairness is resilient to eavesdropping attacks and malleability attacks, without needing a third-party.

Furthermore, from the standpoint of efficiency, both the number of involved transactions and computation cost should be considered.

- *Number of transactions:* The number of transactions involved in BCPay should be as small as possible.
- *Computation cost:* The computation cost of BCPay should be as low as possible considering resource-constrained users.

4. BCPay: blockchain-based fair payment framework

In this section, we first present the main idea of BCPay, and then describe the design details of BCPay together with its security results.

4.1. Challenge and main idea

According to the adversary model and design goals in Section 3.3, the main challenge to design BCPay is *Robust Fairness* besides efficiency. The basic idea for realizing *Robust Fairness* is as follows.

In the service implementation phase, $\mathcal{S}$ constructs a Merkle tree based on the data from $\mathcal{C}$ and generates a signature on the root of the tree. The signature is then stored on the blockchain, which cannot be changed later, and acts as a “root of trust” in the service checking and payment. The ingredient of ensuring fairness is an **all-or-nothing** checking-proof protocol $\mathcal{CP}_{\mathrm{AON}}$. The idea of $\mathcal{CP}_{\mathrm{AON}}$ lies in two aspects:

1. $\mathcal{S}$ is able to earn the service fee from $\mathcal{C}$ and get his/her guaranty back if and only if he/she provides a valid service implementation proof, denoted as $\mathsf{ServiceProof}$;
2. If $\mathcal{S}$ fails to provide such a proof before a specific time t, $\mathcal{C}$ is able to claim from $\mathcal{S}$ either enough compensation together with his/her service fee refund or enough fines in the form of deposit.

In order to achieve these goals, in BCPay, $\mathcal{C}$ and $\mathcal{S}$ jointly create a deposit transaction $\mathsf{TxProofInit}$, which consists of the service fee from $\mathcal{C}$ and the guaranty from $\mathcal{S}$. In the normal case, $\mathsf{TxProofInit}$ can be completely redeemed by $\mathcal{S}$ based on his/her signature and $\mathsf{ServiceProof}$, and hence the *Soundness* is realized. If $\mathcal{S}$ cannot provide $\mathsf{ServiceProof}$, $\mathsf{TxProofInit}$ can be completely redeemed by $\mathcal{C}$ based on his/her signature and a secret r_S from $\mathcal{S}$. If r_S is replaced with the signature of $\mathcal{S}$, BCPay may suffer from malleability attacks. Because BCPay does not use private channels, $\mathsf{ServiceProof}$ may be eavesdropped by $\mathcal{C}$ before honest $\mathcal{S}$ gets the service fee. As a result, malicious $\mathcal{C}$ can redeem $\mathsf{TxProofInit}$ before honest $\mathcal{S}$, which violates (1) mentioned above. To overcome this problem, in BCPay, $\mathcal{S}$ just makes r_S public after redeeming $\mathsf{TxProofInit}$. Certainly, in this case, malicious $\mathcal{S}$ will not publicize r_S even if he/she fails to provide $\mathsf{ServiceProof}$, and hence $\mathcal{C}$ cannot redeem $\mathsf{TxProofInit}$ to claim compensation, which violates (2) mentioned above. To tackle this issue, in BCPay, $\mathcal{S}$ is required to make a commitment to r_S based on a deposit transaction $\mathsf{TxClaimCommitment}$. The commitment must be opened by $\mathcal{S}$ before time t to redeem $\mathsf{TxClaimCommitment}$. Otherwise, $\mathcal{C}$ can redeem $\mathsf{TxClaimCommitment}$ himself as a punishment to $\mathcal{S}$ after time t. Note that, the order among the involved transactions and the use of r_S make BCPay malleability-resistant.

In addition, to ensure the efficiency of BCPay, we aim to introduce small and constant number of transactions in a service implementation checking and proof. In fact, based on the “root of trust” constructed by $\mathcal{S}$ in the service implementation phase, $\mathcal{C}$ is able to specify the service requirements in terms of the authentication path of the Merkle tree. That is, the service implementation checking can be accomplished in one round $\mathcal{CP}_{\mathrm{AON}}$. Hence, the efficiency of BCPay is assured.

4.2. Design details of BCPay

As we know, the Bitcoin script is simple, stack-based and purposefully not Turing-complete. Unlike the Bitcoin protocol, the Ethereum is a programmable blockchain and it allows users to create their own operations of any complexity they wish [10,19,44]. In other words, the Ethereum blockchain is more flexible than the Bitcoin blockchain. However, in order to achieve easy understanding and keep the exposition simple, we present BCPay following the style of Bitcoin transactions in the same way as [1,2,6,28]. Now, we present the details of BCPay.

[2] Because the design of BCPay does not change the underlying blockchains, traditional attacks on blockchains, such as 51% attacks and Sybil attacks, are not considered in BCPay.

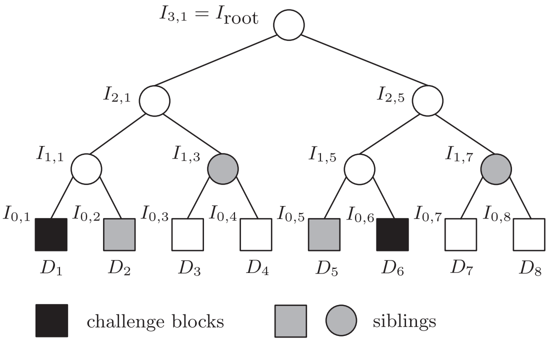


Fig. 4. The example of $\mathcal{T}_3$.

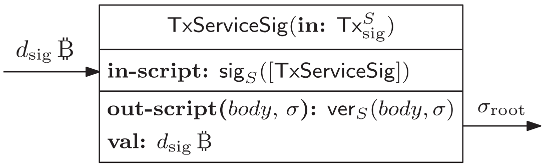

TxServiceSig(**in:** $\mathsf{Tx}^{S}_{\mathrm{sig}}$)
d_{sig} ₿
in-script: sig_S([TxServiceSig])
out-script($body$, σ): $\mathsf{ver}_S(body, \sigma)$
σ_{root}
val: d_{sig} ₿

Fig. 5. The service signature transaction TxServiceSig.

4.2.1. System setup phase

Let H be a cryptographic hash function, such as SHA-256. A secure symmetric encryption algorithm should be chosen for specific services if necessary, such as the PDP service. For simple exposition, we assume $\mathcal{C}$ and $\mathcal{S}$ choose their own ECDSA public-secret key pairs, denoted by (pk_C, sk_C) and (pk_S, sk_S), respectively. $\mathcal{S}$ prepares an unredeemed transaction $\mathsf{Tx}^{S}_{\mathrm{sig}}$ of value d_{sig} ₿, which can be redeemed with sk_S.

4.2.2. Service implementation phase

In order to realize the outsourcing service sv, the following three procedures are performed.

