Blockchain-based Fair Payment Protocols for Cloud Services

Submitted in partial fulfillment of the requirements

for the award of the degree of

DOCTOR OF PHILOSOPHY

Submitted by

Dorsala Mallikarjun Reddy

(Roll No. 716042)

Under the guidance of

Prof. V. N. Sastry

and

Dr. Chapram Sudhakar

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

NATIONAL INSTITUTE OF TECHNOLOGY WARANGAL

TELANGANA - 506004, INDIA

December 2021

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
NATIONAL INSTITUTE OF TECHNOLOGY WARANGAL
TELANGANA - 506004, INDIA

THESIS APPROVAL FOR Ph.D.

This is to certify that the thesis entitled, Blockchain-based Fair Payment Protocols for Cloud Services, submitted by Mr. Dorsala Mallikarjun Reddy [Roll No. 716042] is approved for the degree of DOCTOR OF PHILOSOPHY at National Institute of Technology Warangal.

Examiner

Research Supervisor
Prof. V. N. Sastry
Professor
Center for Mobile Banking
Institute for Development &
Research in Banking Technology
India

Research Supervisor
Dr. Chapram Sudhakar
Associate Professor
Dept. of Computer Science and Engg.
NIT Warangal
India

Chairman
Prof. P. Radha Krishna
Head, Dept. of Computer Science and Engg.
NIT Warangal
India

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
NATIONAL INSTITUTE OF TECHNOLOGY WARANGAL
TELANGANA - 506004, INDIA

CERTIFICATE

This is to certify that the thesis entitled, Blockchain-based Fair Payment Protocols for Cloud Services, submitted in partial fulfillment of requirement for the award of degree of DOCTOR OF PHILOSOPHY to National Institute of Technology Warangal, is a bonafide research work done at the Center for Mobile Banking (CMB), IDRBT by Mr. Dorsala Mallikarjun Reddy [Roll No. 716042] under our supervision during January 2016 to December 2021. The contents of the thesis have not been submitted elsewhere for the award of any degree.

[illegible]

Research Supervisor
Prof. V. N. Sastry
Professor

Center for Mobile Banking
Institute for Development &
Research in Banking Technology
India

Hyderabad
Date: 15-12-2021

[illegible]

Research Supervisor
Dr. Chapram Sudhakar
Associate Professor

Dept. of Computer Science and Engg.
NIT Warangal
India

Warangal
Date: 15-12-2021

DECLARATION

This is to certify that the work presented in the thesis entitled "*Blockchain-based Fair Payment Protocols for Cloud Services*" is a bonafide work done by me under the supervision of Prof. V. N. Sastry and Dr. Chapram Sudhakar. The work was not submitted elsewhere for the award of any degree.

I declare that this written submission represents my ideas in my own words and where others ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea / date / fact / source in my submission. I understand that any violation of the above will be cause for disciplinary action by the institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

[illegible]

Dorsala Mallikarjun Reddy
(Roll No. 716042)
Date: 15-12-2021

ACKNOWLEDGMENTS

I express my deep gratitude and thanks to my supervisors Prof. V. N. Sastry, Professor, Institute for Development and Research in Banking Technology (IDRBT), Hyderabad and Dr. Chapram Sudhakar, Associate Professor, Department of Computer Science and Engineering, National Institute of Technology (NIT) - Warangal for their guidance, continued support and encouragement throughout the research period.

I extend my gratitude to the Doctoral Scrutiny Committee (DSC) members Prof. P. Radha Krishna, Prof. R. B. V. Subramanyam, Dr. R. R. Rout, and Dr. Ch. Venkaiah for their insightful comments and suggestions during oral presentations. I am immensely thankful to Dr. Ch. Sudhakar, Prof. R. B. V. Subramanyam and Prof. P. Radha Krishna, Heads of Dept. of CSE and chairmans of DSC, during my tenure for providing adequate facilities in the department to carry out the research works.

I wish to express my sincere thanks to Prof. N.V. Ramana Rao, Director, NIT Warangal, Prof. D. Janakiram, Director, IDRBT and Dr. A. S. Ramasastri, Ex-Director, IDRBT, Hyderabad, for providing the infrastructure and facilities to carry out the research. I am also very grateful to the Computer Science and Engineering Department faculty members and IDRBT faculty members for their moral support throughout my research work.

On a personal level, I would also like to thank my scholar friends in IDRBT and NIT-Warangal for their valuable suggestions and for extending selfless cooperation. Lastly, I express my gratitude to my family for their unconditional love, support, and prayers to achieve the goal.

Dorsala Mallikarjun Reddy

Dedicated to

My Parents and wife

ABSTRACT

Blockchain has become a prominent technology in recent years, and because of its characteristics like decentralization, immutability, and transparency, it is deemed as an alternative for establishing a trusted platform. Initially introduced through Bitcoin for peer-to-peer financial transactions, Blockchains have recently come to the forefront of the research and industrial communities. Smart contracts have been a major driving factor in the broad adoption of Blockchain technology as they introduce automatic control. Blockchain and smart contracts are becoming popular in many engineering and computer science fields. Cloud computing is one such field that can benefit from adopting Blockchain technology to re-engineer its data centers. Blockchain is expected to be an indispensable tool to fulfill the expectations of cloud systems' performance with minimal costs and management overheads. Many recent works focus on utilizing Blockchain technology for establishing trust and reliability in cloud operations. In cloud computing, it is generally assumed that the user generally trusts the cloud provider for provisioning the service honestly and pays to the provider before actually using the service. However, due to the monetary benefits involved, a rational cloud provider may deviate from provisioning the service honestly. To address this problem, existing solutions comprise trusted parties for fair payments between cloud user and cloud provider. Nevertheless, having trusted parties does not entirely solve the problem, and an additional financial cost is imposed on both the cloud user and the cloud provider. In current literature, fair payments for cloud services is not addressed adequately and hence, this thesis focus on designing fair payment protocols for cloud services without a trusted intermediary between cloud provider and cloud users. We have identified fair payment problems in all three traditional cloud service models and designed solutions for them. For the platform-as-a-service model, we have proposed Blockchain-based fair payment protocols for outsourcing-as-a-service (verifiable computation) and data aggregation-as-a-service (mobile crowdsensing). We have proposed Blockchain-based fair payment protocols for cloud resource allocation and cloud data de-duplication for the infrastructure-as-a-service model. We have designed a Blockchain-based fair rating, charging and billing platform for microservices deployed on a cloud for the software-as-a-service model. Our theoretical analysis shows the fairness guarantees of the designed protocols, and our extensive experimental analysis shows the feasibility of deploying the designed protocols in the real world. Our experimental analysis also shows that the trusted third party in traditional cloud computing can be replaced by a smart contract running on a public Blockchain network with minimal overhead.

Contents

List of Figures

List of Tables

List of Algorithms

List of Notations

$\mathcal{JD}$	Judiciary
$\mathcal{TP}$	Trusted party
B	Block in Blockchain
G, H	Hash functions
D	Difficulty level
CP	Cloud provider
CU	Cloud user
DO	Data owner
DU	Data user
DP	Data provider
$F(x)$	Outsourced function / computation with input x
λ	Security parameter
ek_F, vk_F	Evaluation and verification keys of function F
y, π_y	Output and proof-of-correctness of output
$\tau_i, \tau_r, \tau_a, \tau_c, \tau_{end}$	Blockchain timing parameters
$\$d, \$r, \$f, \$c, \$b, \$pay$	Monetary variables
$u(\cdot)$	Utility function
q_1, q_2	Safe primes
$sk_{CU_i}, sk_{cp}, sk_{\mathcal{A}}$	Secret keys of cloud user, cloud provider and accumulator respectively
$X_{CU_i,t}$	Aggregated input of cloud user at time t
sum_t	Aggregated sum over inputs at time t
MB_i	Mobile device
$Y_{CU_i,t}$	Data collected by cloud user at time t

$K_{CU_i,CP}$	Self generated symmetric key for CU_i and CP
w_{CU_I}	Weight calculated during truth estimation for cloud user
x_m^*	Ground truth value generated for m^{th} sensor
$minDo$	Minimum number of cloud users
$minDv$	Minimum number of data verifiers
th	Threshold
$list_{do}$	List of cloud users who shown intent
$list_{dv}$	List of data verifiers who shown intent
$list_{ho}$	List of honest cloud users
$list_{ma}$	List of malicious cloud users
$list_{hv}$	List of honest data verifiers
$list_{sp}$	List of cloud users who sent spawn message
$list_{co}$	List of cloud users who sent commit message
$\pi^{CP}, \pi^{\mathcal{C}}, \pi^{CU_k}$	Proof-of-correctness of truth discovery algorithm generated by cloud provider, challenger and cloud users respectively
Q	Computation resource capacity
$\mathcal{A}$	Auction mechanism
$P(x)$	Auxiliary pricing function
inv_cap_i	Time invariant capacity requirement
$[a_i, d_i]$	Preferred time duration for accessing cloud resources
l_i	Length of the required time duration
v_i	Valuation function
$b_i(\cdot)$	Concavely increasing function
rId	Request identity
$U[\tau, \tau_{sub}]$	Resource utilization matrix
$allocated[\tau]$	Resource allocation vector
$requests$	A mapping data structure to store all information regarding cloud users requests
$\mathbb{S}$	Cloud storage system
$\mathcal{D}$	Set of data files

$N_d^{CP}(t)$	Number of users having the same data d at time t at a cloud storage provided by cloud provider CP
B_{DEDU}	Smart contract facilitating data deduplication
$U_{CU}^0(t)$	Utility of cloud user when deduplication is not adopted
$U_{CU}^1(t)$	Utility of cloud user when deduplication with B_{DEDU} is adopted
$U_{CP}^0(t)$	Utility of cloud provider when deduplication is not adopted
$U_{CP}^1(t)$	Utility of cloud provider when deduplication with B_{DEDU} is adopted
$SF_{CU}^{CP}(t)$	Storage fee
$SC_{CP}^{CU}(t)$	Storage cost
$EF_{CU}^{CP}(t)$	Extra fee
$n_d^{CP}(t)$	Data deduplication rate
$I_{CP}(t)$	Cost of deploying B_{DEDU} smart contract
tag	Hash value of data file
$uTAB$	A mapping data structure to store information regarding user requests
$AF^{CP}(t)$	Fee paid by a cloud provider to another cloud provider
r_j	Meter value of j^{th} resource
$\vec{U}(CU_i, \tau_s, \tau_e)$	Usage vector of cloud user
R_{CU_i}	Reputation of cloud user
R_{CP}	Reputation of cloud provider
$\mathcal{E}$	Set of errors
w_e	Weight associated with error e
$S(\tau)$	State of the cloud operating environment
$P_{r_i}(\tau)$	Price of the resource at time τ
D_{CU_i}	Discount offered to cloud user
$\$p_{es}$	Estimated cost of running a microservice chain
$list_{recUsage}$	A mapping data structure to store user's service requests

Glossary

5G	5th Generation
ACM	Association for Computing Machinery
APIs	Application Programming Interfaces
AWS	Amazon Web Services
BC	Blockchain
BFT	Byzantine Fault Tolerance
BI	Business Intelligence
BIC	Billing Contract
CaaS	Computation-as-a-Service
CAB	Collaborative Auditing Blockchain
CBVC	Challenge-based Verifiable Computation
CDA	Continuous Double Auction
CE	Convergent Encryption
DaaS	Data aggregation-as-a-Service
DB	Database
DCR	Decisional Composite Residuosity
DEC	Decryption
DEDU / dedup	Data Deduplication
DHT	Data Hash Table
DPS	Data Preservation System
DPoS	Delegated Proof-of-Stake
DRM	Dispute Resolution Mechanism

EDC	Error data contract
EHR	Electronic Health Records
ENC	Encryption
ETSI	European Telecommunications Standards Institute
FPP	Fair Payment Protocol
HDFS	Hadoop Distributed File System
IaaS	Infrastructure-as-a-Service
IEEE	Institute for Electrical and Electronic Engineers
IoT	Internet of Things
IPFS	Interplanetary File System
ISH	Inner State Hash
LAMB	Long term Auction for Mobile Auction
MANO	Management and Network Orchestration
MCS	Mobile Crowd Sensing
MEC	Message exchange contract
MITM	Man-In-The-Middle
N/A	Not Available
NFV	Network Function Virtualization
NIST	National Institute for Standards and Technology
OKS	Oblivious Keyword Search
PaaS	Platform-as-a-Service
PBVC	Proof-based Verifiable Computation
PHD	Personal Health Data
PoP	Proof-of-Ownership
PoS	Proof-of-Stake
PoW	Proof-of-Work
PP	Payment Protocol
PPA	Privacy Preserving Aggregation
PRF	Pseudo Random Function
QoS	Quality of Service

RBAC	Role-based Access Control
RBVC	Replication-based Verifiable Computation
RCB	Rating, Charging and Billing
RCC	Rating and Charging Contract
RRC	Registration and Reputation Contract
RSA	Rivet-Shamir-Adleman
SaaS	Software-as-a-Service
SC	Smart Contract
SDC	Service discovery contract
SE	Searchable Encryption
SGX	Software Gaurd Extension
SLA	Service Level Agreement
trans	Transaction
TDA	Truth Discovery Algorithm
TEE	Trusted Execution Environment
TKG	Trusted Key Generator
TP	Trusted Party
UDC	Usage data contract
URL	Uniform Resource Locator
VCG	Vickrey-Clarke-Groves
VLAN	Virtual Local Area Network
VMM	Virtual Machine Manager
VNF	Virtual Network Function
VNFO	Virtual Network Function Orchestrator
XACML	Extensible Access Control Markup Language
zk-SNARK	Zero Knowledge Succinct Non-Interactive Argument of Knowledge

Chapter 1

Introduction

In this Chapter, we present an introduction to cloud computing, cloud service models, cloud pricing models and payment models. Then, we discuss fair payments and hierarchy of fairness guarantees. We also present fundamentals and main characteristics of Blockchain technology and their potential applications to cloud computing. Later, we outline the motivation, aim, and objectives of the thesis. At the end of the Chapter, we list our contributions and organization of the thesis.

1.1 Cloud Computing

Cloud computing provides on-demand network access of configurable computing resources enabling individuals and enterprises to pay only for the resources or services they use. A cloud computing system is defined by the following set of properties:

(a) **On demand self service**: Cloud users access cloud resources as per need without any human intervention.

(b) **Broad access**: Standardized mechanisms and protocols are defined for accessing cloud resources.

(c) **Resource pooling**: Cloud resources are pooled into shared resources by a cloud provider and are allocated to customers on demand.

(d) **Rapid elasticity**: Resources are allocated and released dynamically according to the need of the user.

(e) **Measured service**: The resource usage is measured automatically allowing users to monitor, control and report resource usage.

1.1.1 Cloud Service Models

National Institute of Standards and Technology (NIST) [1] defines three layers within cloud computing stack as shown in Figure 1.1: (a) Infrastructure-as-a-service (IaaS) (b) Platform-as-a-service (PaaS) and (c) Software-as-a-services (SaaS).

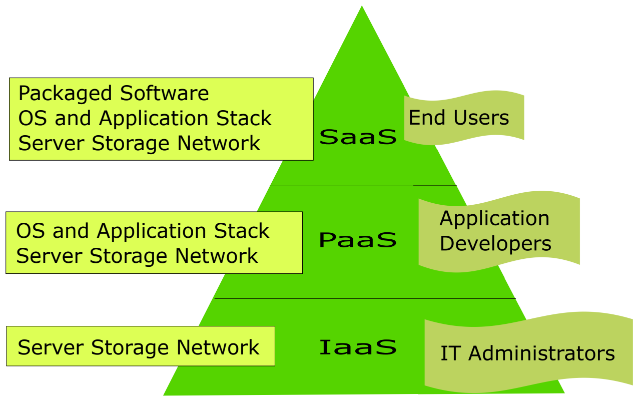

Figure 1.1: Layers in cloud stack

(a) **Infrastructure-as-a-service**: It is an instant computing infrastructure administered over the Internet. IaaS acts as a backbone for PaaS and SaaS. Some of the standard components in IaaS includes data centre physical building / plant, networking firewalls, computing servers, and storage disks. Well trained IT administrators manage the IaaS components while the general public / enterprises purchase, install, configure and manage their software—operating systems, middleware and applications. The typical examples of IaaS providers are Amazon web services, Microsoft Azure, Rackspace, and VMware.

(b) **Platform-as-a-service**: PaaS is a development and deployment environment in the cloud. Any person / enterprise can develop and host their applications for their clients without worrying about the management of underlying hardware. It includes all the components of IaaS and also development tools, middleware, database management systems, and business intelligence (BI) services. The typical examples of PaaS include Force.com, Heroku, Google App Engine, AWS Elastic Beanstalk, and Apache Stratos.

(c) **Software-as-a-service**: SaaS allows user to connect to and use cloud-based applications over the Internet. The applications are accessible through a web browser or application programming interfaces (APIs). The typical examples include Dropbox, Salesforce, Google workspace, and Office 365.

1.1.2 Cloud Pricing Models and Payment Models

1.1.2.1 Pricing Models

Unlike telecoms or electricity services, there is no standard pricing unit in cloud computing. However, cloud providers offer a combination of one or more of the following pricing models:

1. **Subscription Model**: In this model, the services are sold on a fixed time period as monthly / yearly basis (For example: per mailbox or per-app license). The customers are billed for all the resources, whether used or not.

2. **Pay-as-you-go / Pay-per-use**: In this model, a user starts with a zero payment, provisions services on demand, and gets charged based on the actual consumption. This approach is the most attractive model as the user pays only for the services consumed.

3. **Pay per user**: This model is very similar to the subscription model. The price is based on the number of active users provisioning a service (For example: per user per month).

1.1.2.2 Payment Models

Consider the following example scenario to understand the payment models in cloud computing.

Alice, a cloud user, wants to acquire a cloud service (For example a Virtual machine) from Bob, a cloud provider. Alice and Bob do not necessarily trust each other. Alice visits the catalogue offered by Bob and selects interested VM / VMs. Bob displays the price to be paid by Alice according to the adopted pricing model (see section 1.1.2.1). To avail the service, Alice has to pay for the resources provided by Bob. As depicted in Figure 1.2, over the years, different payment models have evolved to facilitate payments between Alice and Bob.

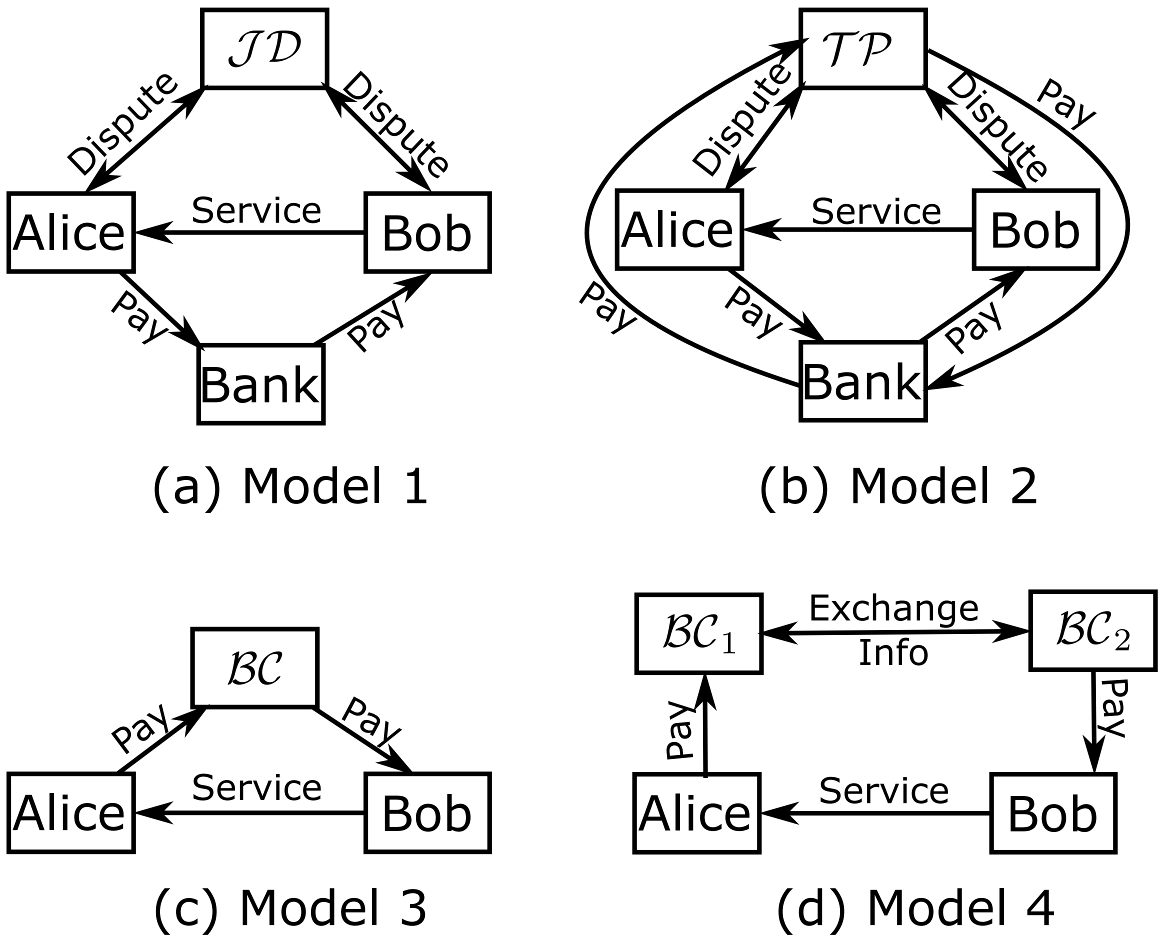

Figure 1.2: Cloud payment models

- **Model 1**: In this model, a service level agreement (SLA) is negotiated between Alice and Bob. Then, Alice pays for the service through a bank. After receiving the payment, Bob provides the service according to the SLA. If Bob does not adhere to the SLA, then Alice approaches legal judiciary ($\mathcal{JD}$) to resolve disputes caused due to non-adhering to SLA. The disadvantage in this model is that Alice has to pay Bob before getting the service. After receiving the payment, Bob may refuse to provide the requested service or may provide only a partial service. Another disadvantage is that the dispute resolution process through $\mathcal{JD}$ is tedious and costly.

- **Model 2**: Similar to model 1 in this model also, an SLA is negotiated between Alice and Bob. They both recruit a trusted third party ($\mathcal{TP}$) to resolve disputes. Alice pays to $\mathcal{TP}$ through the bank then Bob provides the service according to the SLA. At the end of the service, the $\mathcal{TP}$ verifies whether Bob adhered to the SLA or not. Based on the verification result, the $\mathcal{TP}$ pays full / partial / zero amount to Bob. The disadvantage in this model is the cost induced by the $\mathcal{TP}$ because the services of $\mathcal{TP}$ are not free. The second disadvantage is difficulty in finding an ideal $\mathcal{TP}$, which will behave honestly without any prejudice at all times, is difficult.
- **Model 3**: In this model, the full SLA or some part of the SLA is transcoded as a smart contract and deployed on a public Blockchain network (BC). Alice transfers payment to the smart contract, and then Bob provides the service to Alice. At the end of the service, the smart contract pays to Bob or refunds to Alice according to their interactions and the rules encoded in the smart contract. This model has several advantages when compared to previous models: (1) This model limits / eliminates the role of banks during payments. (2) The disputes are resolved with respect to the rules encoded in smart contracts; therefore, the services of $\mathcal{JD}$ / $\mathcal{TP}$ are not required. (3) The interactions with smart contracts and the operations of smart contracts are publicly visible to both Alice and Bob instilling the trust on the payments.
- **Model 4**: This model is similar to model 3 but additionally incorporates the interoperability of the Blockchain networks.

1.1.3 Fair Payments

We follow the definition of fairness introduced by Asokan [2] and later enhanced by Pagina and Vogt [3]. We assume a fair payment protocol is executed by two parties, cloud user

Alice and cloud provider Bob.

Payment protocol (PP): In general, a payment protocol consists of the following steps: (1) Alice requests a service or a product provided by Bob. (2) Bob provides the requested service or product to Alice as agreed. (3) Alice sends payment to Bob as agreed. (4) Bob sends a receipt of payment to Alice. We assume that a payment protocol has two possible termination states, either $success$ or $abort$.

Fair payment protocol (FPP): A PP is said to be FPP if it satisfies the following three requirements:

(a) **Effectiveness**: If both Alice and Bob behave according to PP and do not abort during the PP, then at the end of the PP, Alice has received the service and Bob has received the payment as agreed and PP has reached a $success$ termination state.

(b) **Termination**: If Alice and Bob follow the PP honestly, then PP will eventually reach either a $success$ or an $abort$ termination state.

(c) **Fairness**: If at least one of the two parties does not behave according to the PP, then no honest party gains or loses anything valuable. In other words, Alice receives service if and only if Bob receives payment otherwise the party (Alice or Bob) behaving honestly do not lose anything valuable.

Pagina *et al*. [4] have defined the hierarchy of fairness guarantees from F_0 to F_6 as shown in Figure 1.3.

(a) F_0: No Fairness

(b) F_1: Fairness can only be achieved outside of the system by an external dispute resolution entity by providing compensation for a suffered disadvantage.

(c) F_2: Fairness can only be guaranteed outside of the system by an external dispute resolution entity with eventual cooperation of the other party.

(d) F_3: Fairness can only be guaranteed outside of the system by an external dispute resolution entity without further cooperation of the other party.

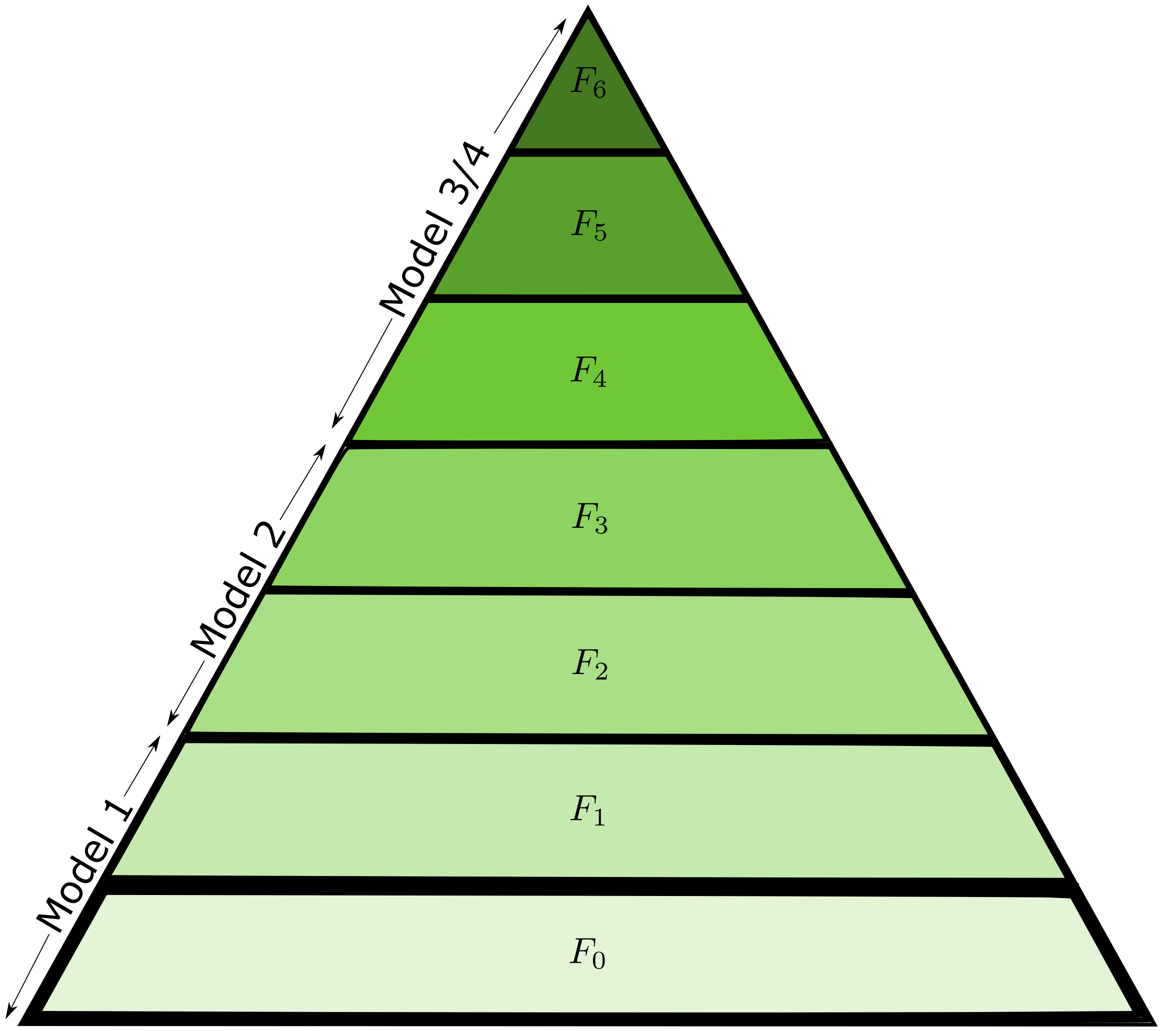

Figure 1.3: Fairness hierarchy from [4]

(e) F_4: Fairness can be achieved automatically by the system through providing compensation for a suffered disadvantage.

(f) F_5: Fairness can be guaranteed automatically by the system with eventual cooperation of the other party.

(g) F_6: Fairness can be guaranteed automatically by the system without further communication with the other party.

Fair payment models constructed for hierarchy levels F_1 - F_3 have traditionally required the presence of a trusted third party. The fairness levels F_4 - F_6 are considered as strong fairness guarantees, and they are thought to be impossible to achieve [3]. However, the advances in Blockchain and smart contracts shows that they can emulate the role of the mediating third party and facilitate fair payments between Alice and Bob automatically without the influence of any party.

1.2 Blockchain Technology

The term Blockchain has been introduced with the peer-to-peer electronic cash system known as Bitcoin [5] designed by an anonymous person / group under the pseudonym Satoshi Nakamoto. Although the initial purpose of the Blockchain technology was to facilitate peer-to-peer financial transactions without trusted intermediaries, its fundamental concepts are used as building blocks to construct many decentralized applications in the fields of digital assets [6], smart contracts [7], Internet of things (IoT) [8], public services [9], cloud computing [10], security services [11], reputation systems [12] and 5G networks [13]. The fundamental components in Blockchain technology are as follows [14]:

(a) **Peer-to-peer transmission**: All the communications (transactions) occur directly between peers without a central entity. Each peer stores and forwards information to all other peers.

(b) **Public key cryptography**: Every peer in the Blockchain network is associated with a public-private key pair. The public key is used to identify a peer uniquely in the network, and the corresponding private key is used to sign the transactions during transfer of the assets (financial or non-financial) owned by the public key.

(c) **Distributed database (blockchain)**: In Blockchain network, the transactions are stored in a one-way append distributed ledger also called as blockchain. Every peer in the network has access to information stored at the blockchain, and no single peer can control the data stored at the blockchain. A block in blockchain consists of two parts: block header and block body. The block header consists of:

    (i) **Block version**: It shows which set of block validations rules have to be followed during block validation.

(ii) **Merkle tree root hash**: The hash of the Merkle tree root constructed with the transactions in the block.

(iii) **Timestamp**: The current time as seconds in the universal time since January 1, 1970.

(iv) **Nonce**: A 4-byte value computed during block creation.

(v) **nBits**: Represents the current target threshold of the block hash.

(vi) **Parent / Previous block hash**: A 256-bit hash value of the parent block.

The structure of the block is shown in Figure 1.4.

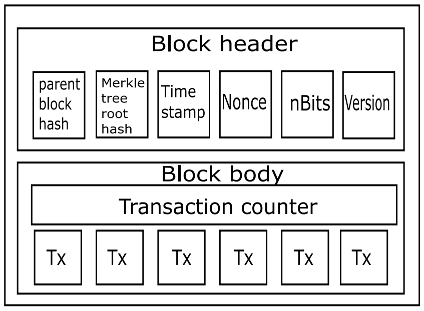

Figure 1.4: Block structure

A blockchain is a sequence of blocks which are linked cryptographically. Figure 1.5 illustrates an example of blockchain.

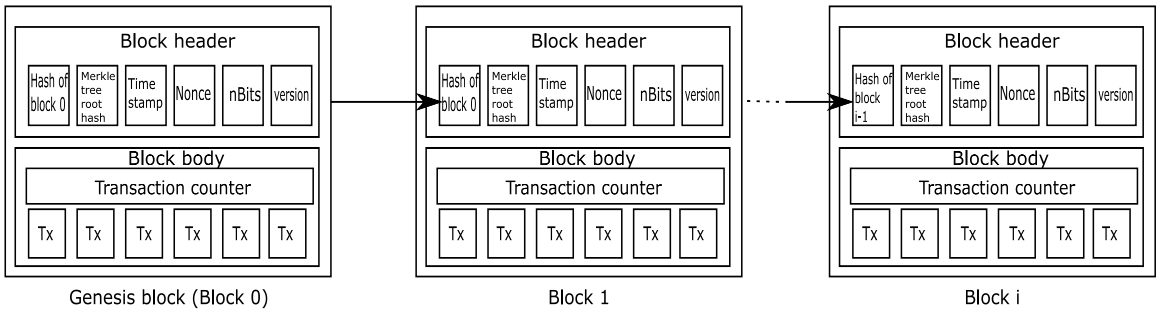

Figure 1.5: Distributed ledger (blockchain) as a sequence of blocks

(d) **Consensus algorithm**: A secure consensus algorithm is executed by a set of decentralized peers known as miners to agree on a common global state of the blockchain. Consensus algorithm guarantees the security of the blockchain. Bitcoin [5] uses the

Proof-of-Work (PoW) consensus algorithm, and most of the public Blockchain networks (commonly called as alt-coins / Nakamoto-style ledgers) proposed later follows Bitcoin's PoW with little / no modifications. In PoW Blockchain networks, a block B is of the form

$$B = \langle h, t, c, mh, ts, nb, v \rangle$$

where:

$h \in \{0,1\}^s$	is the hash of the parent / previous block.
$t \in \{0,1\}^*$	is the set of transactions included in this block.
$mh \in \{0,1\}^s$	is the Merkle hash of t.
$ts \in \mathbb{N}$	is the number of seconds elapsed since the last UNIX epoch.
$nb \in \mathbb{N}$	is the difficulty target for this block.
$v \in \mathbb{N}$	is the version of the block validation rules.
$c \in \mathbb{N}$	is the nonce.

Every block B in a blockchain must satisfy the condition

$$(G(c, H(h, t, mh, ts, nb, v)) < D) \text{ and } (c \leq q)$$

where:

$G(\cdot), H(\cdot)$	are cryptographic hash functions which gives outputs of strings of length s bits.
$D \in \mathbb{N}$	is known as block difficulty level set by consensus algorithm.
$q \in \mathbb{N}$	is the maximum value of nonce.

The following condition must be satisfied to add a new block $B^j = \langle h^j, t^j, c^j, mh^j, ts^j, nb^j, v^j \rangle$ to a blockchain with $B^i = \langle h^i, t^i, c^i, mh^i, ts^i, nb^i, v^i \rangle$ as right most block.

$$h^j = G(c^i, H(t^i, mh^i, ts^i, nb^i, v^i)$$

PoW algorithm makes the miners compete to generate a new block periodically. The miners are rewarded for mining new blocks in the form of currency native to the

Blockchain network (bitcoins in the case of Bitcoin). The PoW algorithm is composed of several transactions and block validation rules. The following two rules ensure the correctness of the execution of transactions: (1) The miners verify all the received transactions before adding them to a block, and (2) A miner after receiving a new block verifies the validity of block as well as transactions in that block before adding it to their local blockchain. These verification steps make the Blockchain network trusted for correctness. The hardness in solving PoW puzzle makes the blockchain immutable, and a large number of participating miners ensure availability of Blockchain network. Several consensus algorithms are proposed in the literature, and one can refer to [7] for the analysis and comparison of different consensus algorithms.

(e) **Smart contracts**: A smart contract (SC) is a program deployed and stored in a Blockchain. A smart contract can hold many contractual clauses between mutually distrusted parties. Similar to transactions, the smart contract is also executed by miners and, its execution correctness is guaranteed by miners running the consensus protocol. Assuming that the underlying consensus algorithm of a Blockchain is secure, the smart contract can be thought of as a program executed by a trusted global machine that will faithfully execute every instruction [15]. The complete analysis, applications, design patterns and limitations of the smart contracts are found in [16]. An example of a smart contract between two parties A and B is given in Figure 1.6. The parties in smart contracts can add the contract terms and set parameters according to their own needs.

```
Contract Example()
Begin
IF $A$ initiates a transaction AND $condition(i)$ is met
        Set timestamp OR trigger event
ELSE
        transaction failed, $A$ and $B$ state regressed and ended the transaction
IF $condition(j)$ is met AND no timeout
        $B$ confirmed the transaction AND quit
IF TIMEOUT
        transaction failed, $A$ and $B$ state regressed and ended the transaction
END
```

Figure 1.6: Example of a smart contract [17].

1.2.1 Main characteristics of Blockchain

We discuss some of the main characteristics of Blockchain and their potential applications to cloud computing.

(a) **Decentralization**: Decentralization in Blockchain refers to a lack of centralized authority for managing identities, accounts, balances, databases, and code execution. The consensus algorithms like Proof-of-work, Proof-of-Stake, etc., ensure the security of the Blockchain without any trusted authorities. This feature is essential in cloud computing, especially when the cloud provider and user are mutually distrusted. As no central authority controls the Blockchain, Blockchain-enabled cloud computing services instil more trust and give more control to cloud users than traditional cloud computing services.

(b) **Immutability**: The complexity in solving the PoW puzzle makes the data stored in the blockchain immutable. The blocks in the blockchain are chained cryptographically. To change a transaction / data in some i^{th} block of a blockchain with n^{th} block as the current block, new PoW solutions have to be found for all blocks between i and n. Finding PoW solutions is computationally expensive, especially if i is much smaller than n. This property can be plausibly used to design data integrity and auditing schemes because the traditional cloud computing solutions rely on trusted party or heavy cryptographic primitives.

(c) **Transparency**: All the interactions / transactions stored in the blockchain are publicly viewable by every member. Also, many parties (miners) contribute their computational power to generate new blocks, verify newly generated transactions and blocks. These steps ensure strong transparency which in turn enhances the integrity of the data stored in the blockchain. If Blockchain is adopted in cloud computing and the meta-data of interactions between users and cloud data / services are recorded in the blockchain, then it enhances the trust and openness in cloud computing.

(d) **Persistency**: Every transaction and block are verified for the common good before adding it to the blockchain. Any malicious attempt to destabilize blockchain by adding malicious transactions is not possible due to public verifications. Once the data is stored at blockchain, then that data is persistent and cannot be modified. This property enhances the persistency and reliability of Blockchain-enabled cloud services.

(e) **Auditability**: As the Blockchain data is retrievable publicly, it is subjected to public auditing. This characteristic is required in cloud computing as most of the existing cloud data integrity, and auditing schemes [18] depend on third party auditor and complex cryptographic primitives. The auditability property of Blockchain dramatically reduces the cost of auditing and also eliminates the role of trusted parties in the traditional cloud auditing schemes.

(f) **Security and privacy**: For sending transactions in Blockchain, every participating entity must generate an asymmetric key pair using public-key cryptography. Before sending, every transaction has to be signed with a private key of a sender. During verification of the transaction, the signature is verified with the help of the public key of the sender. The asymmetric key helps preserve the privacy, ownership and non-repudiation properties. However, in traditional cloud solutions, a trusted key generator (TKG) is required for generating and distributing keys causing the well-known key-escrow problem [19]. The fusion of Blockchain and cloud greatly reduces the dependence on TKG and increases the security of cloud systems. Also, in traditional cloud computing, a trusted party is required for managing access control policies.

> The dependence on the trusted party can be eliminated by transcoding the access control policies as smart contracts. Nevertheless, the drawback of public Blockchain systems is that they cannot provide privacy to the data stored on the Blockchain. In order to mitigate the privacy problem, several privacy-preserving solutions [20] are being explored along with the development of private Blockchain networks.

Although public Blockchain systems possess decentralization, immutability, trust, and transparency properties, they suffer from scalability and privacy problems. To alleviate these problems, private Blockchains and consortium Blockchains are being developed. However, private / consortium Blockchains trade-off decentralization and transparency for the sake of scalability and privacy. The comparisons of public, consortium and private Blockchain is shown in Table 1.1.

S.No	Characteristic	Public	Consortium	Private
1	Decentralization	Yes	No (selected set of nodes spread across multiple organisations)	No (single orgranization)
2	Immutability	Tamper-proof	Cloud be tampered	Cloud be tampered
3	Transparency	Yes	Cloud be public or restricted	Cloud be public or restricted
4	Persistency	Yes	No	No
5	Public Auditability	Yes	No	No
6	Privacy	No	Partial	Yes
7	Smart contracts	Yes	Yes	Yes

Table 1.1: Comparison among public, consortium and private Blockchains

1.2.2 Formal Blockchain Model

The formal Blockchain model was first introduced in [21] and later adopted in [22], for specifying and reasoning about the security of the protocols. We also adopt the same model to describe our protocols.

(a) **Timer**: All parties are aware of the time which progress in rounds. At the beginning of each round, the contract's timer function is executed. The smart contract can also query Blockchain for the current time denoted by variable τ.

(b) **Pseudonymity**: A party can obtain any number of pseudonyms to communicate with a smart contract. Contract wrapper ($\mathcal{G}$) in [22] generates pseudonyms on the request of any party.

(c) **Availability**: We assume that the blockchain is always available to be queried by any party.

(d) **Correctness**: We also assume that all the transactions are verified and only correct transactions are added to a block. The blocks are also verified and only correct blocks are added to Blockchain.

(e) **Currency**: All the monetary variables are prefixed with \$ sign [1]. $ledger[\mathcal{P}]$ denotes the party $\mathcal{P}$'s balance in native cryptocurrency of a Blockchain.

(f) **Variable scope and functions**: A smart contract is written as a set of functions, and each function is invoked with a corresponding message type. We assume that all the variables in a smart contract are globally scoped.

(g) **Wrappers and Programs**: Wrappers contain a set of common features that are applicable for all the ideal and contract functionalities. We use the same wrappers described in [21, 22]. We define our protocols in $\mathcal{G}(Contract)$-hybrid model, where $\mathcal{G}(\cdot)$ is a contract wrapper which models many concepts of the decentralized Blockchains like Bitcoin and Ethereum. The Contract program is the user-defined portion of the contract, i.e., a Contract program contains the business logic, whereas the $\mathcal{G}(\cdot)$ contains the operational semantics. Both are combined to model a real-world smart contract executing on top of a decentralized Blockchain system. Similarly, we have ideal functionality wrappers $\mathcal{F}(\cdot)$ to be used in combination with Ideal programs. All the wrappers are discussed in [22].

[1] Although we are using \$ sign, it can be replaced with any currency symbol.

1.2.3 Ethereum, Solidity, Gas, Truffle framework and Simulation environment

Ethereum [23] is a major Blockchain network supporting smart contracts. A smart contract in Ethereum is a piece of code having its address, balance and state. The execution of the smart contract code changes its state. A smart contract is written as a set of functions. The execution of a function in a smart contract is initiated by sending a transaction to its contract address.

Solidity [24] is one of the scripting languages used to write smart contracts in Ethereum. Solidity is a Turing-complete language; therefore, a wide variety of applications can be developed using Solidity. However, to discourage developers from writing smart contract functions that take a long execution time, Ethereum introduces gas.

A smart contract is compiled into Ethereum opcodes, and each opcode has a predefined cost. The sum of all the opcodes' cost is known as gas. Gas in Ethereum is a form of transaction fee paid in Ether to miners. The unit of gas is gWei (grand Wei) which is equal to 10^{-9} Ethers. Ether is a native cryptocurrency of Ethereum. The amount of gas consumed by a transaction is converted into the Ether and charged from the transaction initiator's Ethereum account and paid to the miner in the form of a transaction fee.

Truffle framework [25] is a development and testing environment for Ethereum smart contracts. Truffle offers automated smart contract compilation and deployment. It also contains a private Ethereum Blockchain known as Ganache which mimics the Ethereum production network. In this thesis, we have implemented proposed protocols using the Solidity and Truffle framework.

We have set up a Ganache Blockchain and Truffle framework simulation environment on a 2.50 GHz Intel Core i5 CPU and a 16 GB RAM machine. We have used the same simulation environment to implement the contracts that we discuss in this thesis.

1.2.4 Need of Blockchain Technology in Cloud Computing

The mapping of Blockchain characteristics and their potential applications in cloud computing is shown in Table 1.2. Each characteristic would enhance the quality of cloud computing from the transparency and trust perspectives showing great potential of using Blockchain in cloud computing. The development of Blockchain-enabled solutions for cloud computing has only recently started and focuses on commercial targets. In the traditional cloud models, users are assumed to trust that the machine hardware, software, and cloud administrator all perform as expected. A wide range of things can go wrong, particularly when one wishes to tie the results of such computations to monetized entities such as smart contracts. Proper economic incentives, the cornerstone of any cryptocurrency, can deter many types of errors occurring in ways that simple task repetition cannot.

S.No	Key characteristics of Blockchain	Description	The potential application to cloud computing
1	Decentralization	No centralized or trusted party controls the Blockchain.	Eliminates the need for trusted parties in the cloud computing environment for services like data auditing, data integrity, data timestamping, data searching, access control, resource allocation, service allocation, service discovery, billing and payments, and federated services.
2	Immutability	The data stored on the Blockchain can not be modified	Every interaction with cloud data / service can be recorded immutably on Blockchain, providing integrity and thus enabling tamper-proof data auditing. The logging of service interactions helps in monitoring user behaviour. As no party can alter the records stored in the append-only ledger, the billing of services based on these records will be fair and correct.
3	Transperency	All the interactions with the Blockchain are publicly available	Cloud provider, application developer, and the end-users can thoroughly check and monitor the transactions with equal rights. No party is deprived of its right to monitor the transactions there by instilling more trust and transparency in the Blockchain-enabled cloud services.
4	Persistency	The data stored in the Blockchain are subject to public verifiability. All the transactions recorded on the Blockchain is verified for correctness and any attempt to maliciously change the state of the Blockchain will be thwarted.	Transactions created from all the interactions with cloud data / services are recorded and verified by the cloud provider and users. This verification enhances the persistency and reliability of Blockchain-enabled cloud services.
5	Auditability	As data is publicly available, it can be traced and audited easily	Data auditing is one of the most critical tasks in cloud computing. Currently, cloud provider and user mutually distrust each other; hence a trusted party has to be required for performing the data auditing tasks. As data is available publicly, Blockchain eliminates the trusted party and enables provider and user to trace and audit data on their own.
6	Security and privacy	Blockchain systems employ public-key cryptography for authentication and non-repudiation. Access controls can be transcoded into smart contracts for authorization. Privacy for data can be provided either by employing private Blockchain or some known encryption techniques.	Blockchain supports secure cloud computing by providing distributed trust models with authentication and data privacy. Blockchain helps in protecting the cloud service end-users privacy by masking the real identity of end-users with a pseudonym generated through public-key cryptography. Blockchain also helps in protecting access control policies of cloud data / services from unauthorized entities.
7	Smart contracts	A smart contract can be thought as a program executed by a trusted global machine (Blockchain network) that will correctly execute every instruction	A broad spectrum of cloud computing applications can be designed with smart contracts. For example, the service layer agreements can be transcoded into smart contracts and deployed on Blockchain for better trust, transparency and reliability of cloud services.

Table 1.2: Key characteristics of Blockchain and their potential applications to cloud computing

In Table 1.3, we present the major obstacles to cloud computing and the opportunity for Blockchain to address the obstacles of cloud computing.

S.No	Obstacle	Opportunity for Cloud	Opportunity for Blockchain
1	Service availability	Use multiple cloud providers	Blockchain can be a reliable communication medium which will instill trust between multiple cloud providers.
2	Data lock-in	Standardize APIs	Blockchain offers tamper-proof storage for publishing standardized APIs.
3	Data confidentiality and auditability	Deploy Encryption, VLANs, and Firewalls;	Blockchain offers tamper-proof storage to publish access control policies defined on data stored at the cloud. Blockchain offers tamper-proof storage to store meta-data of the data / services stored / running at the cloud helps tamper-proof auditing.

4	Data transfer bottlenecks	FedExing Disks;
		Data Backup / Archival;
			Efficient data migration tracking systems can be constructed using Blockchain.

5	Reputation-fate sharing	Offer reputation-guarding services like those for email	Fair and open reputation management schemes can be constructed using Blockchain.
6	Software licensing	Pay-for-use licenses; Bulk use sales	Fair billing and payments schemes without trusted intermediaries can be constructed using Blockchain.

Table 1.3: Some of the obstacles and opportunities for the growth of cloud computing and Blockchain.

1.3 Motivation, Aim and Objectives of Proposed Work

In cloud computing, it is generally assumed that the users always trust the provider for provisioning the service honestly and pays to the provider before actually using the service. However, due to the monetary benefits involved, a rational provider may deviate from provisioning the service honestly. In order to address the fair payment problem, existing solutions comprise trusted parties for fair payments between user and provider. Nevertheless, having trusted parties do not solve the problem completely, and an additional financial cost is imposed on both user and provider. Blockchain, with its innovative properties like decentralization, immutability, transparency and smart contracts, emulate the trusted parties. In recent years, fair payment protocols without trusted parties using Blockchain technology are being explored. In current literature, fair payments for cloud services is not addressed adequately, and this motivates us to develop fair payment protocols for cloud services using Blockchain technology.

1.3.1 Aim

In this thesis, our aim is to design Blockchain-based fair payment protocols for cloud services between cloud users and cloud provider.

1.3.2 Objectives

The main objectives of this thesis are stated as follows:

- To understand the main characteristics of Blockchain technology and its potential applications to cloud computing.
- To do literature survey, identify research gaps and address them.
- To understand the existing cloud computing payment models and provide Blockchain-based fair payment solutions for cloud services involving: (a) Platform-as-a-Service (b) Infrastructure-as-a-service and (c) Software-as-a-service.
- To find future directions in the field of Blockchain-based cloud computing.

1.4 Overview of the Contributions of the Thesis

This thesis presents the following contributions:

1. Presented a comprehensive literature survey on Blockchain-based cloud services and Blockchain-based fair payment models (**Chapter 2**).

2. Proposed a fair incentivized model for proof-based verifiable computation which shows that the cost of running a smart contract is negligible when both cloud user and provider are honest (**Chapter 3**).

3. Proposed a fair incentivized model for replication-based verifiable computation, which shows that smart contracts are an efficient way to send payment to honest cloud service providers and also to penalize malicious cloud service providers (**Chapter 3**).

4. Proposed a Blockchain-based fair payment protocol for monetizing mobile crowdsensing data. Proposed a new key establishment protocol using smart contracts as a communication channel for mobile crowdsensing that does not require a trusted key generator. Proposed a smart contract-based quality-aware incentivization model for paying data providers in mobile crowdsensing (**Chapter 4**).

5. Proposed a Blockchain-based online virtual machine allocation auction which focuses on fair payments and correctness of the auction algorithm (**Chapter 5**).

6. Proposed a new incentive model for cloud data de-duplication which is individually rational and incentive compatible for both cloud user and provider (**Chapter 6**).

7. Proposed a fair payment protocol for cloud data de-duplication which emphasizes correctness of de-duplication rate and fair payments between cloud user and provider (**Chapter 6**).