- *Service subscription:* $\mathcal{C}$ preprocesses service-related local data $\mathtt{data}_0$ and sends the result $\mathtt{data}_1$ to $\mathcal{S}$ for subscribing to sv. Note that the preprocessing is specified by concrete outsourcing services. For example, PDP involves encryption and hashing. Without loss of generality, suppose $\mathtt{data}_1$ consists of $n = 2^{\ell}$ data blocks and $\mathtt{data}_1 = \{D_1, D_2, \ldots, D_{2^{\ell}}\}$.
- *Service enforcement:* Upon receiving the subscription data $\mathtt{data}_1$ from $\mathcal{C}$, $\mathcal{S}$ first enforces sv based on $\mathtt{data}_1$. In order to prove the sv implementation to $\mathcal{C}$ and earn the service fee in the subsequent phases, a Merkle tree $\mathcal{T}_{\ell}$ is built by $\mathcal{S}$ after the service enforcement, where ℓ denotes the height of the tree and the leaf nodes have a height of 0. In $\mathcal{T}_{\ell}$, each interior node has a hash value. For $1 \le i \le \ell$, the jth node of height i has a value

$$I_{i,j} = H(I_{i-1,j} \parallel I_{i-1,j+2^{i-1}}),$$

where $I_{i-1,j}$ and $I_{i-1,j+2^{i-1}}$ represent the hash values of the left child and the right child of $I_{i,\,j}$, respectively. Furthermore, if $i = \ell$, $I_{i,j} = I_{\ell,1}$ is the root node, which is also denoted by I_{root}. If $1 \le i < \ell$ and $I_{i,\,j}$ is a left child, then its right sibling is $I_{i,j+2^i}$. Otherwise, $I_{i,\,j}$ is a right child and its left sibling is $I_{i,j-2^i}$. If $i = 0$, $I_{i,j} = I_{0,j}$ represents the jth leaf and $I_{0,j} = D_j$. As an example, $\mathcal{T}_3$ is shown in Fig. 4. Subsequently, $\mathcal{S}$ computes a signature $\sigma_{\mathrm{root}} = \mathsf{sig}_S(I_{\mathrm{root}})$, and stores σ_{root} on the blockchain by broadcasting a service signature transaction TxServiceSig shown in Fig. 5. Here, σ_{root} is publicly output by TxServiceSig based on the opcode OP_RETURN of Bitcoin transactions.[3] Finally, $\mathcal{S}$ sends the transaction ID to $\mathcal{C}$.
- *Preliminary service confirmation:* Upon receiving the transaction ID from $\mathcal{S}$, $\mathcal{C}$ first locates TxServiceSig on the blockchain and gets σ_{root} from OP_RETURN. Then, $\mathcal{C}$ computes I_{root} based on $\mathtt{data}_1$. If $\mathsf{vec}_S(I_{\mathrm{root}}, \sigma_{\mathrm{root}}) = \mathsf{true}$, $\mathcal{C}$ thinks sv has been preliminarily implemented. According to context of the concrete service under consideration, $\mathcal{C}$ could immediately delete $\mathtt{data}_1$ or store $\mathtt{data}_1$ till a successful service checking proof. In any case, $\mathcal{C}$ should store ℓ as metadata, which will be used to specify the service requirements.

4.2.3. Service checking phase

In this phase, $\mathcal{C}$ and $\mathcal{S}$ jointly initiate the service checking based on three sequential sub-phases: the *Challenge Generation Phase*, the *Claim Commitment Phase* and the *Proof Initiation Phase*. Suppose there is an unredeemed transaction Tx^{S}_{0} of value d_0 ₿, which can be redeemed by $\mathcal{S}$ and is used as the penalty of $\mathcal{S}$ in the case of service failure.

[3] In the Ethereum blockchain, each transaction has a data field data, which can also be used to store σ_{root} on the blockchain.

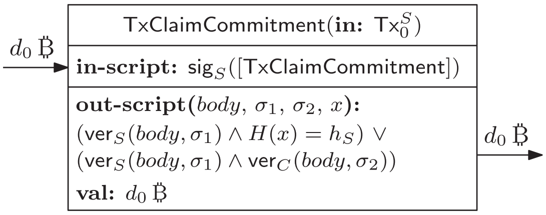

$\mathsf{TxClaimCommitment}(\mathbf{in}\text{:}\ \mathsf{Tx}_0^S)$
d_0 ₿
$\mathbf{in\text{-}script}$: $\mathsf{sig}_S([\mathsf{TxClaimCommitment}])$
$\mathbf{out\text{-}script}(body, \sigma_1, \sigma_2, x)$:
$(\mathsf{ver}_S(body, \sigma_1) \wedge H(x) = h_S) \vee$
$(\mathsf{ver}_S(body, \sigma_1) \wedge \mathsf{ver}_C(body, \sigma_2))$
$\mathbf{val}$: d_0 ₿
d_0 ₿

Fig. 6. The claim commitment transaction TxClaimCommitment.

- *Challenge generation phase:* To check the sv implementation, $\mathcal{C}$ sends a challenge chaldata to $\mathcal{S}$, which specifies the data blocks to be challenged in $\mathcal{T}_\ell$. Suppose

$$\mathsf{chaldata} = (k_1, k_2, \ldots, k_c),$$

which sequentially specifies data blocks $\{D_{k_j}\}_{1\le j\le c}$. For each $k \in \mathsf{chaldata}$, denote by path_k the path from the leaf node $I_{0,\,k}$ to the root I_{root} of $\mathcal{T}_\ell$, and by $\mathsf{AuthenPath}_k$ the authentication path of $I_{0,\,k}$. To be specific, $\mathsf{AuthenPath}_k$ consists of $I_{0,\,k}$ and the sibling nodes corresponding to $I_{0,\,k}$ and the interior nodes on path_k. Define

$$\mathsf{AuthenPath} = \bigcup_{k\in\mathsf{chaldata}} \mathsf{AuthenPath}_k - \bigcup_{k\in\mathsf{chaldata}} (\mathsf{path}_k - I_{0,k}).$$

Denote by chal the ordered version of AuthenPath such that a node with a smaller first index and a smaller second index is placed in the front. Formally, given $I_{i_1,j_1}, I_{i_2,j_2} \in \mathsf{chal}$, I_{i_1,j_1} is in front of I_{i_2,j_2} if $i_1 < i_2$ or $(i_1 = i_2 \wedge j_1 < j_2)$. In addition, denote the challenge index set by $\mathsf{ChalIndex} = \{(i,j)\}_{I_{i,j}\in\mathsf{chal}}$. For example, in Fig. 4, $\mathsf{chaldata} = (1, 6)$, and

$$\begin{aligned}
\mathsf{path}_1 &= \{I_{0,1}, I_{1,1}, I_{2,1}, I_{3,1}\},\\
\mathsf{path}_6 &= \{I_{0,6}, I_{1,5}, I_{2,5}, I_{3,1}\},\\
\mathsf{AuthenPath}_1 &= \{I_{0,1}, I_{0,2}, I_{1,3}, I_{2,5}\},\\
\mathsf{AuthenPath}_6 &= \{I_{0,6}, I_{0,5}, I_{1,7}, I_{2,1}\},\\
\mathsf{chal} &= \{I_{0,1}, I_{0,2}, I_{0,5}, I_{0,6}, I_{1,3}, I_{1,7}\},\\
\mathsf{ChalIndex} &= \{(0,1), (0,2), (0,5), (0,6), (1,3), (1,7)\}.
\end{aligned}$$

Note that ChalIndex can be computed by $\mathcal{C}$ based on the metadata ℓ without knowing chal. Furthermore, suppose there are unredeemed transactions Tx_1^S of value d_1^C ₿ and Tx_1^S of value d_1^S ₿, which can be redeemed by $\mathcal{C}$ and $\mathcal{S}$, respectively.

In order to force $\mathcal{S}$ to compensate $\mathcal{C}$ before a specific time once sv fails, let $d_0 \ge d_1^C + d_1^S$. Here, d_1^C ₿ and d_1^S ₿ denote the service fee of $\mathcal{C}$ and the compensation of $\mathcal{S}$ in the case of service failure, respectively.

- *Claim commitment phase:* Upon receiving chaldata, $\mathcal{S}$ performs $\mathsf{CS.Commit}(\mathcal{S}, \mathcal{C}, d_0, t, r_S)$, where t is a specific time and $r_S \in_R \{0,1\}^*$. Specifically, $\mathcal{S}$ posts a deposit transaction TxClaimCommitment of value d_0 ₿ on the blockchain, which makes a commitment that once the sv implementation does not meet the requirements specified in the *Proof Initiation Phase* and malicious $\mathcal{S}$ refuses[4] to compensate $\mathcal{C}$ in the *Service Payment Phase* before time t, $\mathcal{C}$ is able to claim d_0 ₿ of $\mathcal{S}$ by himself/herself as a penalty in the *Service Claim Phase* after time t. After TxClaimCommitment is included on the blockchain, $\mathcal{S}$ creates the body of the punishment transaction TxFine, which will be used by $\mathcal{C}$ to claim the penalty, signs it and sends the signed body $\mathsf{sig}_S([\mathsf{TxFine}])$ to $\mathcal{C}$. The details of TxClaimCommitment are shown in Fig. 6, where $h_S = H(r_S)$. TxOpen and TxFine will be detailed in the *Service Payment Phase* and the *Service Claim Phase*, respectively. Certainly, if the sv implementation is valid, $\mathcal{S}$ will eventually get his/her deposit back no matter how $\mathcal{C}$ behaves.
- *Proof initiation phase:* If TxClaimCommitment is included on the blockchain with enough confirmations and the signature $\mathsf{sig}_S([\mathsf{TxFine}])$ is received, $\mathcal{C}$ initiates the service proof request based on ChalIndex and σ_{root}, which can be obtained from the blockchain. Generally speaking, $\mathcal{C}$ chooses a single variable x and a variable set $\mathsf{chal}_0 = \{x_{i,j}\}_{(i,j)\in\mathsf{ChalIndex}}$, which can be chosen by $\mathcal{S}$ based on chaldata and $\mathcal{T}_\ell$. Also, $\mathcal{C}$ and $\mathcal{S}$ jointly make a deposit transaction TxProofInit, which specifies the requirements of sv implementation and is finally posted on the blockchain by $\mathcal{S}$. The idea of joint deposit has been used in [1,2]. The joint deposit in TxProofInit consists of the service fee d_1^C ₿ from $\mathcal{C}$ and the guaranty d_1^S ₿ from $\mathcal{S}$, where the guaranty is used as the compensation in the *Service Claim Phase*. Please find the details of TxProofInit in Fig. 7,

[4] Refusing to compensate means that $\mathcal{S}$ does not redeem TxClaimCommitment based on the opening transaction TxOpen before time t, that is, r_S is not revealed by $\mathcal{S}$ before time t.

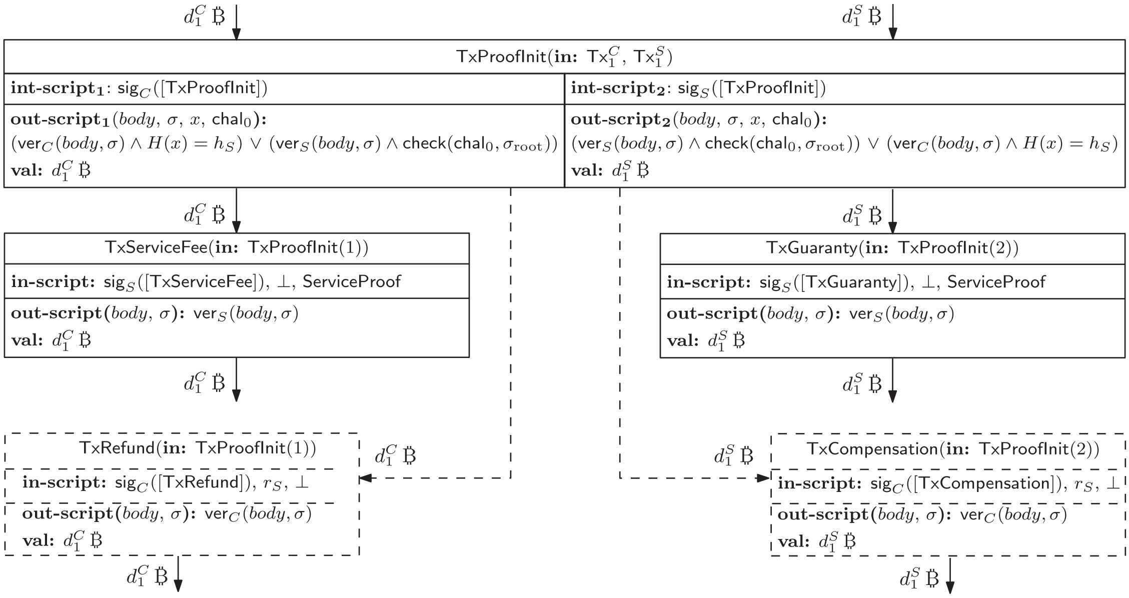

d_1^C ฿
d_1^S ฿
TxProofInit(in: Tx_1^C, Tx_1^S)
int-script$_1$: sig_C([TxProofInit])
int-script$_2$: sig_S([TxProofInit])
out-script$_1$($body$, σ, x, chal_0): ($\mathsf{ver}_C(body,\sigma) \wedge H(x) = h_S$) $\vee$ ($\mathsf{ver}_S(body,\sigma) \wedge \mathsf{check}(\mathsf{chal}_0, \sigma_{\mathrm{root}})$)
val: d_1^C ฿
out-script$_2$($body$, σ, x, chal_0): ($\mathsf{ver}_S(body,\sigma) \wedge \mathsf{check}(\mathsf{chal}_0, \sigma_{\mathrm{root}})$) $\vee$ ($\mathsf{ver}_C(body,\sigma) \wedge H(x) = h_S$)
val: d_1^S ฿
TxServiceFee(in: TxProofInit(1))
in-script: sig_S([TxServiceFee]), $\perp$, ServiceProof
out-script($body$, σ): $\mathsf{ver}_S(body,\sigma)$
val: d_1^C ฿
TxGuaranty(in: TxProofInit(2))
in-script: sig_S([TxGuaranty]), $\perp$, ServiceProof
out-script($body$, σ): $\mathsf{ver}_S(body,\sigma)$
val: d_1^S ฿
TxRefund(in: TxProofInit(1))
in-script: sig_C([TxRefund]), r_S, $\perp$
out-script($body$, σ): $\mathsf{ver}_C(body,\sigma)$
val: d_1^C ฿
TxCompensation(in: TxProofInit(2))
in-script: sig_C([TxCompensation]), r_S, $\perp$
out-script($body$, σ): $\mathsf{ver}_C(body,\sigma)$
val: d_1^S ฿

Fig. 7. The transactions involved in the service implementation checking and proof of BCPay.

$\mathcal{CP}_{\mathrm{AON}}(\mathcal{C}, \mathcal{S}, \{\mathsf{Tx}_1^C, d_1^C\}, \{\mathsf{Tx}_1^S, d_1^S\}, \sigma_{\mathrm{root}}, h_S)$:

$\mathcal{C}$ and $\mathcal{S}$ do the following:

(1) Both $\mathcal{C}$ and $\mathcal{S}$ compute the body of the transaction TxProofInit using Tx_1^C and Tx_1^S as inputs.