8. Proposed a Blockchain-based rating, charging and billing model for microservices which consists of a new decentralized service discovery of microservices, a fair rating and charging platform and a fair billing platform (**Chapter 7**).

1.5 Thesis Organization

The chapters of this thesis are organized as follows:

In Chapter 1, we have presented the preliminaries about cloud computing, fair payments, and Blockchain technology. We have also presented motivation, aim, objective and contributions of the thesis.

Chapter 2 presents a comprehensive literature survey of Blockchain-based cloud services. In particular, the survey focuses on the technical fusion of Blockchain and cloud computing and investigates the recent advances in the field of Blockchain-based cloud services. The Chapter also includes a survey of fair payment protocols constructed using Blockchain technology. At the end of the Chapter, the main findings of the survey and gaps in the literature are discussed.

Chapter 3 presents fair payment protocols for outsourcing a computation. It provides fair payment protocols for outsourcing a computation of two types: (1) Proof-based verifiable computation and (2) Replication-based verifiable computations. The honest computation from the provider is obtained by imposing monetary penalties, and our proposed protocols guarantee to pay to the honest provider.

Chapter 4 discusses fair payment protocols for mobile crowdsensing. It presents two protocols: The first protocol uses the services of a trusted party, whereas the second protocol eliminates the trusted party by using a smart contract-based key generation algorithm. The two protocols ensure that a data provider receives fair payments from the cloud provider for his / her data contribution towards the crowdsensing task.

Chapter 5 presents a fair payment protocol for virtual machine allocation. It shows that the untrusted auctioneer in traditional cloud computing for allocating virtual machines can be replaced by smart contract running on a public Blockchain network. The protocol ensures that the provider receives the pay if and only if the cloud user receives the requested virtual machines.

Chapter 6 discusses a fair payment protocol for data de-duplication. The Chapter presents a new incentivization scheme for cloud data de-duplication. The protocol assures correct and fair discounts on the storage fee for cloud users.

Chapter 7 provides a fair payment protocol for microservices which includes the rating, charging and billing aspects of the microservices.

Chapter 8 concludes the thesis and outlines future research directions.

Chapter 2

Literature Survey

Blockchain and smart contracts are becoming popular in many engineering and computer science fields. Cloud computing is one such field that can benefit from adopting Blockchain technology to re-engineer its data centres. Blockchain is expected to be an indispensable tool to fulfill the performance expectations for cloud systems with minimal costs and management overheads. We observe that the integration of Blockchain and cloud is rarely addressed in the literature. So in this Chapter, we try to address the gap in the literature by discussing the existing Blockchain-based cloud services and present a road map for further integrating Blockchain and cloud.

2.1 Comparison of existing Blockchain surveys

Many studies have surveyed Blockchain technology from various perspectives e.g., digital currencies [26], privacy [20], smart contracts [16], consensus mechanisms [27], IoT [28, 29], cloud computing [30, 10], edge computing [31], and 5G [13]. The comparison of existing survey works on Blockchain in various domains is presented in Table 2.1. We observe that only a few works have discussed Blockchain from the perspective of cloud computing and also, none of the works have discussed the Blockchain-based cloud applications from the perspective of service models. In contrast, we discuss state-of-the-art Blockchain-based cloud systems in all the three service models Iaas, PaaS and SaaS.

S.No	Related surveys	Topic	Key contribution	Limitations and open issues
1	Yang *et al.* (2019) [31]	Blockchain and edge	The authors investigated the integration of Blockchain-based edge computing and its challenges.	There is a lack of discussion on service-based cloud computing. The paper emphasizes on the integration of edge and IoT and have not considered the cloud.
2	Ali *et al.* (2019) [29]	Blockchain and IoT	The authors discussed various shortcomings of integration of IoT and Cloud and investigated solutions for those shortcomings using Blockchain.	The paper only provides the concept of Blockchain for IoT, and a detailed cloud service-based analysis is missing.
3	Dai *et al.* (2019) [28]	Blockchain and IoT	The authors investigated the integration of Blockchain with IoT.	The authors have focused more on IoT and discussed cloud only as a backend to IoT networks.
4	Nguyen *et al.* (2020) [13]	Blockchain and 5G	The authors discussed the adoption of Blockchain in 5G technology, 5G IoT and 5G services.	The authors have discussed some of the Blockchain-based cloud computing models from the perspective of 5G networks. However, they have not analysed from the perspective of cloud services.
5	Xie *et al.* (2020) [30]	Blockchain and cloud exchanges	The authors discussed the integration of Blockchain and cloud exchanges.	The authors have focused on generalised cloud services, and a detailed service-based analysis is missing.
6	Gai *et al.* (2020) [10]	Blockchain and cloud	The authors discussed the fusion of cloud services with Blockchain.	Although most aspects of cloud computing are covered some important topics like computation-as-a-service and data aggregation-as-a-service are not discussed.
7	This Chapter	Blockchain and cloud	We present a comprehensive survey of Blockchain-based cloud services. We present the works published in all the three service models IaaS, PaaS and SaaS of cloud computing.	-

Table 2.1: Comparison of surveys on Blockchain in various domains

2.2 Systematic Literature Survey

Our objective is to explore the existing works on Blockchain-based cloud services, and hence we adopted the systematic literature mapping given by Yli-Huumo *et al.* [32]. The adopted mapping process is illustrated in Figure 2.1. In the first step, we define research questions related to the integration of Blockchain and cloud computing, which are listed in Table 2.2. By analyzing the key research questions, we discuss the merits of the Blockchain and the cloud computing technologies and observe the pivotal issues of cloud computing that can be solved with the help of Blockchain technology. After defining the scope of the research, we have searched for the articles with the terms Blockchain and cloud computing as query strings. We have collected papers from well-known peer-reviewed article publishing platforms including (1) IEEE Xplore (2) Science direct (3) Springer Link and (4) ACM digital library. Then, based on the article title and abstract, we have filtered out some irrelevant papers and retained only papers related to Blockchain-based cloud services. Then, with the help of keywords in the articles, we have categorized papers into three sets based on the type of cloud service they have discussed. Finally, we have extracted information from the articles that needed to address the research questions of this mapping study. Technical dimensions of this survey are presented in Figure 2.2.

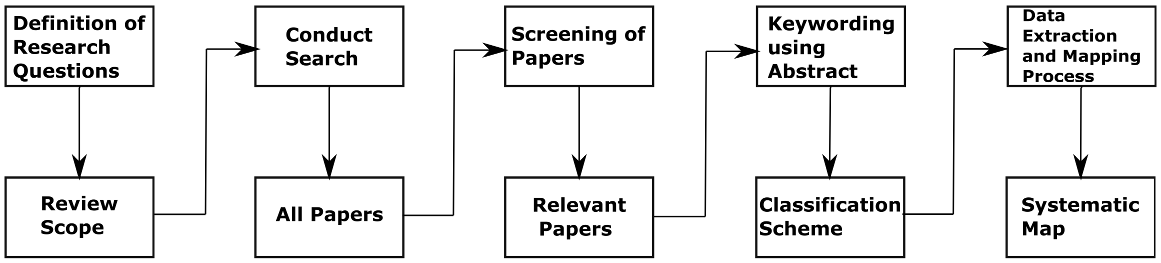

Figure 2.1: The systematic literature mapping process [32]

S.No	Research questions	Motivations
1	What are the current challenges of cloud computing ?	Cloud computing severely suffers from trust and transparency due to centralized architecture. Currently, cloud providers and users have to depend on trusted third parties to resolve disputes which arise due to violation of service level agreements.
2	What are the key requirements of cloud computing ?	The key requirements of cloud computing include availability, elasticity, data security, manageability federated systems, on-demand integration, multi-tenancy, resource management etc.
3	What are the key characteristics of Blockchain ?	The key characteristics of Blockchain include decentralization, immutability, transparency, persistency, auditability, security and smart contracts
4	What are the potential benefits of Blockchain for cloud computing ?	Cloud computing could benefit from Blockchain to build innovative applications providing trust, transparency and giving more control to cloud users.
5	What are the key advantages of Blockchain over the traditional technique for supporting cloud computing ?	Blockchain offers new features like decentralization, trust, transparency and immutability which does not exist in traditional cloud computing. Significantly, Blockchain limits the dependency of the cloud provider and users on trusted third parties.
6	How is the current Blockchain-cloud research progress ?	Many recent works discussed Blockchain-based cloud services for data management, access controls, resource management, data security, service management, etc., across all the three domains SaaS, PaaS and IaaS of cloud computing.

Table 2.2: Research questions and analysis of literature

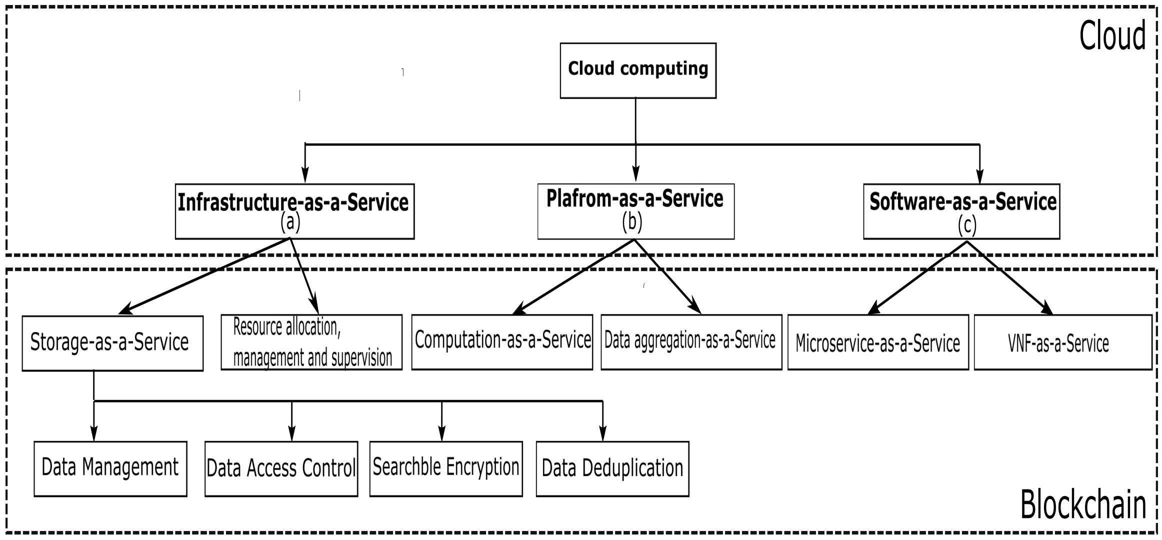

Figure 2.2: Technical dimensions of this survey

2.3 Blockchain-based Cloud Services

2.3.1 Blockchain-based Infrastructure-as-a-Service (IaaS)

IaaS is one of the most significant and fast growing cloud computing service model. A cloud provider under IaaS offers computing resources like virtual machines, storage disks, network devices, load balancers and firewalls. Most of the existing works on Blockchain-based IaaS focus on storage management and computational resource management. Therefore in this section we would first discuss the challenges in existing cloud storage and resource management and then elaborate the existing Blockchain-based storage and resource management works. Finally, we conclude the section with the summary of our findings.

2.3.1.1 Storage-as-a-service

2.3.1.1.1 Issues in traditional cloud Storage-as-a-Service

Outsourcing data to a remote cloud has become a common practice [33]. Every day large amounts of data are being generated, and data explosion is predicted when 5G and IoT networks are deployed worldwide. More and more users are adopting cloud storage to store their personal data, and many enterprises are also moving their data to the cloud in order to reduce on-premise costs. We categorize the entities involved in storage-as-a-service into three types: (1) Cloud provider (CP), (2) Data owner (DO), and (3) Data user (DU). CP is a rational and untrusted party with a large storage capacity. DO stores his data at cloud managed by CP. DO also authorizes DU to read / retrieve his / her data stored at the cloud. A DO has very little, or no control over his data as CP may give access of DO's data to third parties for monetary benefits. In today's world, CP will always try to reduce its costs adversely effecting data properties like:

(a) *Data integrity*: CP may modify stored data without the DO's knowledge.

(b) *Data auditability*: CP may not record or tamper the recorded data about the actions performed on DO's data.

(c) *Data searching*: When requested for a data search operation, CP may return incorrect or partial search results.

(d) *Data sharing and access control*: CP may not adhere to the access policy stated by a DO or may change the already stored access policy without DO's knowledge.

(e) *Data migration*: CP may not comply with storage location policy and may migrate DO's data without his knowledge.

(f) *Data deduplication*: CP saves huge storage cost due to data deduplication, and a rational CP may not pass those benefits to DO.

The list of Blockchain-based cloud data management works in literature is given in Figure 2.3.

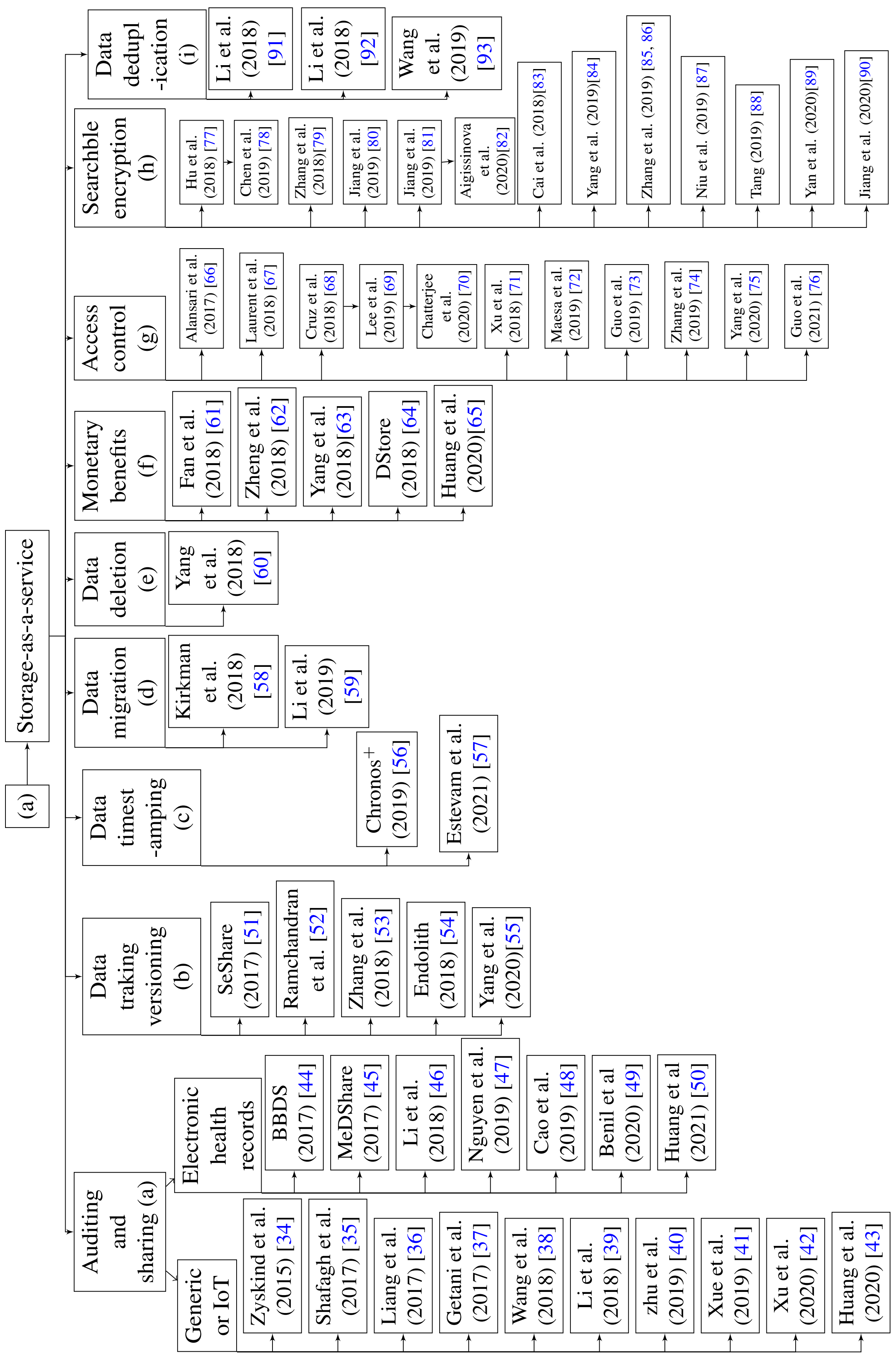

Figure 2.3: List of works in Blockchain-based storage-as-a-service.

In the following subsections, we discuss how Blockchain-based methods enforce the above-discussed properties [1].

2.3.1.1.2 Blockchain-based cloud data management

(a) Data auditing and sharing: Zyskind *et al.* [34] have presented a personal data management model in which the encrypted data is stored at a cloud, and the meta-data about the stored data is embedded into a Blockchain along with the access control permissions. The data storage and retrieval requests are sent as transactions (T_{data}) to Blockchain, and similarly, the changes in access control policies are also requested through transactions (T_{access}). During the verification of T_{data} for data retrieving, the access policy stored in the Blockchain is used to check whether the transaction initiator has necessary permissions to retrieve the data. As meta-data about the data and all the actions performed on the data are recorded in the immutable ledger this model ensures correct data auditability. Since the access control rules are stored in the Blockchain, the model provides fair data sharing and policy verification[2]. Shafagh *et al.* [35] and Liang *et al.* [36] applied the same technique of [34] for IoT stream data and data collected from drones respectively.

Gaetani *et al.* [37] have designed a two-tier Blockchain model to ensure the integrity of data stored in the cloud federation environment. The first tier Blockchain adopts a light-weight consensus protocol for fast performance by storing every operation performed on the federated database. The second tier Blockchain adopts proof-of-work (PoW) to ensure the integrity of the data stored at the first-tier Blockchain. Wang *et al.* [38] have developed a data-sharing mechanism by leveraging Blockchain as a communication channel to distribute the secret key used in attribute-based encryption of the data. They also store encrypted data indexes in smart contracts to increase trust on search operation.

Li and Zhou [39] have adopted Blockchain to store all the logs generated while storing a data block in cloud. However, storing logs increases search time during data auditing. To solve this problem, the authors introduced a proxy node which analyzes the blocks in

[1]To simplify the discussion; we purposely omit all the cryptographic concepts involved in securely storing, retrieving, auditing, searching, deleting and deduplicating of the data stored in the cloud.

[2]Most of the works presented later follow [34] for data auditing and sharing with minor / major modifications. We will discuss the necessary modifications wherever applicable.

Blockchain and forms an index. This indexing will reduce time when a DO searches for all the historical changes of his data. Zhu *et al.* [40] have used a smart contract to validate and trace modifications to cloud data. They have designed a delegated proof-of-stake (DPOS) [94] like consensus mechanism with a Blockchain node having a veto power to detect and override the attempts of adding erroneous data to Blockchain. However, a centralized entity with a veto power opposes the cause of using Blockchain.

One of the main challenges in traditional cloud data auditing is that a trusted auditor may compute the auditing result ahead of time and provide the evidence that he has executed the auditing protocol correctly. So, the challenge messages generated during the auditing must not depend on the DO or auditor and cannot be predefined (truly random challenge messages are required). To solve this problem, Xue *et al.* [41] presented an auditing model which uses random nonces generated during block creation of Bitcoin Blockchain. The auditor has to include the nonce of a particular block specified by the DO while generating challenge messages required for auditing. In [41], the auditors also publish the auditing results to public Blockchain for further tracing and auditing in future. Xu *et al.* [42] uses smart contracts as an arbitrator to resolve disputes regarding the integrity of data stored in the cloud and penalize misbehaving parties. Huang *et al.* [43] have introduced collaborative auditing Blockchain (CAB) running a credit-based consensus algorithm to increase the trust on data auditing results.

Electronic Health Records (EHR) auditing and sharing: Xia *et al.* have presented two Blockchain-based secure medical data sharing methods namely BBDS [44] and MeDShare [45]. In BBDS, the authors employ a light-weight permissioned Blockchain with a new Block structure to enhance the scalability and in MeDShare, the authors propose to use smart contracts for storing access policies and routed the access requests through Blockchain for secure data sharing. As smart contract tracks every action on data stored in the cloud, trust-less auditing is possible. Li *et al.* [46] have constructed a data preservation system (DPS) to store and share personal health data securely. DPS allows the users to submit the hash of the unpreserved data to Blockchain. Later, the users can check and validate the preserved data by retrieving hash from the Blockchain. Nguyen *et al.* [47] have introduced a trusted manager who on receiving a data request verifies the request with access controls stored in smart contracts and take appropriate action based on the verification result. Cao *et al.* [48] have adopted Blockchain to protect patient's health data outsourced to the cloud from illegal modifications by doctors. Benil and Jasper [49] have presented an authorized Blockchain-based data sharing, and auditing model for personal health records. Recently, Huang *et al.* [50] have proposed a Blockchain-based solution for identifying manipulation of EHR data. They have constructed proof-chain to store users' manipulation logs on EHR data. At the later stage, the logs stored in proof-chain are used as evidence for rights protection.

(b) Data tracking and versioning: In SeShare [51], multiple DO's shares and modify cloud data and record those changes in the Blockchain for avoiding conflicts arising due to file sharing. Ramachandran *et al.* [52] have designed a document management system called DataProv which contains two types of smart contracts: Document_Track and Vote. The Document_Track contract facilitates operations like adding a document, granting and revoking access to the document, and tracking the changes in the document. Any changes to the document are recorded in the ledger only through a voting process conducted by the vote contract. Zhang *et al.* [53] have designed a document life-cycle management method where all stages of document management like creation, modification and ownership transfer are recorded in the Blockchain to ensure integrity and auditability. Endolith [54] stores meta-data of the data stored in the cloud using a smart contract enabling data auditing, validation, tracking and versioning.

To provide high availability and to avoid accidental loss of data, multiple replicas of data may be stored at multiple clouds. However, managing multiple-replicas correctly without loss of integrity of every replica is a challenging task. This challenge is considered by Yang *et al.* [55], where a modification record table is maintained to track file changes, and similar to [41] uses the random nonces from Blockchain to generate challenge messages during auditing.

(c) Data time-stamping: A tamper-proof, and correct time-stamping of files is required before storing them in the cloud. However, in traditional cloud computing, a trusted time-stamping server is setup, which may deviate from its operation by colluding with DO or CP. To mitigate this problem, a time-stamping service based on Ethereum Blockchain known as Chronos$^+$ is developed by Zhang *et al.* [56]. For secure and correct time-stamping for a file, a DO has to retrieve the hash values of φ- successive blocks that are most recently accepted on the Ethereum Blockchain and include those hash values in the file. This procedure enables the users to prove that the file was generated no longer than the physical time of the last block of φ- successive blocks. However, miners to some extent can affect the block time-stamp. To mitigate this problem, Estevam *et al.* [57] have proposed a time-stamping service which combines smart contracts and distinct time providers. They have achieved a time-stamping accuracy of milliseconds.

(d) Data migration: DO do not have control over migration of his / her data between cloud providers. Kirkman and Newman [58] uses smart contracts to record data migration between cloud providers. DO can know the current position and status of their data by querying the smart contract. Another data migration process between a group of connected cloud data centres is discussed by Li *et al.* [59]. In their method, every data duplication and data migration operation between cloud providers are recorded on the Blockchain.

(e) Data deletion: Yang [60] have introduced a cloud data deletion method, where the cloud provider generates a proof of deletion and publishes it on a Blockchain. Then, the proof is verified publicly making the deletion operation transparent.

(f) Monetary benefits: Fan *et al.* [61] have stored encrypted data in the cloud and its access policies on Blockchain, but they have innovated in sharing the data by adopting the DPOS [94] consensus mechanism. The miners in [61] are rewarded with data of their interest. When a DU needs data, he / she has to send a request to the miner holding that data. The miner verifies the access policy on data and sends the data to the DU if and only if the access policy is satisfied. DU has to obtain the key to decrypt the data from the DO. This setup reduces congestion and latency in the network as miners are distributed around the world, and DUs are not required to send requests to a centralized cloud. Zheng *et al.* [62] have designed a smart contract-based market place for selling and buying personal data. They have designed a crypto-token known as Personal Health Data coin (PHD coin) to pay a DO in exchange for personal data. Some DOs do not share / sell their data but allows the queries on the data. In this model, the data is protected with differential privacy techniques, and a threshold limit is set on the number of queries allowed before leaking privacy.

Yang *et al.* [63] have employed smart contracts to manage privacy budget. Smart contract may accept / reject a query based on the remaining privacy budget. If a query is accepted, then an anonymization service outputs result by adding sufficient noise to the actual query result, and the privacy budget in the smart contract is updated accordingly. DStore [64] explores the idea of using empty disks of home users (lessors) to form a distributed cloud. A single data file may be stored on several disks stationed at different locations across the globe under different lessors causing data auditing problems. A smart contract is used to check data auditing results, to pay storage fee to lessors, and to impose fines on the lessors if they provide incorrect auditing results. Huang *et al.* [65] have extended their work of [51] with fair incentives so that the users who contributed to the meta-data stored in Blockchain will get their incentives fairly.

The comparison of different data management models is presented in Table 2.3.

Paper	Data	Blockchain Platform / Consensus algorithm	Off-chain data storage	Data access control through BC	Data sharing	Data integrity / auditing	Data deletion	Data versioning	Data time-stamping
Zyskind *et al.* [34]	Generic	Bitcoin	DHT	Yes	Yes	Yes	No	No	No
Shafagh *et al.* [35]	IoT	Bitcoin	DHT	Yes	Yes	Yes	No	No	No
Gaetani *et al.* [37]	Generic	Two tier: Consortium PoW	Distributed Cloud	No	No	Yes	No	No	No
Alansari *et al.* [66]	Generic	Ethereum	Federated cloud	Yes	Yes	Yes	No	No	No
BBDS [44]	EHR	Not discussed	DB	Yes	Yes	No	No	No	No
MeDShare [45]	EHR	Ethereum	DB	Yes	Yes	Yes	No	No	No
Wang *et al.* [38]	Generic	Ethereum	DHT	No	Yes	No	No	No	No
Zhang *et al.* [53]	Generic	Ethereum	DB	No	No	Yes	No	Yes	Yes
Liang *et al.* [36]	Drone	PoW	DB	No	No	Yes	No	No	No
SeShare [51]	Generic	Not discussed	DB	No	Yes	Yes	No	Yes	No
yang *et al.* [60]	Generic	Not discussed	DB	No	No	No	Yes	No	No
Endolith [54]	Generic	Ethereum	HDFS	No	No	Yes	Yes	Yes	No
DataProv [52]	Generic	Ethereum	DB	No	No	Yes	No	Yes	No
Fan *et al.* [61]	ESR	Voting-based	DB	Yes	Yes	No	No	No	No
Kirkman *et al.* [58]	Generic	Ethereum	DB	Yes	Yes	No	No	No	No
Li *et al.* [59]	Generic	Private Ethereum	Federated cloud	No	Yes	Yes	No	No	No
Li *et al.* [39]	Generic	Ethereum	Aliyun	No	No	Yes	No	No	No
Zheng *et al.* [62]	EHR	Ethereum	DB	No	Yes	No	No	No	No
Yang *et al.* [63]	Privacy sensitive data	Hyperledger (BFT)	DB	Yes	Yes	No	No	No	No
DStore [64]	Generic	Ethereum	Distributed cloud	No	No	Yes	No	No	No
Zhu *et al.* [40]	Generic	Ethereum	DB	No	No	Yes	No	Yes	No
Xue *et al.* [41]	Generic	Bitcoin	DB	No	No	Yes	No	No	No
Nguyen *et al.* [47]	EHR	Ethereum	DHT	Yes	Yes	Yes	No	No	No
Cao *et al.* [48]	EHR	Ethereum	DB	No	No	Yes	No	No	No
Li *et al.* [46]	EHR	Ethereum	DB	No	Yes	Yes	No	No	No
Chronos+ [56]	Generic	Ethereum	DB	No	No	No	No	No	Yes
Xu *et al.* [42]	Generic	Two tier: Consortium Ethereum	DB	No	No	Yes	No	No	No
IPANM [65]	Generic	Ethereum	DB	No	Yes	Yes	No	No	No
Yang *et al.* [55]	Generic	Not discussed	DHT	No	No	Yes	Yes	Yes	No
Benil *et al.* [49]	EHR	Ethereum	DB	No	Yes	Yes	No	No	No
Huang *et al.* [43]	Generic	Credit-based	DB	No	No	Yes	No	No	No
Huang *et al.* [50]	EHR	PBFT	Cloud	No	Yes	Yes	No	No	No
Estevam *et al.* [57]	Generic	Ethereum	DB	No	No	No	No	No	Yes

Table 2.3: Comparison of Data Management models. DHT - Distributed hash table / Interplanetary file system (IPFS) in cloud. DB - Database in cloud. HDFS - Hadoop distributed file system.

(g) Blockchain-based cloud data access controls In traditional cloud computing, the access policies are stored in the cloud, which is assumed to process the access requests honestly according to the policy. However, in practice, the cloud provider may deviate from the access policy and may reject access to legitimate DO / DUs or allow access to illegitimate DO / DUs.

Alansari *et al.* [66] have presented an identity and access control system for federated cloud by integrating Blockchain and Intel SGX [95] technologies. Blockchain ensures non-tampering of access policies, and Intel SGX protects the confidentiality and integrity of the policy enforcement process. Laurent *et al.* [67] have recommended creating a smart contract with a specific access control list for every data block outsourced to a remote cloud. So, a DU is authorized based on the challenge-response protocol played between him and remote cloud based on the information stored in the smart contract. Cruz *et al.* [68] have designed role-based access control using smart contracts (RBAC-SC) to verify access permission across different organizations effectively. Every organization creates a smart contract which consists of functionalities for storing user-role assignments, modifying assignments, and revoking assignments. Similar to [67] during authorization, a challenge-response protocol is executed that verifies the ownership of roles based on information stored in the smart contract. Lee *et al.* [69] have also introduced RBAC with smart contracts without the challenge-response protocol. Chatterjee *et al.* [70] have decoupled access control logic with the business logic of the smart contract and proposes a dynamic role-based access control model using smart contracts.

As the RBAC model suffers from scalability, heterogeneity, and spontaneity problems, Xu *et al.* [71] have transcoded capability-based access control policy as smart contracts to support hierarchical and multi-hop delegation in a federated environment. In their work, the smart contract manages federated delegation relationships and capability tokens. CP issues authorization and revocation tokens through smart contracts so that the other nodes in the federation accept or reject the access requests. Maesa *et al.* [72] have codified attribute-based access control policies as smart contracts. They have adopted XACML [96] for defining policies, and the smart contracts are considered as an executable version of XACML policy. The smart contract also manages the attributes representing the features of subjects required to evaluate the policy. Guo *et al.* [73] have introduced multi-authority attribute-based access control with smart contracts. Guo *et al.* [76] have designed an efficient traceable attribute-based encryption scheme for fine grained data sharing on Blockchain.

Zhang *et al.* [74] have designed a smart contract-based access control framework with both static access rights control and dynamic rights validation. They have developed three types of contracts: (1) Access control contracts (ACCs) (2) Judge Contract (JC) and (3) Register Contract (RC). For every object-subject pair, an ACC is created to support adding, deleting and updating access controls. ACC also reports misbehaviour of subjects to JC. JC on receiving ACC request initiates a misbehaviour judging method and returns appropriate penalty. RC manages all the ACCs and the judging methods.

All the access control frameworks discussed till now suffer from privacy problems because the access data stored in Blockchain is publicly available. To mitigate this problem, Yang *et al.* [75] have presented AuthPrivacyChain in which the access control policies are encrypted and stored in the Blockchain. Whenever a cloud receives the access request, it retrieves and decrypts the policy from a smart contract and verifies the access request. The cloud also publishes logs of all the access requests on the Blockchain.

(h) Blockchain-based cloud data searchable encryption In tradition searchable encryption (SE), CP is assumed to perform the search service and return the search results correctly. However, in practice, CP is untrusted and rational entity which may indulge in fraudulent activity and may return partial or incorrect search results. At the same time, the DU may act as malicious and refuse to pay the service fee after receiving the correct search results. This situation leads to the problem of service-payment unfairness and mutual distrust between the DU and CP.

Hu *et al.* [77] have presented a Blockchain-based solution to address the limitations of searchable encryption. The search index is stored in a smart contract, and the search algorithm is also modeled as a smart contract functionality. These two steps ensure the integrity of the index and correctness of search results. They also introduced the notion of fairness in SE such that a DU receives the search results if and only if he pays for the search operation and the DO receives the payment if the search token for the requested keyword is sent to the smart contract. Chen *et al.* [78] have adopted [77] for EHR data. They build an index using complex logical expressions facilitating DU to construct queries like "(disease = 'disease name') AND (num1 $\leq$ age $\leq$ num2)". In both the works, the search functionality is invoked by the DO inducing unnecessary cost to him.

Zhang *et al.* [79] have presented a Bitcoin-like transaction model for obtaining fairness in search results. In their model, CP commits a transaction with enough deposit on Bitcoin so that if it provides wrong results or aborts during the subsequent phases, the DU can claim the deposit committed by CP. Similarly, DU also commits the payment, which will be transferred to CP on providing correct search results or refunded in case the CP behaves maliciously. Jiang *et al.* [80] have applied a stealth authorization method to achieve privacy-preserving access authorization delivery through smart contract. In their method, first, DO sends a stealth authorization information to a smart contract and later this information is retrieved by DU to construct a search token. With that search token, DU calls the search functionality to obtain search results, thereby reducing the burden on DO. Another advantage is that once the DU is authorized, he can search the same keyword multiple times without contacting DO.

Storing index and performing search operations are costly as they consume more computing resources and time, leading to verifiers dilemma [97]. To reduce search cost, Jiang *et al.* [81] have introduced bloom filter based search to find out low-frequency keyword in the multi-keyword search and filter the encrypted database using the keyword. Since the selected keyword is of low frequency, most of the keywords are excluded from the result, thus reducing the search cost significantly. Another bloom filter based method developed on Hyperledger fabric is discussed by Aigissinova *et al.* [82]. Cai *et al.* [83] presented a reliable SE model with negligible cost when both DU and CP are honest. DU verifies the correctness of the search result returned by a CP (Inter-planetary file system (IPFS) service peer), and if found that the result is not correct, then he invokes a dispute resolution mechanism. When a dispute has raised a set of volunteer nodes known as arbiter shard perform the search operation independently. They use a Byzantine voting mechanism to reach consensus among the arbiter shard nodes. The smart contract rewards / penalize the DU and CP according to the results returned by arbiter shard. Yang *et al.* [84] have designed a set of smart contracts interacting with each other to facilitate fair payments between DO, DU and CP.

Contrary to other smart contract-based works, Yang *et al.* [84] have proposed to verify the search results computed by CP using a smart contract for correctness. To further reduce the cost incurred due to public Blockchain, Zhang *et al.* [85, 86] have employed a consortium chain whose major stakeholders are CPs. Encrypted files are stored in Inter-Planetary file system (IPFS) which returns the hash of the stored file as URL. Through transactions, the consortium chain stores the mapping of URLs with the corresponding plain-text. The Merkle root hash of the state of the consortium chain is committed periodically on a public Blockchain guaranteeing its integrity. Apart from maintaining the consortium chain, the CPs also provide computing power to DU for search operations and provide resultant IPFS hash pointers of the queried data to the DU.

Niu *et al.* [87] have also adopted the permissioned Blockchain model for securely searching and sharing electronic health records. Tang [88] has discussed several limitations of directly adopting Blockchain to solve problems in searchable encryption. More importantly, he has identified two privacy leakages: (a) Search pattern leakages and (b) Access pattern leakages. He has also pointed out that storing search index and search results forever in the immutable ledger may lead to attacks in future which are currently unknown. He has presented two new methods by storing and searching the encrypted indexes in centralized servers, thus by limiting the Blockchain to ensure only the fairness. He has employed N servers all will perform search operation on locally stored indexes and commits the search result on the Blockchain. At a later stage, the servers reveal the commitment and the smart contract compares all the results returned by servers for similarity. If all are equal, then each server receives payment for computing search operation correctly; otherwise, an off-line arbitration mechanism is initiated. To overcome employing third-party arbitration, the author improved the initial design by using public-key encryption and zero-knowledge proofs, where the entire dispute resolution will happen through a smart contract.

Sometimes a data owner / user may send a misspelt keyword for searching the index. As the above mentioned works fail to handle misspelt keywords, Yan *et al.* [89] have designed a verifiable fuzzy keyword SE [98] to handle misspelt keywords. Similar to [84], in [89] also the search results are computed by CP and are verified by smart contract for correctness. However, the authors applied RSA accumulators [99] for verifying the search results. A DO after generating index computes an accumulated value $acc(C)$ for a document set C and sends this value to a smart contract. CP, after receiving search token computes search results along with verification evidence proof $pf(C)$ and sends proof to a smart contract. The verify functionality of smart contract takes both $acc(C)$ and $pf(C)$ along with search results as input and outputs search results if CP has sent the correct result otherwise outputs $\perp$ (error) indicating wrong results or wrong proof. Their design also supports fair payments and penalizes malicious entities.

In most of the works discussed till now, the search token is generated by DO, and in some works, DU generates search token based on authorization information sent by the DO. However, in both models, DO knows about DU's keyword revealing their private interest. To avoid this retrieval information leakage and hide data user's keyword Jiang *et al.* [90] have constructed searchchain that aims to assure private search over authorized keywords with unchanged retrieval order. They modified the oblivious keyword search (OKS) [100] as OKS with authorization (OKSA) through which DU can search privately within an authorized set of keywords. Further, they use ordered multi-signatures (OMS) while generating blocks to commit the sequence of retrieval transactions. Comparison of Blockchain-based searchable encryption systems is presented in Table 2.4.

(i) Blockchain-based cloud data de-duplication Most of the data being uploaded to the cloud is redundant [101] and thus wasting large storage space. To avoid redundancy and save storage costs, CP use data de-duplication technique. However, the issues with existing de-duplication techniques are:

(a) *Correct de-duplication rate*: As CP save storage costs by adopting de-duplication, proper incentives on storage fee are required for DO to adopt de-duplication. In general, the incentives on storage fee are calculated based on the de-duplication rate. A rational CP to increase its profits may not compute the de-duplication rate correctly and thus making the DO pay higher fees even though he opted for the de-duplication.

S.No	**Paper**	**Platform**	**Storage location**	**Keyword**	**Search**	**Verification**	**Fair payments**
1	Hu *et al.* [77]	Ethereum	Cloud	Single	Smart contract	Not required	Yes
2	Zhang *et al.* [79]	Bitcoin	Cloud	Single	Cloud	Bitcoin Script	Yes
3	Cai *et al.* [83]	Ethereum	Decentralized	Single	Decentralized node	Data user / Arbitar Shard	Yes
4	Zhang *et al.* [85]	Hybrid (consortium and public)	IPFS	Multiple	Cloud	Data user	No
5	Jiang *et al.* [80]	Ethereum	Cloud / IPFS	Single	Smart contract	Not required	No
6	Chen *et al.* [78]	Ethereum	Cloud	Single	Smart contract	Not required	Yes
7	Niu *et al.* [87]	Permissioned	Cloud	Multiple	Permissioned Blockchain	Not required	No
8	Jiang *et al.* [81]	Ethereum	Cloud	Multiple	Smart contract	Not required	No
9	Yang *et al.* [84]	Ethereum	Cloud	Multiple	Cloud	Smart contract	Yes
10	Tang [88]	Ethereum	Cloud	Single	Cloud	Arbitar / smart contract	Yes
11	Aigissinova *et al.* [82]	Hyperledger Fabric	Cloud	Single	Smart contract	Not required	No
12	Jiang *et al.* [90]	Searchchain	Decentralized	Single	Not discussed	Data user	No
13	Yan *et al.* [89]	Ethereum	Cloud	Single	Cloud	Smart contract	Yes

Table 2.4: Comparison of Blockchain-based Searchable encryption systems

(b) *Fair payments*: An honest CP should receive storage fee if and only if an honest DO receives the file link of the data requested to store at the CP.

(c) *Cross-CP de-duplication*: Currently, to perform cross-CP de-duplication, a trusted party have to be recruited, which maintains a central repository storing meta-data of all the files stored at different CPs. However, having a trusted party introduces a single point of failure and finding an idle party which will behave honestly at all times is difficult.

Li *et al.* [91] have designed CloudShare to enable cross-CP data de-duplication. When a CP receives an upload request for file a f, it checks whether it possesses a copy of f. If the check is valid, then it simply adds the requested DO to the set of DOs that are already registered to the file f. If the check is not valid, then it queries the private Blockchain maintained by a set of CPs whether any CP has a copy of f. If a CP_j holds the copy of f, then the DO is added to set of the DOs that are already registered to the file f at CP_j. Otherwise, it asks the DO to upload the file and the ownership information is recorded in the private Blockchain. Collaborating through Blockchain eliminates the need for a central repository and saves a lot of storage spaces to CP and also saves bandwidth to DO.

A smart contract-based de-duplication method is presented by li *et al.* [92]. Before uploading the file, DO downloads the meta-data of the files stored at a CP from the Blockchain and performs the duplication check locally. If a duplicate is found, then the DO requests the smart contract to register him as an owner to the duplicated file. The smart contract in-turn sends a script to DO. DO signs and send the script to the CP. Then, the CP completes the script and sends it to the smart contract, which then adds the DO as an owner to the requested file. Wang *et al.* [93] have used a smart contract to facilitate fair payments for de-duplication between CP and a DO. They replaced the trusted party for payments in traditional de-duplication with a smart contract. In their work, the DO has an option to initiate the penalty transaction when the CP behave maliciously without sending the file link to the DO.

2.3.1.1.3 Summary

In traditional Storage-as-a-Service model, the cloud provider is assumed as an honest-but-curious party who behaves honestly at all times. Later, to have more trust in the cloud, some works presented a trusted auditor to audit the data and a trusted access manager for access control on data. Nevertheless, hiring a trusted party is costly and finding a trusted party which will behave honestly at all times is difficult. Also, having a one more centralized entity in the loop makes the system more vulnerable to a single point of failure. In this section, we have discussed works which replace the trusted party with Blockchain for auditing and access control. Also, we have discussed the works realizing the correctness through Blockchain in search results returned by the cloud. Outsourcing search queries to smart contract yield a correct and immutable result and requires no further verifications. Further, we have discussed works obtaining correctness and fairness in cloud data de-duplication.

2.3.1.2 Resource allocation, management and supervision

Resource allocation has been one of the most widely studied problems in cloud computing. Allocation of resources to users involves decision making concerning when, what, how much and where to allocate the available resources [102]. The resource management life cycle comprises resource advertisement, allocation, monitoring and freeing. During resource allocation, several factors, like resource utilization, pricing, availability, quality of service, etc., are considered. Once the resources are allocated through some mechanism, it is essential to monitor the state of the allocated resources. We categorize the entities involved in resource allocation and supervision as cloud user CU and cloud provider CP.

2.3.1.2.1 Issues in cloud resource allocation, management and supervision

The issues in cloud resource allocation, management and supervision are as follows:

(a) CP advertises a resource for rent / lease. However, in conventional cloud computing, users cannot verify the authenticity of the advertised resources.

(b) A rational CP may allocate the same resource to one or more users leading to overloading of resource, thereby severely affecting the quality of service.

(c) Resources are allocated by a resource allocator which is a part of CP. The allocator may behave maliciously during the allocation and may not run the allocation algorithm correctly without any prejudice.

(d) CP is in full possession of the logs generated during the resource usage. The provider may tamper the logs to cover up his lapses during the resource provisioning. The logs may also contain sensitive information which can be accessed by an untrusted provider.

The list of Blockchain-based resource allocation and supervision schemes is given in Figure 2.4.

2.3.1.2.2 Blockchain-based resource allocation, management and supervision

In Blockchain-based resource allocation, most of the works focus on resource pricing because pricing models increase the total utility of the cloud provider [102]. Among several resource pricing models, auction-style pricing mechanisms [117, 118, 119] have gained more interest as they reflect the underlying trends in demand and supply of cloud resources. Gu *et al.* [103] have designed a Vickrey–Clarke–Groves (VCG)-based auction [120] mechanism using smart contracts where a CU posts his request by sending a transaction to the

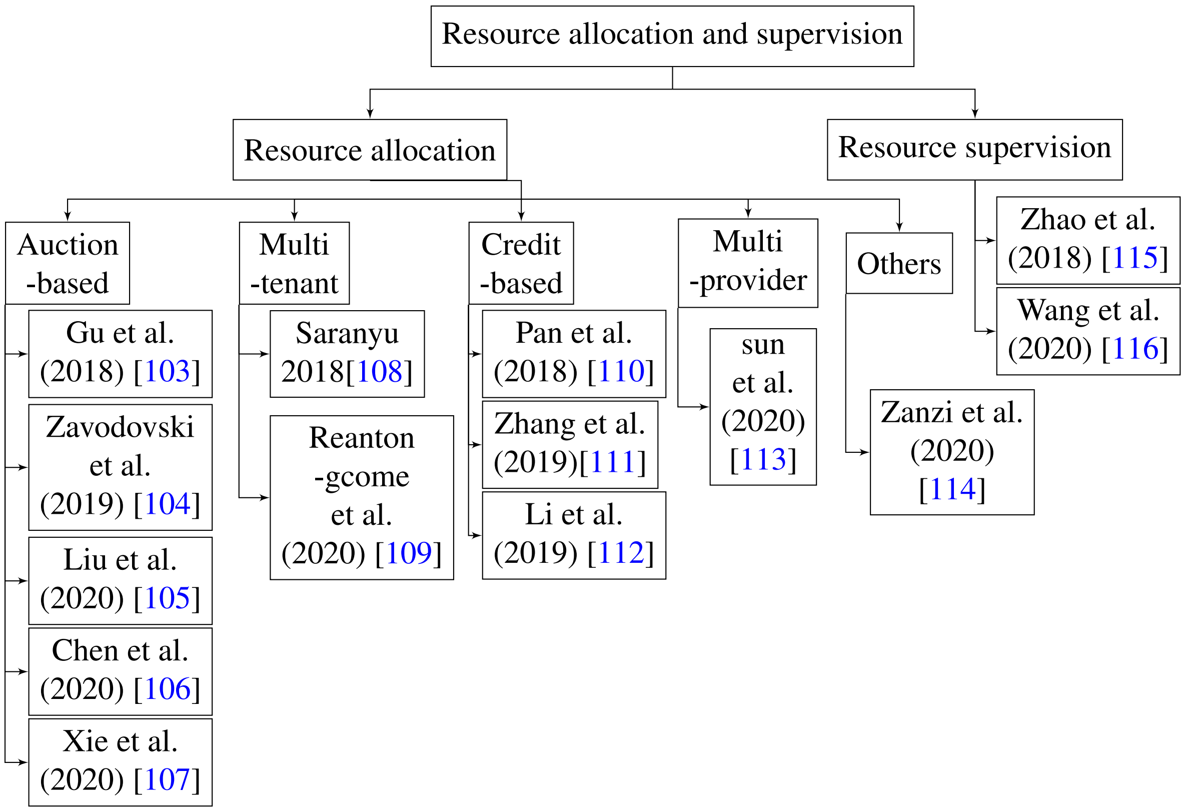

Figure 2.4: List of works in Blockchain-based Resource allocation and supervision

smart contract. CP responds with a sealed-bid which is revealed at the end of the bidding round. Then the smart contract computes the winner with the highest bid, and the winner pays price equal to the highest bid of losers. The smart contract also handles the payments to providers. Zavodovski *et al.* [104] have transcoded a dominant strategy incentive compatible (DISC) double auction [121] method as a smart contract. They have also used a two-phase bidding protocol similar to [103] where bids are committed initially and revealed later. The smart contract computes the matching of CUs with CPs according to the auction rules. Another two-phase bidding double auction method is presented by Liu *et al.* [105], where a long-term auction for mobile blockchain (LAMB) is modeled as a smart contract to determine optimal matching for users and providers.

A Blockchain-based combinatorial auction method for VM allocation is discussed by Chen *et al.* [106]. They realize auction fairness (bids once committed cannot be modified) and trade fairness (the honest provider receives pay if and only if the honest user receives the requested VMs). They use the ladder mechanism presented in [122] to obtain trade fairness. A continuous double auction (CDA) method is modeled as a smart contract by Xie *et al.* [107]. In [107], the providers and users submit asks and bids respectively at any time to a smart contract. The smart contract runs CDA auction, and if there is a deal, then the result is broadcast to everyone.

Zanzi *et al.* [114] have introduced NSBchain to allocate network slice resources to CU through smart contracts. NSBchain is responsible for slice allocation, enforcing policy, billing and resource management. Reantongcome *et al.* [109] also have considered smart contracts for resource allocation in multi-CU scenario. They have addressed the co-resident attack by auditing the activities of malicious users logged in the Blockchain. Zhang *et al.* [111] have developed two smart contracts namely smart trading contract (STC) for facilitating resource trading and smart loan contract (SLC) for facilitating the resource constraint users to borrow coins from banks and use these coins to pay to providers for consuming resources.

Saranyu [108] is an application build on Quorum [123] to provide user management through smart contracts. Initially, users and providers have to be registered with a smart contract. And then a mapping between users and providers will happen through a smart contract which results in issuing delegation rights to users. The smart contract also monitors the resource usage, and this information is used for billing users. The billing is settled in the native cryptocurrency of underlying Blockchain technology. Overall, Saranyu offers four services: identity management, authentication of users, authorization on resource exploitation and charing. Kempf *et al.* [124] constructed a cloud market place based on [108] improving the transparency and auditability of cloud market place.

Pan *et al.* [110] have discussed a credit-based resource management model in which the provider's resources are mapped with internal currency coins (credit coins). When a user's account is created on a Blockchain, it is bootstrapped with some initial credit coins. The amount of coins a user is holding determines the number of resources he can obtain from the provider. The provider keeps an account of debits and credits of a user and provides the resources correspondingly (other factors like priority, application type, past behaviour, etc., are also considered). The interactions like registration, requesting and allocating resources triggers the smart contracts for secure logging and auditing. Another credit-based resource allocation model is discussed by Li *et al.* [112]. The users can lend coins from other users or providers and pay them to providers for fast computing resource trading. All the lending and payment transactions are recorded on the Blockchain for secure transaction history, and the credit values are automatically adjusted according to their lending and payment transactions. Recently, sun *et al.* [113] have presented a Blockchain-based cooperative method where the providers work cooperatively to provide resources to users. They have modelled break-even and break-even free double auctions as smart contracts to determine the price and to allocate resources.

Zhao *et al.* [115] have designed a hybrid Blockchain, namely Mchain to log all the activities of VM (VM measurements) on Mchain to provide better transparency, integrity, auditability and controllability on allocated cloud resources. Mchain is a two-layered Blockchain where the first layer outputs a semi-finished block, and the second layer takes the output from the first layer and generates a mature block which is stored on Mchain. They are two types of blocks in Mchain: data block (VM measurements data) and policy block (VM's user-defined access policy). In the first layer, a candidate block is broadcast, and every other node in the Blockchain validates all the data in the block and signs the block if validated. If every node signs the block, then consensus on the candidate block is achieved, and it is sent to the second layer. In the second layer, all the valid nodes begin the PoW mining tasks, and when a nonce is found, the semi-finished block will become mature and is broadcast in the Blockchain network. Then every node will validate the block and adds it to their local copy of Mchain.