(2) $\mathcal{C}$ signs the transaction TxProofInit and sends the signature sig_C([TxProofInit]) to $\mathcal{S}$. According to the definition of check, $\mathcal{C}$ does not need to know I_{root}.

(3) If $\mathcal{S}$ does not receive sig_C([TxProofInit]) before time $t - 4\max_B$, where $\max_B$ is the maximal delay between broadcasting a transaction and including it on the blockchain, then $\mathcal{S}$ redeems Tx_1^S and quits. Otherwise, $\mathcal{S}$ signs TxProofInit and broadcasts it.

(4) $\mathcal{C}$ waits until TxProofInit is included on the blockchain. If TxProofInit is not included on the blockchain before time $t - 3\max_B$, $\mathcal{C}$ immediately redeems Tx_1^C and quits.

(5) $\mathcal{S}$ provides the service proof ServiceProof, and broadcasts TxServiceFee and TxGuaranty to earn the service fee d_1^C ฿ and get back the guaranty d_1^S ฿, respectively.

(6) If ServiceProof cannot pass check before time $t - 2\max_B$, $\mathcal{C}$ performs the *Service Claim Phase*.

(7) Otherwise, ServiceProof passes check before time $t - 2\max_B$, and hence $\mathcal{C}$ obtains a valid outsourcing service from $\mathcal{S}$. $\mathcal{C}$ quits and $\mathcal{S}$ opens his/her commitment.

Fig. 8. The all-or-nothing checking-proof protocol $\mathcal{CP}_{\mathrm{AON}}$.

in which $\mathsf{check}(\mathsf{chal}_0, \sigma_{\mathrm{root}}) \triangleq \mathsf{vec}_S(I^*_{\mathrm{root}}, \sigma_{\mathrm{root}})$ and I^*_{root} is computed based on chal_0 in the same way as I_{root} is computed based on chal according to the construction of $\mathcal{T}_\ell$. Obviously, hashing and ECDSA signature verification are involved in the output script. More details of the service proof are given based on a checking-proof protocol $\mathcal{CP}_{\mathrm{AON}}$ performed by $\mathcal{C}$ and $\mathcal{S}$, which is described in Fig. 8 and additionally involves the *Service Payment Phase* and the *Service Claim Phase*. We call $\mathcal{CP}_{\mathrm{AON}}$ an **all-or-nothing** protocol in the sense that either the service fee and the guaranty are redeemed by $\mathcal{S}$ at the same time or more deposit of $\mathcal{S}$ will be paid to $\mathcal{C}$.

4.2.4. Service payment phase

In this phase, if $\mathcal{S}$ can provide a valid proof ServiceProof before the specific time $t - 2\max_B$ to prove that the sv implementation meets the requirements, $\mathcal{S}$ can earn the service fee d_1^C ฿ of $\mathcal{C}$ and get his/her guaranty d_1^S ฿ back by redeeming TxProofInit based on TxServiceFee and TxGuaranty, respectively. The transactions TxServiceFee and TxGuaranty are shown in Fig. 7. Furthermore, $\mathcal{S}$ performs $\mathsf{CS.Open}(\mathcal{S}, \mathcal{C}, d_0, t, r_S)$, in which $\mathcal{S}$ opens the claim commitment made in the *Claim Commitment Phase* by posting the opening transaction TxOpen on the blockchain before time t. The details of TxOpen are shown in Fig. 9. Note that TxOpen redeems TxClaimCommitment and hence $\mathcal{S}$ can get his/her commitment deposit back. Finally, $\mathcal{S}$ quits.

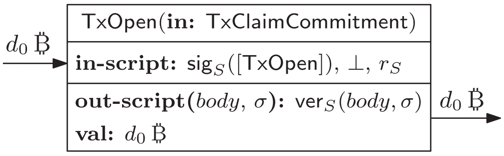

d_0 ₿
TxOpen(**in:** TxClaimCommitment)
in-script: $\mathsf{sig}_S([\mathsf{TxOpen}]), \perp, r_S$
out-script$(body, \sigma)$**:** $\mathsf{ver}_S(body, \sigma)$
val: d_0 ₿
d_0 ₿

Fig. 9. The opening transaction TxOpen.

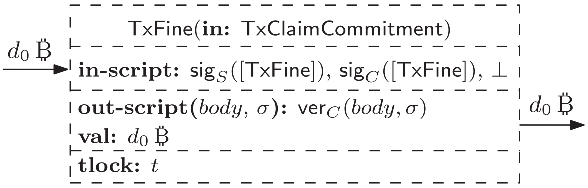

d_0 ₿
TxFine(**in:** TxClaimCommitment)
in-script: $\mathsf{sig}_S([\mathsf{TxFine}]), \mathsf{sig}_C([\mathsf{TxFine}]), \perp$
out-script$(body, \sigma)$**:** $\mathsf{ver}_C(body, \sigma)$
val: d_0 ₿
tlock: t
d_0 ₿

Fig. 10. The punishment transaction TxFine.

4.2.5. Service claim phase

Suppose $\mathcal{S}$ fails to provide a valid service implementation proof ServiceProof in the *Service Payment Phase* before time $t - 2\max_B$, BCPay comes to the *Service Claim Phase*. In this phase, $\mathcal{C}$ is able to get enough deposit from $\mathcal{S}$ no matter how $\mathcal{S}$ behaves. Two cases should be taken into account.

- *Case 1.* $\mathcal{S}$ refuses to pay the compensation d_1^S ₿ to $\mathcal{C}$, that is, $\mathcal{S}$ does not open the claim commitment made in the *Claim Commitment Phase* and TxOpen is not included on the blockchain before time t. In this case, $\mathcal{C}$ performs CS.Fine$(\mathcal{S}, \mathcal{C}, d_0, t, r_S)$, in which $\mathcal{C}$ gets the penalty d_0 ₿ by posting the punishment transaction TxFine on the blockchain, and then quits. The detail of TxFine is given in Fig. 10.
- *Case 2.* $\mathcal{S}$ refuses to pay the penalty d_0 ₿ to $\mathcal{C}$, that is, $\mathcal{S}$ opens the claim commitment by posting the opening transaction TxOpen on the blockchain before time t. In this case, $\mathcal{C}$ gets both the refund d_1^C ₿ and the compensation d_1^S ₿ by immediately posting the refund transaction TxRefund and the compensation transaction TxCompensation on the blockchain, respectively. Then $\mathcal{C}$ quits. The details of TxRefund and TxCompensation are given in Fig. 7.

4.3. Security analysis

In this section, we present the results of security analysis of BCPay in Theorems 1 and 2. As mentioned before, eavesdropping attacks and malleability attacks are considered and no third-party is involved in BCPay.

Theorem 1. *Based on the collision-resistance of the adopted hash function H and the unforgeability of ECDSA, BCPay satisfies the property of soundness.*

Proof. Suppose both $\mathcal{C}$ and $\mathcal{S}$ are honest and they follow the procedures of BCPay. We show that even outside adversaries make eavesdropping attacks and malleability attacks, $\mathcal{C}$ and $\mathcal{S}$ will always obtain the required service implementation and the corresponding service fee at the end, respectively. As a matter of fact, in the service enforcement procedure of the service implementation phase, $\mathcal{S}$ computes a signature σ_{root} which is stored on the blockchain by broadcasting the service signature transaction TxServiceSig. After a challenge is generated by $\mathcal{C}$ in the challenge generation phase, $\mathcal{S}$ makes a commitment based on CS.Commit. Subsequently, $\mathcal{C}$ and $\mathcal{S}$ perform the all-or-nothing checking-proof protocol $\mathcal{CP}_{\text{AON}}$, in which only the proof initiation phase and the service payment phase are involved if both parties are honest. In the proof initiation phase, after $\mathcal{C}$ initiates the service proof based on σ_{root}, $\mathcal{C}$ and $\mathcal{S}$ make a joint deposit transaction TxProofInit, which is finally posted on the blockchain by $\mathcal{S}$. In the service payment phase, $\mathcal{S}$ provides a service implementation proof ServiceProof to earn the service fee from $\mathcal{C}$ and get his/her current guaranty back by redeeming TxProofInit based on TxServiceFee and TxGuaranty, respectively. According to the definitions of check and ServiceProof in Section 4.2, if a ServiceProof, which is deduced by outside adversaries based on eavesdropping attacks and malleability attacks, can pass check, then either a hash collision is found or ECDSA is forgeable. In other words, if the adopted hash function is collision-resistant and ECDSA is unforgeable, it is ensured that check has the value true only if ServiceProof meets the service requirements specified in the proof initiation phase. Therefore, if $\mathcal{C}$ and $\mathcal{S}$ are honest and follow the procedures of BCPay, they will always obtain the required service implementation and the corresponding service fee, respectively. □

Theorem 2. *BCPay satisfies the property of robust fairness without needing a third-party if the adopted hash function H is collision-resistant and ECDSA is unforgeable.*

Proof. As mentioned in Section 3.3, no private channels are required in BCPay. So, eavesdropping attacks and malleability attacks may be made by a malicious party to undermine the fairness for the honest party. In the following, we first prove the robust fairness for $\mathcal{C}$ against malicious $\mathcal{S}$, and then consider the robust fairness for $\mathcal{S}$ in the case of malicious $\mathcal{C}$.

Case 1. Suppose $\mathcal{C}$ is honest and $\mathcal{S}$ is malicious. In this case, $\mathcal{S}$ aims to get the service fee from $\mathcal{C}$ without providing a valid service implementation proof in terms of the requirements specified by $\mathcal{C}$ before time $t - 2\max_B$. At the same time, $\mathcal{S}$ is reluctant to pay compensation and penalty to $\mathcal{C}$. Assume that $\mathcal{C}$ does not get a valid service implementation proof from $\mathcal{S}$ in terms of his/her requirements before time t, which means the service implementation proof ServiceProof is invalid. Hence, the joint deposit transaction TxProofInit cannot be redeemed by $\mathcal{S}$ based on TxServiceFee and TxGuaranty before time $t - 2\max_B$ in the service payment phase. According to the definitions of check and ServiceProof in Section 4.2, we know $\mathcal{S}$ cannot get the service fee d_1^C ₿ from $\mathcal{C}$ unless $\mathcal{S}$ is able to forge an ECDSA signature or find a collision of the hash function H. Furthermore, $\mathcal{S}$ may make malleability attacks by eavesdropping transactions on the public channel. However, the attacks are meaningless because the transactions involved in BCPay are posted on the blockchain in order and $\mathcal{C}$ is still able to claim enough compensation or penalty from $\mathcal{S}$. Please refer to Figs. 7 and 8 for more details.

Specifically, if $\mathcal{S}$ refuses to pay the compensation d_1^S ₿ to $\mathcal{C}$, which means $\mathcal{S}$ does not open the claim commitment made in the claim commitment phase by broadcasting TxOpen based on CS.Open before time t, $\mathcal{C}$ performs CS.Fine, in which $\mathcal{C}$ gets the penalty d_0 ₿ with $d_0 \geq d_1^C + d_1^S$ by posting the punishment transaction TxFine on the blockchain. If $\mathcal{S}$ refuses to pay the penalty to $\mathcal{C}$, which means $\mathcal{S}$ opens the claim commitment by performing CS.Open to post TxOpen on the blockchain before time t, $\mathcal{C}$ can claim both the refund d_1^C ₿ and the compensation d_1^S ₿ by posting TxRefund and TxCompensation on the blockchain, respectively. Accordingly, in any case, if malicious $\mathcal{S}$ fails to provide a valid service implementation proof, $\mathcal{C}$ is able to claim enough compensation besides the service fee refund or penalty from $\mathcal{S}$ no matter how $\mathcal{S}$ behaves.

Generally speaking, the robust fairness for $\mathcal{C}$ is ensured in BCPay without needing a third-party if the hash function H is collision-resistant and ECDSA is unforgeable.

Case 2. Suppose $\mathcal{S}$ is honest and $\mathcal{C}$ is malicious. In this case, $\mathcal{C}$ aims to obtain a valid service implementation proof in terms of his/her requirements before time $t - 2\max_B$ without paying the corresponding service fee to $\mathcal{S}$. Assume that $\mathcal{S}$ provides a valid service implementation proof in terms of the requirements of $\mathcal{C}$ before time t. It follows that the service implementation proof ServiceProof is valid. According to the details of BCPay, $\mathcal{C}$ only puts service fees in the generation of the joint deposit transaction TxProofInit, which can be successfully redeemed by $\mathcal{S}$ in the service payment phase based on TxServiceFee and TxGuaranty before time $t - 2\max_B$ only if ServiceProof is valid. In fact, malicious $\mathcal{C}$ may try to eavesdrop TxServiceFee and TxGuaranty on the public channel to get the service proof ServiceProof together with sig_S([TxServiceFee]) and sig_S([TxGuaranty]), respectively. After that, $\mathcal{C}$ mauls the joint deposit transaction TxProofInit to prevent $\mathcal{S}$ from earning the corresponding service fee. As we know, however, the service payment phase is behind the proof initiation phase in BCPay, hence this malleability attack is meaningless.

On the other hand, malicious $\mathcal{C}$ may try to claim compensation or penalty from $\mathcal{S}$ after ensuring that the service proof is valid in terms of his/her requirements. Obviously, it is infeasible for $\mathcal{C}$ to claim compensation from $\mathcal{S}$ because TxGuaranty has been posted on the blockchain. In particular, $\mathcal{C}$ cannot redeem TxProofInit before $\mathcal{S}$ unless he/she finds a collision of H or forges an ECDSA signature. According to the service payment phase of BCPay, CS.Open is immediately performed by $\mathcal{S}$ to open the claim commitment and hence to get the punishment deposit back before time t. From the property of transaction lock-time, it follows that $\mathcal{C}$ cannot get a penalty from $\mathcal{S}$ even if malleability attacks are made.

Therefore, the robust fairness for $\mathcal{S}$ is ensured in BCPay without needing a third-party if H is collision-resistant and ECDSA is unforgeable. □

5. Performance evaluation

In this section, we evaluate the performance of our proposed BCPay in terms of the number of involved transactions and computation cost.