In previously discussed works when the providers advertise their resources, there is no mechanism for users to verify the provider's claims regarding available resources. Wang *et al.* [116] have designed a hybrid Blockchain network consisting of a public Ethereum network and consortium Blockchains. A provider before joining the consortium network must brace for public verification of their resource capabilities on Ethereum network. The consortium Blockchain is used for real-time monitoring of computing resources. They have also introduced a token called ResourceCoin (RCoin) reflecting the available resources at the provider. The comparison of resource allocation and supervision methods is presented in Table 2.5.

Category	Paper	Platform	Blockchain Objective
Auction-based Resource Allocation	Gu *et al.* [103]	Ethereum	VCG auction
	Zavodovski *et al.* [104]	Ethereum	Double auction (DISC)
	Liu *et al.* [105]	Ethereum (DPOS)	Double auction (LAMB)
	Chen *et al.* [106]	Ethereum	Combinatorial auction
	Xie *et al.* [107]	Ethereum	Double auction (Continous)
Multi-tenant Resource Allocation	Reantongcome *et al.* [109]	Ethereum	Resource allocation and activity logging
	Saranyu [108]	Quorum	Resource allocation and activity logging
Credit-based Resource Allocation	Zhang *et al.* [111]	Ethereum	Resource allocation and coin loaning
	Pan *et al.* [110]	Permissioned Ethereum	Resource allocation and activity logging
	Li *et al.* [112]	PoW	Resource allocation
Multi-provider Resource Allocation	sun *et al.* [113]	Ethereum	Double (break-even and break-even free)
Resource Supervision	Zhao *et al.* [115]	Mchain	VM measurments
	Wang *et al.* [116]	Hybrid (Ethereum and Consortium)	Resource allocation and monitoring
	Zanzi *et al.* [114]	Hyperledger	Network slice allocation

Table 2.5: Comparison of Blockchain-based resource allocation methods

2.3.1.2.3 Summary

In this section, we have discussed various Blockchain-based resource allocation and supervision systems. We observe that the untrusted resource allocator in traditional cloud computing can be replaced by a Blockchain to allocate and price the resources correctly and fairly. The inclusion of Blockchain increases the trust in resource allocation and pricing. We have also discussed works which log resource usage into an immutable ledger, thereby eliminating the odds for the untrusted provider to tamper the logs.

2.3.2 Blockchain-based Platform-as-a-Service

PaaS support businesses to develop and host applications giving the developers freedom to concentrate on building software without having to worry about the operating system, software updates, infrastructure and storage. As PaaS is a connector between IaaS and SaaS, the literature of Blockchain-based PaaS overlaps with that of Blockchain-based IaaS and SaaS. Therefore, in this section, instead of discussing PaaS, we rather focus on the literature of Blockchain-based PaaS applications developed and hosted using PaaS. Mainly, we focus on two of the most widely developed PaaS applications: (1) Computation-as-a-service (CaaS) and (2) Data aggregation-as-a-service (DaaS). In CaaS, a cloud provider sets up an execution environment and a user outsources a computation to cloud expecting output. For example, user outsources a data mining task along with input to a cloud which executes the task and returns the mining results. Similarly, in DaaS, a cloud provider sets up a data collection application which requests specific data from users. For example, a cloud provider sets up a mobile crowdsensing task and publishes the task details publicly. Interested participants read the task requirements and send the requested data. The cloud platform processes the received data and obtains meaningful results. The list of Blockchain-based platform-as-a-service works is presented in Figure 2.5.

2.3.2.1 Computation-as-a-service

2.3.2.1.1 Issues in CaaS

Outsourcing computation has become a common practice. A resource-constrained user outsources a resource-intensive computation to large computational systems like a cloud. The cloud computes and returns the result of the outsourced computation. The user pays a pre-agreed amount of money to cloud for using its computational resources. In today's world, an agreement between the user and the cloud provider about the cost and other QoS parameters is needed before outsourcing the computation. In many cases, the cost of consuming resources has to be paid before actually using cloud service. The user to have greater confidence in the computation performed by the cloud, he has to verify the result of the computation for correctness.

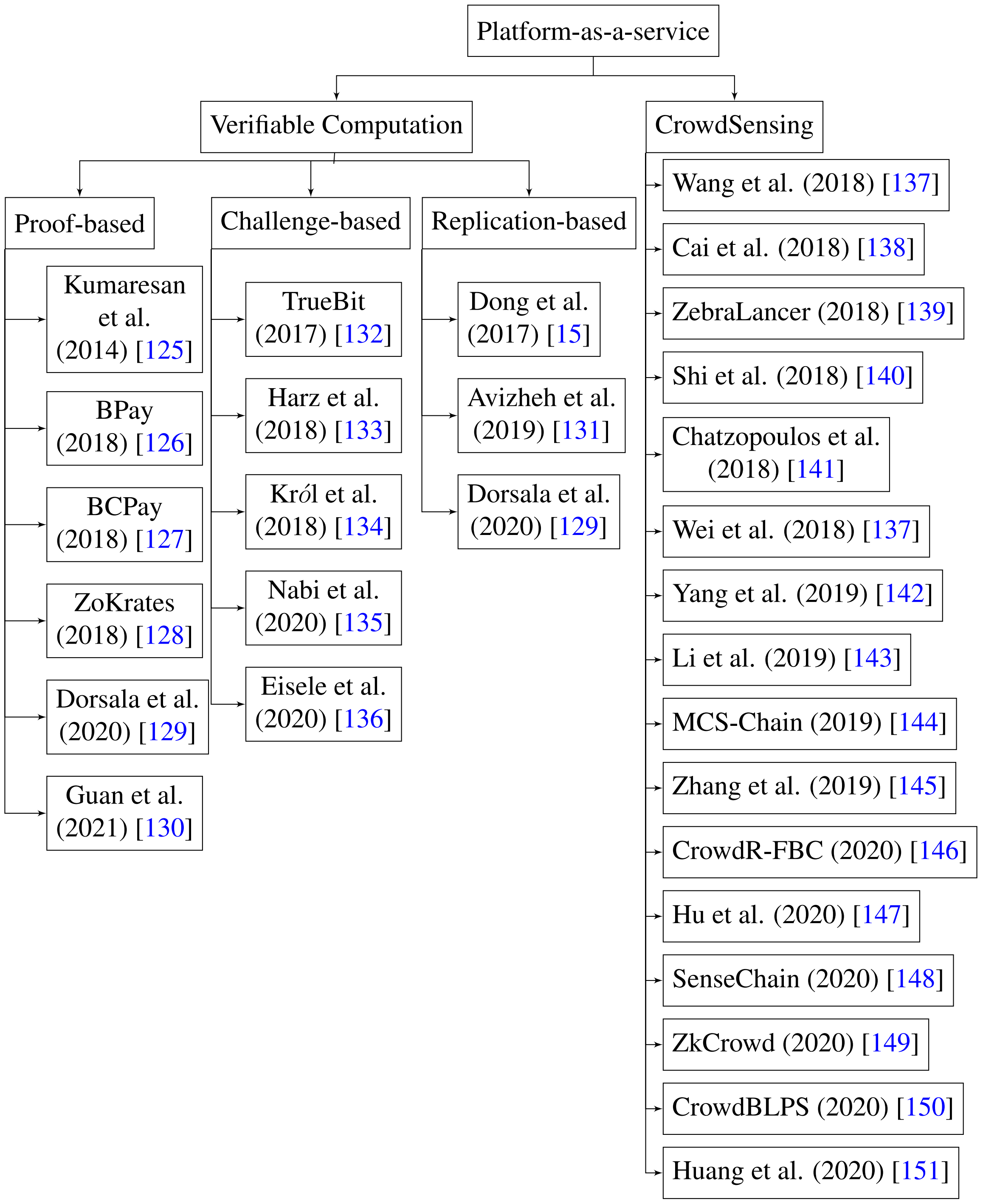

Figure 2.5: List of works in Blockchain-based platform-as-a-service.

In traditional cloud computing, it is not possible for a user to pay to a cloud only after verifying the correctness of the returned result. In general, the cloud is assumed to be trusted by the user. However, the cloud may behave rational and to save computing resources; it may not compute the result correctly. Then, the only option for the user is to get back his payment by going through a cumbersome legal process which may take additional resources and time.

Nevertheless, with the advent of Blockchain technology and smart contracts, a new cloud computing paradigm has evolved where the cost for outsourcing a computation will be paid to the cloud provider if and only if the cloud provider computes and sends the result of the outsourced computation correctly. The verification of correctness of result depends on the type of the verification technique employed.

2.3.2.1.2 Types of verifiable computation techniques

There are three techniques to verify the correctness of the result computed by the cloud: 1) Proof-based methods (PBVC), 2) Replication-based methods (RBVC), and 3) Challenge-based methods (CBVC). In PBVC, the cloud has to submit a proof of correctness along with the result of the computation. Some of the well known proof-based systems are summarized by Walfish *et al.* [152]. In RBVC, the computation is outsourced to multiple clouds, and the results from them are compared for similarity. If the comparison is a success, then the user accepts the result; otherwise, a dispute-resolution protocol is initiated to identify the malicious cloud who submitted the wrong result. Some of the recent works in RBVC are Belenkiy *et al.* [153], Cannetti *et al.* [154] and Kupcu [155]. In CBVC, the computation is outsourced to only one cloud, and any public party can challenge the result computed by the cloud. If no party challenges the result, then the user accepts it. Otherwise, a dispute resolution mechanism is executed.

2.3.2.1.3 Blockchain-based Verifiable Computation

Kumaresan *et al.* [125] have introduced the notion of Blockchain-based verifiable computation. In their work, the user creates a Bitcoin output script containing a pre-agreed pay amount. Then the computation is outsourced to a cloud. The script can be redeemed either by providing the correct output of the outsourced computation or by providing some pre-shared secrets. Zhang *et al.* have presented two works, namely BPay [126] and BCPay [127], for outsourcing computations using Bitcoin scripts. Similar to [125], the authors constructed a robust, fair payment model where a cloud provider receives the payment for computation if and if it produces a valid proof of correctness of the computation. A scalable solution using smart contracts is presented by Eberhardt *et al.* [128]. The user and the cloud provider runs a one-time setup process after which the user generates a verifying contract and deploys it on the Ethereum network. The cloud computes the outsourced computation and sends the proof of correctness consisting of witness to verifying contract. Dorsala *et al.* [129] constructed a fair payment model using [128] as a verification contract and shows that the cost of running a fair verifiable method on top of Ethereum is negligible when both user and cloud provider are honest. Guan *et al.* [130] have also realized fairness in outsourcing polynomial computation using smart contracts.

Dong *et al.* [15] have outsourced the same computation to two clouds and created a prisoner's dilemma between them to avoid collusion. They have constructed three contracts to extract the correct result from two rational clouds: (1) Prisoner's contract, which rewards honest cloud and punishes malicious cloud. (2) However, the clouds can collude and solve the prisoner's dilemma using colluder's contract. (3) Traitor's contract provides additional bounty to the honest cloud to counter collusion. The work in [15] assumes the user as trusted and requires a trusted third party to resolve disputes when the outputs of the clouds do not match. The work by Avizheh *et al.* [131] assumes a rational user and uses a verification game to resolve disputes between the two clouds. The clouds have to construct a Merkle tree with the intermediate states of the computation and have to return the Merkle root hash along with the result of the computation. Then, the smart contract randomly generates two different indexes, one for each cloud and asks the clouds to submit Merkle proof. The smart contract identifies the malicious cloud by constructing the Merkle root from the Merkle proofs submitted by clouds and matches it against the Merkle root submitted earlier. However, [131] does not discuss the collusion attack where the clouds collude to provide the same incorrect result, thereby increasing their payoff.

In Dorsala *et al.* [129], the clouds have to submit an inner sate hash (ISH) [155] along with the result of the computation. A lazy cloud cannot submit a correct ISH without actually computing the result. Even the copy attack cannot be performed because the clouds have to commit a hash of the random number while showing intent and this random has to be concatenated to ISH while submitting the result. In case of disputes, a verifying contract is executed, and its cost is collected from the malicious cloud. However, running a verifying contract is costly, and due to the verifier's dilemma [97], the costly verifying contracts can jeopardize the Ethereum network. The major drawback of RBVC methods is that the user has to pay the cost of computation to every cloud provider, which is a major burden to the user.

In TrueBit [132], the computation is outsourced to a single cloud who submits the result to a smart contract. Then challengers are invited to challenge the correctness of the result. If there is a challenge, then a verification game is initiated through a series of rounds, where each round recursively checks a smaller and smaller subset of the computation. The challengers are rewarded for finding the errors and penalized for wrong alarms. To encourage the challengers to take part in the system, the system occasionally forces the honest cloud to submit wrong results (forced errors) and offers a big bounty to verifiers for finding the errors. Harz *et al.* [133] have presented a method similar to TrueBit except that a cloud and challengers are randomly assigned, and all will compute the computation and report the result to a smart contract, and any disputes in the results are resolved by a dispute resolution protocol similar to TrueBit. In [133], the probability of finding false computation depends on the number of challengers recruited for a computation. Król *et al.* [134] have presented a method using a trusted execution environment (TEE) [95]. Challengers are not required in their method because the honest behaviour of the cloud is enforced by executing the outsourced computation in a TEE. The shortcomings like collusion and sybil attacks of [133] are discussed by Nabi *et al.* [135]. The authors introduce a random audit of results returned by clouds and penalize malicious clouds by sharing their pre-committed deposit as a reward to diligent clouds. Although [135] is better method than [133], it cannot eliminate the collusion attacks completely. More recently, Eisele *et al.* [136] have proposed use of trusted mediators for dispute resolution. Comparison of verifiable computation schemes is presented in Table 2.6.

Three practical systems, namely Golem [156], iExec [157], and SONM [158] all built on Ethereum for outsourcing computation to large computational systems. Golem uses RBVC and log analysis for checking correctness whereas iExec employs an Intel SGX for correct computation. SONM currently supports verification only by users, and no verification method is adopted. One may refer to [159] for a detailed analysis of these three systems.

S. No	**Paper**	**PB / RB / CB**	**Platorm**	**Verifier**	**Dispute resolution**	**Penalities**	**Fair Payment**
1	Kumaresan *et al.* [125]	PB	Bitcoin	Bitcoin scripts	Not required	yes	yes
2	Dong *et al.* [15]	RB	Ethereum	Third-party	TTP	yes	yes
3	Teutsch *et al.* [132]	CB	Ethereum	Verifiers	Verification game	yes	yes
4	Harz *et al.* [133]	CB	Ethereum	Verifiers	TTP	no	yes
5	Eberhardt *et al.* [128]	PB	Ethereum	Smart contract	Not required	no	yes
6	Krol *et al.* [134]	CB	Ethereum	Trusted Execution Environments	Not discussed	yes	yes
7	Zhang *et al.* [126]	CB	Bitcoin	Bitcoin scripts	Verification game	yes	yes
8	Zhang *et al.* [127]	CB	Bitcoin	Bitcoin scripts	Verification game	yes	yes
9	Avizheh *et al.* [131]	RB	Ethereum	Smart contract	Verification game using merkle tree	yes	yes
10	Nabi *et al.* [135]	RB	Ethereum	Verifiers	TTP	yes	yes
11	Dorsala *et al.* [129]	PB RB	Ethereum	Smart contracts	Verification contracts	yes	yes
12	Eisele *et al.* [136]	CB	Ethereum	Third-party	TTP	yes	yes
13	Guan *et al.* [130]	PB	Ethereum	Smart contracts	Not required	no	yes

Table 2.6: Comparison of Blockchain-based verifiable computing methods

2.3.2.1.4 Summary

In this section, we have presented works discussing Blockchain-based computation-as-a-service. In traditional cloud computing, the user has to trust the cloud for correct computation of outsourced computation. However, with the Blockchain-based CaaS, the user is no longer required to trust the cloud or depend on trusted third parties for the correctness of computation. The other benefit for the user is that he can pay if and only if the cloud computes correctly. Likewise, the cloud also gets its payment as long as it performs the computation correctly.

2.3.2.2 Data aggregation-as-a-service

In this section, we review Blockchain-based crowdsensing methods as crowdsensing is the most widely explored data aggregation service.

2.3.2.2.1 Issues in crowdsensing

The rapid increase in the number of mobile devices and wearable devices, equipped with multiple sensors (e.g., gyroscope, accelerometer, microphone etc.) led to the emergence of new sensing paradigm known as Mobile Crowdsensing (MCS). In MCS, a cloud provider CP posts a sensing task and a data provider DP responds to the task by sending the data. Although crowdsensing shows prominence in many applications such as transportation [160, 161], healthcare [162], and environment monitoring [163, 164], it suffers from the following problems:

(a) *Privacy*: Sensing data reveals users personal information, and hence privacy is the utmost priority in MCS.

(b) *Rewards*: The rewards to a DP depends on factors like sensitivity of data, sensing time, data quality, etc., The CP cannot be trusted to compute rewards correctly.

(c) *Fairness*: The CP may not pay to DP after receiving the data or a DP may not provide data after receiving pay.

2.3.2.2.2 Blockchain-based crowdsensing solutions

Li *et al.* [143] have presented a Blockchain-based framework known as CrowdBC in which a set of smart contracts are employed to perform sensing task posting, data receiving, reputation management and reward assignment operations without a centralized entity. In CrowdBC, CP evaluates the quality of the data submitted by DPs, and the evaluated results are given to smart contract based on which the rewards and reputation values are calculated. To achieve fairness and preserve privacy, Wang *et al.* [137] have presented a k-anonymity privacy protection method where DPs form as a group of k members and submit their data as group data for quality-evaluation. Miners evaluate the data and pay the group according to the quality of the data. However, the members in a group trust each other and the group leader is expected to distribute the rewards correctly to group members.

Another work aiming to monetize sensing data fairly without compromising privacy is discussed by Cai *et al.* [138]. They employ two servers as mediators between CP and DPs. The two servers execute a protocol based on secret sharing and garbled circuits to establish ground truth on encrypted data. Then, each server encrypts the learned ground truth and commits it along with a masked encryption key ($key = XOR(k, mask)$, k is actual encryption key) on a smart contract. CP interested in buying the providers data sends two keys encrypted with the corresponding server's public key along with payment to the smart contract. If the cloud server accepts the CP's offer, it will encrypt the $mask$ with the key sent by the CP ($sn = SENC(mask, s)$, s is the key sent by the CP). In order to avail the pay sent by the CP, the cloud server must reveal the masked key key and sn. Then, smart contract verifies the commitments and sends rewards to the server, which will be shared to providers according to the quality of their data with respect to the estimated ground truth. The method is fair in between servers and CP, but the DP still needs to trust the servers for correct computation of ground truth.

In ZebraLancer [139], the DPs encrypt their data with CP's public key and send it to the smart contract. CP decrypts the data off-chain and computes the rewards to DPs based on the quality of the data provided. Then, the CP computes a proof of correctness using zk-SNARK about the computation of rewards and sends this proof to the smart contract. The smart contract consists of zk-SNARK verification algorithm and returns true if and only if the CP computed the rewards correctly. If the CP fails to produce correct proof, the reward deposited by him is distributed to providers. ZebraLancer provides both privacy and fairness, but zk-SNARK requires a setup phase where either a trusted entity or all the participating entities jointly should establish a common reference string. Also, the verification time increases with an increase in the number of DPs limiting the applicability of the system. Shi *et al.* [140] have developed a decentralized application (Dapp) known as MPC-SToken, where a fault-tolerant incentivization mechanism is modeled as a functionality of a smart contract to facilitate payments between CP and DP. Chatzopoulos *et al.* [141] have employed a trusted party (Internet service providers) to protect the location privacy of the DPs by masking their real-identities with pseudonyms. However, employing a trusted, centralized party opposes the cause of using Blockchain.

To protect location privacy, a hybrid Blockchain model is discussed by Yang *et al.* [142]. The authors have proposed to use a public Blockchain to publish task and assume a trusted agent who retrieves the tasks from the public chain and publishes them to a private Blockchain. DPs are allowed to transact with multiple private chains to thwart re-identifications attack caused due to DPs transaction history. The trusted agent submits the sensory data to the public Blockchain on behalf of the DPs, and the payments are transferred according to the employed quality estimation method. MCS-Chain [144] is similar to [143], except that a new light-weight consensus mechanism suitable for mobile crowdsourcing is designed. Unlike in PoW, the generation of new blocks in the presented consensus mechanism is determined by the total amount of payment records waiting to be stored in the next block. The mechanism is guaranteed to eliminate forks by enforcing rules based on time, reputation and awards. Miners compute the reputation values by running a trust evaluation algorithm based on feedback received from all the participating entities.

Zhang *et al.* [145] have put forwarded a Blockchain-based MCS model without the use of smart contracts. Initially, CP commits an incentive policy on a Blockchain and announces the task. Then the DPs encrypt their sensing data with a homomorphic encryption key generated by the CP and send the commitment of the ciphertext to the Blockchain. Later, the providers and CP reveal the committed data and the incentive policy, respectively, by sending a transaction to Blockchain. Then, the CP decrypts the data, evaluates it and computes the rewards accordingly, which will be transferred to DPs through Blockchain. However, the model lacks fairness since the CP receives the data before actually paying DPs.

CrowdR-FBC [146] proposed to use fog nodes as an intermediary between CPs and DPs. The fog nodes are responsible for maintaining the Blockchain network. A CP announces the task on an MCS platform which then encrypts the task and sends it to the fog nodes. The fog nodes select the providers based on the reputation maintained in the Blockchain and send the encrypted task to provider who then decrypts, computes the task and sends the encrypted result back to fog nodes. Then the fog nodes forward the encrypted results to MCS platform along with pseudo-ID of the DPs. Then the MCS platform decrypts the result, computes the new reputation values and sends them to fog nodes. The fog node maps the reputation values to real-world identities and updates them on the Blockchain network. This approach solves the privacy problem as the MCS platform does not know the real identities of DPs, and the fog nodes do not know the task or the data. However, the work does not discuss the collusion between crowdsourcing platform and the fog nodes which will seriously affect the privacy of the providers, and also did not discuss the incentive policy for different participating entities. Wei *et al.* [165] have presented the use of consortium Blockchain and constructed an incentive method based on reputation, quality of data and provider's valuation. Similar to [145], the method also lacks fairness.

Hu *et al.* [147] have transcoded a three-stage Stackelberg [166] game as a smart contract which pays the DPs according to their category (instant and monthly). A fair payment model similar to [143] is constructed using smart contracts in SenseChain [148] except that the platform is constructed for multiple CPs and multiple DPs and reputation is maintained for both CPs and DPs. A hybrid Blockchain platform for MCS is presented by Zhu *et al.* [149]. A public chain running DPOS consensus and many private subchains running PBFT consensus are used to publish and record the information of public and private crowdsourcing tasks, respectively. After the private task is completed, the leader of the subchain generates a zk-SNARK proof about the all the subchain blocks of a task and sends it to the public chain where a zk-SNARK functionality verifies the proof and records it into Blockchain ledger. CrowdBLPS [150] use Blockchain to preserve location-privacy of the DPs, and their main emphasis is on modeling an optimized DPs selection as a smart contract. Huang *et al.* [151] have introduced Blockchain-based MCS into smart factories where DPs record the noise of the machines through their mobile and submit the data on the Blockchain expecting a reward. They have designed a new reward mechanism, namely dynamic reward ranking as a smart contract. Comparison of Blockchain-based MCS methods is given in Table 2.7.

S. No.	Paper	Blockchain platform	Registration	Provider selection criteria	Quality evaluation	Reward distribution	Reputation Feedback		Worker Privacy	Data Privacy	Fair Payments
							R	W			
1	CrowdBC [143]	Ethereum	Yes	Reputation	SC (Miners)	Equal	Yes	Yes	Psudoanonymity	Encryption	No
2	Wang *et al.* [137]	Bitcoin	No	Open	Miners	Quality-aware	No	No	k-anonymity		No
3	Cai *et al.* [138]	Ethereum	No	Open	SC (Miners)	Quality-aware	No	No	Psudoanonymity	Encryption	Yes
4	ZebraLancer [139]	Ethereum	Yes	Open	Requestor	Quality-aware	No	No	ZKSNARKs		Yes
5	MPCSToken [140]	Ethereum	No	Auction	No	Fault Tolerance Incentivisation Mechanism (FTIM)	No	No	No	No	No
6	Chatzopoulos *et al.* [141]	Ethereum	Yes	Auction	Third-party	Equal	No	Yes	Pseudoanonymity with TTP	No	Yes
7	Yang *et al.* [142]	-	No (public BC) Yes (private BC)	Open	SC (Miners)	Equal	No	No	Private Blockchain run by third-party	No	No
8	MCS-Chain [144]	MCS-Chain	Yes	Bargain and Reputation	Requestor	Equal	Yes	Yes	Pseudoanonymity	Encryption	No
9	Zhang *et al.* [145]	Transaction -based	No	Open	Requestor	Equal	No	No	Pseudoanonymity	Homomorphic encryption	No
10	CrowdR-FBC [146]	CrowdR-FBC	Yes	Reputation	Third-party	No	No	Yes	Pseudoanonymity	Encryption	No
11	Wei *et al.* [165]	Consortium	Yes	Auction	Requestor	Quality-aware, Bid-based and Reputation-based	No	Yes	Consortium Blockchain run by trusted agents	Encryption	No
12	Hu *et al.* [147]	Ethereum	Yes	Reputation	SC (Miners)	Quality-aware and Reputation-based	No	Yes	Pseudoanonymity	No	Yes
13	SenseChain [148]	Ethereum	Yes	Reservation	SC (Miners)	Quality-aware	Yes	Yes	Pseudoanonymity	No	Yes
14	zkCrowd [149]	Hybrid (DPOS, PBFT)	Yes	Capacity	No	Equal	No	No	ZK-SNARK (private tasks)		No
15	CrowdBLPS [150]	Ethereum	Yes	Location	Requestor	Quality-aware	No	No	Pseudoanonymity	Encryption	No
16	Huang *et al.* [151]	Ethereum	No	Open	SC (Miners)	Dynamic reward ranking	No	No	Pseudoanonymity	No	Yes

Table 2.7: Comparison of Blockchain-based crowdsensing systems

2.3.2.2.3 Summary

In this section, we have discussed several Blockchain-based data aggregation-as-a-service works which focus on preserving the privacy of the DP, computing correct rewards for DPs and realizing financial fairness. Some of the works use the pseudo-anonymity feature of the Blockchain networks to preserve the privacy of the DP. However, pseudo-anonymity is not sufficient to preserve the privacy of the DP [167]. In some works, the reward distribution depends on the quality of the data unearthed by executing a truth discovery algorithm (TDA). However, the TDA algorithm is either executed by a cloud or modeled as a smart contract. To have greater confidence in executions by cloud, it has to be verified by using methods discussed in section 2.3.2.1.3. Also, modeling complex TDA algorithms as smart contracts may lead to verifiers dilemma [97].

2.3.3 Blockchain-based Software-as-a-Service

Software-as-a-service allows users to access the software through the Internet on a subscription basis. However, most of the SaaS applications are migrating from a monolithic architecture to microservice architectures. Therefore, in this survey, we have considered two emerging SaaS models: (1) Microservices and (2) Virtual network functions. The list of Blockchain-based software-as-a-service works is presented in Figure 2.6.

2.3.3.1 Microservice-as-a-service

2.3.3.1.1 Challenges in Microservice-as-a-service

Containerization technology decomposes the traditional monolithic applications into a suite of small services known as microservices each running in its own process and communicating through lightweight mechanisms. However, existing microservices architecture suffers from the following problems:

1. Microservices are advertised on a centralized platform which may be untrusted, and hence a trusted registry and service discovery is required for the advertisement and discovery of microservices.

2. Microservices lacks trusted communication platform for exchanging messages. They

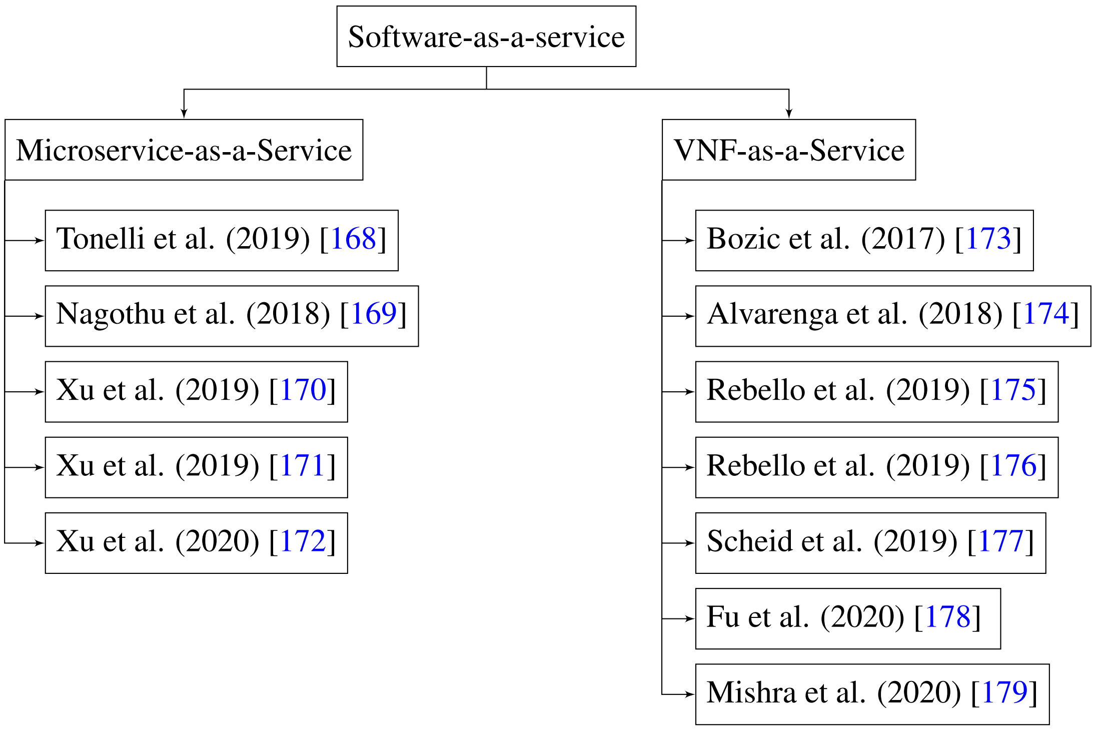
Software-as-a-service
Microservice-as-a-Service
Tonelli et al. (2019) [168]
Nagothu et al. (2018) [169]
Xu et al. (2019) [170]
Xu et al. (2019) [171]
Xu et al. (2020) [172]
VNF-as-a-Service
Bozic et al. (2017) [173]
Alvarenga et al. (2018) [174]
Rebello et al. (2019) [175]
Rebello et al. (2019) [176]
Scheid et al. (2019) [177]
Fu et al. (2020) [178]
Mishra et al. (2020) [179]

Figure 2.6: List of works in Blockchain-based software-as-a-service.

either depend on secret channels or public-key cryptography for secure and authenticated message exchanges.

2.3.3.1.2 Blockchain-based Microservices

Tonelli *et al.* [168] have mapped the existing microservices architecture to smart contracts architecture. They have also demonstrated a case study by modeling the existing microservices-based system as a set of smart contracts and tested them on Ethereum Blockchain. Nagothu *et al.* [169] have adopted Blockchain for a smart surveillance system constructed as a set of microservices. They have employed Blockchain for securing messages exchanged between microservices and also to provide integrity to the microservice database by periodically storing the hash of the database in the Blockchain. Xu *et al.* [170] have constructed BlendMAS; a Blockchain-based decentralized microservices architecture for smart public safety. They have divided the permissioned Blockchain into two sets of microservices: mining services and security policy services. Mining services are responsible for running the consensus algorithms, and security services are responsible for identity management, access controls etc. These decentralized security microservices work as a service cluster to offer a scalable, flexible and lightweight data sharing and access control mechanism. Similar approach is followed in [171] for constructing decentralized marketplace and in [172] for constructing decentralized multi-domain avionics systems.

2.3.3.1.3 Summary

Monolithic cloud applications are migrating towards microservices and incorporating Blockchain into existing microservice architecture which enables trust and transparency. Specifically, a public immutable microservice registry can be constructed using Blockchain. Also, Blockchain, with its peer-to-peer messaging architecture, can act as a trusted overlay for exchanging microservices messages.

2.3.3.2 Virtual network function-as-a-service

Cloud computing infrastructures contain a number of servers with storage and compute capabilities. With the advancement of technology and increase in demand of cloud servers, the existing cloud infrastructure is being enhanced to support European Telecommunications Standards Institute (ETSI) Network Function Virtualization (NFV) architecture [180]. The core objective of NFV is to decouple the physical network infrastructure from the service functions that run on them. NFV mainly consists of three components: NFV Infrastructure (NFVI), Virtual Network Functions (VNF) and Management and Network Orchestration (MANO). In NFV, a cloud service is decomposed into a set of VNFs, which could be then deployed as a software running on one or more physical servers [181]. VNFs can be easily managed as they can be relocated and instantiated at different network locations on-demand.

2.3.3.2.1 Issues in virtual network function-as-a-service

In NFV, the VNF orchestrator (VNFO) receives the request from the user for setting up a new VNF or scaling the already running VNF. Then, the VNFO relays the request to a virtual machine manager (VMM). However, the communications between the VNFO and VMM occurs through the unauthenticated channel leading to a variety of attacks [182]. Another issue in VNF is that multiple cloud providers may cooperatively deploy NFVs leading to data leakage problems.

2.3.3.2.2 Blockchain-based virtual network function-as-a-service

The communication between VNFO and VMM is generally secured by public-key cryptography systems that depend on trusted third parties. The communication involves orchestration requests regarding how to create, modify, destroy or migrate the VMs. A compromised communication channel may cause malicious creation, modification, destruction, or migration of VMs. Hence, a secure, reliable and tamper-resistant interface is required to relay requests from orchestrator to VMM.

Bozic *et al.* [173] have built a Blockchain-based Virtual Machine Orchestration Authentication (VMOA) system to relay messages from orchestrator to VMM securely. In VMOA, the orchestrator before sending a request to VMM, sends the request as a transaction to a Blockchain network which is then stored in a secure and immutable ledger. When VMM receives the request, it uses the information stored in the ledger for authenticating the orchestrator and allocates the VMs only on successful authentication. The authors propose to build VMOA using Hyperledger Fabric. Alvarenga *et al.* [174] have proposed secure configuration management of VNFs by logging the signed VNF configuration and management information on the Blockchain. They employed a BFT consensus algorithm [183] and designed two types of transactions: a configuration transaction to install configurations on VNFs and a configuration request transaction to request the configuration state of a particular VNF. These transactions are appended to the Blockchain aiding traceability and accountability of VNF configuration updates. Besides, they have also presented a secure VNF migration through transactions on the Blockchain.

Another problem in NFV is to identify a malicious or faulty VNF configuration because the presence of a single malicious VNF configuration can affect the entire service function chain. Further, if the orchestrator is compromised, it may overwrite the log to hide its malicious activity / threats. In order to mitigate the activities of malicious VNFO, Rebello *et al.* [175, 176] have presented to connect every VNF in a network with Blockchain to log all operations during execution of a service chain. The recording of operations in Blockchain provides tamper-proof auditing of orchestration operations which helps in identifying threats caused due to malicious VNF. In [175], the authors built BSec-NFVO for providing non-repudiation, auditability and integrity of orchestration operations in multi-tenant NFV environment. In [176], the authors have demonstrated the applicability of their proposal by adopting network slices as a use case where all the VNFs in a particular network slice are connected to a particular Blockchain. They have exploited the Hyperledger Fabric channels to create different network slices running their own Blockchains in isolation.

Fu *et al.* [178] have put forwarded an idea of adopting Blockchain for synchronizing messages between distributed NFV-MANO (Management and Network Orchestration) systems. In their work, the NFV-enabled edge servers perform Blockchain computations along with assigned tasks. Each NFV-MANO system collects their local messages, divides them into transactions and broadcasts them. A Blockchain node tracks all the transactions from the last synchronized state of the MANO systems, and a consensus process will start after receiving the synchronization request from MANO systems. They have used BFT protocol [183] for consensus in which one of the NFV-enabled nodes is designated as Blockchain primary node and others as replica nodes. The primary node broadcasts the pre-prepared messages consisting of transactions collected from the last synchronized state of the MANO systems. Next, the replica nodes send a prepare message to all the Blockchain nodes. Then, all the nodes send a commit message to all other nodes. Finally, the blockchain nodes reply to all NFV-MANO systems with the new validated messages. Thus, all the NFV-MANO systems are message synchronized.

Another critical factor in NFV is the trust in VNF orchestrator as it oversees the end-to-end VNF lifecycle management. Mishra *et al.* [179] have developed a series of smart contracts to increase transparency in the operations of VNF orchestrator for end-to-end VNF lifecycle management. The smart contracts act as a market place for third-party VNF developers to advertise their VNF packages and acts as a trusted platform for edge clouds to buy the advertised packages. The VNF orchestrator will validate the package sent by developers and requests sent by edge clouds. Upon successful validation, the orchestrator selects the best suitable VNF from the available VNF pool and migrates that VNF from cloud to the requested edge cloud. The system also contains a reputation / feedback contract to eliminate malicious VNF packages.

Although the above-discussed works solve some of the problems in NFV through Blockchain, the limitation from the perspective of end-users is how to ensure that the VNF package acquired by the end-user is not malicious and not tampered. Currently, the end-users rely on a trusted centralized database to trust that the VNF packages are not malicious and not tampered. However, the centralized database may be compromised or becomes a single point of failure. Scheid *et al.* [177] have presented a VNF package repository called BUNKER by replacing the centralized database with a smart contract. In BUNKER, a package creator registers its new VNF packages with a smart contract by sending the hash value of a VNF package. Users interested in a VNF package must obtain a license from the smart contract by transferring the necessary payment. After obtaining the license, the users can retrieve the VNF source code from the external VNF storage and can verify the integrity of the obtained package by querying the smart contract. BUNKER also has a smart contract-based reputation / feedback mechanism to avoid users submitting malicious VNF packages.

2.3.3.2.3 Summary

NFV offers end-to-end services by chaining VNFs between competing cloud infrastructures in a trustless environment. In this section, we have discussed the limitations of existing VNF architecture and their solutions with Blockchain technology.

2.4 Observations and Problems Identified

In this survey, we observe that many works attempt to improve the existing cloud architecture with the help of novel features exhibited by Blockchain technology. We notice that the centralization of cloud cannot be entirely abolished, preferably the degree of centralization can be decreased through the adoption of Blockchain. The decrease in the centralization gives cloud users more control and increase the trust and transparency in cloud computing. In this section, we list some of the open issues and future directions of the emerging Blockchain-enabled cloud computing field.

1. We observe that most of the Blockchain-enabled IaaS research is focused on storage-

as-a-service. However, there is a lack of focus on other IaaS services such as network firewalls, security, and load balancing.

2. In general, the resources are acquired by users in pay-per-use / pay-as-you-go model. These models benefit the IaaS provider as the user pays in advance before using the services. However, with the advent of Blockchain, fair payments models are being constructed where the users can pay only for the resources they obtained. Fair payments models using Blockchain in IaaS are largely unexplored, and we expect in future a great potential of useful work on constructing fair payment models.

3. In recent years, different cloud providers are mutually providing resources giving rise to a federated paradigm. Blockchain with its properties can benefit federated cloud computing to inculcate trust among untrusted cloud providers.

4. We have observed that the existing Blockchain-based resource allocation schemes mostly follow commit and reveal methods during auctions. However, there is a gap in designing online resource allocation schemes using Blockchain.

5. Although, many works are proposed in Blockchain-based verifiable computations they either suffer from practicality or huge cost and hence efficient schemes have to be designed.

6. We have observed that the one of the least explored but most important aspect of cloud computing is rating, charging and billing of cloud services. Blockchain, with its novel properties, can ensure transparency and trust in the billing of cloud services.

2.5 Summary

In this Chapter, we have surveyed existing Blockchain-based cloud computing models. Our comprehensive survey maps the existing Blockchain-based cloud services to the most primitive cloud service models IaaS, PaaS and SaaS. To be specific, we have explored Blockchain-based storage-as-a-service, resource allocation, computation-as-a-service, data aggregation-as-a-service, microservice-as-a-service and VNF-as-a-service. We have also listed open issues and future directions which will motivate the interested researchers and practitioners to put focused research efforts into this promising area.

Chapter 3

Fair Payment Protocols for Outsourcing Computation under Platform-as-a-Service

With the advent of cloud computing, outsourcing a computation has become a common practice. A cloud user to have greater confidence in computations performed by the cloud / cloud provider should verify the correctness of the results returned. Two approaches are followed to verify the correctness of results: (1) Proof-based verifiable computing and (2) Replication-based verifiable computing. The first approach uses cryptography techniques, whereas the second approach follows game-theoretic methods. Although both approaches are almost perfect solutions to verify the results, they do not discuss fairness in verifiable computing.

In this Chapter, we consider fairness in verifiable computing, which means that the provider receives the user's payment for an outsourced computation if and only if the user receives the correct output of the computation. The contributions of this Chapter are as follows:

(a) We have designed a new fair incentivized model for proof-based verifiable computation. We show that the cost of running a smart contract is negligible when both user and provider are honest.

(b) We have designed a new fair incentivized model for replication-based verifiable computation for a two-provider case and a multi-provider case. We obtain honest computation from the provider by imposing monetized penalties.

(c) We show that smart contracts are an efficient way to send the reward to honest providers and penalize malicious providers. Using smart contracts, we are emulating trusted entities like banks for payments between user and provider.

(d) We have listed the financial and transactional cost of the designed smart contracts by implementing them in Solidity [24] using Truffle framework [25].

3.1 Verifiable Computation

A resource constraint cloud user (CU) outsources a computation to a cloud provider (CP), who gets paid in return to deliver the correct output of the computation. The output returned by the provider is verified by the user or by a third-party. The work performed to verify the output must be lesser than the work required to compute the output. The user accepts the output of the computation if and only if its correctness is verified.

Definition 3.1.1. *A proof-based verifiable computation consists of a set of three algorithms [184]:*

(a) $Keygen(F, 1^\lambda) \rightarrow (ek_F, vk_F)$: A randomized key generation algorithm takes the function F to be outsourced, and a security parameter λ; It outputs a public evaluation key ek_F and a public verification key vk_F.

(b) $Compute(ek_F, x) \rightarrow (y, \pi_y)$: This is a deterministic algorithm that takes ek_F and x as input. x is input of the function F. It outputs $F(x) \rightarrow y$ and a proof π_y of y's correctness.

(c) $Verify_{vk_F}(x, (y, \pi_y)) \rightarrow \{0, 1\}$: Given the verification key vk_F, x, y and π_y, the deterministic verification algorithm outputs 1 if $F(x) = y$ and 0 otherwise.

Definition 3.1.2. *A replication-based verifiable computation consists of a set of three algorithms:*

(a) $Outsource(F)$*: CU outsources F on input x. Let $CP_1, CP_2, ..., CP_n$ be the set of cloud providers who have shown intent to compute $F(x)$.*

(b) $Compute(F, x) \rightarrow y_i$*: Every $CP_i \in \{CP_1, CP_2, ..., CP_n\}$ computes $F(x)$ and outputs $y_i = F(x)$.*

(c) $Verify(y_1, ..., y_n) \rightarrow \{0, 1\}$*: Given all the providers' outputs, the deterministic algorithm outputs 1 if outputs of all the providers are equal and 0 otherwise.*

Definition 3.1.3. *A fair verifiable computation between two parties CU and CP must provide the following guarantee:*

(a) ***Fast verification****: The work performed to verify the correctness of output of F is less than the work performed to compute F.*

(b) ***Fair payments****: CP obtains pay from CU if and only if CU receives the correct output of the computation from CP.*

3.2 Proof-based Incentivized Outsourced Computation (IOC) using Smart Contracts (PBIOC)

In this section, we discuss fair incentivization of proof-based verifiable computation. As discussed earlier, a public verifiable computation scheme consists of two parties, a cloud user CU and a cloud provider CP. CU runs $Keygen$ algorithm (see Definition 3.1.1), and CP runs $Compute$ algorithm. The $Verify$ algorithm is executed by a CU or by a trusted third party TP. There are three possible approaches to incentivize verifiable computation.

Case 1: A contract is signed between CU and CP, such that CU runs $Verify$ algorithm and pays CP for using its services, if and only if $Verify$ algorithm returns 1. An honest CP receives pay only if CU is honest. In this case, CP has to put trust in CU for honest payment.

Case 2: CU subscribes to CP's service by transferring some pay to it and asking it to run $Compute$ algorithm. Here a legal contract may be made between these two parties containing all necessary clauses. CP may or may not adhere to the legal contract. If CP does not adhere to a legal contract and sends an incorrect output to CU, CU's only way to get back his pay is going through the cumbersome legal process. In this case, CU has to put trust in CP for honest computation.

Case 3: Let CU and CP, recruit a third party TP similar to Model 2 in 1.1.2.2. CU sends pay to TP and asks the CP to run $Compute$ algorithm. CP sends the output to TP. Now, TP runs $Verify$ algorithm; if it returns 1, TP sends pay to CP; otherwise, it will refund CU's payment. Here CU trusts TP for honest verification and CP trust TP for honest payment. A special scenario where the dis-honest CP can also be penalized by asking CP to deposit some pay with TP, before claiming the pay for computation.

In case 3, both CU and CP put their trust in third party. However, use of TP services comes with a cost, and TP may not guarantee to behave honestly every time. As the public Blockchains are trusted for correctness and availability, they can emulate the trusted third party functionality. The Blockchains also offer programmability to create and run small programs known as smart contracts. Now, we show our proof-based verifiable computation using smart contracts.

3.2.1 $PBIOC$ contract clauses

$PBIOC$ is an outsourcing contract between CU and CP. The high-level idea is if CU and CP behave honestly, then CU will get the output $F(x)$, and CP will get the pay for computing $F(x)$. Otherwise, a verifying contract $PBIOCV$ is invoked, and payment is made according to the $PBIOCV$ contract's result.

The clauses in the $PBIOC$ contract are as follows:

(i) CU prepares two contracts $PBIOC$ and $PBIOCV$ where the $Verify$ algorithm from Definition 3.1.1 is modeled as $PBIOCV$ contract and is executed only in case of disputes.

(ii) CU chooses a function $F(\cdot)$, and an input x. CP agrees to compute $F(x)$.

(iii) CU agrees to pay $\$r$ to CP for the correct computation of $F(x)$. CU deposits a reward $\$r$ and a safety deposit of $\$c$ with the $PBIOC$ contract. CU also sends (ek_F, vk_F, x) to CP in off-chain mode.

(iv) CU and CP agree on timing parameters $\tau < \tau_i < \tau_c < \tau_a < \tau_{end}$, where τ is the current time.

(v) CP must send a deposit $\$d$ to $PBIOC$ before τ_i. If CP fails to deposit $\$d$ before τ_i, then the contract is terminated, and CU's deposit ($\$r$ + $\$c$) is refunded.

(vi) CP computes $F(x)$ and sends commitment of the output to the smart contract before τ_c. CP sends (y, π_y) to CU in off-chain mode. If CP fails to deliver the commitment of the output before τ_c, then the deposit $\$d$ made by him and the deposit ($\$r$ + $\$c$) is sent to CU, and the contract is terminated.

(vii) If CU agrees to CP's output before τ_a, then ($\$d + \$r$) is sent to CP, and $\$c$ is sent to CU, and the contract is terminated.

(viii) If $\tau > \tau_a$ and the contract is not terminated, then the $PBIOCV$ contract is invoked.

(ix) If $PBIOCV$ returns CP as honest before $\tau < \tau_{end}$, then ($\$r + \$c + \$d$) is sent to CP. If $PBIOCV$ returns CP as lazy before $\tau < \tau_{end}$, then ($\$r + \$c + \$d$) is sent to CU, and the contract is terminated.

(x) If $\tau > \tau_{end}$ and the contract is not terminated, then ($\$r + \$c + \$d$) is sent to CP, and contract is terminated.

3.2.2 PBIOC Protocol

PBIOC protocol is presented in Figure 3.1. $PBIOC$ smart contract functionalities executed by Blockchain are presented as Algorithms 3.1 to 3.6. CU chooses a function F, an input x and generates an evaluation key ek_F and a verification key vk_F. CU also chooses timing parameters $\tau_i, \tau_c, \tau_a, \tau_{end}$. These timing parameters are required to enforce timely computation and avoid locking the funds if one party refuses to move forward in the protocol. The contract can always query the underlying blockchain for the current time[1]. CU sends the chosen parameters to CP. He also sends the parameters along with $\$r$ and $\$c$ to

[1]Most smart contracts use block number / block timestamp as a timer

BC invoking Algorithm 3.1. $\$r$ is the reward to be given to CP and $\$c$ is the deposit to avoid CU's malicious behavior. Algorithm 3.1 stores all the parameters and changes the state to *Created*. If CP is willing to compute the outsourced task, he will send a deposit $\$d$ to BC invoking Algorithm 3.2. Algorithm 3.2 stores the deposit and changes the contract state to *Intent*. CP sends the commitment of $F(x)$ result to BC invoking Algorithm 3.3. Algorithm 3.3 stores the parameters sent by CP and changes the state to *Committed* CP also sends the result to CU. Depending on the behavior of CU there are three cases as follows:

Case 1: CU verifies the result locally, and if the result is correct, he sends a transaction invoking Algorithm 3.4. In this case, Algorithm 3.4 sends $\$r$ to CP along with his deposit and also sends CU's deposit to CU and the contract is terminated.

Case 2: CU verifies the result locally and if the result is correct and he does not send any transaction to BC. In this case, CP reveals the output parameters by invoking Algorithm 3.5, which internally invokes another contract $PBIOCV$. The $PBIOCV$ verifies the output and sends the result to Algorithm 3.6. If CP has performed honest computation he receives ($\$r + \$d + \$c$); otherwise, CU will receive ($\$r + \$d + \$c$).

Case 3: If neither CU invokes Algorithm 3.4 nor CP invokes Algorithm 3.5 before respective timeouts, then CU receives ($\$r + \$d + \$c$). If $PBIOCV$ fails to send result before timeout then CP receives ($\$r + \$d + \$c$).

PBIOC protocol

For cloud user CU

1. To create a outsourcing task
   (a) Run $Keygen(F, 1^{\lambda}) \rightarrow (ek_F, vk_F)$.
   (b) Compute $E \leftarrow H(ek_F)$, $V \leftarrow H(vk_F)$, $X \leftarrow H(x)$.
   (c) Send $\textbf{trans}^{CU}_{create}$ =(E, V, X, τ_i, τ_c, τ_a, τ_{end}, $\$r$, $\$c$) to BC.
   (d) Send (ek_F, vk_F, x) to CP.
2. On receiving (”verify”,y, π_y) from CP.
   (a) Run $Verify_{vk_F}(x, (y, \pi_y))$.
   (b) If $Verify_{vk_F}(x, (y, \pi_y)) \rightarrow 1$, then send $\textbf{trans}^{CU}_{agree}$ to BC.

For cloud provider CP

3. Verify $PBIOC$ contract and task details.
4. To participate in the outsourcing task, send $\textbf{trans}^{CP}_{intent} = (\$d)$ to BC.
5. To send the output
   (a) Run $Compute(ek_F, x) \rightarrow (y, \pi_y)$.
   (b) Compute $Y \leftarrow H(y)$ and $P \leftarrow H(\pi_y)$, where H is a hash function.
   (c) Send $\textbf{trans}^{CP}_{commit} = (Y, P)$ to BC.
   (d) send (”verify”,y, π_y) to CU.
6. If CU has not sent message $\textbf{trans}^{CU}_{agree}$ to BC before τ_a, then send $\textbf{trans}^{CP}_{verify} = (ek_F, vk_F, x, y, \pi_y)$ to BC.