5.1. Number of transactions

As for BCPay, in the *Service Implementation Phase*, only one transaction TxServiceSig is required. In the *Service Checking Phase*, transactions TxClaimCommitment and TxProofInit are involved. In the *Service Payment Phase*, transactions TxOpen, TxServiceFee and TxGuaranty are needed. In the *Service Claim Phase*, either the transaction TxFine or transactions TxRefund and TxCompensation are created. Note that TxServiceFee and TxGuaranty can be replaced with one transaction because they only need signatures of the server. Similarly, TxRefund and TxCompensation can also be combined into one transaction. Accordingly, as shown in Fig. 11, the number of involved transactions is small and constant and it is affected neither by the height of the data tree nor by the number of challenge data blocks.

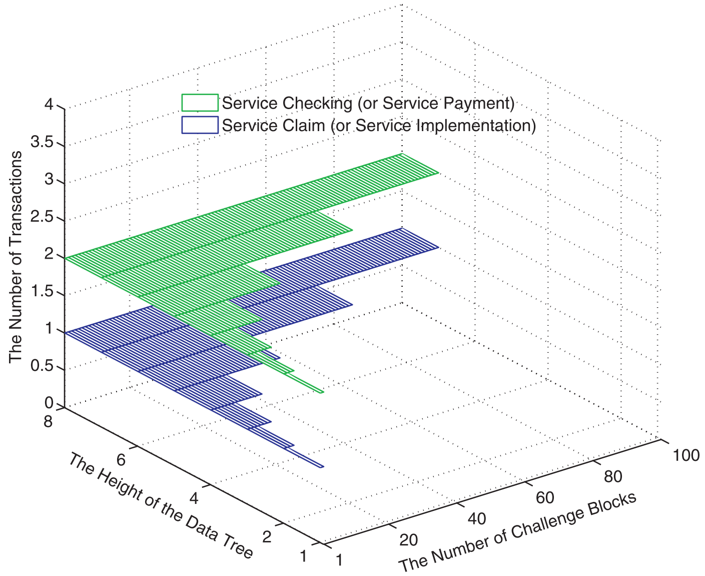


Fig. 11. The number of transactions in BCPay.

5.2. Computation cost

In BCPay, the most common operations are hashing and ECDSA signature operations. Considering that the computation cost of a hashing is far less than that of an ECDSA signature, we take ECDSA signature into account in the following. In our experiments, we evaluate the computation time of the ECDSA signature used in transactions on a virtual machine (3.6 GHz single-core processor and 6 GB DDR3-1600 RAM memory) based on Ubuntu 16.04 LTS and OpenSSL 1.0.2g. In particular, a specific elliptic curve called secp256k1 with the equation $y^2 = x^3 + 7$ is adopted, which is used by Bitcoin and can also be used in Ethereum. Additionally, in the following figures, we display the computation time with data trees of height 7, 10, 13, and 16, respectively. In any case, the number of challenge data blocks can reach 100 ($< 2^7$).

BCPay is very efficient because the computation cost is not related to the height of the data tree and the number of challenge data blocks. If the server is honest, the client only participates in creating the transaction TxProofInit in the *Service Checking Phase*, and hence only one ECDSA signature is needed. The computation time of the client is presented in Fig. 12(a). On the other hand, if the server is malicious, the client only computes one ECDSA signature in the *Service Claim Phase*. The corresponding claim time for the client is presented in Fig. 12(b). In BCPay, the server creates TxServiceSig, TxClaimCommitment, TxOpen, TxProofInit, TxServiceFee and TxGuaranty. Note that, even if TxServiceFee and TxGuaranty are combined, the number of ECDSA signatures is not reduced. In addition, the creation of TxFine also needs a signature of the server. Therefore, the server has to perform 7 ECDSA signature operations in BCPay. Computation time of the server is presented in Fig. 12(c).

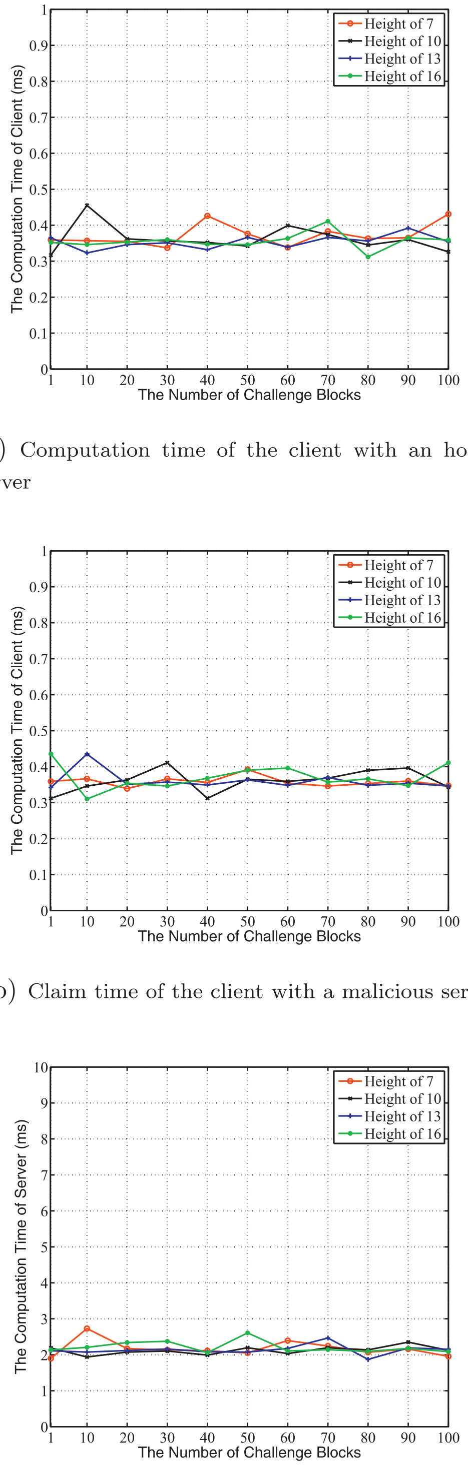
6. Decentralized applications of BCPay

BCPay is a blockchain-based fair payment framework. In this section, we show how to realize two important decentralized applications based on BCPay.

6.1. Blockchain-based PDP

In the case of sv= PDP, according to the details of BCPay, we only need to display the *Service Implementation Phase*, which is implemented based on the following three procedures.

- *Service subscription:* Let $\mathsf{data}_0 = \{F_1, F_2, \ldots, F_{2^\ell}\}$ be the plaintext data collection. $\mathcal{C}$ encrypts data_0 based on a symmetric encryption algorithm and sends the resulting ciphertext data collection data_1 to $\mathcal{S}$ for subscribing the PDP service. Suppose $\mathsf{data}_1 = \{D_1, D_2, \ldots, D_{2^\ell}\}$ in which D_k is the ciphertext of F_k for $1 \le k \le 2^\ell$.
- *Service enforcement:* Upon receiving the subscription data data_1 from $\mathcal{C}$, $\mathcal{S}$ constructs a Merkle tree $\mathcal{T}_\ell$.
Subsequently, $\mathcal{S}$ computes $\sigma_{\text{root}} = \mathsf{sig}_{\mathcal{S}}(h_r)$, and stores σ_{root} on the blockchain by broadcasting TxServiceSig as shown in Fig. 5. Finally, $\mathcal{S}$ sends the transaction ID to $\mathcal{C}$.
- *Preliminary service confirmation:* Upon receiving ID from $\mathcal{S}$, $\mathcal{C}$ first locates TxServiceSig on the blockchain and gets σ_{root}. Then, $\mathcal{C}$ computes h_r based on data_1. If $\mathsf{vec}_{\mathcal{S}}(h_r, \sigma_{\text{root}})$ evaluates to true, $\mathcal{C}$ stores the height ℓ of the Merke tree. Based on



(a) Computation time of the client with an honest server

(b) Claim time of the client with a malicious server

(c) Computation time of the server

Fig. 12. Computation time in BCPay.

his/her practical outsourcing strategies, such as redundant outsourcing, $\mathcal{C}$ immediately deletes data_1 or stores data_1 until a successful service checking proof.