For Blockchain BC: Set $state \leftarrow Init$, $\$reward \leftarrow 0$, $\$deposit \leftarrow \{\}$

7. On receiving $\textbf{trans}^{CU}_{create}$ execute $PBIOC.create(E, V, X, \tau_i, \tau_c, \tau_a, \tau_{end}, \$r, \$c)$
8. On receiving $\textbf{trans}^{CP}_{intent}$ execute $PBIOC.intent(\$d)$
9. On receiving $\textbf{trans}^{CP}_{commit}$ execute $PBIOC.commit(Y, P)$
10. On receiving $\textbf{trans}^{CU}_{agree}$ execute $PBIOC.agree()$
11. On receiving $\textbf{trans}^{CP}_{verify}$ execute $PBIOC.verify(ek_F, vk_F, x, y , \pi_y)$
12. On receiving $\textbf{trans}^{PBIOCV}_{result}$ execute $PBIOC.result(Honest)$

Timer

```
if $\tau > \tau_{end}$ and $state != Terminated$ then
    if $state = Created$ then
        set $ledger[CU] \leftarrow ledger[CU] + \$reward + (\$c, CU)$
    if $state = Intent \,||\, Committed$ then
        set $ledger[CU] \leftarrow ledger[CU] + \$reward + (\$c, CU) + (\$d, CP)$
    if $state = Dispute$ then
        set $ledger[CP] \leftarrow ledger[CP] + \$reward + (\$c, CU) + (\$d, CP)$
    set $state \leftarrow Terminated$
```

Figure 3.1: PBIOC protocol

Algorithm 3.1 $PBIOC.create$

```
Input: $E, V, X, \tau_i, \tau_c, \tau_a, \tau_{end}, \$r, \$c$
Output: Success or Failure message
1: if $state = Init$ then
2:     if $\tau < \tau_i < \tau_c < \tau_a < \tau_{end}$ then
3:         if $ledger[CU] \geq \$r + \$c$ then
4:             $ledger[CU] \leftarrow ledger[CU] - (\$r + \$c)$;
5:             $\$reward \leftarrow \$r$;
6:             $\$deposit \leftarrow \$deposit \cup (\$c, CU)$;
7:             $state \leftarrow Created$;
8:             return (Success, Task created)
9:         else return (Failure, Balance is low)
10:    else return (Failure, Bad timing parameters)
11: else return (Failure, State is not $Init$ )
```

Algorithm 3.2 $PBIOC.intent$

```
    Input: $\$d$
    Output: Success or Failure message
 1: if $state = Created$ then
 2:     if $\tau < \tau_i$ then
 3:         if $ledger[CP] \geq \$d$ then
 4:             $ledger[CP] \leftarrow ledger[CP] - \$d$
 5:             $\$deposit \leftarrow \$deposit \cup (\$d, CP)$
 6:             $state \leftarrow Intent$
 7:             return (Success, Intent success)
 8:         else return (Failure, Balance is low)
 9:     else return (Failure, Intent timeout)
10: else return (Failure, State is not $Created$)
```

Algorithm 3.3 $PBIOC.commit$

```
    Input: $Y$, $P$
    Output: Success or Failure message
 1: if $state = Intent$ then
 2:     if $\tau < \tau_c$ then
 3:         Store $Y, P$
 4:         $state \leftarrow Committed$
 5:         return (Success, Commit success)
 6:     else return (Failure, Commit timeout)
 7: else return (Failure, State is not $Intent$)
```

Algorithm 3.4 $PBIOC.agree$

```
    Input: $\phi$
    Output: Success or Failure message
 1: if $state = Committed$ then
 2:     if $\tau < \tau_a$ then
 3:         set $ledger[CP] \leftarrow ledger[CP] + \$reward + (\$d, CP)$
 4:         set $ledger[CU] \leftarrow ledger[CU] + (\$d, CU)$
 5:         $state \leftarrow Terminated$
 6:         return (Success, Agree success)
 7:     else return (Failure, Agree timeout)
 8: else return (Failure, State is not $Committed$)
```

Algorithm 3.5 $PBIOC.verify$

```
Input: $ek_F$, $vk_F$, $x$, $y$ , $\pi_y$
Output: Success or Failure message
1: if $state! = Terminated$ then
2:     if $\tau_a < \tau < \tau_{end}$ then
3:         if $E = H(ek_f) \wedge V = H(vk_f) \wedge X = H(x) \wedge Y = H(y) \wedge P = H(\pi_y)$ then
4:             Invoke $PBIOCV$
5:             set $state \leftarrow Dispute$
6:             return (Success, Dispute raised)
7:         else return (Failure, Open commitment Failed)
8:     else return (Failure, Verify timeout)
9: else return (Failure, Contract is $Terminated$)
```

Algorithm 3.6 $PBIOC.result$

```
Input: $Honest$
Output: Success or Failure message
1: if $state = Dispute$ then
2:     if $\tau < \tau_{end}$ then
3:         if $Honest = true$ then
4:             set $ledger[CP] \leftarrow ledger[CP] + \$reward + (\$c, CU) + (\$d, CP)$
5:         else
6:             set $ledger[CU] \leftarrow ledger[CU] + \$reward + (\$c, CU) + (\$d, CP)$
7:         set $state \leftarrow Terminated$
8:         return (Success, Dispute resolved)
9:     else return (Failure, Contract timeout)
10: else return (Failure, State is not $Dispute$)
```

If both CU and CP are honest, then the execution cost of PBIOC protocol is minimal and also have the privacy of their inputs and outputs. However, if anyone party is dishonest, then PBIOCV contract is invoked. As we have already discussed the $Verify$ algorithm from Parno *et al.* [184] is modeled as PBIOCV contract, running it is costly, and the privacy of the results no longer exists. The inherent problem with the proof-based systems is the provider's overhead, and the verifier's[2] cost per instance is high. The verification time for the state-of-the-art verifiable computation scheme [184] is 9ms and takes 288 bytes of storage. In our protocol, even if we do not consider the user overhead in converting the function into a boolean circuit, the verification of the proof (i.e., execution of PBIOCV contract) by the Blockchain network will consume a huge amount of time. The verifiers in public Blockchain systems can choose the transactions which will go into a block and then

[2]miners in Blockchain

into a blockchain. Since the average block generation time is very low, it is not practical for the verifiers to accept transactions which would take a long time. Therefore, a verifier will avoid the verification of the transactions which consumes huge computational resources [97]. A generic PBIOCV contract can be very complex, costly and even may not be feasible to deploy in current Blockchain networks.

3.3 Replication-based Incentivized Outsourced Computation using Smart Contracts

In this section, we discuss achieving fairness and correctness in verifiable computation by outsourcing the same task to multiple providers.

3.3.1 Economic model

Let CU outsources a computation to multiple providers CP_1,...,CP_n. The providers will compute the outsourced computation and return the output. There is a chance that providers may use different algorithms other than the one prescribed by the user and yet deliver the correct output with negligible probability. There might be an algorithm which gives higher utility than the specified algorithm and yet deliver the correct output. The user always wants the providers to compute the prescribed algorithm. For example, the user does not outsource "search for an element in a given input set". He will outsource a "particular searching algorithm along with the input". Another example is when the output range of the outsourced algorithm is binary. In this case, the provider can guess the output with a 50% probability without running the algorithm. To prevent providers from using different algorithms, Belenkiy *et al.*[153] introduced the concept known as the inner state of an algorithm..

Definition 3.3.1. *Assuming the algorithm is composed of a finite number of atomic operations and each atomic operation takes some state information as input and produces another state information as output. The inner state of an algorithm is defined as the concatenation of all the input and output states of the atomic operations of an algorithm, along* *with the definition of the algorithm in terms of atomic operations.*

Inner State Hash(ISH): An l-bit hash function takes an inner state of an algorithm as input and maps it into an l-bit random string. Even if the algorithm has many number of steps / its output is large, the hash value is always short having a constant length. The probability of producing the same hash value without using the algorithm prescribed by the user is negligible, i.e., $neg = O(2^{-l})$.

Definition 3.3.2. *The algorithm used by the provider to complete the assigned work that outputs the correct answer with probability q is known as q-algorithm [155]. If a provider runs the algorithm prescribed by the user, then $q = 1$. Similarly, if the provider uses any algorithm other than the user prescribed, then $q < 1$.*

The cost of running a q-algorithm is $cost(q)$. As $q = 1$ for running the prescribed algorithm, we denote the cost of honest computation as $cost(1)$. The high-level description of our approach is that CU outsources the same computation to multiple providers. The providers compute and return the outputs along with the inner state hash. CU compares all the inner state hashes and the outputs sent by providers. If they all are equal, he accepts one of the output, otherwise re-outsources the algorithm. CU can not be trusted by the providers, so he posts a smart contract on to a distributed ledger initiating it with some pay. Similarly, the providers submit their inner state hash value and output to the smart contract. Now smart contract verifies the outputs; if all are equal, providers will receive the payment. Interestingly, even if all the providers submit the same incorrect output, they all get the pay. We categorize the providers into three categories:

(a) **Honest provider**: An honest provider performs the computation exactly (running the prescribed algorithm) as prescribed by the user.

(b) **Rational provider**: A rational provider performs the computation precisely as prescribed by the user as long as his utility of computing the original algorithm is more than the utility of doing anything else.

(c) **Malicious provider**: A malicious provider has two objectives: (1) Making the user accept the incorrect result or (2) Making the user re-outsource the task.

The providers who use the algorithm as prescribed by the user be called diligent (all honest providers and some rational providers who behave honestly) and who uses a q-algorithm are called lazy (all rational who behaves maliciously and all malicious). The utilities of the providers in a two-provider case are given in Table 3.1.

Other / This provider	**Diligent**	**Lazy**
Diligient	$u(1) = \$r - cost(1)$	$u(q) = \$r * q - \$d * (1 - q) - cost(q)$
Lazy	$u(1) = \$r - cost(1)$	$u(q) = \$r - cost(q)$

Table 3.1: Utilities in two-provider case from [153]

3.3.2 Two providers Case (TUIOC Contract)

3.3.2.1 TUIOC Contract Clauses

TUIOC is an outsourcing contract signed between three parties, a CU and two providers (CP_0, CP_1). We follow the prisoner's dilemma model presented by Belenkiy *et al.* [153] and kupcu [155]. We assume $F(x)$ is deterministic, and there exists a smart contract TUIOCV which can compute $F(x)$ and return (y, ish) as output. CU pays a bounty $\$b$ along with reward $\$r$ for an honest provider in case of disputes.

The clauses in the TUIOC contract are as follows:

(i) CU prepares two contracts $TUIOC$ and $TUIOCV$. $TUIOCV$ is executed only in case of disputes.

(ii) CU chooses a function $F(\cdot)$ and an input x. CU agrees to pay $\$r$ to each provider for the correct computation of $F(x)$. CU also agrees to pay $\$b$ to the honest provider in case of disputes.

(iii) All the three parties agree on timing parameters $\tau < \tau_i < \tau_c < \tau_{end}$.

(iv) Each CP_i, $i \in \{0, 1\}$ must pay a deposit of $\$d$ before τ_i. If any CP_i fails to deposit, then the contract is terminated and any deposits made are refunded.

(v) Each CP_i computes $F(x)$ and delivers the output to the contract before τ_c. If any CP_i fails to deliver output by τ_c, then its output is set as NULL. If both providers fails to send output before τ_c, then $2 * (\$d + \$r) + \$b$ is sent to CU and the contract is terminated.

(vi) At $\tau_c < \tau < \tau_{end}$, the contract compares the outputs delivered by providers for equality. If both the outputs are equal, then $\$d + \$r$ is sent to each CP_i, $\$b$ is sent to CU and the contract is terminated. Otherwise $TUIOCV$ contract is invoked.

(vii) $TUIOCV$ computes $F(x)$ and returns the result before τ_{end}.

(viii) The smart contract compares the results sent by $TUIOCV$ with the outputs delivered by providers.

(1) If TUIOCV and CP_i outputs are same, then $(\$d + \$r + \$b)$ is sent to CP_i, $\$d + \$r$ is sent to CU, and the contract is terminated.

(2) If TUIOCV output is not matching with any of the CP_i's output, then $2 * (\$d + \$r) + \$b$ is sent to CU, and contract is terminated.

(ix) If $\tau > \tau_{end}$ and the contract is not terminated, then $(\$d + \$r)$ is sent to each CP_i, $\$b$ is sent to CU, and the contract is terminated.

3.3.2.2 TUIOC protocol

Unlike in Küpçü[155] and Belenkiy *et al.*[153], we do not assume the user acts diligently. In our model, the providers do not trust the user for payment, and the user does not trust the providers for correct computation. All the three entities trust the underlying Blockchain for correct computation of the smart contract. TUIOC protocol is presented in Figure 3.2. $TUIOC$ smart contract functionalities executed by Blockchain are presented as Algorithms 3.7 - 3.11. CU chooses a function F, an input x, timing parameters τ_i, τ_c, τ_r, τ_{end} and publishes these parameters on a public platform. He also sends these parameters along with a pay $\$r$, and a bounty $\$b$ to BC invoking Algorithm 3.7. Bounty is given to the honest provider in case of disputes. Algorithm 3.7 stores the parameters and changes the state to *Created*. Two interested cloud providers can show intent to compute the outsourcing task by sending their deposits to BC invoking Algorithm 3.8. Algorithm 3.8 adds a cloud provider to a worker set and if the two cloud providers have shown the intent, then the state is changed to *Compute*. After showing intent the cloud providers have to send the commitment of the output along with hash of the inner state to BC invoking Algorithm 3.9. Algorithm 3.9 stores the commitments sent by the cloud providers and if both the cloud providers sent their commitments, then the state is set as *Reveal*. There are three cases depending on the behavior of the providers

Case 1: Both the providers have committed the output. In this case, the providers have to reveal the committed outputs by invoking the Algorithm 3.10. Algorithm 3.10 verifies the revealed parameters against the commitments, and if the commitments are correct, then they are stored. If both the provider's commitments are correct, then the state is set as *Pay* . There are three cases depending on the behavior of the providers

Case 1.1 Both the providers have revealed the output. In this case, the outputs are compared for equality. If both the outputs are equal, then ($\$r + \$d$) is sent to each provider and $\$b$ is sent to CU. Else, $TUIOCV$ contract is invoked which will return the output invoking Algorithm 3.11. Algorithm 3.11 compares the output returned by $TUIOCV$ with the outputs submitted by the cloud providers. If the output of any of the cloud provider matches, then it is rewarded and the other provider is financially penalized. Algorithm 3.11 also sets the state to *Terminated*.

Case 1.2: Only one of the provider have revealed the output. In this case, the output of the other provider is set as ϕ and the $TUIOCV$ contract is invoked.

Case 1.3: None of the providers have revealed the output. In this case, $2*(\$r+\$d)+\$b$ is sent to CU.

Case 2: Only one of the provider have committed the output. In this case, the contract state is changed to Reveal.

Case 3: None of the providers have committed the output. In this case, $2 * (\$r + \$d) + \$b$ is sent to CU.

TUIOC protocol

Let (G, P, Q) be the public parameters generated through a trusted setup such that

```
$G$ is an order-$q$ elliptic curve group over $\mathbb{F}_p$, $P$ and $Q$ are random generators of $G$.
For cloud user $CU$
  1. To create an outsourcing task send **trans**$^{CU}_{create}$ = ($F$, $x$, $\tau_i$, $\tau_c$, $\tau_r$, $\tau_{end}$, $\$r$, $\$b$) to $BC$
For cloud provider $CP_i$
  2. To participate in the outsourcing task, send **trans**$^{CP_i}_{intent}$ = ($\$d$) to $BC$.
  3. To send the output
     (a) Run $compute(F, x) \to (y_i, ish_i)$
     (b) Generate two random numbers $s_1 \in_R Z_q$ and $s_2 \in_R Z_q$
     (c) Compute $cm_{y_i} \leftarrow y_i P + s_1 Q$ and $cm_{ish_i} \leftarrow ish_i P + s_2 Q$
     (d) Send **trans**$^{CP_i}_{commit}$ = ($cm_{y_i}$, $cm_{ish_i}$) to $BC$
  4. To reveal the output send **trans**$^{CP_i}_{reveal}$ = ($y_i$, $ish_i$, $s_1$, $s_2$) to $BC$
For Blockchain:
  5. On receiving **trans**$^{CU}_{create}$ execute $TUIOC.create(F, x, \tau_i, \tau_c, \tau_r, \tau_{end}, \$r, \$b)$
  6. On receiving **trans**$^{CP_i}_{intent}$ execute $TUIOC.intent(\$d)$
  7. On receiving **trans**$^{CP_i}_{commit}$ execute $TUIOC.commit(cm_{y_i}, cm_{ish_i})$
  8. On receiving **trans**$^{CP_i}_{reveal}$ execute $TUIOC.reveal()$
  9. On receiving **trans**$^{TUIOC}_{dispute}$ execute $TUIOC.dispute()$
Timer
    **If** $\tau > \tau_r$ and $state = Pay$ **then**
        **If** $y_0 = y_1$ and $ish_0 = ish_1$ **then**
            Set $ledger[CP_0] \leftarrow ledger[CP_0] + \$r + \$d$
            Set $ledger[CP_1] \leftarrow ledger[CP_1] + \$r + \$d$
            Set $ledger[CU] \leftarrow ledger[CU] + \$b$ and $state \leftarrow Terminated$
        Else send **trans**$^{TUIOC}_{dispute}$ to TUIOCV and set $State \leftarrow Dispute$
    **If** $\tau > \tau_i$ and $state = Created$ **then**
        Set $ledger[CU] \leftarrow ledger[CU] + 2 * \$r + \$b$
        **If** $|worker| \neq 0$ **then**
            Set $ledger[CP_i] \leftarrow ledger[CP_i] + \$d, \forall\, CP_i \in workers$
        Set $state \leftarrow Aborted$
    **If** $\tau > \tau_c$ and $state = Compute$ and $|commitment| \neq 0$ **then** set $state \leftarrow Reveal$
    **If** $\tau > \tau_c$ and $state = Compute$ and $|commitment| = 0$ **then**
        Set $ledger[CU] \leftarrow ledger[CU] + 2 * (\$r + \$d) + \$b$ and $state \leftarrow Aborted$
    **If** $\tau > \tau_r$ and $state = Reveal$ and $|output| \neq 0$ **then** set $state \leftarrow Pay$
    **If** $\tau > \tau_r$ and $state = Reveal$ and $|output| = 0$ **then**
        Set $ledger[CU] \leftarrow ledger[CU] + 2 * (\$r + \$d) + \$b$ and $state \leftarrow Aborted$
    **If** $\tau > \tau_{end}$ and $state = Pay || Dispute$ **then**
        Set $ledger[CP_0] \leftarrow ledger[CP_0] + \$r + \$d$
        Set $ledger[CP_1] \leftarrow ledger[CP_1] + \$r + \$d$
        Set $ledger[CU] \leftarrow ledger[CU] + \$b$ and $state \leftarrow Terminated$
```

Figure 3.2: TUIOC protocol

Algorithm 3.7 $TUIOC.create$

```
    Input: $F$, $x$, $\tau_i$, $\tau_c$, $\tau_r$, $\tau_{end}$, $\$r$, $\$b$
    Output: Success or Failure message
 1: if $state = Init$ then
 2:     if $\tau < \tau_i < \tau_c < \tau_r < \tau_{end}$ then
 3:         if $ledger[CU] \geq 2 * \$r + \$b$ then
 4:             Set $ledger[CU] \leftarrow ledger[CU] - (2 * \$r) - \$b$;
 5:             Set $state \leftarrow Created$
 6:             return (Success, Task created)
 7:         else return (Failure, Balance is low)
 8:     else return (Failure, Bad timing parameters)
 9: else return (Failure, State is not $Init$ )
```

Algorithm 3.8 $TUIOC.intent$

```
    Input: $\$d$
    Output: Success or Failure message
 1: if $state = Created$ then
 2:     if $\tau < \tau_i$ then
 3:         if $ledger[CP_i] \geq \$d$ then
 4:             if $CP_i \notin worker$ then
 5:                 set $ledger[CP_i] \leftarrow ledger[CP_i] - \$d$
 6:                 set $worker \leftarrow worker \cup CP_i$
 7:                 if $|worker| = 2$ then
 8:                     set $state \leftarrow Compute$
 9:                 return (Success, Intent success)
10:             else return (Failure, Duplicate provider)
11:         else return (Failure, Balance is low)
12:     else return (Failure, Intent timeout)
13: else return (Failure, State is not $Created$)
```

Algorithm 3.9 $TUIOC.commit$

```
    Input: $cm_{y_i}$,$cm_{ish_i}$
    Output: Success or Failure message
 1: if $state = Compute$ then
 2:     if $\tau < \tau_c$ then
 3:         if $CP_i \in worker$ then
 4:             if $(CP_i, *, *) \notin commitment$ then
 5:                 set $commitment \leftarrow commitment \cup (CP_i, cm_{y_i}, cm_{ish_i})$
 6:                 return (Success, Commit success)
 7:                 if $|commitment| = 2$ then
 8:                     set $state \leftarrow Reveal$
 9:             else return (Failure, Duplicate commitment)
10:         else return (Failure, Wrong provider)
11:     else return (Failure, Commit timeout)
12: else return (Failure, State is not $Compute$)
```

Algorithm 3.10 $TUIOC.reveal$

```
Input: $\phi$
Output: Success or Failure message
 1: if $state = Reveal$ then
 2:     if $\tau < \tau_r$ then
 3:         if $(CP_i, *, *) \in commitment$ then
 4:             if $(CP_i, *, *) \notin output$ then
 5:                 if $cm_{y_i} = y_iP + s_1Q$ and $cm_{ish_i} = ish_iP + s_2Q$ then
 6:                     set $output \leftarrow output \cup (CP_i, y_i, ish_i)$
 7:                     if $|output| = 2$ then
 8:                         set $state \leftarrow Pay$
 9:                     return (Success, Reveal success)
10:                 else return (Failure, Open commitment failed)
11:             else return (Failure, Duplicate output found)
12:         else return (Failure, Not committed)
13:     else return (Failure, Reveal timeout)
14: else return (Failure, State is not $Reveal$)
```

Algorithm 3.11 $TUIOC.dispute$

```
Input: $\phi$
Output: Success or Failure message
 1: if $state = Dispute$ then
 2:     if $\tau_r < \tau < \tau_{end}$ then
 3:         if $y_t = y_0$ and $ish_t = ish_0$ then
 4:             set $ledger[CP_0] \leftarrow ledger[CP_0] + \$r + \$d + \$b$
 5:             set $ledger[CU] \leftarrow ledger[CU] + \$d + \$r$
 6:         else
 7:             if $y_t = y_1$ and $ish_t = ish_1$ then
 8:                 $ledger[CP_1] \leftarrow ledger[CP_1] + \$r + \$d + \$b$
 9:                 $ledger[CU] \leftarrow ledger[CU] + \$d + \$r$
10:             else set $ledger[CU] \leftarrow ledger[CU] + 2 * (\$r + \$d) + \$b$
11:         set $state \leftarrow Terminated$
12:         return (Success, Dispute resolved)
13:     else return (Failure, Contract timeout)
14: else return (Failure, State is not $Dispute$)
```

We have already defined that the cost of honest computation is $cost(1)$, and the cost of running a q-algorithm is $cost(q)$. Let the cost of running the TUIOCV contract be $cost(V)$. We assume that the deposit by a provider $\$d \geq cost(V)$. $\$d$ compensates the user for the $cost(V)$. The analysis of $TUIOC$ protocol is presented in Table 3.2. If the providers collude and sent the same incorrect output, then their utility will be maximum. If no collusion occurs, then the best strategy for the providers is to act diligently. Colluding of the providers is avoided by offering a bounty for acting diligently such that

$\$r - cost(1) + \$b > \$r - cost(q)$. This bounty is given to the diligent provider only when the outputs are different. Unfortunately, this bounty is a burden to the user as this is an extra payment apart from the reward.

Entities			**Utility**				
s.no	CP_0	CP_1	output	$\mu(CU)$	$\mu(CP_0)$	$\mu(CP_1)$	Clause
1	1	1	CO	-	$\$r - cost(1)$	$\$r - cost(1)$	vi
2	1	0	CO	$\$d - cost(V)$	$\$r + \$b - cost(1)$	$-\$d - cost(q)$	viii.1
3	0	1	CO	$\$d - cost(V)$	$-\$d - cost(q)$	$\$r + \$b - cost(1)$	viii.1
4*	0	0	WO	-	$\$r - cost(q)$	$\$r - cost(q)$	vi
5**	0	0	WO	$2 * \$d - cost(V)$	$-\$d - cost(q)$	$-\$d - cost(q)$	viii.2

Table 3.2: Analysis of TUIOC: 1-Diligent, 0-Lazy, CO-Correct Output, WO-Wrong Output, $cost(V)$ is cost of executing TUIOCV Contract. Here, we assume TUIOCV contract is invoked by CU and hence the $cost(V)$ is paid by CU. *when both providers return the same incorrect outputs. ** when the providers return different incorrect results.

3.3.3 Multiple-provider Case (MUIOC)

We extend the two providers case to n providers case and try to achieve fair incentivized outsourced computation. If $n = 2$, then this case is similar to a two providers case, hence we assume $n > 2$.

3.3.3.1 MUIOC Contract Clauses

The high-level overview of the multiple-providers case is that the user outsources the job on a public platform along with the smart contract address. Interested providers show their intent by sending some deposit to the contract. The protocol proceeds in rounds, in each round some $k < n$ number of providers are randomly hired. The selected providers compute the job and submit their outputs. The contract verifies all the received outputs, and if all the outputs are same, then the contract is terminated by sending appropriate pay to each provider who computed correctly. Otherwise, the contract re-outsources the job to a different set of providers until a correct output is obtained. We assume at least one honest provider is hired per round.

The clauses in the $MUIOC$ contract are as follows:

(i) CU prepares two contracts $MUIOC$ and $MUIOCV$. $MUIOCV$ is executed only in case of disputes.

(ii) CU chooses $F(\cdot)$ and an input x. CU agrees to pay a minimum of $\$r/k$ to each CP_i for correct and timely computation of $F(x)$. He also chooses timing parameters $\tau < \tau_i < \tau_c < \tau_{end}$ and k, p, where k is the number of providers hired per round and p is the time required per round.

(iii) As a condition, the providers who wish to compute $F(x)$ must pay a deposit of $\$d$ before τ_i. Let $W \subseteq CP$ be all the providers who paid deposits before τ_i. Let $\$d'$ be the sum of all the deposits made by providers. If $|W| \leq 2$ after τ_i, then the contract is terminated, and the reward is refunded to CU and any deposits made by the providers are also refunded.

(iv) Until $|W| < k$

(a) Smart contract generate a random subset of providers $\Omega_r \subset W$ and notify them to compute $F(x)$.

(b) Every $CP_i \in \Omega_r$ computes $F(x)$ and delivers its output before τ_c. If any $CP_i \in \Omega_r$ fails to send the result by τ_c, it is marked as cheated and its output is set as $NULL$.

(c) Smart contract compares all the outputs obtained in an r^{th} round for equality.

(1) If all the outputs are equal, then every $CP_i \in \Omega_r$ is marked as honest and also all the providers in previous rounds who sent the same output are marked as honest. $\frac{\$d'}{|W|}\left(\frac{r*k}{r*k-|m|}\right) + \frac{\$r}{r*k-|m|}$ is sent to each honest provider, where $|m|$ is the size of malicious providers marked as cheated and lost their deposit for behaving maliciously. $\$d'/|W|$ is sent to all the providers who have not hired in any round and the contract is terminated.

(2) Otherwise, all the providers in r^{th} round are marked as cheated and the parameters are updated as $r = r + 1$, $\tau_c = \tau_c + p$, $W = W - \Omega_r$.

(v) If $|W| < k$, then $MUIOCV$ contract is invoked and the output of the $MUIOCV$ contract is compared with all the hired providers output. The providers who sent the output same as $MUIOCV$ are marked as honest, and all the providers who sent a different output are marked as cheated.

3.3.3.2 MUIOC protocol

MUIOC protocol is presented in Figure 3.3. $MUIOC$ smart contract functionalities executed by Blockchain are presented as Algorithms 3.12 - 3.16. CU chooses a function F, an input x, timing parameters τ_i, τ_c, τ_r, τ_{end} and publishes these parameters on a public platform. He also chooses two more parameters k and p, where k is the number of providers hired per round, and p is the time required per round. He sends all these parameters along with $\$r$ to BC invoking Algorithm 3.12. The pay $\$r$ is shared among the honest providers. Algorithm 3.12 stores all the parameters and sets the state as *Created*. Interested providers show intent by sending deposits to BC invoking Algorithm 3.13. Algorithm 3.13 stores a provider's deposit and adds the provider to a list and increments the count of the providers. After the intent time is expired, a random set of providers is selected to perform the outsourcing task. The selected cloud provider has to compute and send the commitment of the output and inner state hash to BC invoking Algorithm 3.14. Algorithm 3.14 stores the commitment sent by every provider. Depending on the behavior of the providers there are three cases as follows:

Case 1: All the selected providers have committed the output. In this case, the providers have to reveal the output by invoking Algorithm 3.15. Algorithm 3.15 stores the output received from a provider. Depending on the behavior of the providers there are two cases as follows:

Case 1.1: All the providers who committed the output have revealed the output by invoking Algorithm 3.15. In this case, after the reveal timeout, the outputs are compared for equality. If all the outputs are equal, then the providers receive the agreed payment. The providers in the previous rounds who sent the same output will also get the agreed payment. Otherwise, all the providers are marked as cheated, and a new round begins with a random selection of providers. If there are not enough providers for the new round, then $MUIOCV$ contract is invoked, and pay is distributed according to the output sent by the $MUIOCV$ contract invoking Algorithm 3.16. Algorithm 3.16 computes honest and malicious lists according to the output sent by $MUIOCV$.

Case 1.2: Only a subset or none of the providers have revealed the output. All the providers are marked as cheated, and a new round begins the same as the previous case.

Case 2: Only a subset of providers have sent the output. In this case, the aborting providers' commitment is set as NULL, and the rest of the providers will reveal the output.

Case 3: None of the providers has sent the output. In this case, all the providers' commitment and output is set as NULL, and a new round begins with a random set of providers. If there are not enough providers for the new round, then $MUIOCV$ contract is invoked, and pay is distributed according to the output returned by the $MUIOCV$ contract.

MUIOC protocol

Let (G, P, Q) be the public parameters generated through a trusted setup such that G is an order-q elliptic curve group over $\mathbb{F}_p$, P and Q are random generators of G

For cloud user CU

1. To create an outsourcing task send $\textbf{trans}^{CU}_{create}$ = $(F,\ x,\ \tau_i,\ \tau_c,\ \tau_r,\ \tau_{end},\ \$r)$ to BC

For cloud provider CP_i

2. To participate in the outsourcing task send $\textbf{trans}^{CP_i}_{intent}$ = ($\$d$) to BC
3. To commit the output
   (a) Compute $(ish_i, y_i) \leftarrow F(x)$
   (b) Generate two random numbers $s_1 \in_R Z_q$ and $s_2 \in_R Z_q$
   (c) Compute $cm_{y_i} \leftarrow y_iP + s_1Q$ and $cm_{ish_i} \leftarrow ish_iP + s_2Q$
   (d) Send $\textbf{trans}^{CP_i}_{commit}$ =(cm_{y_i}, cm_{ish_i}) to BC
4. To reveal the output send $\textbf{trans}^{CP_i}_{reveal}$ = (y_i, ish_i, s_1, s_2) to BC

For Blockchain:

5. On receiving $\textbf{trans}^{CU}_{create}$ execute $MUIOC.create(F, x, \tau_i, \tau_c, \tau_r, \tau_{end}, \$r)$
6. On receiving $\textbf{trans}^{CP_i}_{intent}$ execute $MUIOC.intent(\$d)$
7. On receiving $\textbf{trans}^{CP_i}_{commit}$ execute $MUIOC.commit(cm_{y_i}, cm_{ish_i})$
8. On receiving $\textbf{trans}^{CP_i}_{reveal}$ execute $MUIOC.reveal(y_i, ish_i, s_1, s_2)$
9. On receiving $\textbf{trans}^{MUIOC}_{dispute}$ execute $MUIOC.dispute(y_t, ish_t)$

```
Timer
    If $\tau > \tau_i$ and $state = Created||Discord$ then
        If $|CP| \geq k$ then
            select a random subset $\Omega_r \subset CP$ of size $k$
            set $CP \leftarrow CP - \Omega_r$ and $state \leftarrow Compute$
            send ("compute",$\Omega_r$) to all parties
    If $\tau > \tau_r$ and $state = Compute$ then
        $(status, y, ish) \leftarrow compare(\Omega_r, output)$
        If $status = agreed$ then
            while $r > 0$
                $(H, M) \leftarrow getHonest(r, \Omega_r, output, y, ish)$
                $r \leftarrow r - 1$
            set $state \leftarrow Agreed$
        Else set $\tau_c \leftarrow \tau_c + p, \tau_r \leftarrow \tau_r + p, r \leftarrow r + 1, state \leftarrow Discord$
    If $\tau > \tau_c$ and $state = Discord$, then send ("dispute") to $MULIOCV$ and set
    $state \leftarrow Dispute$
    If $\tau > \tau_{end}$ then
        If $state = Agreed$ then
            set $\$mdeposit \leftarrow \$deposit/(n - |M| - |CP|)$ and $\$deposit \leftarrow$
            $\$deposit - \$mdeposit$
            for every $CP_i \in H$
                set $ledger[CP_i] \leftarrow ledger[CP_i] + (\$deposit/n) + (\$reward/(k * (r) - |M|)) +$
                $(\$mdeposit/|H|)$
            for every $CP_i \in CP$ set
                $ledger[CP_i] \leftarrow ledger[CP_i] + (\$deposit/n)$.
            set $state \leftarrow Terminated$
        If $state = Created$ then
            set $ledger[CU] \leftarrow ledger[CU] + \$reward$
            for every $CP_i \in CP$ set $ledger[CP_i] \leftarrow ledger[CP_i] + \$deposit/n$
            set $state \leftarrow Aborted$
        If $state = Dispute$ then
            for every $CP_i \in CP$ set $ledger[CP_i] \leftarrow ledger[CP_i] + \$deposit/n$
            while $r > 0$
                for every $CP_i \in \Omega_r$ set $ledger[CP_i] \leftarrow ledger[CP_i] + (\$deposit/n) + (\$reward/(r * k))$
                set $r \leftarrow r - 1$
            set $state \leftarrow Terminated$.
```

Figure 3.3: MUIOC protocol

Algorithm 3.12 $MUIOC.create$

```
    Input: $F$, $x$, $\tau_i$, $\tau_c$, $\tau_r$, $\tau_{end}$, $\$r$
    Output: Success or Failure message
 1: if $state = Init$ then
 2:     if $\tau < \tau_i < \tau_c < \tau_r < \tau_{end}$ then
 3:         if $ledger[CU] \geq \$r$ then
 4:             Set $ledger[CU] \leftarrow ledger[CU] - \$r$
 5:             Set $\$reward \leftarrow \$r$
 6:             Set $state \leftarrow Created$
 7:             return (Success, Task created)
 8:         else return (Failure, Balance is low)
 9:     else return (Failure, Bad timing parameters)
10: else return (Failure, State is not $Init$ )
```

Algorithm 3.13 $MUIOC.intent$

```
    Input: $\$d$
    Output: Success or Failure message
 1: if $state = Created$ then
 2:     if $\tau < \tau_i$ then
 3:         if $ledger[CP_i] \geq \$d$ then
 4:             if $\$d \geq \$reward$ then
 5:                 if $CP_i \notin CP$ then
 6:                     Set $ledger[CP_i] \leftarrow ledger[CP_i] - \$d.$
 7:                     Set $\$deposit \leftarrow \$deposit + \$d.$
 8:                     Set $CP \leftarrow CP \cup CP_i.$
 9:                     Set $n \leftarrow n + 1$
10:                     return (Success, Intent success)
11:                 else return (Failure, Duplicate request)
12:             else return (Failure, Deposit not enough)
13:         else return (Failure, Balance is low)
14:     else return (Failure, Intent timeout)
15: else return (Failure, State is not $Created$)
```

Algorithm 3.14 $MUIOC.commit$

```
    Input: $cm_{y_i}$, $cm_{ish_i}$
    Output: Success or Failure message
 1: if $state = Compute$ then
 2:     if $\tau \leq \tau_c$ then
 3:         if $(CP_i, *, *) \notin commitments$ then
 4:             if $CP_i \in \Omega_r$ then
 5:                 Set $commitments \leftarrow commitments \cup (CP_i, cm_{y_i}, cm_{ish_i})$
 6:                 return (Success, Committed)
 7:             else return (Failure, Wrong provider)
 8:         else return (Failure, Duplicate commitment)
 9:     else return (Failure, Commit timeout)
10: else return (Failure, State is not $Compute$ )
```

Algorithm 3.15 $MUIOC.reveal$

```
Input: $y_i, ish_i, s_1, s_2$
Output: Success or Failure message
1: if $\tau_c < \tau < \tau_r$ then
2:     if $CP_i \in \Omega_r$ then
3:         if $(CP_i, *, *) \notin output$ then
4:             if $cm_{y_i} = y_iP + s_1Q$ and $cm_{ish_i} = ish_iP + s_2Q$ then
5:                 Set $output \leftarrow output \cup (CP_i, y_i, ish_i)$
6:                 return (Success, Revealed)
7:             else return (Failure, Reveal failed)
8:         else return (Failure, Duplicate request)
9:     else return (Failure, Wrong provider)
10: else return (Failure, Reveal timeout )
```

Algorithm 3.16 $MUIOC.dispute$

```
Input: $y_t$, $ish_t$
Output: Success or Failure message
1: if $state = Dispute$ then
2:     if $\tau_c < \tau < \tau_{end}$ then
3:         while $r > 0$ do
4:             Set $(H, M) \leftarrow getHonest(r, \Omega_r, output, y_t, ish_t)$
5:             Set $r \leftarrow r - 1$
6:         Set $state \leftarrow Agreed$
7:     else return (Failure, Contract timeout)
8: else return (Failure, State is not $Dispute$ )
```

Excepted number of rounds: The contract outsources the job until all the providers hired in a round returns the same output. Let E be the probability of getting different outputs in a single round then the expected number of rounds the task is outsourced is $\frac{1}{1-E}$. The contract outsources the task at least once. If the outputs returned in the first round are not equal, then the contract outsources the task again, this will happen with a probability of E. Again if the outputs are not equal, then the contract outsources with a probability of E^2 and so on till all the outputs in a round are equal. The expected number of rounds the task is outsourced is

$$1 + E + E^2 + \cdots = \frac{1}{1-E}$$

3.4 Simulation Results and Discussions

The simulation environment is discussed in Section 1.2.3. The actual tasks are treated as black boxes, and the contracts do not need to know their internal states. The contracts are called before, during or after the execution of the tasks. We have implemented all the contracts in private Ethereum network which mimics the Ethereum production network. However, our goal of this implementation is to deploy it in public Blockchain networks in real scenarios.

3.4.1 Implementation of PBIOC

We ran our experiments multiple times, and each transaction's computational and financial cost is listed in Table 3.3. It may be observed that the contract deployment is consuming a large amount of gas, but this will be amortized over multiple agreements between user and provider. We have designed our contract to be used for multiple outsourcing tasks so that the user and provider can run multiple agreements on a single deployment. For implementation feasibility, we have modified our PBIOC contract in Figure 3.1, because the current blockchain networks do not support the scheduled function calls, which are executed when the timer expires. The Timer functionality in Figure 3.1, and Algorithm 3.6 is implemented as a Payout functionality. The results in the Table 3.4 shows that the cost of running the PBIOCV contracts even for small inputs is high when compared to Table 3.3. Hence, when both cloud user and provider are honest overhead of running a PBIOC contract is minimal.

The lack of possibility of running heavy cryptography operations on Ethereum blockchain limits the PBIOCV contract's implementation. Recently, a new library ZoKrates [128] using zkSNARKs[184] is introduced to perform heavy computations off-chain, and the proof of off-chain computations is verified on the Blockchain network. The user and provider run an off-chain setup phase to establish a common reference string (CRS) to derive a proving key and verification key. The $PBIOCV$ contract is generated by the user using verification key, and the user deploys the $PBIOCV$ contract and sends its address to the provider. The provider computes the witness and generates a proof. The provider then sends this proof to

Algorithm 3.5. We ran the experiments for three problems 1) Sorting integers 2) Primality test on non-prime numbers and 3) Searching in an array of integers. The costs of deploying and running the $PBIOCV$ contracts is shown in Table 3.4.

Function	**Caller**	**Cost in Gas**	**Cost in $**
Deployment	user	2117086	0.565
Create	user	302889	0.080
Intent	provider	92255	0.024
Commit	provider	85709	0.022
Agree	user	41905	0.011
Verify	provider	30275 +cost(PBIOCV)	0.008+cost(PBIOCV)
Payout	Anyone	37742	0.010

Table 3.3: Costs of interacting with PBIOC Contract. We have approximated the gas price as 1 Gwei and 1 ETH = $267.76, which are the real world costs in June 2019. We have rounded off the cost in $ value up to three decimals.

PBIOCV contract	**Deployment Cost in Gas**	**Execution Cost in Gas**
Sort(input size 10)	1281467	736975
Primality Test*	1005959	563902
Search(input size 10)	1311503	999455

Table 3.4: Deployment and Execution costs of $PBIOCV$ contracts. * The factors multiplication test is conducted only for non-prime number. The primality test for prime numbers is not possible with current Zokrates implementation as modulo operations are expensive on prime fields.

3.4.2 Implementation of TUIOC

3.4.2.1 Inner State Hash computation

The provider has to send the inner state hash and the algorithm's output to prove that he executed the prescribed algorithm given by the user. We have created an AspectJ aspect which hashes all the inputs and outputs of a method without modifying the actual java code. The user outsources the java code, which is compiled along with AspectJ aspect which, when executed, returns the algorithm's output along with inner state hash. We have computed the inner state hash using a Merkle tree [185]. The input string and output string of all the algorithm methods will become leaf nodes in the Merkle tree (i.e., two nodes for

Functions	**Caller**	**Cost in Gas**	**Cost in $**
Deployment	user	2526981	0.67
Create	user	64424	0.01
Intent	provider	64320	0.01
Commit	provider	118142	0.031
Reveal	provider	386452	0.10
Dispute	Anyone	40364	0.01
Payout	Anyone	39640	0.01

Table 3.5: Costs of interacting with TUIOC Contract. We have approximated the gas price as 1 Gwei and 1 ETH = $267.76 which are the real world costs in June 2019.

one method call). Once the execution is completed, a Merkle tree is constructed with all the leaf nodes. The root of the Merkle tree is returned as the inner state hash of an algorithm.

The providers will compute the commitments using Pedersen [186] commitments based on the public parameters given in Cryptocon [187]. We also use the Cryptocon contract for verifying the commitments submitted by the providers. We have invoked several instances of the TUIOC contract, and each transaction's computational and financial cost is listed in Table 3.5.

As we have discussed earlier, our TUIOC contract is simple and invokes TUIOCV contract only in case of disputes. We have modeled 1) Merge sort 2) Primality test and 3) Binary search as TUIOCV contracts and deployed them on Ethereum Blockchain. Merge sort contract takes an array of integers as an input and outputs sorted array of numbers. We have used a recursive merge sort implementation and hence we could not test it for large input sizes due to the limit on the number of call stacks of a smart contract in Ethereum Blockchain. The number of call stacks limit is hardcoded as 1024 stack frames which limit the number of function calls allowed. We also could not test for sorting fractional numbers as Ethereum does not support operations on floating-point numbers. The Merge sort contract is tested for input sizes from 5 to 100, where all the numbers in a single test are generated randomly, and the transaction costs are listed in Figure 3.4.

The second TUIOCV contract we have deployed will check whether the given number is prime or not. We have tested the primality contract for integers up to 10000. The transaction cost of prime numbers is listed in Figure 3.5 and transactional cost of non-prime numbers is given in Figure 3.6. The gas cost to test non-prime number is less than the gas cost for testing prime numbers. Also, the gas cost to test prime numbers increases as we move to higher numbers. Observe that the gas consumption of the non-prime number varies dramatically due to the adoption of Fermat's little theorem for primality test. To reduce the gas cost, we have included a condition "if $num\%2 == 0 \, || \, num\%3 == 0$, then return $false$" where num is the input to primality test. This statement consumes an almost constant amount of gas for all the non-prime numbers, which are having 2 or 3 as one factor. For all the remaining non-prime numbers we check the condition "$a^{num-1} \equiv 1$ mod (num)" for some random $1 < a < num - 1$. This condition is checked minimum once and a maximum of $\sqrt{num}$ times.

As a third TUIOCV contract, we have implemented and tested the binary search. The binary search contract is tested for input sizes from 100 to 1000 elements. The transactional costs for searching an element that does not exist in the input are shown in Figure 3.7.

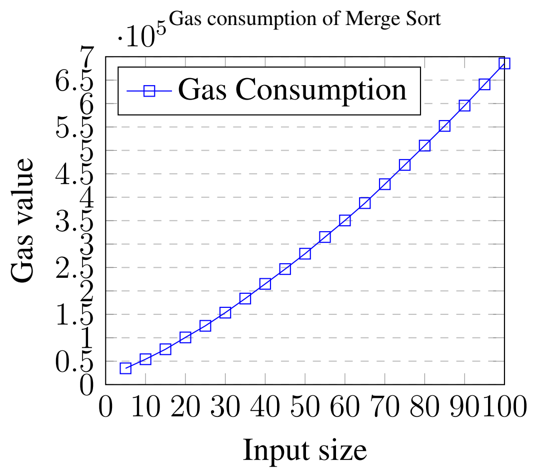

Figure 3.4: Gas consumption of Merge Sort

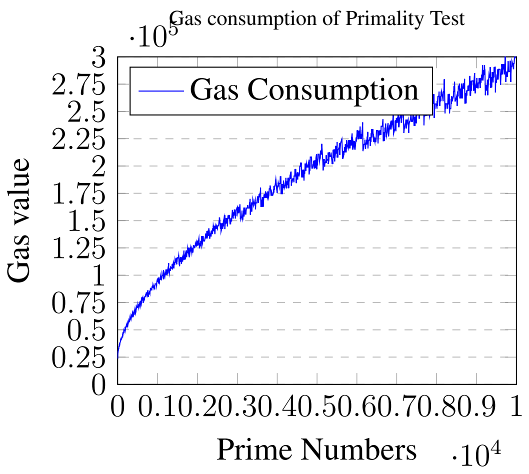

Figure 3.5: Gas Consumption of prime numbers

3.4.3 Implementation of MUIOC

We have deployed our MUIOC contract in the Ethereum Blockchain and tested it under different parameters. The first parameter we have varied is the number of providers. We have varied the number of providers from 10 to 90. The second parameter we have varied is the number of lazy providers who intend to compute the task. The percentage of lazy providers varies from 10% to 50%. The third parameter we have varied is the number

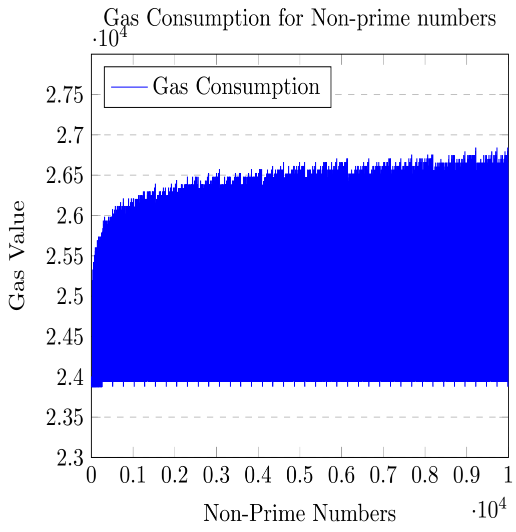

Figure 3.6: Gas Consumption of non-prime numbers

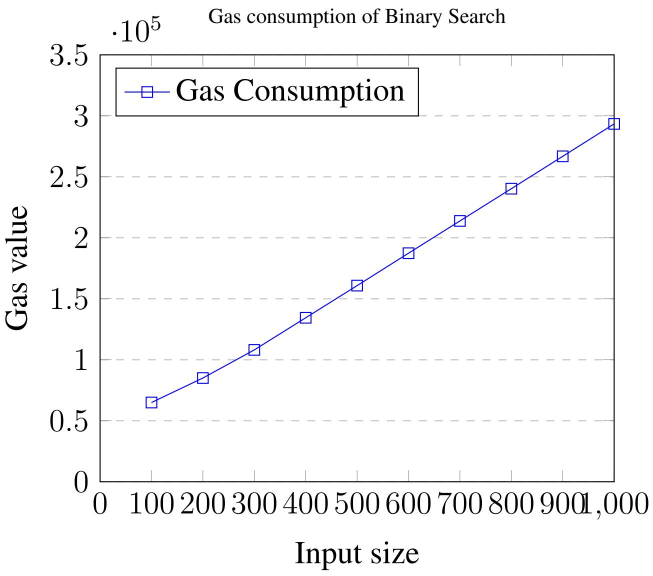

Figure 3.7: Gas Consumption of Binary Search

of providers selected per round. We have tested our contract for the number of providers selected per round from 3 to 5. In Table 3.6, we show the cost of transactions sent to the MUIOC contract. The timer functionality in Figure 3.3 is implemented as compute and payout functionalities. Private Ethereum Blockchain does not support a large number of accounts. We have simulated the contract in Java for n=100 to 1000 and showed the honest providers' pay in all the above scenarios in Figure 3.8.

Function			**Deployment**	**Create**	**Intent**	**Commit**			**Reveal**		
Cost in Gas			2908677	126179	150966	66576			33838		
Cost in $			0.77662	0.0337	0.04032	0.01778			0.00902		
Function			**Compute**	**Dispute**	**Payout**				**Compute**	**Dispute**	**payout**
n=10	k=3	gas	219779	165631	131866	n=60	k=3	gas	424140	2483879	660366
		$	0.05869	0.04422	0.03522			$	0.11323	0.6632	0.17633
	k=4	gas	265589	190749	132541		k=4	gas	471637	2522335	661041
		$	0.07092	0.05092	0.03538			$	0.12592	0.67345	0.17649
	k=5	gas	314773	220080	133216		k=5	gas	519135	2562952	661716
		$	0.08405	0.05877	0.03556			$	0.1386	0.68432	0.17667
n=20	k=3	gas	259300	373656	237566	n=70	k=3	gas	465350	3353794	766066
		$	0.06923	0.09978	0.06344			$	0.12426	0.89546	0.20455
	k=4	gas	306797	403904	238241		k=4	gas	512847	3378912	766741
		$	0.08192	0.10784	0.0636			$	0.13692	0.90217	0.20471
	k=5	gas	355153	421436	238916		k=5	gas	560345	3425172	767416
		$	0.09484	0.11251	0.06379			$	0.1496	0.91453	0.2049
n=30	k=3	gas	300510	700721	343266	n=80	k=3	gas	506560	4331463	871766
		$	0.08023	0.18709	0.09166			$	0.13526	1.15651	0.23277
	k=4	gas	348007	733021	343941		k=4	gas	554057	4381718	872441
		$	0.09292	0.19571	0.09182			$	0.14794	1.16991	0.23293
	k=5	gas	395505	755170	344616		k=5	gas	601555	4356671	873116
		$	0.1056	0.20164	0.09201			$	0.16063	1.16324	0.23312
n=40	k=3	gas	341720	1177093	448966	n=90	k=3	gas	547770	5409704	977466
		$	0.09123	0.31429	0.11988			$	0.14626	1.44439	0.26099
	k=4	gas	389217	1203750	449641		k=4	gas	595267	5440978	978141
		$	0.10392	0.32141	0.12004			$	0.15895	1.45275	0.26115
	k=5	gas	436715	1238211	450316		k=5	gas	643609	5497498	978816
		$	0.1166	0.3306	0.12023			$	0.17184	1.46783	0.26134
n=50	k=3	gas	382930	1770453	554666	n=100	k=3	gas	584859	6537458	1072596
		$	0.10223	0.47272	0.1481			$	0.15617	1.74551	0.28638
	k=4	gas	430427	1828916	555341		k=4	gas	632356	6613876	1073271
		$	0.11492	0.48832	0.14827			$	0.16885	1.76591	0.28657
	k=5	gas	478769	1801817	556016		k=5	gas	679854	6628843	1073946
		$	0.12784	0.48108	0.14845			$	0.18153	1.76989	0.28673

Table 3.6: Costs of running MUIOC Contract

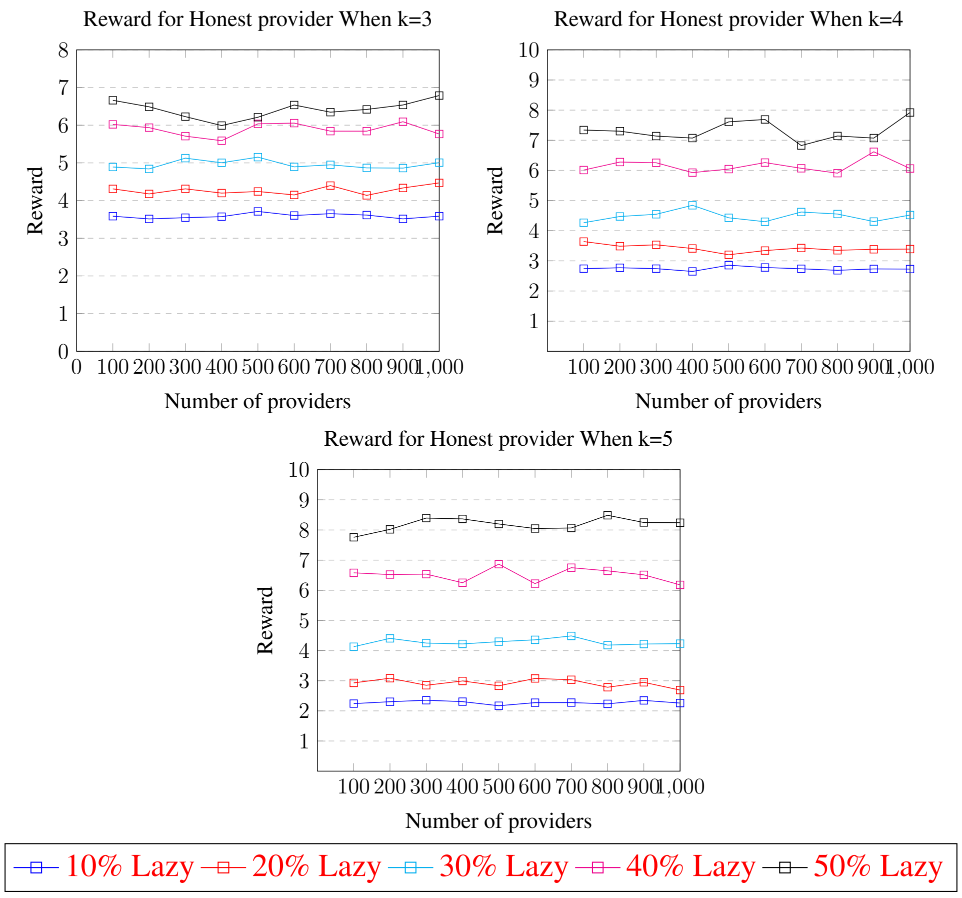

Figure 3.8: Reward for honest cloud provider in different scenarios. The reward for the computation is set as 10 Ethers, which is shared by all the honest providers.

3.5 Summary

Blockchains and smart contracts provide new solutions to the problems which are thought as hard to solve. One such issue is achieving fair payments for verifiable computations without a trusted intermediary. We have designed fair payment protocols using smart contracts for two types of verifiable computations: (1) Proof-based verifiable computation and (2) Replication-based verifiable computation. Our experiments show that in fair proof-based verifiable computation, the overhead of using a smart contract is minimal when both the user and the provider are honest. We have achieved fairness in replication-based verifiable computation by imposing fines on cheating providers and offering bounties to honest providers. We have shown that monetized penalties are an efficient way to deter cheating providers. We ran experiments for both types of verifiable computations and presented the transactional and financial costs of interacting with smart contracts. Our work could serve as a founding stone to future works that can design more robust, secure smart contracts for fair verifiable computations with fewer rounds of interactions. However, our protocols do not provide privacy to the output. One area of future work will mainly focus on the development of fair protocols for verifiable computations using smart contracts which will also provide privacy of the inputs and outputs of an outsourced problem.

Chapter 4

Fair Payment Protocols for Mobile Crowdsensing under Platform-as-a-Service

A typical mobile crowdsensing data marketplace consists of a cloud provider, an aggregation platform / aggregator to publish sensing tasks and many cloud users with mobile devices. The cloud provider sends sensing tasks to the platform, which then publishes the task. If a cloud user is interested in the task, he can participate in it, expecting incentives to his data contribution. However, the cloud provider does not know the cloud users in advance and may not be confident about the generated data by users due to differences in cloud users age groups, smart-phone / watch capabilities etc. In this case, aggregated statistics like sum, min, max, standard deviation, and variance about the aggregated data collected from all the cloud users help the cloud provider know the dataset's dispersion. The aggregation platform generates aggregated statistics. Nevertheless, the cloud users do not trust a cloud provider / aggregation platform and may not be willing to send the data for computing statistics without proper incentives. Motivated by this challenge in this Chapter, we design two protocols that compute aggregated statistics on private data. After knowing the statistics, if the cloud provider is interested in buying, the data is revealed only after being correctly paid to cloud users. The contributions of this Chapter are as follows:

(a) We propose two novel protocols to facilitate fair payments for monetizing mobile crowdsensing data. As a first protocol, we show a naive protocol requiring a trusted key dealer to establish encryption keys. As a second protocol, we design a more robust and trusted key dealer free system. We design a new key establishment protocol using a smart contract as a communication channel that does not require a trusted key dealer. We also design a new incentivization model that pays a cloud user based on the quality of the data and impose penalties on the cloud users who do not follow the intended protocol.

(b) We have implemented the proposed smart contracts using Solidity [24] and tested the smart contracts for the MotionSense dataset [188]. We have also presented the transactional and financial costs of interacting with smart contracts.

4.1 Privacy-preserving aggregation

Definition 4.1.1. *A privacy-preserving aggregation model consists of a set of three algorithms [189]:*

(a) $Setup(1^\lambda) \rightarrow (N, H, \{sk_{CU_i}\}_{CU_i \in CU}, sk_\mathcal{A})$*: It is run by a trusted key dealer* TP*. He chooses two safe primes* q_1 *and* q_2 *and computes* $N = q_1 * q_2$*. He also chooses a hash function* $H : \mathbb{Z} \rightarrow \mathbb{Z}^*_{N^2}$*.* TP *sets the public parameters as* $\mathcal{P}_{JL} = (N, H)$*, and distributes to each cloud user* $CU_i \in CU$ *a secret key* $sk_{CU_i} \in [0, N^2]$ *and sends* $sk_\mathcal{A} = -\sum_{i=1}^{n} sk_{CU_i}$ *to the untrusted aggregator* $\mathcal{A}$ *where* $n = |CU|$*.*

(b) $Encrypt(\mathcal{P}_{JL}, sk_{CU_i}, x_{CU_i,t}) \rightarrow C_{CU_i,t}$ *: At a time period* t*, each cloud user* CU_i *encrypts his private input* $x_{CU_i,t}$ *using the secret key* sk_{CU_i} *and outputs*

$$C_{CU_i,t} = (1 + x_{CU_i,t}N) \cdot H(t)^{sk_{CU_i}} \; mod \; N^2.$$

(c) $Aggregate(\mathcal{P}_{JL}, sk_\mathcal{A}, C_{CU_1,t}, ..., C_{CU_n,t}) \rightarrow sum_t = \sum_{i=1}^{n} x_{CU_i,t}$ *: Up on receiving*

$C_{CU_i,t}$ *from every cloud user, the untrusted aggregator computes*

$$P_t = \prod_{i=1}^{n} C_{CU_i,t} H(t)^{sk_A} \; mod \; N^2$$

$$sum_t = \frac{P_t - 1}{\dot{N}}$$

The correctness of the algorithms is as follows

$$\prod_{i=1}^{n} C_{CU_i,t} H(t)^{sk_A} \equiv \prod_{i=1}^{n} (1 + x_{CU_i,t} N)$$

$$\equiv 1 + (\sum_{i=1}^{n} x_{CU_i,t} \; mod \; N) N (mod \; N^2)$$

If $\sum_{i=1}^{n} x_{CU_i,t} < N$, *then* $sum_t = \frac{P_t - 1}{N} = \sum_{x=1}^{n} x_{CU_i,t}$. *One may refer to [189] for the proof of aggregator obliviousness.*

Definition 4.1.2. *A fair data aggregation protocol must provide the following guarantees:*

(a) ***Aggregator obliviousness****:The protocol is aggregator obliviousness if the aggregator learns nothing about cloud users' private data except the aggregated statistics.*

(b) ***Aggregator unforgeability*** *: The protocol is said to be aggregator unforgeable if aggregation operation solely depends on the inputs of the cloud users and no other party can influence the result of the aggregation operation.*

(c) ***Fair payments****: The protocol is said to be financially fair if (1) the cloud users receive the payments for their data contribution and (2) the cloud provider receives the data*

4.1.1 Entities

A Blockchain-based framework for privacy-preserving aggregation of mobile crowdsensing data has the following entities:

(a) **Mobile device** (MB_i): A mobile device consists of m sensors $M = (S_1, S_2, ..., S_m)$[1].

[1]Each sensor may produce more than one value. For example, an accelerometer produces x-axis, y-axis and z-axis values.

At each specific interval of time $[t_s, t_e]$, a mobile device MB_i produces a set of data points $Y_{CU_i,[t_s,t_e]}$.

$$Y_{CU_i,[t_s,t_e]} = \begin{array}{c} \\ t_s \\ \vdots \\ t_e \end{array} \begin{array}{c} \begin{array}{ccc} S_1 & S_2 \cdots\cdots & S_m \end{array} \\ \begin{bmatrix} y_{1,s} & y_{2,s} \cdots\cdots & y_{m,s} \\ \vdots & \vdots & \vdots \\ y_{1,e} & y_{2,e} \cdots\cdots & y_{m,e} \end{bmatrix} \end{array}$$

Each data point $y \in Y_{CU_i,[t_s,t_e]}$ is a sensor reading containing sensitive information of a cloud user. From now onwards we represent the time $[t_s, t_e]$ with t.

(b) **Cloud users** (CU_i): A cloud user collects data from the mobile device MB_i and generates a vector $X_{CU_i,t} = [\sum_{i=s}^{e} y_{1,i}, ..., \sum_{i=s}^{e} y_{m,i}]$[2]. To protect the confidentiality of every value in $X_{CU_i,t}$ against the eavesdroppers, CU_i encrypts every value in $X_{CU_i,t}$ using a secret key sk_{CU_i} and sends the resulting cipher text vector $C_{CU_i,t}$ to the smart contract[3].

(c) **Cloud provider** (CP): A cloud provider CP creates and deploys the smart contract on a public Blockchain network. It will distribute an amount of $\$p$ to all the cloud users if the data delivered generates desired aggregated statistics.

(d) **Smart contract**: A smart contract acts as an aggregation platform, which computes the aggregated statistics on receipt of $C_{CU_i,t}$ from every cloud user.

(e) **Blockchain Network**(BC): A Blockchain network is maintained by a set of peers known as miners who execute the smart contract according to an underlying consensus algorithm.

(f) **Inter-planetary file system** ($IPFS$): As storing large amounts of data in smart contract incurs a high financial cost, we use a distributed storage network $IPFS$ to store $Y_{CU_i,t}$, and $X_{CU_i,t}$ is sent to the smart contract for computing aggregated statistics. The $IPFS$ network returns the hash of the data stored as a URL.

[2]In this Chapter, we consider sum as the aggregation operation, and hence the cloud user need not send the entire data points.

[3]Even though $X_{CU_i,t}$ consists of accumulated values, it can still reveal some private information about the cloud user, and hence encryption is required.

4.2 A naive trusted party based fair payment protocol for privacy-preserving aggregation of mobile crowdsensing data ($FairNaivePPA$)

In this section, we design a naive protocol to facilitate fair payments for privacy-preserving aggregation of mobile crowdsensing data where the aggregator services are modeled as a smart contract deployed on a public Blockchain network.

4.2.1 $FairNaivePPA$ contract clauses

The $FairNaivePPA$ is a crowdsourcing contract signed between a cloud provider CP and a set of cloud users CU. The high-level idea is that if CP and every $CU_i \in CU$ are honest, then CP will get the sensing data and every $CU_i \in CU$ gets a pre-agreed payment.

The clauses in the $FairNaivePPA$ contract are as follows:

(i) A cloud provider CP creates a smart contract $FairNaivePPA$ for monetizing mobile sensing data where the $aggregate$ algorithm from Definition 4.1.1 is modeled as one of the contract's functionalities. CP deploys the smart contract on a public Blockchain network and publishes the contract address on a public platform (like a website / bulletin board).

(ii) All parties agree on timing parameters $\tau_i < \tau_c < \tau_a < \tau_b < \tau_r$ and a pay $\$p$, distributed to cloud users for their data contribution.

(iii) As a condition, a cloud user CU_i who wishes to participate in the sensing task must pay a deposit of $\$d$ before τ_i. The safe deposit is required to ensure the cloud user participation until the end of the protocol. Let $list_{CU}$ be the set of users who have shown intent to contribute data. If $list_{CU} = 0$, then $\$p$ is refunded, and the contract is terminated.

(iv) After $\tau > \tau_i$, a trusted key dealer TP generates a set $\{sk_{CU_1}$,..., $sk_{CU_{|CU|}}$, $sk_{CP}\}$ of $|CU|$+1 keys such that $\sum_{i=1}^{|CU|} sk_{CU_i} + sk_{CP} = 0$. TP sends a key sk_{CU_i} to every $CU_i \in CU$ and also sk_{CP} to CP through a secure channel.

(v) Every $CU_i \in CU$ generates data $Y_{CU_i,t}$ containing m attributes according to the CP's specification. The data is encrypted with a self-generated symmetric key, and the encrypted data is stored at $IPFS$. The data is also aggregated, and the aggregated data is encrypted with the key received from TP. Encrypted aggregated data and $IPFS$ URL are sent to the contract before $\tau < \tau_c$. If any $CU_i \in CU$ fails to send the encrypted data before τ_c, then the CU_i's deposit is forfeited. The forfeited deposit is distributed to all other cloud users equally along with their deposits. $\$p$ is refunded to CP, and the contract is terminated. If no CU_i has sent the encrypted data, all deposits and $\$p$ are sent to CP, and the contract is terminated.

(vi) After $\tau > \tau_c$, CP sends the key sk_{CP} to the smart contract. The contract performs the aggregation operation on the encrypted data and generates aggregated statistics. If CP fails to send the key before τ_a, $\$p$ is distributed to every $CU_i \in CU$ along with their deposits, and the contract is terminated.

(vii) After learning the aggregated statistics of the data, if CP is willing to buy the data, it should send its willingness to the contract before τ_b. Otherwise, $\$p$ is refunded to CP, deposits are refunded to cloud users, and the contract is terminated.

(viii) If CP is willing to buy the data, then every $CU_i \in CU$ should reveal the sk_{CU_i} before τ_r. The contract will distribute $\$p$ to every CU_i along with their deposit $\$d$. If any CU_i fails to reveal the key[a] before τ_r, then their deposit is forfeited. The deposits and the pay shares of forfeited cloud users are sent to CP. CP after learning sk_{CU_i} can decrypt the encrypted url and obtain the encrypted data. The encrypted data can be decrypted with the respective self-generated symmetric key $K_{CU_i,CP}$.

[a]As shown in Figure 4.1 the actual key is not revealed. But, to simplify the high-level description, we assume that the key is revealed.

4.2.2 $FairNaivePPA$ Protocol

$FairNaivePPA$ protocol is presented in Figure 4.1. $FairNaivePPA$ smart contract functionalities executed by Blockchain are presented as Algorithms 4.1 to 4.6. CP publishes the details of the sensing task and timing parameters τ_i, τ_c, τ_a, τ_b, τ_r at a public platform. It sends all these parameters along with $\$p$ to BC invoking Algorithm 4.1. Algorithm 4.1 stores the provider's pay and sets the state as *Intent*. Interested users show intent by sending deposits to BC invoking Algorithm 4.2. Algorithm 4.2 stores the deposit sent by a user and adds the user to a list and then increments the user count. The intended users have to generate the data according to the given specifications, and the aggregated data has to be sent to BC invoking Algorithm 4.3. Algorithm 4.3 stores the commitment sent by a cloud user. Depending on the behavior of the users, there are three cases as follows:

Case 1: All the intended users have committed the data. In this case, the provider has to send the secret key to initiate the computation of aggregated statistics. Depending on the behavior of the provider, there are two cases as follows:

Case 1.1: The provider has sent the secret key invoking Algorithm 4.4. In this case, the Algorithm 4.4 computes the aggregated statistics. Depending on the behavior of the provider, there are two cases as follows:

Case 1.1.1: After learning the aggregated statistics, the provider is willing to buy. In this case, the provider will send a transaction invoking Algorithm 4.5 which sets the state as *Reveal*. The users have to reveal the data to get the payment and their deposits by invoking Algorithm 4.6. After revealing the data by an user, Algorithm 4.6 add that user to a list of honest users and sets the state as *Buy*. The deposits and the aborted users' pay shares are sent to the provider, and the contract is terminated. At the end of the protocol, CP retrieves the dataset $Y_{CU_i,t}$ of every cloud user in the following way:

After obtaining CK_{CU_i} from every $CU_i \in CU$, CP computes

$$sk_{CU_i} = DEC_{K_{CU_i,CP}}(CK_{CU_i})$$

$$url = DEC_{sk_{CU_i}}(C_{url})$$

obtains $CY_{CU_i,t}$ from $IPFS$ network using url

$$\bar{Y}_{CU_i,t} = DEC_{sk_{CU_i}}(CY_{CU_i,t})$$

$$Y_{CU_i,t} = DEC_{K_{CU_i,CP}}(\bar{Y}_{CU_i,t})$$

Case 1.1.2: After learning the aggregated statistics, the provider is not willing to buy. In this case, the payment is refunded to the provider. The deposits of the users are also refunded, and the contract is terminated.

Case 1.2 The provider has failed to send the secret key. In this case, the pay is equally distributed to the users along with their deposits, and the contract is terminated.

Case 2: Only a subset of the intended users have committed the data. In this case, the aborted users' deposits are equally shared among the committed users, along with their deposits. Pay is refunded to the provider, and the contract is terminated.

Case 3: None of the intended users has committed the data. All the deposits of the users and the pay are sent to the provider, and the contract is terminated.

FairnaivePPA protocol

For trusted key dealer TP

1. To generate and send encryption keys
   (a) Choose two safe primes q_1 and q_2 and compute $N = q_1 * q_2$.
   (b) Choose a hash function $H : \mathbb{Z} \rightarrow \mathbb{Z}^*_{N^2}$ and set the public parameters as $param = (N, H, \mathbb{G}_q, g)$ where g is a generator of group $\mathbb{G}_q$ with prime order q.
   (c) Send a secret key $sk_{CU_i} \in [0, N^2]$ to every user $CU_i \in CU$ through a secure channel.
   (d) Send sk_{CP} to the cloud provider CP through a secure channel such that $\sum_{i=1}^{|CU|} sk_{CU_i} + sk_{CP} = 0$.

For a cloud provider CP

2. To create a sensing task
   (a) Choose a random number $r_{CP} \in \mathbb{Z}_q$ and compute $pk_{CP} = g^{r_{CP}}$
   (b) Send **trans**$^{CP}_{create}$ = (pk_{CP}, $\$pay$, τ_i, τ_c, τ_a, τ_b, τ_r) to BC.
3. To initiate the computation of aggregation statistics, send **trans**$^{CP}_{aggregate}$ = (sk_{CP}) to BC
4. To buy the data send **trans**$^{CP}_{buy}$ to BC

For a cloud user CU

5. To participate in the sensing task
   (a) Choose a random number $r_{CU_i} \in \mathbb{Z}_q$ and compute a $pk_{CU_i} = g^{r_{CU_i}}$.

```
        (b) Send $\textbf{trans}^{CU_i}_{intent}$ = ($pk_{CU_i}$, \$$d$) to $BC$.
    6. To send the encrypted data
        (a) Encrypt $X_{CU_i,t} = [\sum_{i=s}^{e} y_{1,i}, ..., \sum_{i=s}^{e} y_{m,i}]$ as
            $C_{CU_i,t} = [(1+\sum_{i=s}^{e} y_{1,i} * N) \cdot H(t)^{sk_{CU_i}} \bmod N^2, ..., (1+\sum_{i=s}^{e} y_{m,i} * N) \cdot H(t)^{sk_{CU_i}} \bmod N^2]$
        (b) Encrypt $Y_{CU_i,t}$ with shared symmetric key $K_{CU_i,CP} = pk_{CP}^{r_{CU_i}} = pk_{CU_i}^{r_{CP}} = g^{r_{CU_i} r_{CP}}$ such that
            $CY_{CU_i,t} = Enc_{sk_{CU_i}}(Enc_{K_{CU_i,CP}}(Y_{CU_i,t}))$ and send $CY_{CU_i,t}$ to $IPFS$ and obtain $url$.
        (c) Encrypt $url$ with $sk_{CU_i}$ such that $C_{url} = Enc_{sk_{CU_i}}(url)$. Compute $CK_{CU_i} = Enc_{K_{CU_i,CP}}(sk_{CU_i})$
            and $ct_{CU_i} = comm(CK_{CU_i}, s)$ where $s$ is randomly chosen.
        (d) Send $\textbf{trans}^{CU_i}_{commit}$=($C_{CU_i,t}$, $C_{url}$, $ct_{CU_i}$) to $BC$.
    7. To reveal the data send $\textbf{trans}^{CU_i}_{reveal}$=($CK_{CU_i}$, $s$) to $BC$
For Blockchain $BC$: set $state \leftarrow Init$, $list_{CU} \leftarrow \{\}$, $list_{co} \leftarrow \{\}$, $list_{ho} \leftarrow \{\}$, $n \leftarrow 0$, \$$p \leftarrow 0$,
\$$deposit \leftarrow 0$
    8. On receiving $\textbf{trans}^{CP}_{create}$ execute $FairNaivePPA.cretae(pk_{CP}, \$pay, \tau_i, \tau_c, \tau_a, \tau_b, \tau_r)$
    9. On receiving $\textbf{trans}^{CU_i}_{intent}$ execute $FairNaivePPA.intent(pk_{CU_i}, \$d)$
    10. On receiving $\textbf{trans}^{CU_i}_{commit}$ execute $FairNaivePPA.commit(C_{CU_i,t}, C_{url}, ct_{CU_i})$
    11. On receiving $\textbf{trans}^{CP}_{aggregate}$ execute $FairNaivePPA.aggregate(sk_{CP})$
    12. On receiving $\textbf{trans}^{CP}_{buy}$ execute $FairNaivePPA.buy()$
    13. On receiving $\textbf{trans}^{CU_i}_{reveal}$ execute $FairNaivePPA.reveal(CK_{CU_i}, s)$
Timer
    If $\tau > \tau_i$ and $|list_{cu}| = 0$ and $state = Intent$ then
        set $ledger[CP] \leftarrow ledger[CP] + \$p$ and $state \leftarrow Aborted$
    If $\tau > \tau_i$ and $|list_{cu}| \neq 0$ then set $state \leftarrow Commit$
    If $\tau > \tau_c$ and $|list_{co}| = 0$ then
        set $ledger[CP] \leftarrow ledger[CP] + \$p + \$deposit$ and $state \leftarrow Aborted$
    If $\tau > \tau_c$ and $|list_{co}| \neq |list_{cu}|$ then
        set $ledger[CP] \leftarrow ledger[CP] + \$p$
        $\forall\, CU_i \in list_{co}$ set $ledger[CU_i] \leftarrow ledger[CU_i] + \frac{\$deposit}{|list_{co}|}$
        set $state \leftarrow Aborted$
    If $\tau > \tau_c$ and $|list_{co}| = |list_{cu}|$ then set $state \leftarrow Aggregate$
    If $\tau > \tau_a$ and $state = Aggregate$ then
        $\forall CU_i \in list_{cu}$ set $ledger[CU_i] \leftarrow ledger[CU_i] + \frac{\$p}{|list_{cu}|} + \frac{\$deposit}{|list_{cu}|}$
        set $state \leftarrow Aborted$
    If $\tau > \tau_b$ and $state = Buy$ then
        set $ledger[CP] \leftarrow ledger[CP] + \$p$
        $\forall CU_i \in list_{cu}$ set $ledger[CU_i] \leftarrow ledger[CU_i] + \frac{\$deposit}{|list_{cu}|}$
        set $state \leftarrow Aborted$
    If $\tau > \tau_r$ and $state = Reveal$ and $|list_{ho}| = 0$ then
        set $ledger[CP] \leftarrow ledger[CP] + \$p + \$deposit$ and $state \leftarrow Aborted$
    If $\tau > \tau_r$ and $state = Reveal$ and $|list_{ho}| \neq 0$ then
        $\forall CU_i \in list_{ho}$ set $ledger[CU_i] \leftarrow ledger[CU_i] + \frac{\$p}{|list_{cu}|} + \frac{\$deposit}{|list_{cu}|}$
        set $ledger[CP] \leftarrow ledger[CP] + \frac{\$p}{|list_{cu}|} * (|list_{cu}| - |list_{ho}|) + \frac{\$deposit}{|list_{cu}|} *$
        $(|list_{cu}| - |list_{ho}|)$
        set $state \leftarrow Terminated$
```

Figure 4.1: $FairNaivePPA$ protocol

Algorithm 4.1 $FairNaivePPA.create$

```
Input: $pk_{CP}$, $\$pay$, $\tau_i$, $\tau_c$, $\tau_a$, $\tau_b$, $\tau_r$
Output: Success or Failure message
1: if $state = Init$ then
2:     if $\tau < \tau_i < \tau_c < \tau_a < \tau_b < \tau_r$ then
3:         if $ledger[CP] \geq \$pay$ then
4:             set $ledger[CP] \leftarrow ledger[CP] - (\$pay)$;
5:             set $\$p \leftarrow \$pay$;
6:             set $state \leftarrow Intent$;
7:             return (Success, Task created)
8:         else
9:             return (Failure, Balance is low)
10:    else
11:        return (Failure, Bad timing parameters)
12: else
13:    return (Failure, State is not $Init$)
```

Algorithm 4.2 $FairNaivePPA.intent$

```
Input: $pk_{CU_i}$, $\$d$
Output: Success or Failure message
1: if $state = Intent$ then
2:     if $\tau < \tau_i$ then
3:         if $ledger[CU_i] \geq \$d$ then
4:             if $(CU_i, *) \notin CU$ then
5:                 set $ledger[CU_i] \leftarrow ledger[CU_i] - \$d$
6:                 set $\$deposit \leftarrow \$deposit + \$d$
7:                 set $list_{CU} \leftarrow list_{CU} \cup (CU_i, pk_{CU_i})$
8:                 set $n \leftarrow n + 1$
9:                 return (Success, Intent success)
10:            else
11:                return (Failure, Duplicate user)
12:        else
13:            return (Failure, Balance is low)
14:    else
15:        return (Failure, Intent timeout)
16: else
17:    return (Failure, State is not $Intent$)
```

Algorithm 4.3 $FairNaivePPA.commit$

```
    Input: $C_{CU_i,t}, C_{url}, ct_{CU_i}$
    Output: Success or Failure message
 1: if $state = Commit$ then
 2:     if $\tau < \tau_c$ then
 3:         if $(CU_i, *) \in list_{CU}$ then
 4:             if $(CU_i, *, *, *) \notin commitments$ then
 5:                 set $commitments \leftarrow commitments \cup (CU_i, C_{CU_i,t}, C_{url}, ct_{CU_i})$
 6:                 return (Success, Commit success)
 7:             else
 8:                 return (Failure, Duplicate commitment)
 9:         else
10:             return (Failure, Wrong user)
11:     else
12:         return (Failure, Commit timeout)
13: else
14:     return (Failure, State is not $Commit$)
```

Algorithm 4.4 $FairNaivePPA.aggregate$

```
    Input: $sk_{CP}$
    Output: Success or Failure message
 1: if $state = Aggregate$ then
 2:     if $\tau < \tau_a$ then
 3:         set $P_{j,t} \leftarrow \pi_{i=1}^{list_{CU}} C_{CU_i,t} H(t)^{sk_{CP}}$ mod $N^2$  $\forall j \leftarrow 1$ to $m$
 4:         set $sum_{j,t} \leftarrow \frac{P_{j,t}-1}{N}$  $\forall j \leftarrow 1$ to $m$
 5:         set $state \leftarrow Buy$
 6:         return (Success, Aggregate success)
 7:     else
 8:         return (Failure, Aggregate timeout)
 9: else
10:     return (Failure, State is not $Aggregate$)
```

Algorithm 4.5 $FairNaivePPA.buy$

```
    Input: $\phi$
    Output: Success or Failure message
 1: if $state = Buy$ then
 2:     if $\tau < \tau_b$ then
 3:         set $state \leftarrow Reveal$
 4:         return (Success, Buy success)
 5:     else
 6:         return (Failure, Buy timeout)
 7: else
 8:     return (Failure, State is not $Buy$)
```

Algorithm 4.6 $FairNaivePPA.Reveal$

```
Input: $CK_{CU_i}$, $s$
Output: Success or Failure message
1: if $state = Reveal$ then
2:     if $\tau < \tau_r$ then
3:         if $ct_{CU_i} = comm(CK_{CU_i}, s)$ then
4:             set $honest \leftarrow honest \cup CU_i$
5:             set $state \leftarrow Buy$
6:             return (Success, Reveal success)
7:         else
8:             return (Failure, Wrong commitment)
9:     else
10:        return (Failure, Reveal timeout)
11: else
12:     return (Failure, State is not $Reveal$)
```

4.2.3 Limitations of $FairNaivePPA$

(a) **Trusted party** TP: We have assumed a trusted party to generate and distribute the secret keys securely. However, using a TP comes with a cost, and TP may not guarantee to behave honestly every time (key escrow problem). Distributed key generation techniques are used to eliminate TP for a key generation. However, these techniques involve generating a single private key that can be computed from shares of all the cloud users increasing the communication cost. Recently, Schindler *et al.* [190] establish a secret key using a smart contract as a communication channel. In the next section, we employ a lighter key establishment protocol similar to [190] to establish secret keys using smart contracts.

(b) **All or nobody**: We have assumed that either all the cloud users or no cloud user who have shown intent to share data will call the commit functionality. In the real world, a malicious cloud user may abort the protocol after showing intent or a cloud user may experience communication problems with the smart contract and may not call the commit functionality. The protocol fails even if one of the cloud users refuses to commit after showing intent. Although the contract imposes monetary penalties on aborting, a malicious cloud user who aborts intentionally causes the entire protocol to be aborted.

(c) **Quality of the data**: The incentive model in $FairNaivePPA$ is static, i.e., if all the cloud users reveal their secret key, then every cloud user will get equal pay. In this model, there is no choice for the cloud provider to pay according to the quality of data. The incentive model will be more meaningful if the cloud provider can pay according to some payment mechanism based on the data quality.

(d) **Robustness**: After intent phase, the number of cloud users participating in the protocol is fixed. The smart contract will compute aggregated statistics correctly only if all the cloud users commit their data. In the next section, we show that the smart contract computes aggregated statistics even if some cloud users abort the protocol prematurely.

(e) **Copy and Paste Attack**: As communications with the smart contract are not through secure channels, protocols interacting with public Blockchain networks suffer from inherent copy and paste attack. A lazy cloud user CU_l will listen to the interactions between an honest cloud user CU_i and smart contract. Then, CU_l copies CU_i's interactions and submit it to the smart contract without actually performing the sensing task. One way to avoid this attack is by using the commit and reveal technique where the commitment of the key is generated using a well-known commitment method like Pedersen commitment [186]. If two cloud users submit the same commitment, then smart contract rejects the second commitment and asks to recompute the commitment with different parameters. In the reveal phase, the commitment is revealed along with the parameters used to generate the commitment. For the sake of simplicity, we are not considering this attack in our protocol.

4.3 A trusted party free fair payment protocol for privacy-preserving aggregation of mobile crowdsensing data ($FairPPA$)

In this section, our first objective is to eliminate the trusted party TP for key generation. We propose a new smart-contract based key generation algorithm in which every cloud user generates encryption keys non-interactively without using the TP. The second objective is adding robustness. Even if some cloud users abort during the protocol execution, the smart contract should compute the correct aggregated statistics. The cloud users are allowed to abort during different phases, and still, our smart contract can compute the aggregated result. However, this increases the number of interactions between cloud users and the smart contract. The third objective is to design a payment mechanism that will pay according to the quality of data produced by the cloud users. In this protocol, we use a truth-finding algorithm and compute weights according to the ground truth and pay a cloud user based on the computed weight. The fourth objective is to introduce fairness through a robust dispute resolution mechanism.

4.3.1 Smart contract based key generation

We use the smart contract as a communication channel for generating keys non-interactively without a TP. The algorithm for key generation is presented in Algorithm 4.7. The algorithm takes the list of cloud users $CU_i \in list_{do}$ obtained from the $FairPPA$ contract and two values r_1, r_2 given by CP during task creation. These values will change for every new sensing task. The algorithm also takes l as input: the expected number of cloud users to be selected as buddies during the key generation. The algorithm outputs a list of buddies and a list of keys established with each buddy. Every $CU_i \in list_{do}$ computes the encryption key as a sum of all the keys established with every buddy. Clearly, if CU_i selected CU_j as a buddy, then $dkey_{CU_i,CU_j} - dkey_{CU_j,CU_i} = 0$. Both of them will add the same value to their encryption key such that one of them adds positive value, and the other will add a negative value. This property cancels out the encryption keys during the aggregation operation. If any $CU_i \in list_{do}$ aborts after showing intent, then a new list of buddies and new encryption keys must be generated according to the new $list_{do}$.

Algorithm 4.7 $FairPPA.keygen$

```
    Input: $list_{cu}, r_1, r_2, l$
    Output: $ind_{CU_i}$,$sk_{CU_i}$
1: foreach $CU_j \in list_{do} - CU_i$ do
2:     $K_{CU_i,CU_j} \leftarrow (pk_{CU_j})^{r_{CU_i}} = (pk_{CU_i})^{r_{CU_j}} = g^{r_{CU_i} r_{CU_j}}$
3:     if $PRF(K_{CU_i,CU_j}, r_1) \leq \frac{l}{|list_{do}|-1}$ then
4:         add $CU_j$ to $ind_{CU_i}$
5: foreach $CU_j \in ind_{CU_i}$ do
6:     $dkey_{CU_i,CU_j} \leftarrow \frac{i-j}{|i-j|} * PRF(K_{CU_i,CU_j}, r_2)$
7:     $sk_{CU_i} \leftarrow sk_{CU_i} + dkey_{CU_i,CU_j}$
```

4.3.2 Truth Discovery Algorithm (TDA)

As the quality of different cloud users' data typically varies, it is sensible to pay more to the users who sense the quality data. However, the data quality is unknown a priori. Therefore, truth discovery algorithm (TDA) is used to find weights and estimate the ground truths. Many TDA algorithms [191, 192, 193] are presented in the literature, and their common procedure is summarized in Algorithm 4.8. The algorithm starts with a random ground-truth value and has mainly two steps: weight calculation and truth estimation. During weight calculation, the ground truth is assumed to be constant, and the weight w_{CU_i} of each user $CU_i \in list_{ho}$ is calculated as

$$w_{CU_i} = \omega \left(\sum_{m \in (s_1,\ldots,s_n)} d(x_m^{CU_i}, x_m^*) \right) \tag{4.1}$$

Where $d(\cdot)$ denotes the function that computes the distance between the user's data $x_m^{CU_i}$, and the estimated ground truth x_m^*. $\omega(\cdot)$ is some monotonically decreasing function. The $d(\cdot)$ and $w(\cdot)$ functions vary with respect to different TDAs. In truth estimation step, the users' weights are assumed to be fixed, and the estimated ground truth x_m^* of each sensor $m \in (s_1, \ldots, s_m)$ is derived as

$$x_m^* = \frac{\sum_{CU_i \in list_{ho}} (w_{CU_i} * x_m^{CU_i})}{\sum_{CU_i \in list_{ho}} w_{CU_i}} \tag{4.2}$$

Clearly, the ground truth x_m^* relies more on the user values with higher weights. The converge criteria is application-specific.

Algorithm 4.8 $FairPPA.TDA$

```
    **Input:** Cloud users' data $\{X_m^{CU_i} | m \in (s_1, ..., s_m), CU_i \in list_{ho}\}$
    **Output:** Estimated ground truth $\{x_m^* | m \in (s_1, ..., s_m)\}$, weights $W = \{w_{CU_i} | CU_i \in list_{ho}\}$, $\pi$
1:  Randomly initialize the ground truth for a sensing task
2:  **repeat**
3:      **for** $CU_i \in list_{ho}$ **do**
4:          update the weight $w_{CU_i}$ based on the current estimated ground truths using Equation 4.1
5:      **for** $m \in (s_1, ..., s_m)$ **do**
6:          Update the estimated ground truth $x_m^*$ based on cloud users' current weights using Equation 4.2
7:  **until** Convergence criterion is satisfied
```

We do not focus on designing new $d(\cdot)$ and $\omega(\cdot)$ functions; instead, we use simple existing functions and design a payment mechanism based on the weights obtained from the TDA. However, directly modeling Algorithm 4.8 as a smart contract is not feasible due to costs involved and may lead to verifier's dilemma [97]. In our protocol, the cloud provider computes the weights and ground truth off-line and sends the weights to the smart contract along with proof of correctness (π) of executing TDA. We use the inner state hash (ISH) [129] to compute the proof-of-correctness. Assuming that the TDA algorithm is composed of a finite number of atomic operations and each atomic operation takes some state information as input and produces another state information as output. The inner state of an algorithm is defined as the concatenation of all the input and output states of the atomic operations of an algorithm, and the definition of the algorithm in terms of atomic operations. An l-bit hash function takes an inner state of an algorithm as input and maps it into an l-bit random string called as inner state hash. ISH helps in detecting the cheating behavior accurately as the slightest deviation from the correct computation can be detected accurately.

4.3.3 Payment Mechanism

The payment mechanism described in Algorithm 4.9 takes the weights calculated in Algorithm 4.8 as input. It produces the amount of pay to be received by each cloud user who followed the protocol honestly.

Algorithm 4.9 $FairPPA.payment$

```
    Input: Weights $W = \{w_{CU_i} | CU_i \in list_{ho}\}$, pay $\$p$
    Output: payment for each cloud user $\{\$p_{CU_i} | CU_i \in list_{ho}$
1: foreach $CU_i \in list_{ho}$ do
2:     $\bar{p}_{CU_i} \leftarrow \left[ \left( \frac{w_{CU_i}}{\sum_{i=1}^{|list_{ho}|} w_{CU_i}} \right) * 100 \right]$
3:     $\$p_{CU_i} \leftarrow \left[ \bar{p}_{CU_i} * \left( \frac{\$p}{\sum_{i=1}^{|list_{ho}|} \bar{p}_{CU_i}} \right) \right]$
```

4.3.4 Dispute Resolution Mechanism (DRM)

Dispute resolution mechanism is discussed in Algorithm 4.10 and Algorithm 4.11. It is executed when a CP miscalculates weights, and a cloud user challenges these incorrect weights. To resolve the dispute, we adopt a byzantine voting mechanism such that when a dispute is raised, every $CU_k \in list_{dv}$ runs Algorithm 4.8 locally and returns results to $FairPPA$ contract. Algorithm 4.10 compares the weights sent by data verifiers with the weights sent by both cloud provider and challenger. Algorithm 4.10 rewards or penalizes cloud users, data verifiers and cloud provider according to the results of the comparison. Algorithm 4.11 is invoked when there is no consensus among the data verifiers. Algorithm 4.11 computes the weights and compares the weights with weight returned by verifiers, provider and challenger. It rewards or penalizes the users, verifiers, provider and challenger according to the comparison results. If both the cloud provider and the challenger are malicious, $\$f_{CP}, \$f_C$ are added to $corpus$. As executing TDA using smart contracts is costly, the $corpus$ fund is used to compensate the party calling TDA smart contract functionality.

Algorithm 4.10 $FairPPA.DRM$

```
    Input: $W^{CP}$ and $\pi^{CP}$, $W^{\mathcal{C}}$ and $\pi^{\mathcal{C}}$, $\forall CU_k \in list_{hv}\ (W^{CU_k}, \pi^{CU_k})$, $minDv$, $th$, $\$p$, $\$deposit_{do}$, $\$deposit_{dv}$, $\$f_{CP}$, $\$f_{\mathcal{C}}$, $list_{ho}$, $list_{ma}$, $list_{hv}$
    Output: $\phi$
 1: if $|list_{hv}| > minDv$ then
 2:     foreach $CU_k \in list_{hv}$ do
 3:         if $W^{CU_k} = W^{CP} \wedge \pi^{CU_k} = \pi^{CP}$ then
 4:             $v_{CP} \leftarrow v_{CP} + 1$
 5:         else
 6:             if $W^{CU_k} = W^{\mathcal{C}} \wedge \pi^{CU_k} = \pi^{\mathcal{C}}$ then
 7:                 $v_{\mathcal{C}} \leftarrow v_{\mathcal{C}} + 1$
 8:     if $v_{CP} \geq th$ then
 9:         $\forall CU_i \in list_{ho}$ set $\$p_{CU_i} \leftarrow PM(W^{CP}, \$p)$
10:         $\forall CU_i \in list_{ho}$ set $ledger[CU_i] \leftarrow ledger[CU_i] + \frac{\$deposit_{do}}{|list_{ho}|+|list_{ma}|} + \$p_{CU_i}$
11:         $\forall CU_k \in list_{hv}$ set $ledger[CU_k] \leftarrow ledger[CU_k] + \frac{\$deposit_{dv}}{|list_{hv}|} + \frac{\$f_{\mathcal{C}}}{|list_{hv}|}$
12:         Set $ledger[CP] \leftarrow ledger[CP] + \frac{\$deposit_{do}}{|list_{ho}|+|list_{ma}|} * (|list_{ma}|) + \$f_{CP}$
13:     else
14:         if $v_{\mathcal{C}} \geq th$ then
15:             $\forall CU_i \in list_{ho}$ set $\$p_{CU_i} \leftarrow PM(W^{\mathcal{C}}, \$p)$
16:             $\forall CU_i \in list_{ho}$ set $ledger[CU_i] \leftarrow ledger[CU_i] + \frac{\$deposit_{do}}{|list_{ho}|+|list_{ma}|} + \$p_{CU_i}$
17:             $\forall CU_k \in list_{hv}$ set $ledger[CU_k] \leftarrow ledger[CU_k] + \frac{\$deposit_{dv}}{|list_{hv}|} + \frac{\$f_{CP}}{|list_{hv}|}$
18:             Set $ledger[CP] \leftarrow ledger[CP] + \frac{\$deposit_{do}}{|list_{ho}|+|list_{ma}|} * (|list_{ma}|)$
19:             Set $ledger[\mathcal{C}] \leftarrow ledger[\mathcal{C}] + \$f_{\mathcal{C}}$
20:         else
21:             Run $FairPPA.DRM2(W^{CP}, \pi^{CP}, W^{\mathcal{C}}, \pi^{\mathcal{C}}, \forall CU_k \in list_{hv}\ (W^{CU_k}, \pi^{CU_k}))$
22: else
23:     Run $FairPPA.DRM2(W^{CP}, \pi^{CP}, W^{\mathcal{C}}, \pi^{\mathcal{C}}, \forall CU_k \in list_{hv}\ (W^{CU_k}, \pi^{CU_k}))$
24: Set $State \leftarrow Terminated$
```

Algorithm 4.11 $FairPPA.DRM2$

```
    Input: $W^{CP}$ and $\pi^{CP}$, $W^{\mathcal{C}}$ and $\pi^{\mathcal{C}}$, $\forall CU_k \in list_{hv}$ $(W^{CU_k}, \pi^{CU_k})$
    Output: $\phi$
 1: Compute $(W^{SC}, \pi^{SC}) \leftarrow TDA(X_m^{CU_i} | m \in (s_1, ..., s_m), CU_i \in list_{ho})$
 2: Compute $\$p_{CU_i} \leftarrow PM(W^{SC}, \$p)$
 3: $\forall CU_i \in list_{ho}$ set $ledger[CU_i] \leftarrow ledger[CU_i] + \frac{\$deposit_{do}}{|list_{ho}|+|list_{ma}|} + \$p_{CU_i}$
 4: Set $ledger[CP] \leftarrow ledger[CP] + \frac{\$deposit_{do}}{|list_{ho}|+|list_{ma}|} * (|list_{ma}|)$
 5: if $W^{CP} = W^{SC}$ and $\pi^{CP} = \pi^{SC}$ then
 6:     set $ledger[CP] \leftarrow ledger[CP] + \$f_{CP}$
 7: else
 8:     if $W^{\mathcal{C}} = W^{SC}$ and $\pi^{\mathcal{C}} = \pi^{SC}$ then
 9:         Set $ledger[\mathcal{C}] \leftarrow ledger[\mathcal{C}] + \$f_{\mathcal{C}}$
10:     else
11:         Set $\$corpus \leftarrow \$f_{CP} + \$f_{\mathcal{C}}$
12: foreach $CU_k \in list_{hv}$ do
13:     if $W^{CU_k} = W^{SC}$ and $\pi^{CU_k} = \pi^{SC}$ then
14:         set $list_{hhv} \leftarrow list_{hhv} \cup CU_k$
15: foreach $CU_k \in list_{hhv}$ do
16:     set $ledger[CU_k] \leftarrow ledger[CU_k] + \frac{\$deposit_{dv}}{|list_{hhv}|}$
```

4.3.5 $FairPPA$ contract clauses

The proposed system has four key entities: cloud provider, cloud user, verifiers and miners (Blockchain nodes) -see also Figure 5.3. The system consists of three phases: (1) Task creation phase (TCP) (2) Spawn and sensing phase (SSP) and (3) Data sharing and reward distribution phase (DRP).

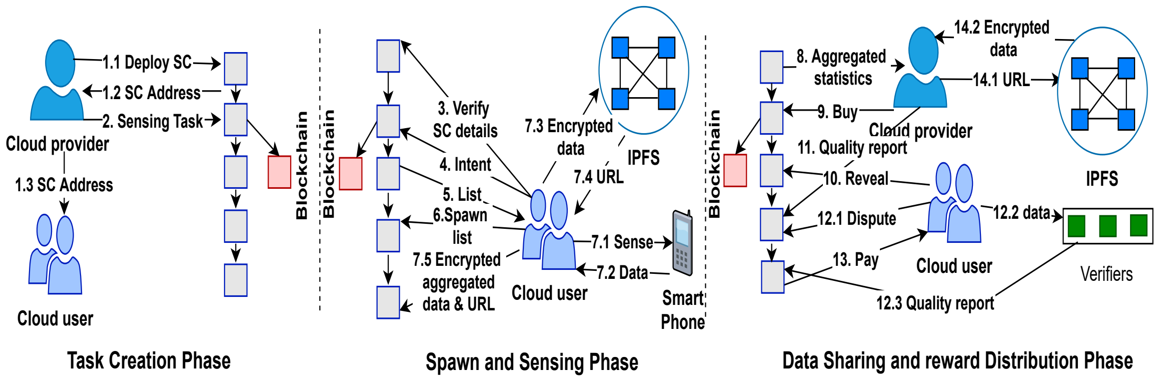

Figure 4.2: Overview of the proposed $FairPPA$ protocol

The clauses in $FairPPA$ contract are as follows:

Task creation phase (TCP)

(i) A cloud provider CP designs, develops and deploys a smart contract $FairPPA$ in a public Blockchain network and publish the contract address on a public platform (like a website / bulletin board).

(ii) CP initiates the $FairPPA$ with necessary parameters to which every participant has to adhere. He also initializes it with a pay $\$p$ distributed to the cloud users for participating in the sensing task.

Spawn and sensing phase (SSP)

(iii) A cloud user CU_i verifies the contract and task details with the publicly available contract address.

(iv) To participate in the sensing task, a CU_i has to send an intent message to $FairPPA$ along with safety deposit $\$d_{CU_i}$. The safe deposit $\$d_{CU_i}$ is required to avoid abrupt aborts of the cloud users during the protocol. If no CU_i has shown the intent, $\$p$ is refunded to CP, and the contract is terminated.

(v) $FairPPA$ contract randomly categorizes every CU_i into one of the two sets $list_{do}$ and $list_{dv}$. The first set computes the task, and the second set is assigned as verifiers during dispute resolution. Every cloud user receives $list_{do}$ and $list_{dv}$ by querying the $FairPPA$.

(vi) After receiving $list_{do}$, every $CU_i \in list_{do}$ generates a list of buddies ind_{CU_i} using $list_{do}$ and pseudo-random function. Each CU_i computes a value sk_{CU_j} for every $CU_j \in ind_{CU_i}$. The encryption key is computed as $sk_{CU_i} = \sum_{CU_j \in ind_{CU_i}} sk_{CU_j}$. The CU_i who sends the buddies list ind_{CU_i} to $FairPPA$ is added to $list_{sp}$. If no CU_i has sent the buddies list, then all the deposits of cloud users, and $\$p$ are sent to CP. The deposits of $\forall CU_k \in list_{dv}$ are refunded and the contract is terminated. If any CU_i fails to send the buddies list, then that CU_i is removed from $list_{do}$. $list_{sp}$ is set as empty, and a new ind_{CU_i} is requested until all the cloud users in $list_{do}$ have sent the buddies list.

(vii) Every $CU_i \in list_{sp}$ generates data $Y_m^{CU_i,t}$ according to the cloud provider's specification and stores the encrypted $Y_m^{CU_i,t}$ in IPFS. Aggregates $Y_m^{CU_i,t}$ in to $X_m^{CU_i}$ and encrypts it with a key generated in the previous step and sends the encrypted data to $FairPPA$, which will add CU_i to $list_{co}$. If any $CU_i \in list_{sp}$ fails to send the encrypted data, then he is removed from $list_{do}$. $list_{sp}, list_{co}$ are set as empty. A new ind_{CU_i} is requested until all the cloud users in $list_{do}$ have sent the buddies list and also sent the encrypted data. If no CU_i has sent the encrypted data, all the deposits of $CU_i \in list_{sp} \cup list_{do}$ along with $\$p$ are sent to CP. The contract is terminated after refunding deposits to every $CU_k \in list_{dv}$.