Note that a challenge-response mechanism is needed in the traditional PDP. In blockchain-based PDP, $\mathcal{C}$ can challenge $\mathcal{S}$ based on the *Service Checking Phase* of BCPay for data integrity. $\mathcal{S}$ can response to $\mathcal{C}$ based on the *Service Payment Phase*. To further support data dynamics, the user only needs to store the structure of the Merkle tree as metadata in *Preliminary Service Confirmation*.

6.2. BCOC: blockchain-based outsourcing computation

In this section, we show the application of BCPay in outsourcing computation, and propose a blockchain-based outsourcing computation scheme, denoted as BCOC. BCOC can be used in fog computing, where a fog user with limited resources wants to outsource distributed computation tasks to the fog node. For consistency, we use $\mathcal{C}$ and $\mathcal{S}$ to represent the fog user and the fog node, respectively. Based on the definition in [26], a distributed computation involves a function, a screener and a payment scheme. However, a trusted third-party is introduced in [26]. In BCOC, we realize both the screener and the payment based on blockchain and no third-party is required. Formally, let f be a one-way function from X to Y, denoted as f: $X \mapsto Y$. Suppose $y^* = f(x)$ for $x \in X$. Note that multiple such x may exist. Given f and y^* only, the objective of the computation is to discover all such x by exhaustive search of the domain X. According to the design details of BCPay, we show the outsourcing of inverting a hash function. Based on the original procedures of BCPay, $\mathcal{C}$ and $\mathcal{S}$ further perform the following.

- *System setup phase:* $\mathcal{C}$ specifies a task $\mathsf{task} = (f, X, y^*)$, where $X = \{x_1, x_2, \ldots, x_N\}$.
- *Service implementation phase:*
  - *Service subscription:* $\mathcal{C}$ sends task to $\mathcal{S}$.
  - *Service enforcement:* For $1 \le i \le N$, $\mathcal{S}$ computes $y_i = f(x_i)$. Without loss of generality, suppose

    $$\{x \in X | f(x) = y^*\} = \{x_1, x_2, \ldots, x_{2^\ell}\} \triangleq X^*,$$

    where $\ell \le \log N$. $\mathcal{S}$ constructs a Merkle tree based on X^* as before and sends ℓ to $\mathcal{C}$.
  - *Preliminary service confirmation:* $\mathcal{C}$ stores ℓ.
- *Service checking phase:*
  - *Challenge generation phase:* As before.
  - *Claim commitment phase:* As before.
  - *Proof initiation phase:* $\mathcal{C}$ and $\mathcal{S}$ jointly create $\mathsf{TxProofInit}$ based on check and y^* by putting $H(x'_k) = y^*$ for $k \in \mathsf{chaldata}$ in the output script, where the value of x'_k is from x_k provided by $\mathcal{S}$ in the *Service Payment Phase*. That is, the service proof provided by $\mathcal{S}$ in the *Service Payment Phase* should satisfy the basic correctness requirement besides check.
- *Service payment phase:* $\mathcal{S}$ provides $\{x_k\}_{k \in \mathsf{chaldata}}$ besides $\mathsf{ServiceProof}$.
- *Service claim phase:* As before.

7. Conclusion

In this paper, we introduced BCPay, a blockchain based fair payment framework for outsourcing services in cloud computing. Specifically, we presented the system architecture, specifications and adversary model, and described the design details of BCPay. Our security analysis indicated that BCPay enjoys *Soundness* and *Robust Fairness*. Our performance analysis showed that BCPay is very efficient in terms of the number of involved transactions and computation cost. To illustrate the applications of BCPay, we presented a blockchain-based PDP scheme and a blockchain-based outsourcing computation protocol based on BCPay.

Acknowledgments

We are grateful to the editors and reviewers for their invaluable suggestions. This research is supported by the National Key R&D Program of China (2017YFB0802000), the AXA Research Fund, the National Natural Science Foundation of China (Nos. 61772418, 61472472, 61402366), the Natural Science Basic Research Plan in Shaanxi Province of China (No. 2015JQ6236). Yinghui Zhang is supported by New Star Team of Xi'an University of Posts and Telecommunications (No. 2016-02).

References

[1] M. Andrychowicz, S. Dziembowski, D. Malinowski, L. Mazurek, Fair two-party computations via bitcoin deposits, in: International Conference on Financial Cryptography and Data Security (FC), Springer, 2014, pp. 105–121.

[2] M. Andrychowicz, S. Dziembowski, D. Malinowski, L. Mazurek, Secure multiparty computations on bitcoin, in: IEEE Symposium on Security and Privacy (SP), IEEE, 2014, pp. 443–458.

[3] M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, M. Zaharia, A view of cloud computing, Commun. ACM 53 (4) (2010) 50–58.

[4] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, D. Song, Provable data possession at untrusted stores, in: Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS), ACM, 2007, pp. 598–609.

[5] G. Ateniese, R. Di Pietro, L.V. Mancini, G. Tsudik, Scalable and efficient provable data possession, in: Proceedings of the 4th International Conference on Security and Privacy in Communication Netowrks (SecureComm), ACM, 2008, pp. 1–10.

[6] G. Ateniese, M.T. Goodrich, V. Lekakis, C. Papamanthou, E. Paraskevas, R. Tamassia, Accountable storage, in: International Conference on Applied Cryptography and Network Security (ACNS), Springer, 2017, pp. 623–644.

[7] G. Ateniese, S. Kamara, J. Katz, Proofs of storage from homomorphic identification protocols, in: International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT), Springer, 2009, pp. 319–333.

[8] I. Bentov, R. Kumaresan, How to use bitcoin to design fair protocols, in: International Cryptology Conference (CRYPTO), Springer, 2014, pp. 421–439.

[9] F. Bonomi, R. Milito, J. Zhu, S. Addepalli, Fog computing and its role in the internet of things, in: Proceedings of the first edition of the MCC workshop on Mobile cloud computing, ACM, 2012, pp. 13–16.

[10] V. Buterin, A Next-Generation Smart Contract and Decentralized Application Platform. White paper, 2014, pp. 1–36.

[11] M. Campanelli, R. Gennaro, S. Goldfeder, L. Nizzardo, Zero-knowledge contingent payments revisited: attacks and payments for services, in: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS), ACM, 2017, pp. 229–243.

[12] B. Carbunar, M.V. Tripunitara, Payments for outsourced computations, IEEE Trans. Parallel Distrib. Syst. 23 (2) (2012) 313–320.

[13] X. Chen, J. Li, X. Huang, J. Ma, W. Lou, New publicly verifiable databases with efficient updates, IEEE Trans. Dependable Secure Comput. 12 (5) (2015) 546–556.

[14] X. Chen, J. Li, J. Ma, W. Lou, D.S. Wong, New and efficient conditional e-payment systems with transferability, Fut. Gen. Comput. Syst. 37 (2014) 252–258.

[15] X. Chen, J. Li, J. Ma, Q. Tang, W. Lou, New algorithms for secure outsourcing of modular exponentiations, IEEE Trans. Parallel Distrib. Syst. 25 (9) (2014) 2386–2396.