Data sharing phase (DSP)

(viii) The smart contract performs the aggregation operation on the encrypted data and generates aggregated statistics.

(ix) CP after learning the aggregated statistics of the data, if he is interested in buying the data, he will send a buy transaction to $FairPPA$ along with a deposit $\$f_{CP}$. This deposit is required to ensure honest behavior of CP during dispute resolution. If CP is not interested in buying, then he will not send any further transactions. $\$p$ is refunded to CP, and the deposits of $CU_i \in list_{co}$ and $CU_k \in list_{dv}$ are refunded, and the contract is terminated. The cloud users aborted during the SSP phase get a pay of $\$0$ and their deposits are distributed equally to $CU_i \in list_{co}$.

(x) Every $CU_i \in list_{co}$ reveals the IPFS address of the $Y_m^{CU_i,t}$ through $FairPPA$, which adds CU_i to $list_{ho}$.

(xi) CP runs a truth discovery algorithm (TDA) to find weights $W^{CP} = \{w_{CU_i} | CU_i \in list_{co}\}$ and sends W^{CP} along with proof-of-correctness of the computation π^{CP} to $FairPPA$. If CP fails to send W^{CP} then a dispute resolution mechanism is initiated.

(xii) The cloud users who do not have trust on CP may optionally run TDA locally, and if found that CP has not computed the weights correctly sends a challenge request to $FairPPA$ along with a deposit $\$f_{\mathcal{C}}$ and new weights $W^{\mathcal{C}}$ and new proof $\pi^{\mathcal{C}}$.

(xiii) If no challenge is raised then the $FairPPA$ will distribute the pay to every $CU_i \in list_{ho}$ according to W^{CP}. The cloud users aborted during the SSP and DSP phases get a pay of \$0 and their deposits are sent to CP. $\$f_{CP}$ is refunded to CP. If any cloud user raises a dispute, then a dispute resolution mechanism (DRM) is initiated.

(xiv) CP after learning sk_{DO_i}, retrieves the encrypted data from IPFS and decrypts it locally to obtain $Y_m^{CU_i,t}$

4.3.6 $FairPPA$ contract phases

4.3.6.1 Task creation phase

Before creating a task, CP designs, develops and sends a smart contract $FairPPA$ to a public Blockchain network. The Blockchain network deploys $FairPPA$ according to standard contract mining process and returns contract address. CP chooses two safe primes q_1 and q_2 to compute $N = q_1 * q_2$ and a generator g of group $\mathbb{G}_q$ with prime order q. He also chooses a secure pseudorandom function $PRF : \{0,1\}^a \times \{0,1\}^* \rightarrow \{0,1\}^a$, a hash function $H : \mathbb{Z} \rightarrow \mathbb{Z}^*_{N^2}$, a secure symmetric encryption method Enc and a secure commitment method $Comm$. Then, CP prepares and sends a transaction **trans**$_{PP}^{CP} = (N, H, \mathbb{G}_q, g, Enc, Comm, PRF)$ to BC. All the received parameters are stored in $FairPPA$ contract storage. Then, CP chooses three random numbers r_{CP}, r_1, and r_2. Computes $pk_{CP} = g^{r_{CP}}$ and sends a transaction **trans**$_{create}^{CP} = (task_{id}, pk_{CP}, \$p, \tau_{in}, \tau_{sp}, \tau_{co}, \tau_{bu}, \tau_{re}, \tau_{cp}, \tau_{ch}, \tau_{di}, r_1, r_2, minDo, minDv, th, k)$. As $FairPPA$ contract handles multiple sensing task simultaneously, $task_{id}$ is used to differentiate among multiple sensing tasks[4]. Across the phases, the participating entities follow the timing parameters. These timing parameters are required to enforce timely computation and also to avoid locking of funds indefinitely. All the entities are aware of the timer which progresses in rounds and at the beginning of each round the timer functionality is executed. The current time is fetched by querying the underlying Blockchain. $\$p$ is distributed among cloud users according to the quality of their sensed data. The random numbers r_1, r_2 values change for every new sensing task. The parameters $minDo$ and $minDv$ ensures that there is a minimum number

[4]From here on we omit $task_{id}$ for the sake of better clarity.

of cloud users and data verifiers before starting a sensing task. th is known as threshold used during dispute resolution. The parameter k is used to update timing parameters. When received $\textbf{trans}^{CP}_{create}$, the $FairPPA$ asserts $\tau_{in} < \tau_{sp} < \tau_{co} < \tau_{bu} < \tau_{re} < \tau_{cp} < \tau_{ch} < \tau_{di}$. Then, the received parameters are stored in contract storage, and the state of the contract is changed to $Intent$. The formal protocol for TCP is presented in Figure 4.3.

Protocol: Task Creation Phase

For cloud provider CP

1. Send $\textbf{trans}^{CP}_{deploy} = (FairPPA)$ to Blockchain $\mathcal{BC}$. After receiving the $FairPPA_{address}$, publish the address publicly.
2. Choose two safe primes q_1 and q_2 and compute $N = q_1 * q_2$. Choose a hash function $H : \mathbb{Z} \rightarrow \mathbb{Z}^*_{N^2}$, a pseudo-random function $PRF : \{0,1\}^a \times \{0,1\}^* \rightarrow \{0,1\}^a$ a symmetric encryption method Enc and commitment method $Comm$. Choose a generator g of group $\mathbb{G}_q$ with prime order q.
3. Send $\textbf{trans}^{CP}_{PP} = (N, H, \mathbb{G}_q, g, Enc, Comm, PRF)$ to $FairPPA$.
4. Choose a random number $r_{CP} \in_R \mathbb{Z}_q$ and compute $pk_{CP} = g^{r_{CP}}$.
5. Generate two random numbers $r_1, r_2 \in_R \mathbb{Z}_q$ such that $r_1 \neq r_2$.
6. Send $\textbf{trans}^{CP}_{create} = (task_{id}, pk_{CP}$, $\$p$, τ_{in}, τ_{sp}, τ_{co}, τ_{bu}, τ_{re}, τ_{cp}, τ_{ch}, τ_{di}, r_1, r_2, $minDo$, $minDv$, th, $k)$ to $FairPPA$ and publish the task details publicly.

Blockchain

7. On receiving $\textbf{trans}^{CP}_{deploy}$ deploy $FairPPA$ and return $FairPPA_{address}$.
8. On receiving $\textbf{trans}^{CP}_{PP}$, store all the received parameters.
9. On receiving $\textbf{trans}^{CP}_{create}$
   (a) Assert $ledger[CP] \geq \$p$ and $\tau_{in} < \tau_{sp} < \tau_{co} < \tau_{bu} < \tau_{re} < \tau_{cp} < \tau_{ch} < \tau_{di}$
   (b) Set $state \leftarrow Intent$ and store all the received parameters.

Figure 4.3: Task Creation Phase protocol

4.3.6.2 Spawn and sensing phase

An interested cloud user CU_i fetches the contract details and verifies them. He computes sensing cost and expected utility. If the expected utility is more than the sensing cost, then he chooses a random number r_{CU_i} and computes $pk_{CU_i} = g^{r_{CU_i}}$. He prepares and sends $\textbf{trans}^{CU_i}_{intent} = (pk_{CU_i}, \$d_{CU_i})$ to $FairPPA$. The deposit $\$d_{CU_i}$ is required to ensure participation of the cloud user till the end of the protocol. The contract verifies that CU_i has not sent the intent transaction for the same task previously. Then, the contract randomly adds CU_i in to one of two lists $list_{do}$ or $list_{dv}$. Generating true randomness is not possible due to the deterministic nature of smart contracts. As currently, the cloud users cannot predict the exact time-stamp of the new block, we use the result of $blockhash(block.timestamp)$ as a random number which meets our need to randomly assign cloud users to $list_{do}$ and $list_{dv}$. $block.timestamp$ returns the current block timestamp as seconds since UNIX epoch and $blockhash$ is a function that takes an integer as input and returns the hash of that block. After $\tau > \tau_{in}$: (1) if $state = Intent$ and $|list_{do}| < minDo$ and $|list_{dv}| < minDv$, $\$p$ is refunded to CP and the deposits of $CU_i \in list_{do}$ and $CU_k \in list_{dv}$ are also refunded. The state of the contract is set to $Aborted$. (2) If $state = Intent$ and $|list_{do}| \geq minDo$ and $|list_{dv}| \geq minDv$, then the contract state is set as $Spawn$

Every CU_i queries the $FairPPA$ to fetch $list_{do}$ and $list_{dv}$. If $CU_i \in list_{do}$, then they run $(ind_{CU_i}, sk_{CU_i}) \leftarrow KeyGen(CU, r_1, r_2, l)$. Next, CU_i sends $\textbf{trans}^{CU_i}_{spawn} = (ind_{CU_i})$. After receiving $\textbf{trans}^{CU_i}_{spawn}$, the contract checks $\tau < \tau_s$, $state = Spawn$, $CU_i \in list_{do}$, and $CU_i \notin list_{sp}$. If all checks are passed, then the CU_i along with received parameters is added to $list_{sp}$. After $\tau > \tau_{sp}$: (1) if $state = Spawn$ and $|list_{sp}| = 0$, then the contract state is set as $Aborted$ and $\$p$ is refunded to CP along with the deposits of $CU_i \in list_{do}$. The deposits of $CU_k \in list_{dv}$ are refunded. (2) if $state = Spawn$ and $|list_{sp}| \neq |list_{do}|$, then the aborted cloud users are removed from $list_{do}$ and all timing parameters except τ_i are increased by a factor of k. $list_{sp}$ is set as empty. The contract will be in the $Spawn$ state until $list_{sp} = list_{do}$ or $list_{do} = 0$. (3) if $state = Spawn$ and $list_{sp} = list_{do}$, then the contract state is set as $Commit$.

Now, every $CU_i \in list_{sp}$ starts sensing according to the CP specifications and collect the data $Y^{CU_i,t}_m$ from the requested sensors $(s_1, ..., s_m)$ in the requested time interval $[t_s, t_e]$. As $Y^{CU_i,t}_m$ is large data set, it cannot be shared through Blockchain because storage incurs a huge cost in public Blockchain networks. So, CU_i aggregates $Y^{CU_i,t}_m$ into $X^{CU_i}_m = [\sum_{i=s}^{e} y_{1,i}, ..., \sum_{i=s}^{e} y_{m,i}]$ and encrypts $X^{CU_i}_m$ as $C^{CU_i}_m = [(1 + \sum_{i=s}^{e} y_{1,i} * N) \cdot H(t)^{sk_{CU_i}} \bmod N^2, ..., (1 + \sum_{i=s}^{e} y_{m,i} * N) \cdot H(t)^{sk_{CU_i}} \bmod N^2]$. Then, a shared key is computed as $K_{CU_i,CP} = (pk_{CP})^{r_{CU_i}} = (pk_{CU_i})^{r_{CP}} = g^{r_{CU_i} r_{CP}}$. By using symmetric encryption CU_i performs double encryption on $Y^{CU_i,t}_m$ to generate $CY^{CU_i,t}_m = Enc_{sk_{CU_i}}(Enc_{K_{CU_i,CP}}(Y^{CU_i,t}_m))$. The encryption with shared key protects the data from entities other than CP and encryption with sk_{CU_i} protects data from CP till the CU_i reveals it. $CY^{CU_i,t}_m$ is stored at IPFS network which returns the hash of $CY^{CU_i,t}_m$ as an URL url to access it. CU_i encrypts

url and sk_{CU_i} as $C_{url}^{CU_i} = Enc_{sk_{CU_i}}(url)$ and $CK^{CU_i} = Enc_{K_{CU_i,CP}}(sk_{CU_i})$. As a final step in this phase CU_i commits the encryption of sk_{CU_i} as $ct^{CU_i} = comm(CK^{CU_i}, s)$ where s is a randomly chosen. Then, CU_i prepares and sends a transaction $\textbf{trans}_{commit}^{CU_i} = (C_m^{CU_i}, C_{url}^{CU_i}, ct^{CU_i})$ to $FairPPA$. After receiving $\textbf{trans}_{commit}^{CU_i}$, the $FairPPA$ contract verifies (1) $\tau < \tau_{co}$ (2) $CU_i \in list_{sp}$ (3) $CU_i \notin list_{co}$ and (4) $state = Commit$. If all the checks are passed, then CU_i is added to $list_{co}$ along with all the received values. After $\tau > \tau_{co}$: (1) if $state = Commit$ and $|list_{co}|= 0$, then the contract state is set as $Aborted$ and $\$p$ is refunded to CP along with deposits of $CU_i \in list_{do} \cup list_{sp}$. The deposits of $CU_k \in list_{dv}$ are refunded. (2) if $|list_{co}| \neq |list_{sp}|$, then the aborted cloud users are removed from $list_{do}$. $list_{sp}$, $list_{co}$ are set as empty. All timing parameters except τ_{in} are increased by a factor of k, and the contract state is set as $Spawn$. (3) if $list_{co} = list_{sp}$, then the contract computes the aggregated statistics as

$$P_{j,t} \leftarrow \prod_{i=1}^{|list_{co}|} C_j^{CU_i} \mod N^2 \ \forall j = 1\, to\, m$$
$$sum_{j,t} \leftarrow \frac{P_{j,t}-1}{N} \ \forall j = 1\, to\, m$$

$FairPPA$ stores the computed statistics and the contract state is set as Buy. The formal protocol for SSP is presented in Figure 4.4.

Protocol: Spawn and Sensing Phase

<u>For cloud user CU_i</u>

1. Verify $FairPPA$ contract and task details. Calculate sensing cost and utility according to CP's specifications.
2. If satisfied with utility then choose a random number $r_{CU_i} \in_R \mathbb{Z}_q$ and compute $pk_{CU_i} = g^{r_{CU_i}}$
3. Send $\textbf{trans}_{intent}^{CU_i} = (pk_{CU_i}, \$d_{CU_i})$ to $FairPPA$.
4. Query the smart contract for $list_{do}$ and $list_{dv}$. If $CU_i \in list_{do}$, then run $(ind_{CU_i}, sk_{CU_i}) \leftarrow KeyGen(list_{do}, r_1, r_2, l)$ and send $\textbf{trans}_{spawn}^{CU_i} = (ind_{CU_i})$ to $FairPPA$
5. Using smart device collect sensing data from sensors $S_1, .., S_m$ in the time interval $t = [t_s, t_e]$

$$Y_m^{CU_i,t} = \begin{array}{c c} & \begin{array}{cccc} S_1 & S_2 & \cdots\cdots & S_m \end{array} \\ \begin{array}{c} t_s \\ \vdots \\ t_e \end{array} & \begin{bmatrix} y_{1,s} & y_{2,s} & \cdots\cdots & y_{m,s} \\ \vdots & \vdots & & \vdots \\ y_{1,e} & y_{2,e} & \cdots\cdots & y_{m,e} \end{bmatrix} \end{array}$$

6. Aggregates $Y_m^{CU_i,t}$ in to $x_m^{CU_i}$ and Compute
$$C_m^{CU_i} =$$
$$[(1 + \sum_{j=s}^{e} y_{1,j} * N) \cdot H(t)^{sk_{CU_i}} \bmod N^2, ..., (1 + \sum_{j=s}^{e} y_{m,j} * N) \cdot H(t)^{sk_{CU_i}} \bmod N^2]$$
7. Compute a shared key $K_{CU_i,CP} = (pk_{CP})^{r_{CU_i}} = (pk_{CU_i})^{r_{CP}} = g^{r_{CU_i} r_{CP}}$

```
    8. Compute $CY_m^{CU_i,t} = Enc_{sk_{CU_i}}(Enc_{K_{CU_i,CP}}(Y_m^{CU_i,t}))$ and send $CY_m^{CU_i,t}$ to IPFS and receive $url$.
    9. Compute $C_{url}^{CU_i} = Enc_{sk_{CU_i}}(url)$ and $CK^{CU_i} = Enc_{K_{CU_i,CP}}(sk_{CU_i})$ and $ct^{CU_i} = comm(CK^{CU_i}, s)$ where $s$ is a randomly chosen.
    10. Send **trans**$_{commit}^{CU_i} = (C_m^{CU_i}, C_{url}^{CU_i}, ct^{CU_i})$ to $FairPPA$
Blockchain: $list_{do} \leftarrow \{\}$, $list_{dv} \leftarrow \{\}$, $list_{sp} \leftarrow \{\}$, $list_{co} \leftarrow \{\}$, $list_{ma} \leftarrow \{\}$, $\$deposit_{do} \leftarrow 0$, $\$deposit_{dv} \leftarrow 0$
    11. On receiving **trans**$_{intent}^{CU_i}$
        (a) Assert $ledger[CU_i] \geq \$d$ and $\tau < \tau_{in}$ and $CU_i \notin (list_{do} \cup list_{dv})$ and $state = Intent$
        (b) If $(blockhash(block.timestamp))\%2 = 0$ then set $list_{do} \leftarrow list_{do} \cup CU_i$ and $\$deposit_{do} \leftarrow \$deposit_{do} + \$d$. Else, set $list_{dv} \leftarrow list_{dv} \cup CU_i$ and $\$deposit_{dv} \leftarrow \$deposit_{dv} + \$d$.
    12. On receiving **trans**$_{spawn}^{CU_i}$
        (a) assert $\tau < \tau_{sp}$ and $state = Spawn$ and $(CU_i, *) \notin list_{sp}$ and $CU_i \in list_{do}$
        (b) set $list_{sp} \leftarrow list_{sp} \cup (CU_i, ind_{CU_i})$
    13. On receiving **trans**$_{commit}^{CU_i}$
        (a) assert $\tau < \tau_{co}$ and $state = Commit$ and $(CU_i, *) \in list_{sp}$ and $(CU_i, *, *, *) \notin list_{co}$
        (b) set $list_{co} \leftarrow list_{co} \cup (CU_i, C_m^{CU_i}, C_{url}^{CU_i}, ct^{CU_i})$
Timer
    if $\tau > \tau_{in}$ and $state = Intent$ and $|list_{do}| < minDo$ and $list_{dv} < minDv$
        set $ledger[CP] \leftarrow ledger[CP] + \$p$
        $\forall CU_i \in list_{do}$ set $ledger[CU_i] \leftarrow ledger[CU_i] + \frac{\$deposit_{do}}{|list_{do}|}$
        $\forall CU_k \in list_{dv}$ set $ledger[CU_k] \leftarrow ledger[CU_k] + \frac{\$deposit_{dv}}{|list_{dv}|}$ and set $state \leftarrow Aborted$
    if $\tau > \tau_{in}$ and $state = Intent$ and $|list_{do}| > minDo$ and $|list_{dv}| > minDv$ then set $state \leftarrow Spawn$
    if $\tau > \tau_{sp}$ and $state = Spawn$ and $|list_{sp}| = 0$
        set $ledger[CP] \leftarrow ledger[CP] + \$p + \$deposit_{do}$
        $\forall CU_k \in list_{dv}$ set $ledger[CU_k] \leftarrow ledger[CU_k] + \frac{\$deposit_{dv}}{|list_{dv}|}$ and set $state \leftarrow Aborted$
    if $\tau > \tau_{sp}$ and $state = Spawn$ and $|list_{sp}| \neq |list_{do}|$
        set $list_{ma} \leftarrow list_{ma} \cup (list_{do} - list_{sp})$ and $list_{do} \leftarrow list_{do} - list_{ma}$ and $list_{sp} \leftarrow \phi$ and $\tau_{sp} \leftarrow \tau_{sp} + k$ and $\tau_{co} \leftarrow \tau_{co} + k$ and $\tau_{bu} \leftarrow \tau_{bu} + k$ and $\tau_{re} \leftarrow \tau_{re} + k$ and $\tau_{cp} \leftarrow \tau_{cp} + k$ and $\tau_{ch} \leftarrow \tau_{ch} + k$ and $\tau_{di} \leftarrow \tau_{di} + k$
    if $\tau > \tau_{sp}$ and $state = Spawn$ and $|list_{sp}| = |list_{do}|$ then set $state \leftarrow Commit$
    if $\tau > \tau_{co}$ and $state = Commit$ and $|list_{co}| = |list_{sp}|$
        set $P_{j,t} \leftarrow \prod_{i=1}^{|list_{co}|} C_j^{CU_i} \mod N^2 \; \forall j = 1\, to\, m$
        set $sum_{j,t} \leftarrow \frac{P_{j,t}-1}{N} \; \forall j = 1\, to\, m$ and $state \leftarrow Buy$
    if $\tau > \tau_{co}$ and $state = Commit$ and $|list_{co}| \neq |list_{sp}|$
        set $list_{ma} \leftarrow list_{ma} \cup (list_{sp} - list_{co})$ and $list_{do} \leftarrow list_{do} - list_{ma}$ and $list_{sp} \leftarrow \phi$ and $list_{co} \leftarrow \phi$ and $list_{sp} \leftarrow \phi$ and $\tau_{sp} \leftarrow \tau_{sp} + k$ and $\tau_{co} \leftarrow \tau_{co} + k$ and $\tau_{bu} \leftarrow \tau_{bu} + k$ and $\tau_{re} \leftarrow \tau_{re} + k$ and $\tau_{cp} \leftarrow \tau_{cp} + k$ and $\tau_{ch} \leftarrow \tau_{ch} + k$ and $\tau_{di} \leftarrow \tau_{di} + k$ and $state \leftarrow spawn$
    if $\tau > \tau_{co}$ and $state = Commit$ and $|list_{co}| = 0$
        set $ledger[CP] \leftarrow ledger[CP] + \$p + \$deposit_{do}$
        $\forall CU_k \in list_{dv}$ set $ledger[CU_k] \leftarrow ledger[CU_k] + \frac{\$deposit_{dv}}{|list_{dv}|}$ and $state \leftarrow Aborted$
```

Figure 4.4: Spawn and Sensing Phase Protocol

4.3.6.3 Data sharing and reward distribution phase

The cloud provider receives the aggregated statistics by querying the $FairPPA$. Then, he analyzes the statistics and if he is satisfied with the statistics, then he sends $\textbf{trans}_{buy}^{CP}$ to $FairPPA$ along with a deposit of $\$f_{CP}$. After receiving $\textbf{trans}_{buy}^{CP}$, the $FairPPA$ checks whether $\tau < \tau_{bu}$ and $state = buy$. If checks are valid, then the contract state is set as *Reveal*. If CP is not interested in buying the data, then it aborts the protocol by not sending any further transactions. If the contract state is *Buy* even after $\tau > \tau_{bu}$, then the deposits of $CU_i \in list_{co}$ and $CU_k \in list_{dv}$ are refunded. To compensate $CU_i \in list_{co}$ for their honest participation, the deposits of aborted cloud users are distributed equally among $CU_i \in list_{co}$. $\$p$ and $\$f_{CP}$ are refunded to CP. The contract state is set as $Aborted$.

Soon after the contract state is set as *Reveal*, every $CU_i \in list_{co}$ sends $\textbf{trans}_{reveal}^{CU_i} = (X_m^{CU_i}, CK^{CU_i}, s)$ to $FairPPA$. After receiving $\textbf{trans}_{reveal}^{CU_i}$, the contract checks whether (1) $\tau < \tau_{re}$ (2) $state = Reveal$ (3) $CU_i \in list_{co}$ (4) $CU_i \notin list_{ho}$ and (5) $ct^{CU_i} = comm(CK^{CU_i}, s)$. If all checks are valid, then the contract adds CU_i to $list_{ho}$ along with the received data. After $\tau > \tau_{re}$: (1) if $state = Reveal$ and $|list_{ho}| = 0$, then the deposits of $CU_i \in list_{do}$, the deposits of aborted cloud users and $\$p, \$f_{CP}$ are sent to CP. The deposits of $CU_k \in list_{dv}$ are also refunded. The contract state is set as $Aborted$. (2) If $state = Reveal$ and $|list_{ho}| \neq 0$, then the contract state is set as $Compute$.

Now, the CP executes Algorithm 4.8 to find ground truths and the corresponding weights according to the quality of the data. CP prepares and sends $\textbf{trans}_{proof}^{CP} = (W^{CP}, \pi^{CP})$ to $FairPPA$ where π^{CP} is inner state hash (ISH) of the computation. After receiving $\textbf{trans}_{proof}^{CP}$, the contract verify whether $\tau < \tau_{cp}$ and $state = Compute$. If checks are valid, then the contract state is set as $Challenge$ and the received data is stored in contract storage. After $\tau > \tau_{cp}$, if $state = Compute$, then the state of the contract is set as $Dispute$.

If the contract state is $Challenge$, then any $CU_i \in list_{ho}$ who does not have trust on CP can execute Algorithm 4.8 locally. If the locally computed results have any discrepancies with the results sent by CP, then CU_i prepares and sends $\textbf{trans}_{challenge}^{CU_i} = (W^{\mathcal{C}}, \pi^{\mathcal{C}}, \$f_{\mathcal{C}})$ where $W^{\mathcal{C}}, \pi^{\mathcal{C}}$ are locally computed values. After receiving $\textbf{trans}_{challenge}^{CU_i}$, the contract verifies whether (1) $\tau < \tau_{ch}$, (2) $state = Challenge$, and (3) $CU_i \in list_{ho}$. If checks are valid, then the contract state is set as $Dispute$ and received parameters are stored. After $\tau > \tau_{ch}$ and $state = Challenge$ then, the $FairPPA$ executes Algorithm 4.9 with W^{CP} and pays according to the results returned. $\$f_{CP}$ is refunded to CP along with the deposits of the aborted cloud users. The deposits of $CU_i \in list_{ho}$, $CU_k \in list_{dv}$ are refunded. The contract state is set as $Terminated$.

When the contract state is set as $Dispute$, every $CU_k \in list_{dv}$ executes Algorithm 4.8 locally and sends $\textbf{trans}^{CU_k}_{verify} = (W^{CU_k}, \pi^{CU_k})$ to $FairPPA$. After receiving $\textbf{trans}^{CU_k}_{verify}$, the contract verifies whether (1) $\tau < \tau_{di}$, (2) $state = Dispute$ and (3) $CU_k \in list_{dv}$. If all checks are valid, then the contract stores the received data. After $\tau > \tau_{di}$, if $state = Dispute$, then the $FairPPA$ executes Algorithm 4.10. The formal protocol for DRP is presented in Figure 4.5.

Protocol: Data Sharing Phase

For cloud provider CP

1. Query the $FairPPA$ to receive $Sum_{j,t}$ $\forall j = 1$ to m, if satisfied with statistics send $\textbf{trans}^{CP}_{buy} = (\$f_{CP})$ to $FairPPA$.
2. If $\tau > \tau_{re}$ compute $(W^{CP}, \pi^{CP}) \leftarrow TDA(x^{CU_i}_m | CU_i \in list_{ho})$ and send $\textbf{trans}^{CP}_{proof} = (W^{CP}, \pi^{CP})$ to $FairPPA$.
3. For every received CK^{CU_i} do
   (a) Compute a shared key $K_{CU_i,CP} \leftarrow (pk_{CP})^{r_{CU_i}} = (pk_{CU_i})^{r_{CP}} = g^{r_{CU_i} r_{CP}}$
   (b) Compute $sk_{CU_i} \leftarrow Dec_{K_{CU_i,CP}}(CK^{CU_i})$ and $url \leftarrow Dec_{sk_{CU_i}}(C^{CU_i}_{url})$.
   (c) Access $IPFS$ network with url to obtain $CY^{CU_i,t}_m$
   (d) Compute $\bar{Y}^{CU_i,t}_m \leftarrow Dec_{sk_{CU_i}}(CY^{CU_i,t}_m)$ and $Y^{CU_i,t}_m \leftarrow Dec_{K_{CU_i,CP}}(\bar{Y}^{CU_i,t}_m)$

For cloud user CU_i

4. If $FairPPA$'s $state = Reveal$ then send $\textbf{trans}^{CU_i}_{reveal} = (x^{CU_i}_m, CK^{CU_i}, s)$ to $FairPPA$
5. If $FairPPA$'s $state = Challenge$ then run $(W^{C}, \pi^{C}) \leftarrow TDA(x^{CU_i}_m | CU_i \in list_{ho})$.
6. If $W^{CP} \neq W^{C}$ and/or $\pi^{CP} \neq \pi^{C}$ then send $\textbf{trans}^{CU_i}_{challenge} = (W^{C}, \pi^{C})$

For Data Verifier CU_k

7. If $FairPPA$'s state=Dispute then compute $(W^{CU_k}, \pi^{CU_k}) \leftarrow TDA(x^{CU_i}_m | CU_i \in list_{ho})$
8. send $\textbf{trans}^{CU_k}_{verify} = (W^{CU_k}, \pi^{CU_k})$ to $FairPPA$

Blockchain: $list_{ho} \leftarrow \{\}, list_{hv} \leftarrow \{\}$

9. On receiving $\textbf{trans}^{CP}_{buy}$
   (a) assert $ledger[CP] \geq \$f_{CP}$ $\tau < \tau_{bu}$ and $state = Buy$
   (b) set $state \leftarrow Reveal$
10. On receiving $\textbf{trans}^{CU_i}_{reveal}$
    (a) assert $\tau < \tau_{re}$ and $state = Reveal$ and $ct^{CU_i} = comm(CK^{CU_i}, s)$ and $CU_i \in list_{co}$ and $CU_i \notin list_{ho}$
    (b) set $list_{ho} \leftarrow list_{ho} \cup (CU_i, X^{CU_i}_m)$
11. On receiving $\textbf{trans}^{CP}_{proof} = (W^{CP}, \pi^{CP})$
    (a) assert $\tau < \tau_{cp}$ and $state = Compute$
    (b) Store (W^{CP}, π^{CP}) and set $State \leftarrow Challenge$

```
    12. On receiving $\textbf{trans}^{CU_i}_{challenge} = (W^{\mathcal{C}}, \pi^{\mathcal{C}})$
        (a) assert $\tau < \tau_{ch}$ and $state = Challenge$ and $CU_i \in honest$
        (b) Store $(W^{\mathcal{C}}, \pi^{\mathcal{C}})$ and set $state \leftarrow Dispute$
    13. On receiving $\textbf{trans}^{CU_k}_{verify} = (W^{CU_k}, \pi^{CU_k})$
        (a) assert $\tau < \tau_{di}$ and $state = Dispute$ and $CU_k \in list_{dv}$
        (b) set $list_{hv} \leftarrow list_{hv} \cup (CU_k, W^{CU_k}, \pi^{CU_k})$
Timer
    if $\tau > \tau_{bu}$ and $state = Buy$
        set $ledger[CP] \leftarrow ledger[CP] + \$p$
        $\forall CU_i \in list_{co}$ set $ledger[CU_i] \leftarrow ledger[CU_i] + \frac{\$deposit_{do}}{|list_{co}|}$
        $\forall CU_k \in list_{dv}$ set $ledger[CU_k] \leftarrow ledger[CU_k] + \frac{\$deposit_{dv}}{|list_{dv}|}$ and set $state \leftarrow Aborted$
    if $\tau > \tau_{re}$ and $state = Reveal$ and $|list_{ho}| = 0$
        set $ledger[CP] \leftarrow ledger[CP] + \$p + \$deposit_{do} + \$f_{CP}$
        $\forall CU_k \in list_{dv}$ set $ledger[CU_k] \leftarrow ledger[CU_k] + \frac{\$deposit_{dv}}{|list_{dv}|}$ and set $state \leftarrow Aborted$
    if $\tau > \tau_{re}$ and $state = Reveal$ and $|list_{ho}| \neq 0$ set $list_{ma} \leftarrow list_{ma} \cup (list_{co} - list_{ho})$ and $state \leftarrow Compute$
    if $\tau > \tau_{cp}$ and $state = Compute$ set $state \leftarrow Dispute$
    if $\tau > \tau_{ch}$ and $state = Challenge$
        $\forall CU_i \in list_{ho}$ set $\$p_{CU_i} \leftarrow PM(W^{CP}, \$p)$
        $\forall CU_i \in list_{ho}$ set $ledger[CU_i] \leftarrow ledger[CU_i] + \frac{\$deposit_{do}}{|list_{ho}|+|list_{ma}|} + \$p_{CU_i}$
        $\forall CU_k \in list_{dv}$ set $ledger[CU_k] \leftarrow ledger[CU_k] + \frac{\$deposit_{dv}}{|list_{dv}|}$
        set $ledger[CP] = ledger[CP] + \frac{\$deposit_{do}}{|list_{ho}|+|list_{ma}|} * (|list_{ma}|) + \$f_{CP}$ and set $state \leftarrow Terminated$
    if $\tau > \tau_{di}$ and $state = Dispute$ then run $FairPPA.DRM(W^{CP}, \pi^{CP}, W^{\mathcal{C}}, \pi^{\mathcal{C}}, \forall CU_k \in list_{hv}\ (W^{CU_k}, \pi^{CU_k}), minDv, th,$ $\$p$, $\$deposit_{do}$, $\$deposit_{dv}$, $\$f_{CP}$, $\$f_{\mathcal{C}}$, $list_{ho}$, $list_{ma}$, $list_{hv}$).
```

Figure 4.5: Data sharing and reward distribution phase protocol

4.4 Security Guarantees

Theorem 4.4.1. *Our proposed protocol satisfies aggregator obliviousness under Decisional Composite Residuosity (DCR) in random oracle model.*

We show that our protocol can be reduced to aggregator obliviousness protocol in Joye *et al.* [189] which is sufficient to show that our protocol satisfies aggregator obliviousness under DCR assumption. Our protocol is different from [189] in two aspects: First, in [189], the keys are generated by a trusted party and are sent to cloud users securely, whereas in our protocol the cloud users generate their keys. Both the protocols are the same from the attacker's perspective that is the secret key can be compromised if and only if the cloud user is corrupted. Second, in [189] aggregator's key is assumed to be compromised, whereas, in our protocol, there is no aggregator key. So, from the attacker's perspective, it is same that the key is compromised or there is no key at all. In summary, if the protocol in [189] satisfies aggregator obliviousness, then our protocol also satisfies aggregator obliviousness as defined in Definition 4.1.2.

Theorem 4.4.2. *Our proposed protocol satisfies fair payments*

We prove fairness by considering the following cases:

Case 1: CU_i is malicious and aborts after the Buy phase without revealing the actual data. In this case, according to $FairPPA$ contract, his deposit is forfeited, and moreover, the payment for his data contribution is withheld. Here, the CU_i does not receive any payment without revealing the actual data. Thus, fairness holds.

Case 2: CP aborts after learning the aggregate statistics. In this case, according to $FairPPA$ contract the pay is refunded to CP and the deposit of every cloud user is refunded. Here, the CP does not receive data without initiating the payment. Thus, fairness holds.

Case 3: CP aborts after knowing data without computing weights. In this case, according to $FairPPA$ contract, the contract asks a set of verifiers to compute weights and payment to cloud users is made according to the weights returned by verifiers. Here, although the CP receives data, he cannot avoid paying to the cloud users. Thus, fairness holds.

In summary, our protocol satisfies fair payments as defined in Definition 4.1.2

4.5 Implementation and comparisons

The simulation environment is discussed in Section 1.2.3. We have adopted the factory model of solidity [194] to design our contracts. The factory model helps in saving gas costs during deployment of contracts for every new sensing task. The actual sensing tasks are treated as black boxes, and the contracts do not need to know the sensors internal states. The contracts are called before the start of sensing task and after the completion of the sensing task. We have implemented all the contracts in private Ethereum network which mimics the Ethereum production network. However, this implementation aims to deploy the contracts on public Ethereum network in real-world scenarios.

4.5.1 MotionSense Dataset

We have tested the contracts for MotionSense [188] dataset. MotionSense[5] contains time-series data collected by both accelerometer and gyroscope sensors (attitude, gravity, user acceleration and rotation rate). A total of 24 participants performed six activities in 15 trails. The activities are downstairs, upstairs, walking, jogging, sitting and standing. Two different kinds of trails are conducted: (1) Long trails - numbered 1 to 9 with around 2 to 3 minutes duration (2) Short trails - numbered 11 to 16 that are around 30 seconds to 1-minute duration.

4.5.2 Implementation of $FairNaivePPA$

We ran our experiments multiple times, and each transaction's computational and financial cost computed is shown in Table 4.1. We have varied the number of cloud users from 1 to 24 and computed the gas consumption of aggregate and payout functionalities in Figure 4.6. For implementation feasibility, we have modified our $FairNaivePPA$ in Figure 4.1. The current blockchain networks support calling the function in a contract only through an Ethereum account or a function in the same contract or another contract. That is the scheduled function calls which are executed when the timer expires are not possible. The timer is implemented as a payout functionality called by anyone, including a cloud user or cloud provider. Observe that the contract deployment consumes a tremendous amount of gas; this is due to large contract storage usage. Storing data in a contract is expensive in Ethereum network.

[5]The MotionSense dataset consists of data collected from different age groups. So, naturally, there will be a significant difference in sensor readings of different participants. Although we are aware that data quality will vary due to the participant's age, we use this dataset to test proof-of-concept implementation of proposed smart contracts.

Function	**Caller**	**Cost in Gas**	**Cost in $**
Deployment	CP	2501562*	26.491
Create	CP	142536	1.509
Intent	CU_i	69124	0.732
Commit	CU_i	204218	2.162
Aggregate	CP	500203**	5.297
Buy	CP	41497	0.439
Reveal	CU_i	67446	0.714
Payout	Anyone	232359**	2.46

Table 4.1: Costs of interacting with $FairNaivePPA$ contract. We have approximated the gas price as 30 Gwei and 1 ETH = $ 353, which are the real world costs in Oct 2020. We have rounded off the cost in $ value up to three decimals. * - including contract deployment. ** - gas consumption for 24 cloud users.

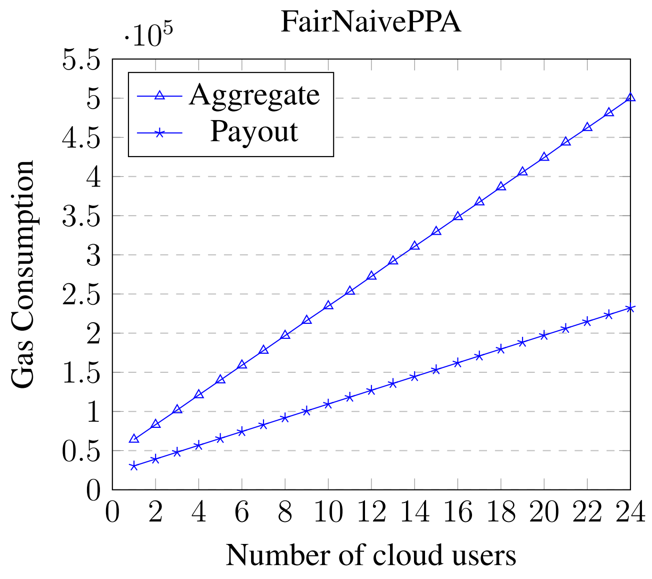

Figure 4.6: Gas Consumption of $FairNaivePPA$ - Aggregate and Payout functionalities

Function	**Caller**	**Cost in Gas**	**Cost in $**
Deploy	CP	3808389	1.485
PP	CP	152537	0.059
Create	CP	252965	0.098
Intent	CU_i	69337	0.027
Spawn	CU_i	188540*	0.073
Commit	CU_i	181492	0.07
Aggregate	CP	409759**	0.159
Buy	CP	41497	0.016
Reveal	CU_i	224142	0.087
Proof	CP	590492	0.23
Challenge	CU_i	590492	0.23
Verify	CU_k	540624	0.213
Payout	Anyone	359838***	0.140

Table 4.2: Costs of interacting with $FairPPA$ contract. We approximated the gas price as 1 Gwei and 1 ETH = $ 390, which are the real world costs in Sep 2020. We have rounded off the cost in $ value up to three decimals. * - As gas consumption by Spawn functionality varies due to the parameter l, we listed the maximum gas consumption value. ** - gas consumption for 24 data owners. *** - gas consumption for an only single iteration of weights computation for 24 data owners.

4.5.3 Implementation of $FairPPA$

The transactional and financial costs of interacting with $FairPPA$ contract are listed in Table 4.2. We have implemented $FairPPA$ as a set of functions such that transaction $\textbf{trans}_x^y$ is sent by a party y to function x. The current Ethereum blockchain network supports calling of the function in a contract only through an Ethereum account or from a function in the same contract or another contract. That is the scheduled function calls which are executed when the timer expires are not possible. So, we implemented timer as two functionalities: $Aggregate$ function, which computes aggregated statistics and the rest of the timer, including DRM is implemented as $Payout$ function. We have taken the $\omega(\cdot)$ and $d(\cdot)$ functions in Equation 4.1 same as [195]. We have implemented Algorithm 4.8 in Java. We created an AspectJ aspect which computes inner state hash without modifying the actual Java code. Algorithm 4.8 is executed along with Aspectj aspect which outputs computed weights and inner state hash. The gas cost for payout functionality in Table 4.2 includes only one round of weight computation.

In Figure 4.7, we show the consumption of gas with respect to the number of data owners. Observe that the gas consumption increases enormously with the increase in the number of data owners.

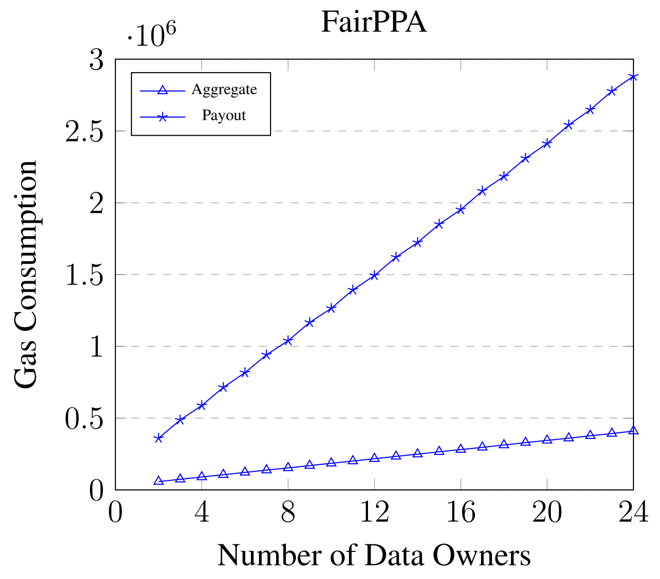

Figure 4.7: Gas Consumption of $FairPPA$ - Aggregate and Payout functionalities. The Payout functionality includes the gas cost of executing $DRM()$.

4.6 Comparison with existing methods

4.6.1 Comparison with privacy-preserving aggregation methods

We compare our methods with some of the state-of-the-art privacy-preserving methods in Table 4.3. Similar to our $FairPPA$ method, the methods in [196, 197, 198] do not require a trusted key dealer. However, they do not have a trusted key establishment platform whereas in our method we use smart contracts as a communication platform for establishing keys. The methods in [199, 200] provides aggregator unforgeability. Nevertheless, [199] require an interactive complexity assumption and [200] require modified computational Diffie–Hellman assumption to provide the aggregator unforgeability. On the other hand, we prove the aggregator unforgeability by assuming the underlying consensus algorithm of a public Blockchain is secure. Observe that our methods also extend PPA towards fair payments when compared to the existing methods.

method	**Trusted key dealer Free**	**Aggregator oblivousness**	**Aggregator unforgeability**	**Dynamic leaves (robustness)**	**Fair Payments**
Shi *et al.* [201]	No	Yes	No	No	No
Acs *et al.* [196]	Yes	Yes	No	Yes	No
Joye *et al.* [189]	No	Yes	No	No	No
Leontiadis *et al.* [197]	Yes	Yes	No	Yes	No
Chen *et al.* [198]	Yes	Yes	No	Yes	No
Benhamouda *et al.* [202]	No	Yes	No	No	No
Leontiadis *et al.* [199]	No	Yes	Yes	No	No
Emura *et al.* [200]	No	Yes	Yes	No	No
$FairNaivePPA$	No	Yes	Yes	No	Yes
$FairPPA$	Yes	Yes	Yes	Yes	Yes

Table 4.3: Comparison of proposed methods with state-of-the-art privacy-preserving methods

method	**Quality evaluation**	**Incentive Mechanism**	**Data owner privacy**	**Data Privacy**	**Aggregator correctness**	**Aggregated Statistics**	**Fair Payments**
CrowdBC [143]	Data buyer	Quality-aware	Psudoanonymity	Encryption	No	No	No
Wang *et al.* [137]	Miners	Quality-aware	k-anonymity		Yes	No	No
Cai *et al.* [138]	SC (miners)	Quality-aware	Psudoanonymity	Encryption	No	Yes	No
ZebraLancer [139]	Data buyer	Quality-aware	Zk-SNARKS	Encryption	Yes	No	Yes
Chatzopoulos *et al.* [141]	TTP	Equal	Pseudoanonymity with TTP	No	No	No	Yes
Wei *et al.* [165]	Data buyer	Quality-aware, Bid-based and Reputation-based	Consortium Blockchain run by trusted agents	Encryption	No	No	No
Hu *et al.* [147]	SC (Miners)	Quality-aware and Reputation-based	Pseudoanonymity	No	Yes	No	Yes
SenseChain [148]	SC (Miners)	Quality-aware	Pseudoanonymity	No	Yes	No	Yes
CrowdBLPS [150]	Data buyer	Quality-aware	Pseudoanonymity	Encryption	No	No	No
$FairNaivePPA$	-	Equal	Pseudoanonymity	Homomorphic encryption	Yes	Yes	Yes
$FairPPA$	SC (Miners)	Quality-aware	Pseudoanonymity	Homomorphic encryption	Yes	Yes	Yes

Table 4.4: Comparison of proposed methods with existing Blockchain-based mobile crowdsensing methods.

4.6.2 Comparison with Blockchain-based mobile crowdsensing methods

The comparison with existing Blockchain-based MCS methods is presented in Table 4.4. Observe that apart from Cai *et al.* [138] method, only our methods provide aggregated statistics. Zebralancer [139], Hu *et al.* [147], and SenseChain [148] provide both aggregator correctness and fair payments; however they do not provide aggregator statistics.

4.7 Summary

We have designed two protocols for fair payments in the privacy-preserving aggregation of the mobile crowdsensing. Our protocols show that the untrusted aggregator in traditional PPA methods can be replaced by smart contracts deployed on a public Blockchain network. Moreover, our protocols guarantee the correctness of the aggregation operation and fair payments without any additional cryptographic operations or trusted intermediaries when compared to traditional PPA methods. We have shown our protocols' feasibility by deploying them on a private Ethereum network and listed the transactional and financial cost of interacting with smart contracts.

Chapter 5

Fair Payment Protocol for Virtual Machine Allocation under Infrastructure-as-a-Service

Cloud computing offers on-demand network access of configurable computing resources (virtual machines (VM)), enabling individuals and enterprises to pay only for the resources or services they use. Resource allocation to cloud users depend on several factors, like resource utilization, resource pricing, availability, and quality of service. Among all the factors, resource pricing mechanisms are widely studied because they increase the cloud provider's utility [102]. In recent years, auction-style resource pricing mechanisms [117, 118, 119] have gained more interest as they reflect the underlying trends in demand and supply of the cloud resources. Auction mechanisms are categorized into two types: (1) off-line and (2) online. In off-line auctions, all the users' requests are collected, and then the auctioneer decides on allocation and price of VMs. (2) In online auctions, users' requests are processed instantly without prior knowledge of future requests. Online auctions are extensively studied than off-line auctions because online setting provides faster services and can efficiently allocate and price the resources [203].

However, most of the existing online auction methods have a resource allocator, as shown in Figure 5.1, which processes user request by executing auction algorithm. In the existing online auction setting, the request allocator is assumed as a trusted entity and does not consider: (1) auction correctness: running the auction algorithm correctly without any prejudice. (2) fairness: the provider receives the usage fee for his service if and only if the user receives his requested resources. One of the significant challenges in an auction is a lack of trust among users and providers. The users do not trust the provider for allocation of VMs, and the provider does not trust the user for getting payment. Therefore the resource allocator must be run by a trusted party to have greater confidence in the auction mechanism. However, in practical situations hiring a trusted party is costly and finding an ideal trusted party that will behave honestly is difficult. The solution is to distribute trust among multiple entities instead of a single centralized trusted resource allocator. The progress in Blockchain technology presents an alternate solution for using the services of a trusted party [10].

In this Chapter, we present a Blockchain-based online auction for cloud VM allocation and pricing. We realize both correctness and fairness of the auction mechanism by taking advantage of two key components of a Blockchain network: smart contracts and cryptocurrency. We achieve correctness by modeling the auction algorithm as a smart contract running on a public Blockchain network. We achieve fairness by carefully encoding the smart contract rules about the payment and allocation of VMs.

Our contributions in this Chapter are summarized as follows:

(a) To the best of our knowledge, we are the first to propose a Blockchain-based online cloud resource auction protocol by leveraging the trust, immutability and correctness properties of the public Blockchain networks.

(b) As most of the existing online auctions are truthful and focus on optimization of utility and social welfare / cost, in this work, we focus on fair payments and correctness of the online auction algorithm by modeling the auction algorithm as smart contract running on a public Blockchain network.

(c) We have implemented the proposed smart contract written in solidity [24] and executed them on a private Ethereum network and on Ropsten test network. We have tested the proposed smart contract and presented the transaction and financial costs

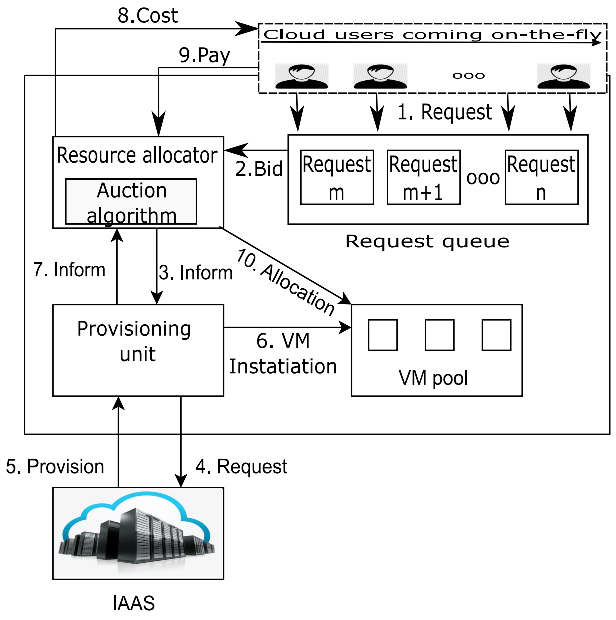

Figure 5.1: Online auction infrastructure and resource allocation flow

of interacting with proposed smart contract. Experimental results show the feasibility of our proposed smart contract with minimal financial overhead.

5.1 Online auction

Let CP be a cloud resource provider who has a large number of computational resources with a fixed capacity Q in an infinite time interval $[0, \infty]$. Let CU_i be a cloud user who would like to use the resources provided by the CP. Let $DeOAA$ be a cloud auctioneer who facilitates and executes auction mechanism $\mathcal{A}$. An auction is said to be an online auction if the allocation and payment for a CU_i's request is determined instantaneously according to some adopted auction mechanism $\mathcal{A}$. A simple online auction algorithm is shown in Algorithm 5.12 and the online auction infrastructure is shown in Figure 5.1.