[16] X. Chen, J. Li, W. Susilo, Efficient fair conditional payments for outsourcing computations, IEEE Trans. Inf. Forensics Secur. 7 (6) (2012) 1687–1694.

[17] X. Chen, J. Li, J. Weng, J. Ma, W. Lou, Verifiable computation over large database with incremental updates, IEEE Trans. Comput. 65 (10) (2016) 3184–3195.

[18] X. Chen, F. Zhang, W. Susilo, H. Tian, J. Li, K. Kim, Identity-based chameleon hashing and signatures without key exposure, Inf. Sci. 265 (2014) 198–210.

[19] E. Community, Ethereum homestead documentation, 2016, Online document. http://ethdocs.org/en/latest/index.html.

[20] G. Developers, Google cloud platform, 2017, Online document. https://cloud.google.com/free/docs/frequently-asked-questions.

[21] C. Dong, Y. Wang, A. Aldweesh, P. Mc Corry, A. vanMoorsel, Betrayal, distrust, and rationality: smart counter-collusion contracts for verifiable cloud computing, in: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS), ACM, 2017, pp. 211–227.

[22] W. Du, J. Jia, M. Mangal, M. Murugesan, Uncheatable grid computing, in: Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS), IEEE Computer Society, 2004, pp. 4–11.

[23] U. Feige, A. Shamir, Witness indistinguishable and witness hiding protocols, in: Proceedings of the twenty-second annual ACM symposium on Theory of computing, ACM, 1990, pp. 416–426.

[24] C. Gao, Q. Cheng, P. He, W. Susilo, J. Li, Privacy-preserving Naive Bayes classifiers secure against the substitution-then-comparison attack, Inf. Sci. 444 (2018) 72–88.

[25] R. Gennaro, C. Gentry, B. Parno, Non-interactive verifiable computing: outsourcing computation to untrusted workers, in: Annual Cryptology Conference (CRYPTO), Springer, 2010, pp. 465–482.

[26] P. Golle, I. Mironov, Uncheatable distributed computations, in: Cryptographers' Track at the RSA Conference (CT-RSA), Springer, 2001, pp. 425–440.

[27] K. Hardson, Monitoring at dropbox, 2017, Online document. https://www.usenix.org/conference/srecon17asia/program/presentation/hardson-hurley.

[28] H. Huang, X. Chen, Q. Wu, X. Huang, J. Shen, Bitcoin-based fair payments for outsourcing computations of fog devices, Fut. Gen. Comput. Syst. 78 (2018) 850–858.

[29] Z. Huang, S. Liu, X. Mao, K. Chen, J. Li, Insight of the protection for data security under selective opening attacks, Inf. Sci. 412 (2017) 223–241.

[30] A. Juels, B.S. Kaliski Jr., Pors: proofs of retrievability for large files, in: Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS), ACM, 2007, pp. 584–597.

[31] R. Khalil, A. Gervais, Revive: rebalancing off-blockchain payment networks, in: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS), ACM, 2017, pp. 439–453.

[32] B. Li, Y. Huang, Z. Liu, J. Li, Z. Tian, S.M. Yiu, Hybridoram: practical oblivious cloud storage with constant bandwidth, Inf. Sci. (2018). Online publication. 10.1016/j.ins.2018.02.019

[33] J. Li, X. Huang, J. Li, X. Chen, Y. Xiang, Securely outsourcing attribute-based encryption with checkability, IEEE Trans. Parallel Distrib. Syst. 25 (8) (2014) 2201–2210.

[34] J. Li, Y. Zhang, X. Chen, Y. Xiang, Secure attribute-based data sharing for resource-limited users in cloud computing, Comput. Secur. 72 (2018) 1–12.

[35] T. Li, J. Li, Z. Liu, P. Li, C. Jia, Differentially private naive bayes learning over multiple data sources, Inf. Sci. 444 (2018) 89–104.

[36] G. Malavolta, P. Moreno-Sanchez, A. Kate, M. Maffei, S. Ravi, Concurrency and privacy with payment-channel networks, in: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS), ACM, 2017, pp. 455–471.

[37] W. Meng, E. Tischhauser, Q. Wang, Y. Wang, J. Han, When intrusion detection meets blockchain technology: a review, IEEE Access 6 (2018) 10179–10188.

[38] P. Merriam, Ethereum alarm clock, 2015, Online document. http://docs.ethereum-alarm-clock.com/en/latest/.

[39] S. Nakamoto, Bitcoin: a peer-to-peer electronic cash system, 1–9, 2008, Online document. http://bitcoin.org/bitcoin.pdf.

[40] Q. Lin, H. Yan, Z. Huang, W. Chen, J. Shen, Y. Tang, An id-based linearly homomorphic signature scheme and its application in blockchain, IEEE Access 6 (2018) 20632–20640.

[41] J. Shen, Z. Gui, S. Ji, J. Shen, H. Tan, Y. Tang, Cloud-aided lightweight certificateless authentication protocol with anonymity for wireless body area networks, J. Netw. Comput. Appl. 106 (2018) 117–123.

[42] W. Song, B. Wang, Q. Wang, C. Shi, W. Lou, Z. Peng, Publicly verifiable computation of polynomials over outsourced data with multiple sources, IEEE Trans. Inf. Forensics Secur. 12 (10) (2017) 2334–2347.

[43] Q. Wang, C. Wang, K. Ren, W. Lou, J. Li, Enabling public auditability and data dynamics for storage security in cloud computing, IEEE Trans. Parallel Distrib. Syst. 22 (5) (2011) 847–859.

[44] G. Wood, Ethereum: a secure decentralised generalised transaction ledger, in: Ethereum Project Yellow Paper, vol. 151, 2014, pp. 1–32. Online document. http://www.cryptopapers.net/papers/ethereum-yellowpaper.pdf.

[45] J. Xu, L. Wei, Y. Zhang, A. Wang, F. Zhou, C.z. Gao, Dynamic fully homomorphic encryption-based merkle tree for lightweight streaming authenticated data structures, J. Netw. Comput. Appl. 107 (2018) 113–124.

[46] H. Yan, J. Li, J. Han, Y. Zhang, A novel efficient remote data possession checking protocol in cloud storage, IEEE Trans. Inf. Forensics Secur. 12 (1) (2017) 78–88.

[47] X. Zhang, Y.A. Tan, C. Liang, Y. Li, J. Li, A covert channel over volte via adjusting silence periods, IEEE Access 6 (1) (2018) 9292–9302.

[48] Y. Zhang, X. Chen, J. Li, D.S. Wong, H. Li, I. You, Ensuring attribute privacy protection and fast decryption for outsourced data security in mobile cloud computing, Inf. Sci. 379 (2017) 42–61.

[49] Y. Zhang, J. Li, X. Chen, H. Li, Anonymous attribute-based proxy re-encryption for access control in cloud computing, Secur. Commun. Netw. 9 (14) (2016) 2397–2411.

[50] Y. Zhang, D. Zheng, X. Chen, J. Li, H. Li, Efficient attribute-based data sharing in mobile clouds, Pervasive Mob. Comput. 28 (2016) 135–149.

[51] Y. Zhang, D. Zheng, R.H. Deng, Security and privacy in smart health: efficient policy-hiding attribute-based access control, IEEE Internet Things J. 5 (3) (2018) 2130–2145.