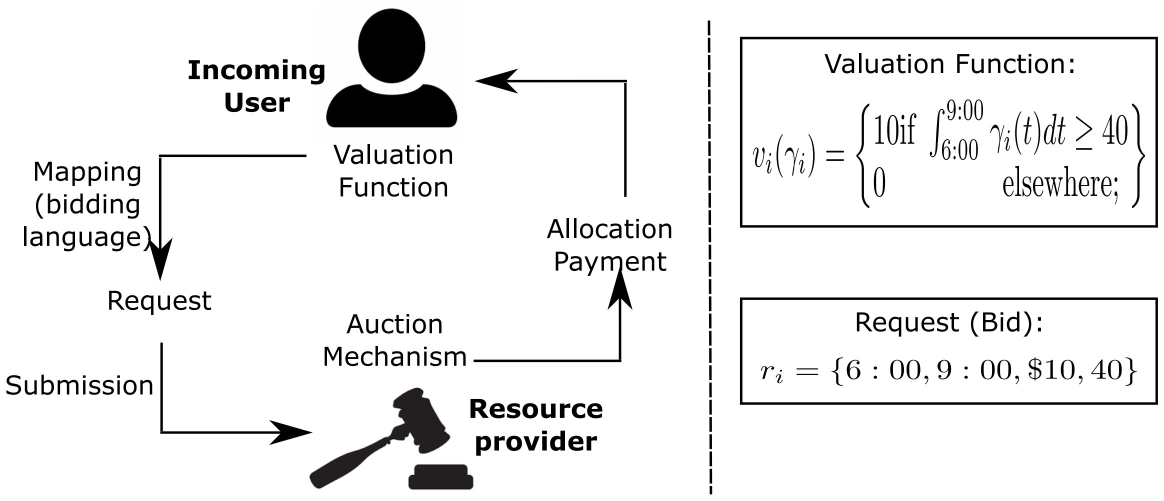
Incoming User
Valuation Function
Mapping (bidding language)
Request
Submission
Auction Mechanism
Allocation Payment
Resource provider
Valuation Function:
$v_i(\gamma_i) = \begin{cases} 10 \text{if } \int_{6:00}^{9:00} \gamma_i(t)dt \geq 40 \\ 0 \quad \text{elsewhere;} \end{cases}$
Request (Bid):
$r_i = \{6:00, 9:00, \$10, 40\}$

Figure 5.2: An illustrative example of online auction [119]

Algorithm 5.12 Simple online auction

```
   Input: A sequence of requests $R = \{r_1, r_2, ..., r_\infty\}$, such that $t_{sub_1} < t_{sub_2} < ... < t_{sub_\infty}$;
       A non decreasing pricing function $P(x)$
   Output: Allocation and payment decision for every request $r_i \in R$.
1: Initialize the utilization rate: $\forall\, t \in [0, \infty], U(t, t_{current}) = 0$;
2: for each request do
3:     if the requested resources are free in the requested time slots then
4:         Find the allocation that maximizes requester's utility;
5:         Compute payment for the allocation using an auxiliary pricing function $P(x)$ and utilization rate
             $U$ of the requested time slots;
6:     else Reject the request;
7:     Reserve the requested resources and update the utilization rate;
```

In the algorithm, initially, utilization rate of the cloud resources for every future time slot is set as zero. The utilization rate is the ratio of allocated and total resources at time $(t, t_{current})$, where $t_{current}$ is the current time and t is the time in future. Price of a resource varies with respect to the utilization rate. The cloud users come on-the-fly and requests the cloud resources. The request consists of information about the start time, end time, valuation and number of resources required. The auction platform process the requests in a sequence. First, the algorithm checks whether the requested resources are available in the requested time slots. If resources are available, then a best allocation is computed according to the user's valuation. The usage fee pay is computed according to a pricing function and the utilization rate at the allocated time slots. An example of the online auction is shown in Figure 5.2. A cloud user requires 40 VMs in every time slot from $6:00$ to $9:00$. He sends his request to the auction platform which determines the allocation and payment pay immediately according to the adopted auction mechanism. He values his request at $\$10$ if all the requested VMs are allocated in the requested time slots. The valuation represents the benefit a cloud user obtains from receiving the cloud resources. Also, utility represent the net profit a cloud user gets from an allocation, that is utility = valuation - usage fee.

Definition 5.1.1. *A fair online auction protocol must provide the following guarantees:*

(a) ***Online****: An auction protocol is said to be online if it provides the flexibility for users to request cloud resources whenever they need, and their requests are processed by the cloud provider instantaneously.*

(b) ***Correctness of auction mechanism****: An auction protocol is said to be correct if the following three factors determine the allocation and payment for any user: (1) the request by CU_i (2) the requests which have been accepted before the CU_i's request (3) auxiliary pricing function $P(x)$. In other words, no party (CU_i / CP / miners) can influence the outcome of the auction except the above three factors.*

(c) ***Fair Payments****: An auction protocol is said to be fair if an honest CP receives the usage fee for the resources it leases if and only if an honest CU_i receives the requested resources.*

5.1.1 Entities

A Blockchain-based fair payment protocol for online auction of cloud resources consists of the following entities:

(a) Cloud resources: A large cloud data center consists of resources like CPU, RAM, storage, and bandwidth.

(b) Cloud Service Provider (CP) : A cloud service provider bundles the cloud resources as virtual machines and leases the virtual machines to users.

(c) Cloud user (CU_i): A cloud user requests for using resources provided by CP. If the request is accepted, then he pays usage fee to CP and uses the allocated resources.

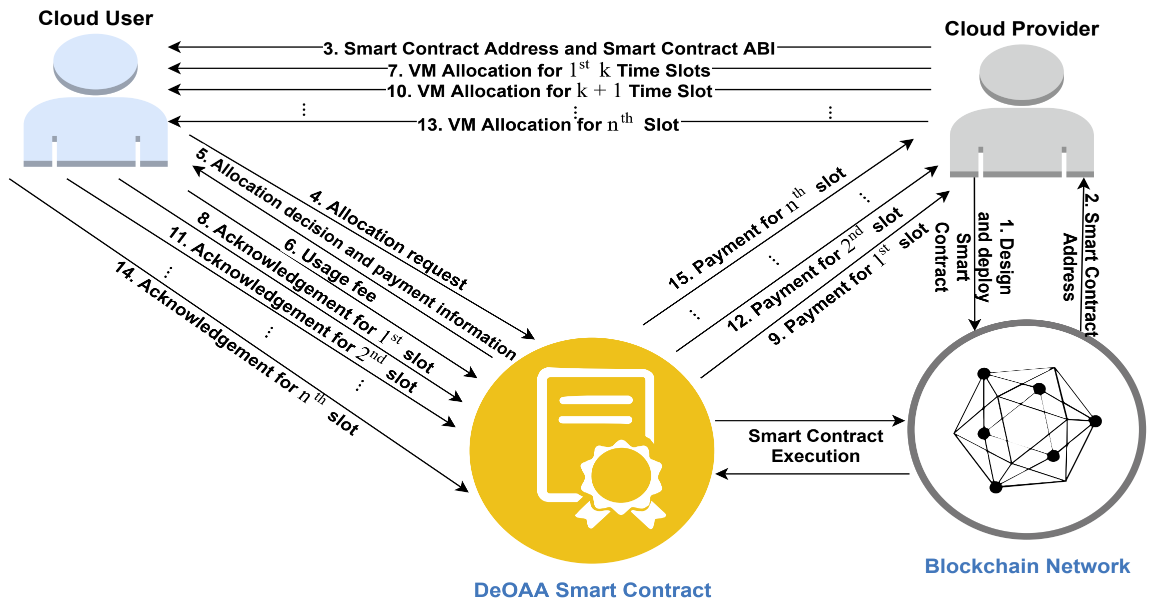
Cloud User
Cloud Provider
3. Smart Contract Address and Smart Contract ABI
7. VM Allocation for 1^{st} k Time Slots
10. VM Allocation for k + 1 Time Slot
13. VM Allocation for n^{th} Slot
4. Allocation request
5. Allocation decision and payment information
6. Usage fee
8. Acknowledgement for 1^{st} slot
11. Acknowledgement for 2^{nd} slot
14. Acknowledgement for n^{th} slot
15. Payment for n^{th} slot
12. Payment for 2^{nd} slot
9. Payment for 1^{st} slot
1. Design and deploy Smart Contract
2. Smart Contract Address
Smart Contract Execution
DeOAA Smart Contract
Blockchain Network

Figure 5.3: Overview of the proposed protocol

(d) Smart Contract: A smart contract emulates the trusted auction platform in Figure 5.1. It receives requests from users and computes the usage fee to be paid by CU_i for using resources provided by the CP.

(e) Blockchain Network (BC): It is maintained by a set of peers known as miners who execute the smart contract functionalities according to an underlying consensus algorithm.

The overview of the proposed protocol is presented in Figure 5.3.

5.2 Bidding language

We adopt the bidding language proposed by Zhang *et al.* [119]; more particularly, we adopt TYPE III users as described in [119]. TYPE III users are resource-aggressive with time-invariant capacity requirements. CU_i may request the auction platform for cloud resources of invariable capacity inv_cap_i for a time length of l_i within a preferred time duration $[a_i, d_i]$ ($l_i \leq d_i - a_i$). The request is organized as: $r_i = \{a_i, d_i, l_i, inv\_cap_i, v_i\}$ where $v_i = b_i(inv\_cap_i) * l_i$ is the user's valuation and $b_i(\cdot)$ is a concavely increasing function. One can refer to [119] for more details about bidding language. Although, we have chosen TYPE III users our smart contract can process other type of requests with simple modifications.

5.3 Decentralized online auction protocol ($DeOAA$)

In this section, we design a fair Blockchain-based online auction protocol by modeling the trusted resource allocator in Figure 5.1 as a smart contract running on a public Blockchain network.

5.3.1 Assumptions

(a) We assume that the resources like CPU, RAM, storage and bandwidth are bundled as Virtual Machines(VM) or instances, and the cloud provider offers only a single instance type. Although we are assuming a single type of VM, generalization to any number of types of VMs is straightforward.

(b) We assume that the user and cloud provider is aware of the discrete timer running on the public Blockchain network, which is different from the real-world timer.

(c) We assume that the number of time slots for a request is more than a predefined parameter k.

(d) We treat the auxiliary pricing function as a black box, as any monotonic pricing function commonly used in traditional online auction mechanisms can also be used in our protocol.

5.3.2 $DeOAA$ contract clauses

The $DeOAA$ is an online auction contract signed between a cloud service provider CP and a cloud user CU_i. The high-level idea is that if both CP and CU_i are honest, CP receives the usage fee for leased resources, and CU_i receives the requested resources.

The clauses in the $DeOAA$ contract are as follows:

(i) A cloud provider CP creates a smart contract $DeOAA$ for online auction and deploys the $DeOAA$ on a public Blockchain network. CP publishes the contract address on an open platform (like a website/bulletin board). CP initializes $DeOAA$ with parameters like the cloud capacity in terms of virtual machines (VM) and an auxiliary pricing function that is used to compute the usage fee for a user.

(ii) After verifying the contract details at the contract address, a user CU_i sends his request to $DeOAA$ along with some safety deposit $\$d$. This safe deposit is required to penalize users for sending false requests. The request includes a preferred start time, preferred end time, required number of time slots, required number of VMs, and valuation of the user.

(iii) As soon as the $DeOAA$ receives the user request, it assigns a submission time to the request. $DeOAA$ finds the best allocation for the CU_i's request, which maximizes CU_i's utility. If there are enough VMs in every time slot to satisfy CU_i's request, then those VMs are reserved, and the corresponding allocation decision and usage fee are communicated to CU_i. If $DeOAA$ cannot find the allocation for a request, then the request's deposit is refunded.

(iv) If VMs are reserved to a request, then the user of that request has to send the usage fee to $DeOAA$ before the actual start time of the request. If the user sends the usage fee successfully, then his deposit $\$d$ is returned.

(v) If the user fails to send the usage fee before the actual start time of the request, CP will send a message to $DeOAA$ to free up the reserved VMs. In this case, CU_i forfeits his deposit which will be sent to CP. Otherwise, CP allocates the VMs for first k time slots. The cloud provider allocates VMs for the next time slot if the user sends acknowledgements of the previous allocation to $DeOAA$. The allocation is like a sliding window. For every acknowledgement received, VMs for the next time slot is allocated by the CP. The process will continue till the end of all the time slots of the request. The user cannot send the acknowledgement of i^{th} slot without sending the acknowledgement of $(i-1)^{th}$ slot. The payment to CP is also divided into parts equal to the number of time slots. The CP receives the usage fee for allocating the resources in a time slot as soon as the $DeOAA$ receives that particular time slot acknowledgement.

(vi) The maximum time slots that can be allocated without acknowledgement are k. For every time slot $i \geq$ (start time + k), if the user fails to send $(i-k)^{th}$ slot acknowledgment before the start of $(i-k-1)^{th}$ slot, the CP will not allocate resources from i^{th} time slot and the resources are freed up for the remaining time slots. The remaining usage fee for the unused time slots is refunded to CU_i.

5.3.3 $DeOAA$ protocol

$DeOAA$ protocol is presented in Figure 5.4. $DeOAA$ smart contract functionalities executed by Blockchain are presented as Algorithms 5.13 to 5.18.

For cloud provider:

1. To initiate the auction process send **trans**$^{CP}_{create}$ = $(capacity, d)$ to BC.
2. To free up the virtual machines send **trans**$^{CP}_{free}$ = (rId) to BC.
3. If CU neither sends acknowledgements for the alloted slots nor sends abrogate transaction, then send **trans**$^{CP}_{CPAbrogate}$ = (rId).

For cloud user:

4. To request virtual machine allocation, send **trans**$^{CU_i}_{request}$ = $(a_i, d_i, l_i, v_i, inv\_cap_i, \$d)$ to BC.
5. To avail the resources, send **trans**$^{CU_i}_{avail}$ = (rId, $\$pay$) to BC.
6. To acknowledge the alloted slot, send **trans**$^{CU_i}_{ack}$ = $(rId, ackNum)$ to BC.
7. If CP fails to allocate the requested virtual machines, then send **trans**$^{CU_i}_{abrogate}$ = (rId).

Blockchain: $state \leftarrow Init$, $Q \leftarrow 0$, $\forall t \in [\tau, \infty]$ $U[t][\tau] \leftarrow 0$, $\forall t \in [\tau, \infty]$ $allocated[t] \leftarrow 0$, $requests \leftarrow \{\}$, $rId \leftarrow 0$, $deposit \leftarrow 0$

8. On receiving **trans**$^{CP}_{create}$ execute $DeOAA.create(capacity, d)$
9. On receiving **trans**$^{CU_i}_{request}$ execute $DeOAA.request(a_i, d_i, l_i, v_i, inv\_cap_i, \$d)$
10. On receiving **trans**$^{CU_i}_{avail}$ execute $DeOAA.avail(rId, \$pay)$
11. On receiving **trans**$^{CU_i}_{ack}$ execute $DeOAA.ack(rId, ackNum)$
12. On receiving **trans**$^{CP}_{free}$ execute $DeOAA.free(rId)$
13. On receiving **trans**$^{CU_i}_{abrogate}$ execute $DeOAA.abrogate(rId)$
14. On receiving **trans**$^{CP}_{abrogate}$ execute $DeOAA.CPAbrogate(rId)$

Figure 5.4: $DeOAA$ protocol

5.3.3.1 Initializing the smart contract

The cloud provider CP deploys the smart contract initialized with global parameters Q, U, $allocated$, $requests$, and rId. Q is the maximum number of VMs that CP can provide for each time slot. We allow reserving resources before the actual start time of a request. We use the utilization matrix U to track utilization rates at each time slot with respect to the submission time of the request. $allocated$ is an array to keep track of the number of VMs allocated at each time slot. $requests$ is a structure to store all the request related parameters, rId is the number of requests submitted, and $deposit$ is the minimum amount of safety deposit to be paid by the users. At this stage, the contract will be in the $Init$ state. CP invokes Algorithm 5.13 to set the parameters Q and $deposit$. Now, the contract state will be changed to $Created$. The create functionality provides the flexibility for the CP to update the parameters Q, and $deposit$ in the future.

Algorithm 5.13 $DeOAA.create$

```
    Input: $capacity$, $d$
    Output: Success or failure message
1: if $state = Init \,||\, Created$ then
2:     Set $Q \leftarrow capacity$
3:     Set $deposit \leftarrow d$
4:     Set $state \leftarrow Created$
```

5.3.3.2 Submission of a request

A cloud user sends his request to BC invoking Algorithm 5.14. After receiving the request, the contract makes sure that the preferred timings of the request are greater than the current time. The contract also checks whether the CU_i has sent the minimum safety deposit. If all the checks are passed, then the contract finds the best allocation that maximizes CU_i's utility according to his request and the current utilization rate. If there is an allocation (i.e., VMs are available for the requested number of time slots), then the actual start time, end time, usage fee to be paid are computed. The $allocated$ array and U matrix are updated according to new allocation, and the request state is set to $Accepted$. If there is no allocation, then the user's deposit is refunded, and the request state is set to $Rejected$. The bidding information is stored as a structure, and the user is notified about the allocation decision and the usage fee. In the end, the requests count is updated.

Algorithm 5.14 $DeOAA.request$

```
    Input: $a_i, d_i, l_i, v_i, inv\_cap_i, \$d$
    Output: Success or failure message
 1: if $state = Created$ then
 2:     if $\tau < a_i \leq d_i$ then
 3:         if $\$d \geq deposit$ then
 4:             if $ledger[CU_i] \geq \$d$ then
 5:                 Set $ledger[CU_i] \leftarrow ledger[CU_i] - \$d$
 6:                 Set $\tau_{sub} \leftarrow \tau$, $end \leftarrow 0$, $start \leftarrow a_i$, $maxUtility \leftarrow 0$, $pay \leftarrow 0$
 7:                 while $end \leq d_i$ do
 8:                     Set $end \leftarrow start + l_i - 1$
 9:                     Set $flag \leftarrow true$
10:                     foreach $t \in [start, end]$ do
11:                         if $allocated[t] + size \geq Q$ then
12:                             Set $flag \leftarrow false$
13:                     if $flag$ then
14:                         Set $isAllocated \leftarrow flag$
15:                         Set $tpay \leftarrow \int_{start}^{end} \int_{U(t,\tau_{sub})}^{U(t,\tau_{sub})+inv\_cap_i/Q} P(x) \cdot Q \, dxdt$ where $t \in [start, end]$
16:                         Set $tUtility \leftarrow v_i - tpay$
17:                         if $tUtility > maxUtility$ then
18:                             Set $maxUtility \leftarrow tUtility$, $pay \leftarrow tpay$
19:                             Set $\tau_r^- \leftarrow start$, $\tau_r^+ \leftarrow end$
20:                         Set $start \leftarrow start + 1$
21:                 if $isAllocated$ then
22:                     foreach $t \in [\tau_r^-, \tau_r^+]$ do
23:                         Set $allocated[t] \leftarrow allocated[t] + inv\_cap_i$
24:                         Set $U(t, \tau_{sub}) = U(t, \tau_{sub}) + allocated[t]/Q$
25:                     Set $rState \leftarrow Accepted$
26:                 else
27:                     Set $pay \leftarrow 0$
28:                     Set $ledger[CU_i] \leftarrow ledger[CU_i] + \$d$
29:                     Set $rState \leftarrow Rejected$
30:                 foreach $t \in [\tau_r^-, \tau_r^+]$ do
31:                     Set $ack[t] \leftarrow false$
32:                 Set $\$payment \leftarrow 0$
33:                 Set $requests[rId] \leftarrow (a_i, d_i, \tau_r^-, \tau_r^+, \tau_{sub}, v_i, inv\_cap_i, rState, pay, ack, \$payment,
                        \$d)$
34:                 Set $rId \leftarrow rId + 1$
35:                 return (Success, Reveal success)
36:             else return (Failure, Not enough balance)
37:         else return (Failure, Not enough deposit)
38:     else return (Failure, Wrong timing parameters)
39: else return (Failure, State is not created)
```

5.3.3.3 Sending the usage fee

The user CU_i upon learning the usage fee, sends payment to BC invoking Algorithm 5.15. After receiving the CU_i's transaction, the contract checks whether the usage fee has been sent before the actual start time of the request. The contract also checks whether the usage fee sent is greater than or equal to the value computed earlier. One more check is on the request state which should be in $Accepted$ state. If all these checks are passed, then the user's deposit is refunded, and the request state is changed to $Payed$.

Algorithm 5.15 $DeOAA.avail$

```
    Input: $rId$, $\$pay$
    Output: Success or failure message
 1: if $requests[rId].rState = Accepted$ then
 2:     if $requests[rId].\tau_r^- > \tau$ then
 3:         if $requests[rId].pay \leq \$pay$ then
 4:             if $ledger[CU_i] \geq \$pay$ then
 5:                 Set $ledger[CU_i] \leftarrow ledger[CU_i] - \$pay$
 6:                 Set $requests[rId].\$payment \leftarrow \$pay$
 7:                 Set $ledger[CU_i] \leftarrow ledger[CU_i] + requests[rId].\$d$
 8:                 Set $requests[rId].rState \leftarrow Payed$
 9:                 return (Success, Avail success)
10:             else return (Success, Avail success)
11:         else return (Failure, Pay not enough)
12:     else return (Failure, Avail timeout)
13: else return (Failure, Request not accepted)
```

5.3.3.4 Acknowledging the allocation

If CP allocates VMs to CU_i, the user sends acknowledgements about the allocated slots to BC invoking Algorithm 5.16. For each acknowledgement received the contract checks whether the received acknowledgement number is between the actual start and end times of the request. The contract also checks whether the previous allocation slot has been acknowledged or not. The acknowledgement for the i^{th} slot cannot be sent before acknowledging the $(i-1)^{th}$ slot unless it is an acknowledgement for the 1^{st} time slot. If all the checks are passed, the CP gets the usage fee for that slot from the contract. If all the acknowledgements are received, then the request state is changed to $Finished$.

Algorithm 5.16 $DeOAA.acknowledge$

```
    Input: $rId$, $ackNum$
    Output: Success or failure message
 1: if $requests[rId].\tau_r^- \leq ackNum \leq requests[rId].\tau_r^+$ then
 2:     if $requests[rId].rState = Payed$ then
 3:         if $ackNum = requests[rId].\tau_r^-$ then
 4:             Set $requests[rId].ack[ackNum] \leftarrow true$
 5:             Set $ledger[CP] \leftarrow ledger[CP] + \frac{requests[rId].\$payment}{requests[rId].l_i}$
 6:             return (Success, Acknowledge success)
 7:         else
 8:             if $requests[rId].ack[ackNum - 1] = true$ then
 9:                 Set $ledger[CP] \leftarrow ledger[CP] + \frac{requests[rId].\$payment}{(requests[rId].l_i)}$
10:                 Set $requests[rId].ack[ackNum] \leftarrow true$
11:                 return (Success, Acknowledge success)
12:         if $ackNum = request[rId].\tau_r^+$ then
13:             Set $requests[rId].rState \leftarrow Finished$
14:     else return (Failure, State is not payed)
15: else return (Failure, Wrong acknowledgement number)
```

5.3.3.5 Freeing up the resources

If CU_i, refuses to send usage fee even though VMs are reserved for his request then CP calls the BC to invoke Algorithm 5.17 to free the allocated resources and claim the deposit made by CU_i. CP can call Free functionality only if the request state is $Accepted$ and only after the actual start time of the request. If checks are valid, then the previously allocated resources are deallocated. The deposit made by CU_i is sent to CP, and the request state is set to $Aborted$.

Algorithm 5.17 $DeOAA.free$

```
    Input: $rId$
    Output: Success or failure message
 1: if $requests[rId].\tau_r^- < \tau$ then
 2:     if $requests[rId].rState = Accepted$ then
 3:         foreach $t \in [start, end]$ do
 4:             Set $allocated[t] \leftarrow allocated[t] - inv\_cap_i$
 5:             Set $U(t, \tau_{sub}) = U(t, \tau_{sub}) - allocated[t]/Q$
 6:         Set $ledger[CP] \leftarrow ledger[CP] + requests[rId].\$d$
 7:         Set $requests[rId].rState \leftarrow Aborted$
 8:         return (Success, Free success)
 9:     else return (Failure, Request not accepted)
10: else return (Failure, Free timeout)
```

5.3.3.6 Abrogate by the user

If CP refuses to allocate VMs or allocates VMs less than requested, then CU_i calls the BC invoking Algorithm 5.18 to redeem the usage fee paid by him. The number of acknowledged slots are calculated, and deallocation of resources is invoked with the last acknowledged slot and the payment belonging to the unacknowledged slots is sent to CU_i. The request state is set to $userAborted$.

Algorithm 5.18 $DeOAA.abrogate$

```
Input: $rId$
Output: Success or failure message
1: if $\tau > requests[rId].\tau_r^-$ then
2:     if $requests[rId].rState = Payed$ then
3:         foreach $t \in [requests[rId].\tau_r^-, requests[rId].\tau_r^+]$ do
4:             if $requests[rId].ack[t] = false$ then
5:                 Set $NotAckSlots \leftarrow NotAckSlots + 1$
6:             Set $lastAckSlot \leftarrow requests[rId].\tau_r^+ - requests[rId].\tau_r^- + NotAckSlots$
7:             if $lastAckSlot + k \leq \tau$ then
8:                 foreach $t \in [start, end]$ do
9:                     Set $allocated[t] \leftarrow allocated[t] - inv\_cap_i$
10:                    Set $U(t, \tau_{sub}) = U(t, \tau_{sub}) - allocated[t]/Q$
11:                if caller is cloud user then
12:                    Set $ledger[CU_i] \leftarrow ledger[CU_i] + \frac{requests[rId].\$payment}{(requests[rId].l_i)} * NotAckSlots$
13:                    Set $requests[rId].rState \leftarrow userAborted$
14:                else
15:                    Set $ledger[CP] \leftarrow ledger[CP] + \frac{requests[rId].\$payment}{(requests[rId].l_i)} * NotAckSlots$
16:                    Set $requests[rId].rState \leftarrow Terminated$
```

5.3.3.7 Abrogate by the cloud provider

If CU_i neither sends acknowledgements nor calls abrogate functionality, then CP calls BC invoking Algorithm 5.18, sending the payment belonging to unacknowledged slots to CP.

5.3.4 Correctness and fairness proofs

Now we prove the correctness and fairness of the proposed protocol defined above.

Theorem 5.3.1. *Our proposed protocol satisfies correctness*

Let $DeOAA$ deployed on a public Blockchain network using PoW as a consensus algorithm. Let b be the current block of the Blockchain which is extended by blocks b_1 and b_2.

Let $tx_{request}$ be the request transaction and tx_{avail} be the avail transaction sent to $DeOAA$ which are eventually placed in b_1 and b_2 respectively. We will prove the correctness by considering the following cases.

Case 1: CU_i influence the auction mechanism to decrease the pay to be paid by him. This case can happen if the CU_i pre-mines two blocks $b_1^{'}$ and $b_2^{'}$ containing transactions $tx_{request}^{'}$ and $tx_{avail}^{'}$ respectively with a modified pay $pay^{'}$. It may be noted that, in this case, CU_i does not broadcast transactions to the network. Now, CU_i broadcast $b_1^{'}$ extending b and $b_2^{'}$ extending $b_1^{'}$ to the Blockchain network. Let all the other miners in the network verify the transactions in $b_1^{'}$ for the common good and add this block to their local distributed ledger if all the transactions in $b_1^{'}$ and their outputs are correct[1]. So, as the output of $tx_{request}$ is pay but not $pay^{'}$ other miners will discard this block without adding new blocks extending $b_1^{'}$. As $b_1^{'}$ is discarded, $b_2^{'}$ is also discarded because it is extending a wrong block. Miners with at least 51% of hash-rate cumulatively required to correctly verify the transactions in the block.

Case 2: CU_i broadcast $tx_{request}$ but pre-mines block $b_2^{'}$ that consists $tx_{avail}^{'}$ with a modified pay $pay^{'}$. In this case, as the actual pay is already stored in $DeOAA$, during $b_2^{'}$ verification, the $pay^{'}$ is compared against pay (Figure 5.15, Line 3). The comparison will fail, and the block is rejected.

Case 3: CP influences the auction mechanism to increase the pay to be paid by CU_i. This case is similar to Case 1 except that now CP mines $b_1^{'}$ and broadcasts it to network.

Similarly, all the transactions with $DeOAA$ are executed correctly; otherwise, those transactions are rejected. Nevertheless, CU_i or CP can influence some miners to include the wrong blocks to their local ledger and generate new blocks extending these wrong blocks. But, CU_i or CP should accumulate more than 50% of hash rate to make the entire network accept wrong blocks which are a difficult task unless they have large mining pools under their control [204].

In summary, as the miners in public Blockchain networks are reasonably assumed to act honestly for the common good and follow the rules encoded in consensus algorithm, it is difficult for CU_i or CP to make the network accept wrong blocks. Considering the above

[1]These rules are encoded in the consensus algorithm.

cases, our protocol satisfies correctness as defined in Definition 5.1.1.

Theorem 5.3.2. *Our proposed protocol satisfies fair payments*

We will prove the fairness by considering the following cases.

Case 1: CP is malicious, and CU_i is honest. In this case, if CP refuses to allocate VMs according to the CU_i's request, then the CU_i will not send acknowledgements to unallocated time slots. CU_i can retrieve his pay paid earlier by sending a message to Abrogate functionality. As we know, $DeOAA$ pays to CP if and only if it receives an acknowledgement for a particular time slot. So, CU_i will not lose any money even if the CP acts maliciously without allocating requested VMs. Thus the fairness holds.

Case 2: CP is honest, and the CU_i is malicious. In this case, CP allocates the requested number of VMs, but CU_i acts maliciously and refuses to send acknowledgements for the allocated VMs. CP waits for k number of time slots, even then if CU_i fails to send acknowledgements of previously allocated VMs, then CP will not allocate any further VMs. Let $\$p$ be the amount to be paid by CU_i for each time slot; then CP will incur a tiny loss of $(k * \$p)$ for the resources allocated for the unacknowledged k time slots. Thus the fairness holds in this case with a $(k * \$p)$ loss to CP.

Case 3: CP is malicious, and CU_i is also malicious. This case is straightforward and similar to the above two cases.

Case 4: CP is honest, and CU_i is also honest. In this case, CU_i sends an acknowledgment to all the time slots that are requested and satisfied by CP. CU_i gets the VMs, and CP receives the payment for the allocated resources. Thus the fairness holds.

In summary, our protocol satisfies the fair payments.

5.4 Implementation

The simulation environment is discussed in Section 1.2.3. We have also deployed the contracts on the Ethereum Ropsten test network.

5.4.1 Floating point numbers

Ethereum solidity smart contract language does not support floating-point numbers. We have taken all the input values to the Request functionality as integers, but the utilization matrix values are floating-point values. Therefore the intermediate values generated during computation of payment are floating-point numbers. To handle these floating-point values, we have used ABDK Math Quad smart contract library[2].

5.4.2 Implementation of $DeOAA$

We ran our experiments multiple times, and each transaction's computational and financial cost is shown in Table 5.1. Observe that the contract deployment consumes enormous amounts of gas, but this is a one time cost for CP. Next, call to the request functionality incurs a huge cost to CU_i, this is due to the computation of payment function in Algorithm 5.14 using Simpson 1/3 rule. We have computed gas cost for Request functionality separately in local Blockchain network by varying both $(d_i - a_i)$ and l_i and listed the costs in Table 5.2. Observe that the costs are enormous and more than the Ethereum block gas limit. Most of these high costs are due to the computation of Algorithm 5.14. To reduce the costs, some other payment computation algorithm may be adopted. Another option to reduce the cost is increasing the step size of the integral. We have chosen 0.01 and 1 as a step size of inner integral and outer integral, respectively.

Function	**Caller**	**Cost in gas**	**Cost in $\$$**
Init (contract deployment)	CP	3,483,962	0.547
Create	CP	27,383	0.004
Request*	CU_i	1,837,360	0.288
Avail	CU_i	58,648	0.009
Acknowledge	CU_i	57,388	0.009
Free*	CP	99,834	0.015
Abrogate**	CU_i	49,684	0.007

Table 5.1: Costs of interacting with $DeOAA$ contract. We have approximated the gas price as 1 Gwei and 1 ETH = $\$$ 157.01, which are the real world costs in April 2020. We have rounded off the cost in $\$$ value up to three decimals. * - depends on $(d_i - ai)$ and l_i. ** - depends on l_i. We have taken $(d_i - a_i) = 5$ and $l_i = 4$

[2]https://github.com/abdk-consulting/abdk-libraries-solidity/blob/master/ABDKMathQuad.sol

$(d_i - a_i)$ l_i	**10**	**15**	**20**	**25**	**30**	**35**	**40**	**45**	**50**
1	2173478	3089484	3990024	4906529	5839211	6740594	7657853	8575347	9493075
2	2665707	3937249	5208757	6481330	7754254	9029398	10302445	11576590	12851086
3	3125388	4813598	6500951	8204563	9878714	11568429	13273542	14949149	16640992
4	3384653	5514963	7614702	9715805	11817699	13920188	16023238	18126993	20231404
5	3497380	6055547	8571281	11086227	13617833	16120191	18609243	21157598	23663326
6	3515706	6442044	9353608	12281743	15210934	18141836	21042406	24004724	26953125
7	3297068	6635919	9932698	13319201	16587845	20022563	23323736	26716148	30049877
8	2925568	6663405	10843283	14185407	17941733	21699664	25459145	29220176	32982758
9	2390432	6541064	10736967	14889765	19059887	23217734	27361583	31476531	35758914
10	1659823	6055547	10843283	15423243	20005551	24589711	29176113	33719694	38355335

Table 5.2: Gas consumption of interaction with Request functionality.

$\boldsymbol{k}$	$\boldsymbol{\tau_{sub}}$	$\boldsymbol{a_i}$	$\boldsymbol{d_i}$
3	$block.number$	$[\tau_{sub}$,500]	$[a_i + k$,min$(a_i + 10$,500]
$\boldsymbol{l_i}$	$\boldsymbol{inv\_cap\_i}$	$\boldsymbol{p_i}$	$\boldsymbol{b_i(\cdot)}$
[1,min$(d_i - a_i + 1, 5)$]	[100,10000]	[1,2]	$p_i * inv\_cap_i * l_i$

Table 5.3: Implementation configuration

5.4.3 Financial overhead

We have computed the proposed protocol's financial overhead by varying the requests received by the $DeOAA$. We consider $Q = 10^4$ (i.e., the provider can host up to 10^4 VMs simultaneously). All the bidding parameters are uniformly distributed, and their possible ranges are presented in Table 5.3. Figure 5.5 shows the financial overhead of $DeOAA$ compared with the results obtained without $DeOAA$. Let E and E_{DeOAA} be the sum of all payments paid by users without $DeOAA$ and with $DeOAA$ respectively such that $E_{DeOAA} = E - Cost(Request + Avail + Acknowledge)$. Observe that the ratios are closer to 1, which means the financial overhead of deploying an online auction using a smart contract is minimal[3].

5.4.4 Deploying on Ropsten test network

We have also deployed $DeOAA$ in Ethereum Ropsten test network, interacted with the deployed contract and the transaction hashes / address are given in Table 5.4. The transactions can be verified using the transaction address at https://ropsten.etherscan.io/. We have tested

[3]We have taken the payment, valuation, cost of a VM and transaction cost in U.S. dollars($). The fractional part of the values in Figure 5.5 may change depending on the denomination of the currency.

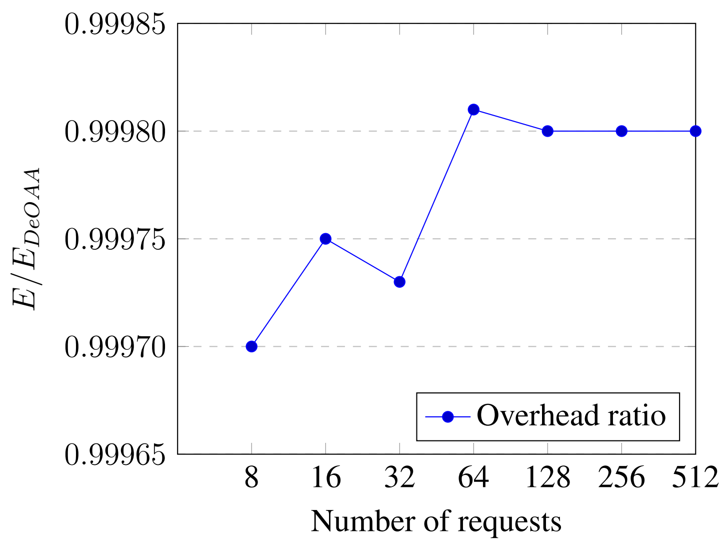

Figure 5.5: Financial overhead of $DeOAA$

the contract for four cases. The contract deployment and create functionalities are called only once, and the remaining functionalities are called according to the test case. All the test cases have the same input to Request functionality. We have taken $(d_i - a_i) = 5$ and $l_i = 4$. Test case 1 is similar to case 4 in theorem 5.3.2 where both CP and CU_i are honest. In test case 2, the user aborts after learning the usage fee and allocation decision. In test case 3, similar to case 1 in theorem 5.3.2, the CP acts maliciously and do not allocate VMs to CU_i. So, CU_i calls the Abrogate functionality to retrieve the usage fee he paid earlier. In test case 4, CP call the Abrogate functionality as CU_i fails to send an acknowledgement and fails to call Abrogate functionality.

5.5 Comparison with existing works

The comparison of the existing smart contract-based auctions with our proposed method is presented in Table 5.5. No existing methods support online auctions. In existing methods, the smart contract has to collect all users' requests and then decide the allocation and price of item / goods. Except [106], no method supports fair payments. In existing method, the cloud service provider receives payment without delivering goods, or the bidder receives goods without paying. Observe that our method does not provide privacy because

Test cases	**Function**	**Transaction hash/address**
	Contract Deployemnt	0xe3a93b4230103826250689267ee64fb71bd8a99452e8aec3f6b0818625a35818
	Create	0x9cf1f8c7e124f43118478e11fc271f3468d3853b8b262443232b466a57b94f76
Case 1	Request	0xc39e7aa99b3f5b8cd64a46e359820681c2512c1f7ab9b0322c2c41845a19d0a1
	Avail	0x8fa2cefb5095cd064b0ba1e24d11cff35db3329c2a2d5f5037d239707299d049
	Acknowledge	0x4152ec82a9e7e0e985065418a668ac207ba775e8ce830a21196b28f7ddddb0b0
		0x231ce9d8868e877afbb7b9cae240cc0ef83f9dbe3fdbe70d90bec34fd0b36859
		0xe51a8d7a09cfa93c232e1981bc4e6496edf0da4572026f46b85c85279b754e3b
		0x43806098c7884e8bc8e27d4000d28fc33f1a04c06378c54c08a2db9e1fdb4965
Case 2	Request	0x2917542fec7aa03c17b0c0ef88e14ddb34766c2b630051f72474e7cf9b7256c3
	Free	0xcb562eb8e47f3cd7db6058e0b8f6ebb09dd8e63d30ce48b9ac215a9d9acca6bc
Case 3	Request	0x13209c72a5f042cbb646cf6e038c927eaabebd559544ad61af14e4bdcb14eca1
	Avail	0xa4f45ab2fd7e3d4479f03ac33d4d74c833f851c2803b361d78d0b5a687143be9
	Acknowledge	0xdbc851a9925b8b5bbcc7f6474afe5c330677d93bb3001b595c2b461e3b409105
	Abrogate	0x6de4d605ab3bf6ea0ff7cb2cbd4bccd1c7b53f773011829238cf90c19840a950
Case 4	Request	0xf4176c7d921937b09336ca35cd8e01c9c0923854c7f054549d76648fa3c853de
	Avail	0x3fbf6922c048bf5437e867d965b62be6dffdc5e41918136a1f17d615aa2e9e6c
	Acknowledge	0x15a017c546ba5c2ecc6d3ee5b6c1ad8dfa23e2f32ada3c2a2bb7191dd41fc5e7
	Abrogate	0x35561ea38c509db15b14eb9adc7b2544d64372110a9d8951748a9dd2de2b67ff

Table 5.4: Transactions on Ethereum Ropsten test network

our scheme is online, which process bids instantly. Instant processing requires bids without applying encryption / hashing. Another reason for lack of privacy is that our auction algorithm is truthful, which means bidder revealing his true valuation will maximize his utility [119]. So, a malicious bidder cannot maximize his utility by copying other bidder's valuation. In some cases, the bidder does not want to reveal his bid to competitors. To achieve this, a bidder has to trust the semi-honest auctioneer similar to Galal *et al.* [205]. However, due to the lack of semi-honest auctioneer, our method cannot provide privacy.

Method	**Auction Type**	**Trading Item**	**Privacy**	**Truthful**	**Online**	**Fair Payments**
Hahn *et al.* [206]	Vickrey	Energy	Timed Commitments	No	No	No
Chen *et al.* [207]	Vickrey	Generic	Timed Commitments	No	No	No
Wu *et al.* [208]	Collusion resistant Vickrey	Generic	Timed Commitments	Yes	No	No
Braghin *et al.* [209]	English, Dutch, FPSB and Vickrey	Generic	Timed Commitments	No	No	No
Galal *et al.* [205]	FPSB	Generic	ZKP	No	No	No
Thakur *et al.* [210]	Double	Energy	No	No	No	No
Blass *et al.* [211]	FPSB	Generic	Goldwasser-Micali encryption and ZKP	No	No	No
Galal [212]	Vickrey	Generic	Trusted execution environment	No	No	No
Hassija *et al.* [213]	Double	Supply-chain	No	No	No	No
Zavodovski *et al.*[104]	Double	Generic	Timed-Commitments	No	No	No
Chen *et al.*[106]	Combinatorial	Cloud VM	Timed-Commitments	Yes	No	Yes
Proposed model	Online	Cloud VM	No	Yes	Yes	Yes

Table 5.5: Comparison of proposed method with existing smart contract-based auction methods.

5.6 Summary

We have designed a decentralized protocol for fair payments in the online cloud auction. Our protocol shows that the untrusted resource allocator in traditional online auction protocols can be replaced by smart contracts running on a public Blockchain network. Moreover, our protocol guarantees the correctness of the auction algorithm and fair payments without any trusted intermediaries. We have shown our protocol's feasibility by deploying the designed smart contract on a private Ethereum network and listed the transactional and financial cost of interacting with smart contracts.

Chapter 6

Fair Payment Protocol for Data de-duplication under Infrastructure-as-a-Service

With the advent of cloud computing, outsourcing data to a remote cloud storage servers has become a common practice [33]. However, most of the data being uploaded is redundant [101] and thus wasting large storage spaces. Storage systems use de-duplication (dedup) techniques to eliminate redundancy. Dedup eliminates the need to upload and store redundant data by verifying whether a data already exists in a cloud storage before each upload. If the check is valid, then the data is not uploaded, and simply the corresponding cloud user account is added to the existing file. The file link is sent to the requested user upon successful verification of Proof-of-Ownership.

Although there are many advantages, dedup introduces interesting challenges. First, to prevent a cloud provider from accessing sensitive information, it is common for cloud users to encrypt their data. If the data is encrypted using conventional encryption techniques, it is difficult to apply de-duplication techniques. Two identical files encrypted with two different keys generate two different cipher-texts which cannot be compared for similarity. Encrypted data dedup schemes are proposed based on convergent encryption [214, 215, 216, 217], secret sharing [218], proof-of-ownerships [219], keyword search [220], and password-authenticated key exchange [218].

Second, it is clear that the greatest beneficiary of the dedup is the cloud provider as it saves storage cost. It is required to motivate cloud users to opt for dedup by offering incentives / discounts on storage fee. Although several schemes are proposed for secure de-duplication in literature, only a few schemes [221, 222, 223, 224, 225, 93] discuss the incentives in de-duplication.

Even though the best encryption and incentive mechanisms are available, the cloud provider has to be trusted by the cloud user for fair computation of storage fee or a trusted party has to be recruited for computing storage fee correctly. However, hiring a trusted party is costly and finding an ideal trusted party that will behave honestly is difficult. The recent progress in Blockchain technology allows a public Blockchain network to emulate the properties of a trusted party. The public Blockchain network is trusted for the immutability of data it possesses, the correctness of the code (smart contract) execution in its environment and its availability.

This Chapter proposes a new Blockchain-based secure cloud storage system where no party can influence the computation of storage fee, and the proposed scheme also provides fair payments. We guarantee correct computation of storage fee even if the cloud provider is untrusted and cloud users are selfish. We employ a convergent encryption (CE) scheme for providing secrecy and a proof-of-ownership (PoP) scheme for proving ownership of duplicated data by a cloud user. Both CE and PoP schemes are black-boxes in our model, and we solely focus on designing a Blockchain-based secure cloud storage system with new incentive mechanism.

We summarize the contributions of this Chapter as follows:

(a) The contributions in this Chapter are two-fold: First, we design a new incentive mechanism, and second, we design a new Blockchain-based dedup scheme by leveraging the immutability, trust, and correctness properties of a public Blockchain network.

(b) The proposed incentive mechanism motivates the cloud users to choose dedup while ensuring profits for cloud provider. Experimental analysis shows that the proposed incentive mechanism is individually rational and incentive compatible for both users and cloud provider.

(c) As most of the existing schemes focus on secure de-duplication, we have proposed a dedup scheme which emphasizes correctness of dedup rate and fair payments between cloud user and cloud providers. We design a smart contract to realize the correctness and fairness of the proposed scheme.

(d) We have implemented the proposed smart contracts using solidity and executed them on a private Ethereum network which emulates the public Ethereum network. We have tested the proposed smart contract for the publicly available dataset and presented the transactional and financial costs of interacting with the smart contract.

6.1 Data de-duplication model

Let $\mathbb{S} = \{\mathcal{C}, \mathcal{U}\}$ be the cloud storage system with a smart contract B_{DEDU} running on a public Blockchain network BC where $\mathcal{C} = \{CP_1, CP_2, ..., CP_C\}$ is the set of cloud providers, and $\mathcal{U} = \{CU_1, CU_2, ..., CU_U\}$ is the set of cloud users.

Cloud providers provide data storage service to cloud users. If a cloud provider receives a data storage request from a cloud user, it will check whether any cloud user has previously stored the same data in its storage. If the check is negative, then it will ask the user to encrypt and upload the data. Otherwise, it will ask the user to send proof-of-ownership of the data. In both the cases, the cloud provider sends a file link to the user to access the stored file.

Many cloud users may exist in the system, and some may request to store the same data. If they all accept the de-duplication, then only one copy of that data is stored in the cloud. Let $\mathcal{D} = \{d_1, d_2, ..., d_D\}$ be the set of data files that the users may wish to store in the cloud. Each $d \in \mathcal{D}$ belongs to at least one user. Let $N_d^{CP}(t)$ represent the number of users having the same data d at time t, therefore, $N_d^{CP}(t) \geq 1$ and $\Sigma_{d \in D} N_d = \mathcal{U}$.

A smart contract B_{DEDU} facilitates fair payments. It assures the users for a correct dedup rate (based on which the storage fee is computed) and assures the cloud providers for a fair payment.

A public Blockchain network BC is maintained by a set of peers known as miners who will execute the B_{DEDU} according to an underlying consensus algorithm.

6.1.1 Convergent Encryption (CE)

A convergent encryption method consists of four algorithms.

(a) $K \leftarrow KeyGen_{CE}(d)$. The key generation algorithm takes the data file d as input and outputs a convergent key K.

(b) $C \leftarrow Encrypt_{CE}(K, d)$. The symmetric encryption algorithm takes the convergent key K and the file d as input and generates a cipher-text C as output.

(c) $d \leftarrow Decrypt_{CE}(K, C)$. The decryption algorithm takes both the convergent key K and cipher-text C as input and outputs the original file d.

(d) $tag \leftarrow TG_{CE}(C)$. TG is a tag generation algorithm which takes the cipher-text C as input and outputs a hash value tag.

6.1.2 Economic model

Both the cloud provider and user are rational, and they try to maximize their utility, and the utility is based on their interactions with the proposed system.

6.1.2.1 Utility of Cloud user

Let a cloud user $CU \in \mathcal{U}$ stores data $d \in \mathcal{D}$ at a cloud managed by a cloud provider $CP \in \mathcal{C}$. Let $U^0_{CU}(t)$ and $U^1_{CU}(t)$ be the utilities of CU when he does not adopt dedup and adopt dedup with B_{DEDU}, respectively.

$$U^0_{CU}(t) = P_{CU}(t) - SF^{CP}_{CU}(t). \tag{6.1}$$

Where $P_{CU}(t)$ is the profit earned by storing data in cloud, $SF^{CP}_{CU}(t)$ is the storage fee CU has to pay to CP.

Now let us define the utility of a user when dedup with B_{DEDU} is adopted.

$$U_{CU}^{1}(t) = P_{CU}(t) - \frac{SF_{CU}^{CP}(t)}{n_{d}^{CP}(t)} - EF_{CU}^{CP}(t) - I_{CU}(t) \tag{6.2}$$

Where $n_d^{CP}(t)$ is the data dedup rate at CP, i.e., the number of users having data d opted for dedup before CP receives CU's request. In our model, the fee depends on two parameters (1) Storage fee ($SF_{CU}^{CP}(t)$) - computed according to the current dedup rate and (2) Extra fee ($EF_{CU}^{CP}(t)$) - a cost paid by the user apart from storage fee to make the cloud incentive compatible. The cost of interacting with the smart contract is represented with $I_{CU}(t)$.

6.1.2.2 Utility of cloud provider

Let $U_{CP}^{0}(t)$ be the utility of a cloud provider when dedup is not adopted.

$$U_{CP}^{0}(t) = \sum_{d\in\mathcal{D}} N_{d}^{CP}(t) * (SF_{CU}^{CP}(t) - SC_{CP}^{CU}(t)) \tag{6.3}$$

where $N_d^{CP}(t)$ is the number users having data $d \in \mathcal{D}$. $SF_{CU}^{CP}(t)$ is the storage fee received by CP and $SC_{CP}^{CU}(t)$ is the cost incurred to CP for storing data.

Let $U_{CP}^{1}(t)$ be the utility of cloud provider when dedup with B_{DEDU} is adopted.

$$\begin{aligned} U_{CP}^{1}(t) = &\sum_{d\in\mathcal{D}}(N_{d}^{CP}(t) - n_{d}^{CP}(t) + 1) * (SF_{CU}^{CP}(t) - SC_{CP}^{CU}(t)) \\ &+ \sum_{d\in\mathcal{D}} n_{d}^{CP}(t) * EF_{CU}^{CP}(t) - \sum_{d\in\mathcal{D}} N_{d}^{CP}(t) * I_{CP}(t) - I_{deploy}(t) \end{aligned} \tag{6.4}$$

where $I_{CP}^{deploy}(t)$ is the cost of deploying smart contract and $I_{CP}(t)$ is the cost of interacting with smart contract.

The summary of cloud user and cloud provider utilities are given in Table 6.1.

Definition 6.1.1. *A fair data de-duplication protocol must provide the following guarantees:*

(a) ***Individually rational*** *(IR-constraint): An incentive mechanism is said to be individu-*

	without dedup	dedup with B_{DEDU}
Cloud user	$U^0_{CU}(t) = P_{CU}(t) - SF^{CP}_{CU}(t)$	$U^1_{CU}(t) = P_{CU}(t) - \frac{SF^{CP}_{CU}(t)}{n^{CP}_d(t)} - EF^{CP}_{CU}(t) - I_{CU}(t)$
Cloud provider	$U^0_{CP}(t) = \sum_{d\in\mathcal{D}} N^{CP}_d(t) * (SF^{CP}_{CU}(t) - SC^{CU}_{CP}(t))$	$U^1_{CP}(t) = \sum_{d\in\mathcal{D}}(N^{CP}_d(t) - n^{CP}_d(t) + 1) * (SF^{CP}_{CU}(t) - SC^{CU}_{CP}(t))$ $+ \sum_{d\in\mathcal{D}} n^{CP}_d(t) * EF^{CP}_{CU}(t)$ $- \sum_{d\in\mathcal{D}} N^{CP}_d(t) * I_{CP}(t) - I_{deploy}$

Table 6.1: Utilities of cloud user and cloud provider

ally rational if a rational user / cloud provider choosing de-duplication with B_{DEDU} obtains a non-negative utility. That is $\forall u \in \mathcal{U}$ with B_{DEDU} $U^1_{CU}(t) \geq 0$ and $\forall c \in \mathcal{C}$ $U^1_{CP}(t) \geq 0$.

(b) ***Incentive-compatibility** (IC-constraint): An incentive mechanism is said to be incentive-compatible if the cloud provider or cloud users cannot gain more profits from not adopting dedup. The best strategy for a cloud user is to opt de-duplication with B_{DEDU}. That is $\forall u \in \mathcal{U}$ $U^1_{CU}(t) - U^0_{CU}(t) \geq 0$. The cloud provider obtains more profits by choosing de-duplication with B_{DEDU}. That is $\forall c \in \mathcal{C}$ $U^1_{CP}(t) - U^0_{CP}(t) \geq 0$.*

(c) ***Correctness**: A dedup protocol is said to be correct if the storage fee for any user is defined by the following three factors: (1) size of the data to be stored (2) dedup rate and (3) auxiliary pricing function $p(|d|)$ at time t set by the cloud provider. In other words, no party (cloud provider / cloud users / miners) should influence the storage fee to be paid by the user except the above three factors.*

(d) ***Uniform payments**: A dedup protocol is said to be uniform if every user holding data d at CP pays the same storage fee irrespective of when their request arrives.*

(e) ***Fair payments**: A dedup protocol is fair if an honest cloud provider receives storage fee if and only if an honest user receives the file link of the data stored in the cloud managed by the cloud provider.*

6.2 Proposed incentive mechanism

To make a cloud provider incentive-compatible, we introduced a new parameter EF_u^c in Equation (6.2). In this section, we find the minimum and maximum values of $EF_{CU}^{CP}(t)$ so that a user / cloud provider obtains non-negative utility when opted for dedup with B_{DEDU}. According to the IC-Constraint in Definition 6.1.1, a cloud provider CP is said to be incentive compatible if

$$U_{CP}^{1}(t) - U_{CP}^{0}(t) \geq 0 \tag{6.5}$$

Substituting the utilities from Table 6.1 in Equation (6.5)

$$\begin{aligned}\sum_{d\in\mathcal{D}}(N_d^{CP}(t) - n_d^{CP}(t) + 1) * (SF_{CU}^{CP}(t) - SC_{CP}^{CU}(t)) + \sum_{d\in\mathcal{D}} n_d^{CP}(t) * EF_{CU}^{CP}(t) \\ - \sum_{d\in\mathcal{D}} N_d^{CP}(t) * I_{CP}(t) - I_{deploy}(t) - \sum_{d\in\mathcal{D}} N_d^{CP}(t) * (SF_{CU}^{CP}(t) - SC_{CP}^{CU}(t) \geq 0\end{aligned}$$

Assuming the cost of interacting with B_{DEDU} is negligible when compared to $SF_{CU}^{CP}(t)$ and $SC_{CP}^{CU}(t)$, we have

$$\begin{aligned}\sum_{d\in\mathcal{D}}(N_d^{CP}(t) - n_d^{CP}(t) + 1) * (SF_{CU}^{CP}(t) - SC_{CP}^{CU}(t)) + \sum_{d\in\mathcal{D}} n_d^{CP}(t) * EF_{CU}^{CP}(t) \\ - \sum_{d\in\mathcal{D}}(N_d^{CP}(t)) * (SF_{CU}^{CP}(t) - SC_{CP}^{CU}(t)) \geq 0\end{aligned}$$

For a single data file we have

$$(1 - n_d^{CP}(t)) * (SF_{CU}^{CP}(t) - SC_{CP}^{CU}(t)) + n_d^{CP}(t) * EF_{CU}^{CP}(t) \geq 0$$

$$n_d^{CP}(t) * EF_{CU}^{CP}(t) \geq (n_d^{CP}(t) - 1) * (SF_{CU}^{CP}(t) - SC_{CP}^{CU}(t))$$

$$EF_{CU}^{CP}(t) \geq \frac{n_d^{CP}(t) - 1}{n_d^{CP}(t)} * (SF_{CU}^{CP}(t) - SC_{CP}^{CU}(t)) \tag{6.6}$$

Now, we find the maximum value of $EF_{CU}^{CP}(t)$ so that a user is incentive-compatible when opted for dedup with B_{DEDU}. According to the IC-Constraint in definition (b), a user CU

is said to be incentive-compatible if

$$U^1_{CU}(t) - U^0_{CU}(t) \geq 0 \tag{6.7}$$

Substituting the utilities from Table 6.1 in Equation (6.7)

$$P_{CU}(t) - \frac{SF^{CP}_{CU}(t)}{n^{CP}_d(t)} - EF^{CP}_{CU}(t) - I_{CU}(t) - P_{CU}(t) + SF^{CP}_{CU}(t) \geq 0$$

Assuming the cost of interacting with B_{DEDU} is negligible when compared to $SF^{CP}_{CU}(t)$, we have

$$SF^{CP}_{CU}(t) - \frac{SF^{CP}_{CU}(t)}{n^{CP}_d(t)} - EF^{CP}_{CU}(t) \geq 0$$

$$EF^{CP}_{CU}(t) \leq \frac{n^{CP}_d(t) - 1}{n^{CP}_d(t)} * SF^{CP}_{CU}(t) \tag{6.8}$$

From (6.6) and (6.8) the minimum and maximum values of $EF^{CP}_{CU}(t)$ are set as

$$EF^{CP}_{CU}(t) = \left[\frac{n^{CP}_d(t) - 1}{n^{CP}_d(t)} * (SF^{CP}_{CU}(t) - SC^{CU}_{CP}(t)), \frac{n^{CP}_d(t) - 1}{n^{CP}_d(t)} * SF^{CP}_{CU}(t)\right] \tag{6.9}$$

when the cost of interacting with B_{DEDU} is considered then

$$\begin{aligned} EF^{CP}_{CU}(t) = \Bigg[& \frac{n^{CP}_d(t) - 1}{n^{CP}_d(t)} * (SF^{CP}_{CU}(t) - SC^{CU}_{CP}(t)) + (n^{CP}_d(t) * I_{CP}(t)) + I_{deploy}, \\ & (\frac{n^{CP}_d(t) - 1}{n^{CP}_d(t)} * SF^{CP}_{CU}(t)) + I_{CU}(t)\Bigg] \end{aligned} \tag{6.10}$$

6.2.1 Blockchain-based de-duplication protocol

In this section, we discuss a Blockchain-based cloud storage system which consists of a smart contract B_{DEDU}, a protocol to interact with B_{DEDU} and a public Blockchain network to deploy B_{DEDU}. At the end of the section, we provide an analysis of our proposed smart contract.

6.3 Fair data de-duplication method

6.3.1 Assumptions

(a) We assume that there are no unintentional system failures that may affect the cloud provider and the user's utilities.

(b) The cloud provider, the cloud, the smart contract and the Blockchain network are available all the time.

6.3.2 B_{DEDU} contract clauses

B_{DEDU} is a contract between a cloud provider CP and a cloud user CU. The high-level idea is that if both CP and CU are honest, then CP will receive the fee[1] paid by CU and CU will receive the file link to access his / her file in the cloud managed by CP. The fee is computed according to rules encoded in B_{DEDU} contract.

The clauses in the B_{DEDU} contract are as follows:

(i) All parties agree on timing parameters $\tau_p < \tau_{c1} < \tau_{c2}$ and two payment parameters: $SF_{CU}^{CP}(t)$ and $EF_{CU}^{CP}(t)$.

(ii) CP creates a smart contract (B_{DEDU}) for facilitating payments for cloud storage de-duplication. CP deploys the B_{DEDU} on a public Blockchain network and announces the smart contract address and smart contract ABI on a public platform (like a website / bulletin board).

(iii) After verifying the contract details at the contract address, a user CU if willing to store data at cloud managed by CP, has to send a request to B_{DEDU} along with some safety deposit $\$d$. This safe deposit is required to penalize CU for sending false requests. CU's request includes a tag computed from the encrypted file, and length of the file in bits.

(iv) After receiving the request, B_{DEDU} checks whether the tag sent by CU is received previously. If the check is valid, it will compute the fee as $(\frac{SF_u^c(t)*|d|}{n_d^{CP}(t)} +$

[1]From here on we call the amount paid by CU as fee which includes both $SF_{CU}^{CP}(t)$ and $EF_{CU}^{CP}(t)$ values.

$EF_{CU}^{CP}(t)$) and send this information to CU. Otherwise, it will compute the fee as $SF_u^c(t) * |d| + EF_{CU}^{CP}(t)$ and sends it to CU. $|d|$ is the length of the data to be stored in bits.

(v) CU must send the fee to B_{DEDU} before $\tau > \tau_p$. If CU fails, then his deposit $\$d$ is sent to CP, and the request is marked as terminated. Otherwise, CU's deposit $\$d$ is refunded.

(vi) CP has to send the confirmation message to B_{DEDU} before $\tau > \tau_{c1}$ acknowledging the receipt of file or correct PoP . Otherwise, the fee paid by CU is refunded, and the request is marked as terminated. CP should send the file link with correct access rights to CU.

(vii) CU has to send the confirmation message to B_{DEDU} before $\tau > \tau_{c2}$ acknowledging the receipt of file link. Otherwise, the fee paid by CU is refunded. If CU has sent the confirmation message before $\tau > \tau_{c2}$ then,

  (a) if CU is the first uploader of d then the fee is sent to CP.

  (b) if CU is not the first uploader of d, then the $EF_{CU}^{CP}(t)$ part of fee is sent to CP and the $SF_{CU}^{CP}(t)$ part of fee is distributed equally among all the users who hold the file link of d before CU. In either case, the request is marked as terminated and the value of $n_d^{CP}(t)$ is incremented.

6.3.3 B_{DEDU} protocol

B_{DEDU} protocol is presented in Figure 6.1. B_{DEDU} smart contract functionalities executed by Blockchain are presented as Algorithms 6.19 to 6.26.

A CP initializes the parameters SF and EF by invoking Algorithm 6.19. CP sets these parameters according to current storage costs and utility. If storage costs vary in future, he can change SF and EF values according to new storage costs. The parameter $interval$ is required to compute timing parameters τ_p, τ_{c1} and τ_{c2}. These timing parameters are required for timely computation of protocol and avoiding indefinite locking of funds in the contract. A CU sends his storage request to BC invoking Algorithm 6.21. His request consists of parameters like tag, $|d|$ and $\$d$. tag is computed using a convergent encryption algorithm, and $|d|$ is the length of the file requested for storage and $\$d$ is a safety deposit.

Algorithm 6.21 computes the pays in two ways. If the tag sent by CU already exists and is in an active state at cloud maintained by CP, then pay is computed according to the current discounted storage fee of file F with tag tag. Otherwise, the pay is computed as $pay = SF * |d| + EF$. Depending on the behavior of the user, there are two cases as follows:

Case 1: CU has sent the storage fee to BC invoking Algorithm 6.21. Depending on the behavior of the user, there are five cases as follows:

Case 1.1: CU has sent the correct file d [2] to CP. Depending on the behavior of CP, there are four cases as follows:

Case 1.1.1: CP has sent the confirmation message to BC invoking Algorithm 6.22. Depending on the behavior of CP, there are two cases as follows:

Case 1.1.1.1: CP has sent the file link to CU. Depending on the behavior of the CU, there are two cases as follows:

Case 1.1.1.1.1: CU has sent the confirmation message to BC invoking Algorithm 6.23. In this case, if file d is not previously stored at CP, then all the pay is sent to CP. Otherwise, the number of owners currently having a link to file d is calculated, and pay is divided into rem and DF components. rem is distributed among the file owners equally, and DF is sent to CP. The new pay to be paid by the next de-duplication requester is computed and stored at contract storage.

Case 1.1.1.1.2: CU has failed to send the confirmation message. This case occurs when CU has not received the file link from CP or CU intentionally / unintentionally fail to send a confirmation message. In this case, the CP can invalidate the link sent to CU and CU can send a transaction to BC invoking Algorithm 6.24, which refunds pay to CU.

Case 1.1.1.2: CP has failed to send the file link to CU. This case is similar to case 1.1.1.1.2 where CU can obtain a refund by invoking Algorithm 6.24.

[2] The correctness of the file is verified using the tag, based on which the payment is computed in Algorithm 6.20.

Case 1.1.2: CP has failed to send the confirmation message to BC. This case is similar to case 1.1.1.1.2 where CU can obtain a refund by invoking Algorithm 6.24.

Case 1.2: CU has sent the incorrect file to CP. In this case, the CP discards the CU's request and will not send any further transactions. CU can obtain its pay invoking Algorithm 6.24.

Case 1.3: CU has sent the correct proof-of-possession to CP. In this case, CP adds CU to the list of owners of file d and sends the file link to CU. From now on this case proceeds similarly to case 1.1.1.1.

Case 1.4: CU has sent the incorrect proof-of-possession to CP. This case is similar to case 1.2.

Case 1.5: CU neither sends file nor proof-of-possession to CP. This case is similar to case 1.2.

Case 2: CU has failed to send the storage fee. In this case, CP sends a transaction to BC invoking Algorithm 6.25 to claim CU's deposit.

B_{DEDU} **protocol**

Let $(KeyGen_{CE}, Encrypt_{CE}, Decrypt_{CE}, TG_{CE})$ be a secure convergent encryption method.

For cloud storage provider CP

1. To set up storage fee send **trans**$^{CP}_{create}$ = ($SPay$, $EPay$, $interval$) to BC.
2. After receiving a storage request
   (a) Assert that CU has sent \$$pay$ to BC with the same tag
   (b) Send **trans**$^{CP}_{cspconf}$ =(tag, $reqNum$) to BC.
   (c) Store d and send the file link (tag, $reqNum$, L) to user CU.
3. After receiving (”proof”, PoP, tag, $reqNum$) from user CU.
   (a) Assert that PoP is the correct proof for tag
   (b) Assert that CU has sent \$$pay$ to BC with the same tag.
   (c) Send **trans**$^{CU}_{cspConf}$ = (tag, $reqNum$) to BC.
   (d) Store d or add CU to the user list of d and send file link (tag, $reqNum$, L) to user CU
4. To claim the storage fee send **trans**$^{CP}_{claim}$ = (tag, $reqNum$) to BC.
5. After receiving a delink request (tag, $reqNum$, L) from user CU

(a) Assert that CU has sent ”deLink” message to BC with same tag and $reqNum$.

(b) Disable the link L for CU or remove d if no other users have active links to d

For user CU

6. To send a storage request
   (a) Compute $K \leftarrow KeyGen_{CE}(d)$.
   (b) Set $C \leftarrow Encrypt_{CE}(K, d)$.
   (c) Set $tag \leftarrow TG_{CE}(C)$.
   (d) Send $\textbf{trans}^{CU}_{request}$ = (tag, $|d|$, \$$d$) to BC.
7. To send storage fee
   (a) Assert that a request message has sent earlier with the same tag.
   (b) Send $\textbf{trans}^{CU}_{pay}$ = $(tag, reqNum, \$pay)$ to BC
   (c) If $SF * |d| + EF = pay$, then send file (d, tag, $reqNum$) to CP. Otherwise send the proof of possession (PoP, tag, $reqNum$) to CP.
8. To confirm receiving file link
   (a) Assert that L is a correct link to File d.
   (b) Assert that CP has sent ”cspConf” message to BC with the same tag.
   (c) Send $\textbf{trans}^{CU}_{usrConf}$ = (tag, $reqNum$) to BC.
9. To request a refund send $\textbf{trans}^{refund}_{CU}$ = (tag, $reqNum$) to BC.
10. To delink a file
    (a) Send $\textbf{trans}^{CU}_{deLink}$ = (tag, $reqNum$) to BC.
    (b) Send (tag, $reqNum$, L) to CP.

Blockchain BC: $SF := 0$, $EF := 0$, $uTAB := \{\}$, $k := 0$

11. On receiving $\textbf{trans}^{CP}_{create}$ execute $B_{Dedu}.create(SPay, EPay, interval)$
12. On receiving $\textbf{trans}^{CU}_{request}$ execute $B_{Dedu}.request(tag, |d|, \$d)$
13. On receiving $\textbf{trans}^{CU}_{pay}$ execute $B_{Dedu}.pay(tag, reqNum, \$pay)$
14. On receiving $\textbf{trans}^{CP}_{cspConf}$ execute $B_{Dedu}.cspConf(tag, reqNum)$
15. On receiving $\textbf{trans}^{CU}_{usrConf}$ execute $B_{Dedu}.usrConf(tag, reqNum)$
16. On receiving $\textbf{trans}^{refund}_{CU}$ execute $B_{Dedu}.refund(tag, reqNum)$
17. On receiving $\textbf{trans}^{CP}_{claim}$ execute $B_{Dedu}.claim(tag, reqNum)$
18. On receiving $\textbf{trans}^{CU}_{deLink}$ execute $B_{Dedu}.deLink(tag, reqNum)$

Figure 6.1: B_{Dedu} protocol

Algorithm 6.19 $B_{Dedu}.create$

```
    Input: $SPay$, $EPay$, $interval$
    Output: $\Phi$
1:  Set $SF \leftarrow SPay$
2:  Set $EF \leftarrow EPay$
3:  Set $k \leftarrow interval$
```

Algorithm 6.20 $B_{Dedu}.request$

```
    Input: $tag$, $|d|$, $\$d$
    Output: Success or failure message along with pay and timing parameters
 1: if $ledger[u] \geq \$d$ then
 2:     Set $ledger[u] \leftarrow ledger[u] - \$d$
 3:     if $(tag, *, *, *) \in uTAB$ then
 4:         if $\exists\, request(*, *, *, *, *, active, *, *) \in uTAB[tag]$ then
 5:             Set $pay \leftarrow uTAB[tag].cPay$
 6:         else
 7:             set $pay \leftarrow SF * |d| + EF$
 8:     else
 9:         Set $pay \leftarrow SF * |d| + EF$
10:         Set $numReq \leftarrow 0$
11:         Set $uTAB \leftarrow uTAB \cup (tag, numReq, *, *)$
12:     Set $ID \leftarrow u$, $rState \leftarrow waitForPay$
13:     Set $\$paid \leftarrow 0$, $reqNum \leftarrow uTAB[tag].numReq$
14:     Set $\tau_{sub} \leftarrow \tau$, $\tau_p \leftarrow \tau_{sub} + k$, $\tau_{c1} \leftarrow \tau_p + k$, $\tau_{c2} \leftarrow \tau_{c1} + k$
15:     Set $uTAB[tag].requests \leftarrow uTAB[tag].requests \cup (ID, \tau_{sub}, \tau_p, \tau_{c1}, \tau_{c2}, rState, pay, \$paid)$
16:     Set $uTAB[tag].cPay \leftarrow pay$
17:     Set $uTAB[tag].numReq \leftarrow uTAB[tag].numReq + 1$
18:     return (Success, $tag$, $SF$, $pay$, $reqNum$, $\tau_p$, $\tau_{c1}$, $\tau_{c2}$)
19: else return (Failure, Not enough balance)
```

Algorithm 6.21 $B_{Dedu}.pay$

```
    Input: $tag, reqNum, \$pay$
    Output: Success or failure message
 1: if $\tau \leq \tau_p$ then
 2:     if $uTAB[tag].requests[reqNum].ID = u$ then
 3:         if $uTAB[tag].requests[reqNum].pay \geq \$pay$ then
 4:             if $uTAB[tag].requests[reqNum].rState := waitForPay$ then
 5:                 if $ledger[u] \geq \$pay$ then
 6:                     Set $ledger[u] \leftarrow ledger[u] - \$pay$
 7:                     Set $uTAB[tag].requests[reqNum].\$paid \leftarrow \$pay$
 8:                     Set $ledger[u] \leftarrow ledger[u] + uTAB[tag].requests[reqNum].\$d$
 9:                     Set $uTAB[tag].requests[reqNum].rState \leftarrow waitForcloudproviderConf$
10:                     return (Success, Pay success)
11:                 else return (Failure, Not enough balance)
12:             else return (Failure, Request state is not $waitForPay$)
13:         else return (Failure, Not enough pay)
14:     else return (Failure, Wrong user)
15: else return (Failure, Pay timeout)
```

Algorithm 6.22 $B_{Dedu}.cspConf$

```
    Input: $tag$, $reqNum$
    Output: Success or failure message
 1: if $\tau \leq \tau_{c1}$ then
 2:     if $uTAB[tag].requests[reqNum].rState = waitForcloudproviderConf$ then
 3:         Set $uTAB[tag].requests[reqNum].rState \leftarrow waitForCliConf$
 4:         return (Success, Provider confirmation success)
 5:     else return (Failure, Request state is not $waitForcloudproviderConf$)
 6: else return (Failure, Provider confirmation timeout)
```

Algorithm 6.23 $B_{Dedu}.usrConf$

```
    Input: $tag$, $reqNum$
    Output: Success or failure message
 1: if $\tau \leq \tau_{c2}$ then
 2:     if $uTAB[tag].requests[reqNum].ID = u$ then
 3:         if $uTAB[tag].requests[reqNum].rState = waitForCliConf$ then
 4:             foreach $i \in [0, uTAB[tag].numReq - 2]$ do
 5:                 if $uTAB[tag].requests[i].rState = active$ then
 6:                     Set $activeRequests \leftarrow activeRequests + 1$
 7:             if $activeRequests = 0$ then
 8:                 Set $ledger[c] \leftarrow ledger[c] + uTAB[tag].requests[rNum].\$paid$
 9:             else
10:                 Set $\$rem \leftarrow uTAB[tag].requests[reqNum].\$paid - EF$
11:                 Set $\$DF \leftarrow uTAB[tag].requests[reqNum].\$paid - \$rem$
12:                 Set $ledger[c] \leftarrow ledger[c] + \$DF$
13:                 foreach $i \in [0, activeRequests]$ do
14:                     if $uTAB[tag].requests[i].rState = active$ then
15:                         Set $ledger[uTAB[tag].requests[i].ID] \leftarrow ledger[uTAB[tag].requests[i].ID] + \frac{\$rem}{activeRequests}$
16:                 Set $uTAB[tag].cPay \leftarrow \frac{SF}{activeRequests+2} + EF$
17:                 Set $uTAB[tag].requests[reqNum].rState \leftarrow active$
18:             return (Success, User confirmation success)
19:         else return (Failure, Request state is not $waitForCliConf$)
20:     else return (Failure, Wrong user)
21: else return (Failure, User confirmation timeout)
```

Algorithm 6.24 $B_{Dedu}.refund$

```
    Input: $tag, reqNum$
    Output: Success or failure message
 1: if $uTAB[tag].requests[reqNum].ID = u$ then
 2:     if $(\tau > \tau_{c1}$ && $uTAB[tag].requests[reqNum].rState = waitForcloudproviderConf) \mid\mid (\tau > \tau_{c2}$ && $uTAB[tag].requests[reqNum].rState = waitForCliConf))$ then
 3:         Set $ledger[uTAB[tag].requests[reqNum].ID] \leftarrow ledger[uTAB[tag].requests[reqNum].ID] + uTAB[tag].requests[reqNum].\$paid$
 4:         return (Success, refund success)
 5:     else return (Failure, Refund timeout)
 6: else return (Failure, Wrong user)
```

Algorithm 6.25 $B_{Dedu}.claim$

```
    Input: $tag, reqNum$
    Output: Success or failure message
1: if $\tau > \tau_p \&\& uTAB[tag].requests[reqNum].rState := waitForPay$ then
2:     Set $ledger[c] \leftarrow ledger[c] + uTAB[tag].requests[reqNum].\$d$
3:     return (Success, Claim success)
4: else return (Failure, Claim timeout)
```

Algorithm 6.26 $B_{Dedu}.deLink$

```
    Input: $tag, reqNum$
    Output: Success or failure message
1: if $uTAB[tag].requests[reqNum].ID = u$ then
2:     if $uTAB[tag].requests[reqNum].rState = active$ then
3:         Set $uTAB[tag].requests[reqNum].rState \leftarrow inActive$
4:         return (Success, DeLink Success)
5:     else return (Failure, Request state is not $active$)
6: else return (Failure, Wrong user)
```

6.3.4 Proofs of B_{DEDU}

Theorem 6.3.1. *Proposed protocol satisfies correctness*

Let $contract\text{-}B_{DEDU}$ is deployed on a Nakamoto-style Blockchain network using Proof-of-Work as a consensus algorithm. Let b be the current block of the blockchain which is extended by block b_1 and then by block b_2. Let $tx_{request}$ and tx_{pay} be the request and pay transactions respectively initiated by a user CU. $tx_{request}$ and tx_{pay} are eventually embedded in b_1 and b_2, respectively. We will prove the correctness by considering the following cases.

Case 1: A user CU influences the execution of the request functionality to decrease the fee f to be paid by him. This can happen if CU assumes the role of a miner and pre-mines two blocks $b_1^{'}$ and $b_2^{'}$ that contain the $tx_{request}^{'}$ and $tx_{pay}^{'}$ transactions respectively with a modified fee $f^{'}$. Observe that in this case, CU does not broadcast $tx_{request}^{'}$ to the network. Now CU broadcast $b_1^{'}$ extending b and $b_2^{'}$ extending $b_1^{'}$ to the Blockchain network. All the other miners in the Blockchain network verify every transaction in $b_1^{'}$ and extends $b_1^{'}$ if and only if all the outputs of every transaction in $b_1^{'}$ are correct. As the output of $tx_{request}$ is f but not $f^{'}$ the miners discard the block $b_1^{'}$. As $b_2^{'}$ is extending the wrong block $b_1^{'}$, it is also discarded. Miners with at least 51% of hash-rate cumulatively required to guarantee the correctness of the transactions in the block.

Case 2: CU broadcasts $tx_{request}$ but pre-mines $b_2^{'}$ that contains $tx_{pay}^{'}$ transaction with modified fee $f^{'}$. In this case, the actual fee f to be paid by CU is already stored in the contract storage. Therefore, during verification of the $tx_{pay}^{'}$ in $b_2^{'}$, the $f^{'}$ received through $tx_{pay}^{'}$ is compared against the stored value. The comparison will fail, and the block $b_2^{'}$ is discarded.

Case 3: A cloud storage provider CP influences the execution of request functionality to increase the fee to be paid by CU. This case is similar to Case 1 except that now CP assumes the role of a miner and broadcasts $b_1^{'}$ consisting the modified fee.

Similarly, all the transactions with B_{DEDU} are executed correctly; otherwise, those transactions are rejected. Nevertheless, CP or CU can influence some miners to include the wrong blocks to their local ledger and generate new blocks extending these wrong blocks. However, CP or CU should accumulate more than 50% of hash rate to make the entire network to accept wrong blocks which is difficult unless they have large mining pools under their control [204].

In summary, as the miners in public Blockchain networks are reasonably assumed to act honestly for the common good and follow the rules encoded in consensus algorithm, it is difficult for CP or CU to make the network to accept wrong blocks. Considering the above cases, our protocol satisfies correctness as defined in Definition 6.1.1.

Theorem 6.3.2. *Proposed protocol satisfies fair payments*

We prove fairness by considering the following cases.

Case 1: CU is malicious and aborts after learning the fee he needs to pay. In this case, according to B_{DEDU} contract clause (v), CU forfeits his deposit, and his data is not stored in the cloud. Here, the CU's data cannot be stored in the cloud unless he pays the fee. Thus the fairness holds.

Case 2: CU fails to send the data d or sends incorrect PoP to CP. In this case, according to B_{DEDU} contract clause (vi), CP refuses to acknowledge the receipt of d / PoP and the fee paid by CU is refunded. Here, the CU cannot obtain the storage link if he fails to send the d or the correct PoP. Thus the fairness holds.

Case 3: CP is malicious and does not acknowledge the receipt of d or correct PoP. This case similar to Case 2. The fee paid by CU is refunded. Here CP cannot obtain the fee without acknowledging the receipt of d or correct PoP. Thus the fairness holds.

Case 4: CP is malicious and does not send the file link to CU. In this case, according to B_{DEDU} contract clause (vii), CU will not acknowledge the receipt of file link. Then the fee paid by CU is refunded. Here, CP cannot obtain the fee without sending the file link to CU. Thus the fairness holds.

Case 5: CU is malicious and do not acknowledge the receipt of the file link. This case is similar to case 4. The fee paid by CU is refunded. If CP does not receive the fee, it disables the file link sent to CU. Here, CU cannot store his data without acknowledging the receipt of the file link. Thus the fairness holds.

In summary, considering the above cases, our protocol holds fairness.

Theorem 6.3.3. *Proposed protocol satisfies uniform payments*

According to clause (iv), the first user pays a fee of $SF_{CU}^{CP}(t) + EF_{CU}^{CP}(t)$ [3]. The second user pays a fee of $\frac{SF_{CU}^{CP}(t)}{2} + EF_{CU}^{CP}(t)$. Due to the second user, the first user gets a refund of $\frac{SF_{CU}^{CP}(t)}{2}$. So at the end each user pays a fee of $\frac{SF_{CU}^{CP}(t)}{2} + EF_{CU}^{CP}(t)$.

The n^{th} user pays a fee of $\frac{SF_{CU}^{CP}(t)}{n} + EF_{CU}^{CP}(t)$ and due to the n^{th} user each of the previous $n-1$ users receive a refund of $\frac{SF_{CU}^{CP}(t)}{n*(n-1)}$. At the end of n^{th} user's payment, the first user pays $SF_{CU}^{CP}(t) + EF_{CU}^{CP}(t) - \frac{SF_{CU}^{CP}(t)}{2} - \frac{SF_{CU}^{CP}(t)}{6}, ..., -\frac{SF_{CU}^{CP}(t)}{n*(n-1)} = \frac{SF_{CU}^{CP}(t)}{n} + EF_{CU}^{CP}(t)$. The second user pays $\frac{SF_{CU}^{CP}(t)}{2} + EF_{CU}^{CP}(t) - \frac{SF_{CU}^{CP}(t)}{6}, ..., -\frac{SF_{CU}^{CP}(t)}{n*(n-1)} = \frac{SF_{CU}^{CP}(t)}{n} + EF_{CU}^{CP}(t)$. Similarly the $(n-1)^{th}$ user pays $\frac{SF_{CU}^{CP}(t)}{n-1} + EF_{CU}^{CP}(t) - \frac{SF_{CU}^{CP}(t)}{n*(n-1)} = \frac{SF_{CU}^{CP}(t)}{n} + EF_{CU}^{CP}(t)$. So, every user having a data d opting for dedup with B_{DEDU} at CP pays the same fee.

In summary, the proposed protocol satisfies uniform payments irrespective of when a user submits his/her request.

[3] Assuming $|d| = 1$.

6.4 Proposed Inter-cloud provider de-duplication protocol

In the previous section, we have proposed a dedup protocol with a single cloud provider. This section proposes a Blockchain-based inter-cloud provider de-duplication protocol, which consists of a root-level smart contract $B_{I\text{-}DEDU}$, a smart contract B_{DEDU} for each cloud provider, protocols to interact with B_{DEDU}, and a public Blockchain network to deploy the smart contracts.

6.4.1 Assumptions

(a) We assume that all the cloud providers charge the same $SF_{CU}^{CP}(t)$ and $EF_{CU}^{CP}(t)$ values.

(b) We assume that no cloud provider will collect extra fee other than $SF_{CU}^{CP}(t)$ or $EF_{CU}^{CP}(t)$ for inter-cloud provider de-duplication. Inter-cloud provider de-duplication is same as a single cloud provider de-duplication from the cloud users perspective.

(c) A cloud provider CP_1 pays a fee of $AF^{CP}(t)$ to a cloud provider CP_2 for accessing the data stored at CP_2.

6.4.2 $B_{I\text{-}DEDU}$

The inter-cloud provider de-duplication is similar to single cloud provider de-duplication except that now a root-level smart contract has information about tags of data stored in different cloud providers.

The clauses in $B_{I\text{-}DEDU}$ are as follows:

(i) An organization $\mathcal{O}$ (like a consortium of cloud providers) creates a smart contract to facilitate inter-cloud de-duplication. $\mathcal{O}$ designs and deploys the $B_{I\text{-}DEDU}$ on a public Blockchain network and shares the smart contract address and ABI with all the registered cloud providers[a].

(ii) Each cloud provider CP_i also deploys a smart contract B_{DEDU}^{i} on a public

Blockchain network and announces the smart contract address and smart contract ABI on a public platform (like a website / bulletin board) and also registers B^i_{DEDU} address with $B_{I\text{-}DEDU}$.

(iii) A user CU sends a request to B^i_{DEDU} similar to clause (3) in Section 6.3.2.

(iv) B^i_{DEDU} performs checks similar to clause (4) in Section 6.3.2. If the check is valid, it will compute the fee to be paid by CU according to the dedup rate and sends this information to CU. Otherwise, it will send a request containing tag to $B_{I\text{-}DEDU}$.

(v) $B_{I\text{-}DEDU}$ checks whether any cloud provider is holding data with the same tag. If the check is not valid, it will return N/A to B^i_{DEDU}. Then B_{DEDU} will compute the fee as $SF^{CP}_{CU}(t) * |d| + EF^{CP}_{CU}(t)$ and send it to CU. Otherwise, $B_{I\text{-}DEDU}$ will send the information about the smart contract B^j_{DEDU} and cloud provider CP_j, holding the data with tag.

(vi) If B^i_{DEDU} receives the info about CP_j, then a request message is sent to B^j_{DEDU} from B^i_{DEDU}. Then B^j_{DEDU} computes the fee according to the dedup rate and sends it to B^i_{DEDU}, which is then forward to user CU by B^i_{DEDU}.

From here on, we assume that inter-cloud de-duplication is found and the contract proceeds as follows:

(vii) If $\tau > \tau_p$ and CU has not sent the fee to B^i_{DEDU}, then B^i_{DEDU} also will not send the fee to B^j_{DEDU}. CU's deposit is forfeited. This deposit is sent to CP_j and the request is marked as terminated by both B^j_{DEDU} and B^i_{DEDU}.

(viii) If $\tau > \tau_{c1}$ and CP_j has not sent the confirmation message to B^j_{DEDU}, then the fee paid by CU is refunded, and the request is marked as terminated by both B^j_{DEDU} and B^i_{DEDU}.

(ix) If $\tau > \tau_{c2}$ and CU has not sent the confirmation message to B^i_{DEDU}, then B^i_{DEDU} will not send confirmation to B^j_{DEDU}. The fee paid by CU is refunded. Else, a fee $AF^{CP}(t)$ is sent to CP_j from CP_i, and the request is marked as terminated by both B^j_{DEDU} and B^i_{DEDU}.

[a]We assume a registration phase is executed before deploying the smart contract and only registered cloud providers can exchange messages with $B_{I\text{-}DEDU}$

The analysis of $B_{I\text{-}DEDU}$ is similar to B_{DEDU} and all the properties satisfied by B_{DEDU} are also satisfied by $B_{I\text{-}DEDU}$. The formal smart contract *contract*-$B_{I\text{-}DEDU}$ is presented in Figure 6.2.

```
                         contract-$B_{I\text{-}DEDU}$
Init:       $list := \{\}$, $tags := \{\}$
Register:   Upon receiving ("register", $B_{DEDU}$, $info$) from a cloud storage provider
            $CP$
            assert $(c, *, *) \notin list$
            set $list := list \cup (c, B_{DEDU}, info)$
setTag:     Upon receiving ("setTag", $tag$) from a contract $B_{DEDU}$
            assert $tag \notin tags$
            set $cloudprovider := list[B_{DEDU}].c$
            set $info := list[B_{DEDU}].info$
            set $tags := tags \cup (tag, B_{DEDU}, cloudprovider, info)$
getTag:     Upon receiving ("getTag", $tag$) from a contract $B_{DEDU}$
            assert $(*, B_{DEDU}, *) \in list$
            if $(tag, *, *, *) \in tags$
              send ("tagFound", $tag$, $tags[tag].B_{DEDU}$, $tags[tag].cloudprovider$,
            $tags[tag].info$) to $B_{DEDU}$
            else send ("N/A", $tag$) to $B_{DEDU}$
```

Figure 6.2: *contract*-$B_{I\text{-}DEDU}$

6.5 Implementation

The simulation environment is discussed in Section 1.2.3.

6.5.1 Implementation of B_{DEDU}

We have tested the proposed smart contract B_{DEDU} multiple times, and each transaction's transactional cost and its equivalent financial cost is shown in Table 6.2. Observe that the contract deployment transaction consumes a large amount of gas; however, this is a one time cost for cloud provider. Next, the create and request functionalities also consumes a large amount of gas due to the modification of contract storage variables. Storing data in a contract is an expensive operation in Ethereum. As the usrConf function executes main tasks like computing dedup rate, sending the storage fee to cloud provider, and sending

Function	**Caller**	**Gas cost**	**Cost in Ether**	**Cost in $**
Init (contract deployment)	cloud provider	187467	0.000187	0.045
Create	cloud provider	143464	0.000143	0.035
Request	User	161168	0.000161	0.039
Pay	User	66558	0.000066	0.016
cspConf	cloud provider	31457	0.000031	0.007
Refund	User	31779	0.000031	0.007
Claim	cloud provider	31549	0.000031	0.007
DeLink	User	29318	0.000029	0.007

Table 6.2: Costs of interacting with B_{DEDU} contract. The gas price is approximated as 1Gwei and 1 Eth = $243.45, which are real-world prices in June 2020. We have rounded off the cost in $ value up to 3 decimals.

refunds to users, the transaction to usrConf also consumes a large amount of gas. The gas consumption of usrConf varies and depends on the number of users opted for de-duplication before the transaction initiator's call. We have listed the gas consumption of usrConf function in Figure 6.3. Observe that the gas consumption increases with the increase in dedup rate, and it reaches more than the block gas limit.

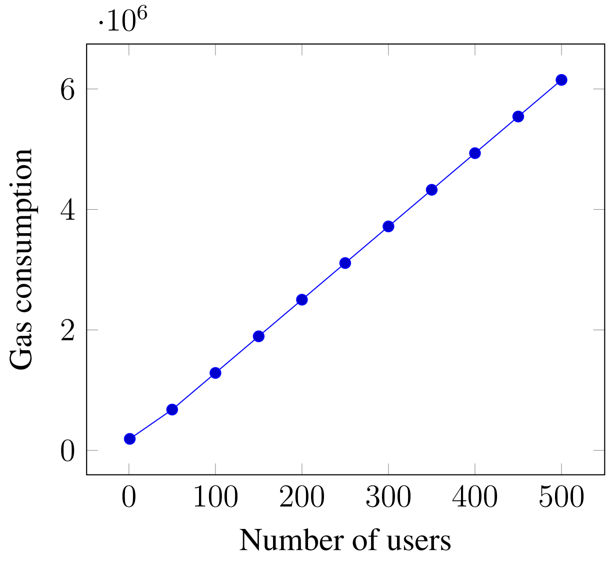

Figure 6.3: Costs of interacting with usrConf functionality

6.5.2 Experiment 1: Finding utility of the users and the cloud provider by varying $n_d^{CP}(t)$ and $EF_{CU}^{CP}(t)$

We have conducted experiments with the values shown in Table 6.3 adopted from Gao *et al.* [226] and Liang *et al.* [225]. The utility of the users and the cloud provider are shown in Figure 6.4 and Figure 6.5, respectively. We have varied $EF_{CU}^{CP}(t)$ from 10% to 50% of $SF_{CU}^{CP}(t)$ in both the figures. We have also varied the de-duplication rate $n_d^{CP}(t)$ as 10%, 50%, 90%, 100% of $N_d^{CP}(t)$. It is observed that both the users and cloud provider obtain non-negative utilities when opted for dedup with B_{DEDU}. The results show that our proposed model is individually rational.

In Figure 6.4, the utilities of the users decreases as the $EF_{CU}^{CP}(t)$ increases. However, for any constant $EF_{CU}^{CP}(t)$ value the average utility of users increases with increase in dedup rate. This result agrees with the general notion of increase in dedup rate with increase in the average utility of users. Figure 6.4 also shows that the user is incentive-compatible that is $U_{CU}^1(t) > U_{CU}^0(t)\ \forall n_d^{CP}(t) > 1$. The results show that the proposed model is not incentive compatible for $n_d^{CP}(t) = 1$, due to the new parameter $EF_{CU}^{CP}(t)$. To make the user incentive compatible when $n_d^{CP}(t) = 1$, the $EF_{CU}^{CP}(t)$ value should be set to zero.

Figure 6.5 shows that the cloud provider is not incentive compatible until $EF_{CU}^{CP}(t) \geq$ 36% of $SF_{CU}^{CP}(t)$. This result is in line with equation (6.6) when the values in Table 6.3 are substituted.

Parameter	Value in Ether
$P_{CU}(t)$	2.165
$SF_{CU}^{CP}(t)$	0.165
$SC_{CP}^{CU}(t)$	0.1
$AF_c(t)$	0.1

Table 6.3: Experiment Settings.

6.5.3 Experiment 2: Testing B_{DEDU} and $B_{I\text{-}DEDU}$ with public dataset

In Experiment 1, we have considered only similar requests where all users have the same data file with the same size. In this experiment, we have chosen a dataset that consists of

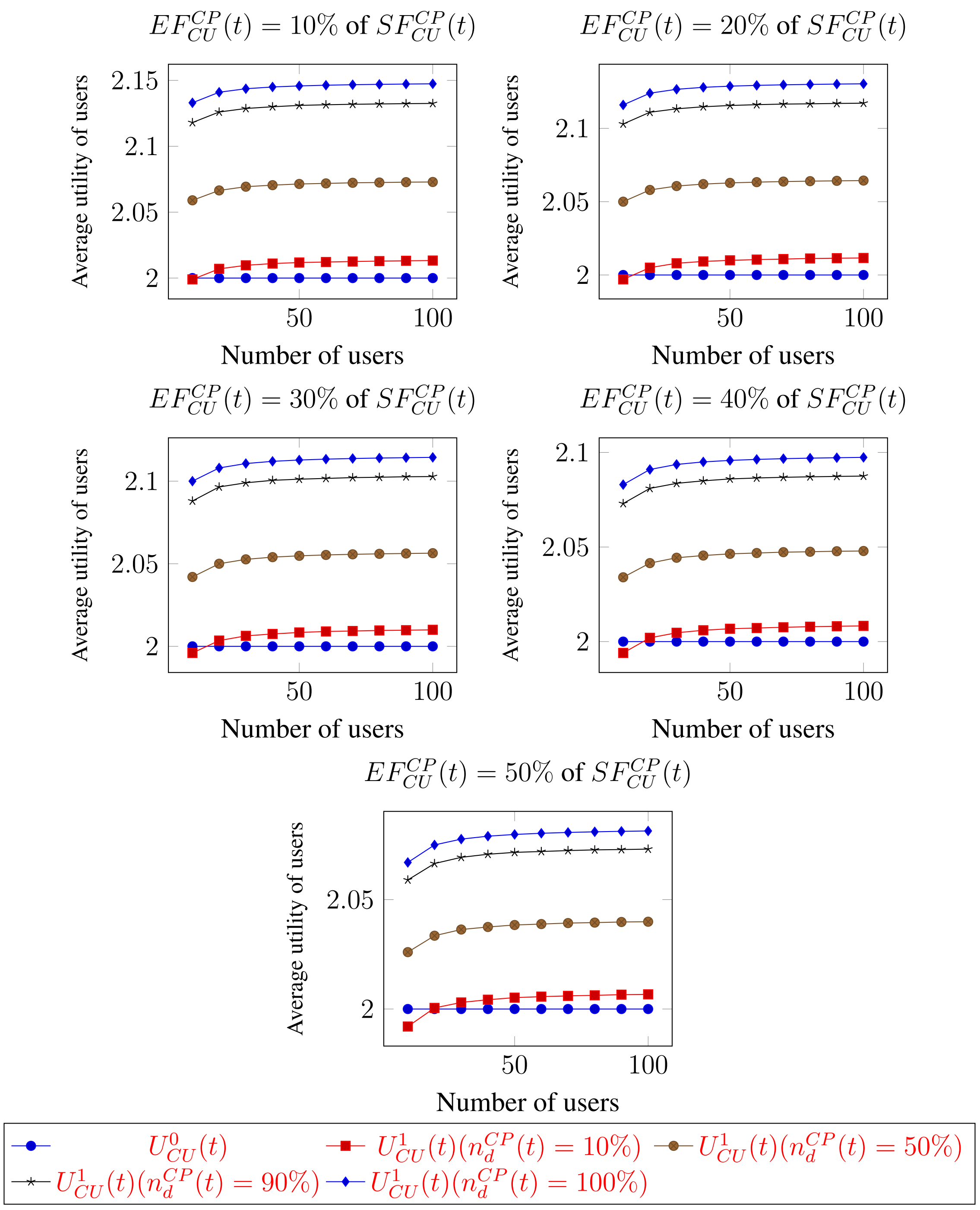

Figure 6.4: The effect of $EF_{CU}^{CP}(t)$ and $n_d^{CP}(t)$ on average utility of the cloud users

information of Debian packages gathered from Debian popularity contest (https://popcon.debian.org/contrib/by_inst). We took a snapshot of the number of packages, the number of installations of a single package, and size of each package as on 7th May

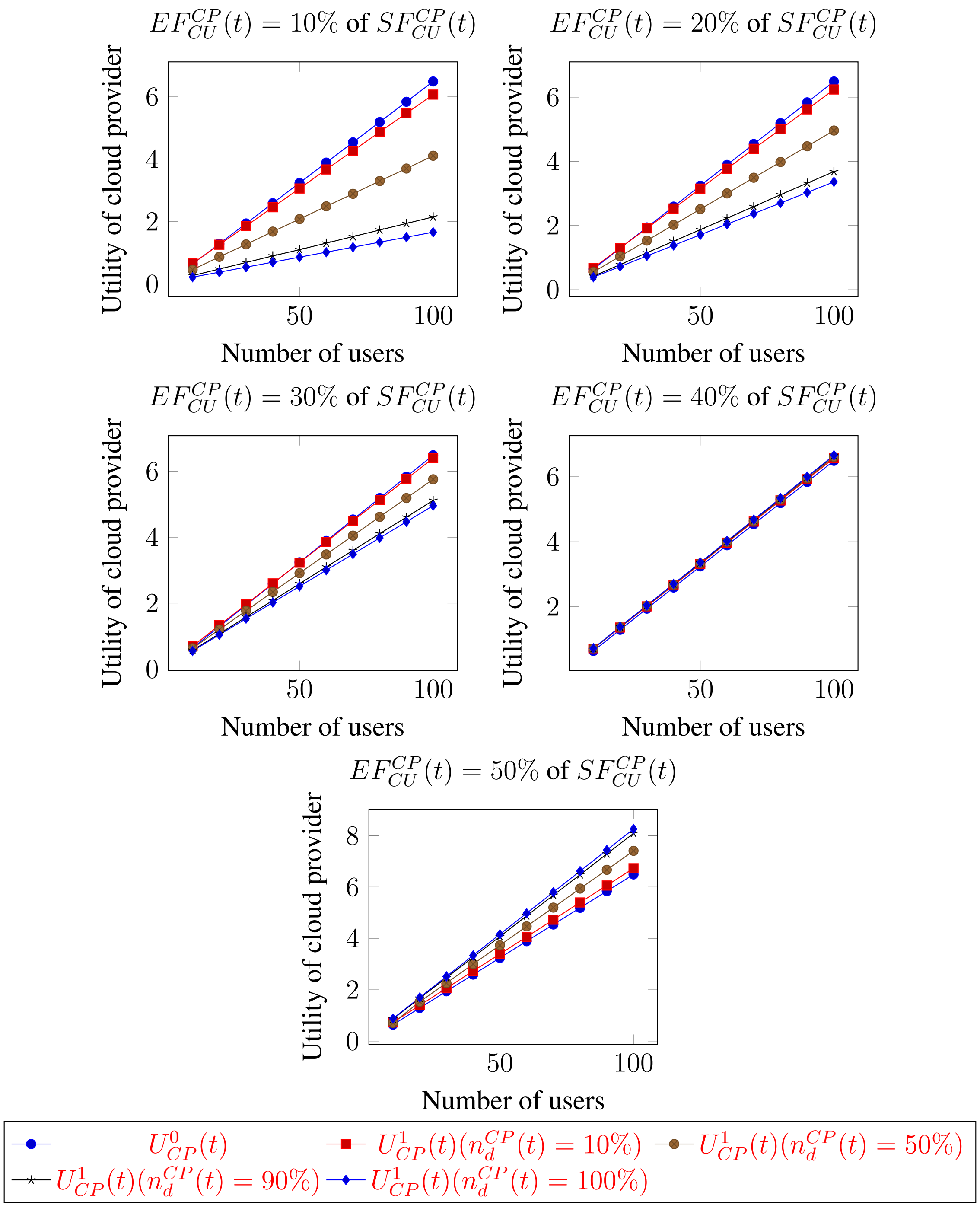

Figure 6.5: The effect of $EF_{CU}^{CP}(t)$ and $n_d^{CP}(t)$ on utility of the cloud provider

2020. Each package represents a data file to be stored in the cloud, and the installations serve as the de-duplication requests. They were a total of 403 packages (data files) and 270738 installations requests (users). The dataset is very diverse as it consists of data files having different sizes and each data file have a different number of installations. We uniformly distributed the requests among two cloud providers CP_1, CP_2 and computed the two cloud providers' utility. As discussed earlier in section 6.4, the inter-cloud provider de-duplication is the same as single cloud provider de-duplication from the user point of view, and there is no change in users' utility. The utility of the cloud provider changes as $U_c^2(t) = U_{CP}^1(t) + AF_{in}^c(t) - AF_{out}^c(t)$. $AF^{CP}(t)$ is the fee paid by a cloud provider CP_i to cloud provider CP_j for accessing the data stored at CP_j. In Figure 6.6, we show the utilities of the above-considered dataset by taking $EF_{CU}^{CP}(t) = 40\%$ of $SF_{CU}^{CP}(t)$ and $n_d^{CP}(t) = 100\% of N_d^{CP}(t)$. We observe that the cloud providers gain more utility when opted for inter-cloud de-duplication. The gain is due to the increase in the dedup rate. Therefore, with our proposed incentive mechanism dedup with $B_{I\text{-}DEDU}$ is more profitable than dedup with B_{DEDU} and dedup with B_{DEDU} is more profitable than no de-duplication.

6.6 Comparison with existing methods

Table 6.4 shows the comparison of our work with [221], [92], [225], and [93] in terms of category, incentives, features of incentives, correctness in the computation of de-duplication rate and fairness. Miao *et al.* [221] provides correctness of dedup rate but uses a trusted party known as de-duplication rate manager. In contrast, our method does not rely on the trusted party and still has correctness. The incentive mechanism in [221] supports only IR-constraint, whereas our method supports both IR-constraint and IC-constraint. The dedup method proposed by Li *et al.* [92] focuses more on the integrity of the deduplicated data stored in the cloud, whereas our method focuses on incentives and fair payment mechanism. The incentive mechanism in Liang *et al.* [225] supports both IR-constraint and IC-constraint, but, an untrusted cloud provider computes the dedup rate. Also, the incentives in [225] are time-variant, whereas our method supports uniform payments. Although

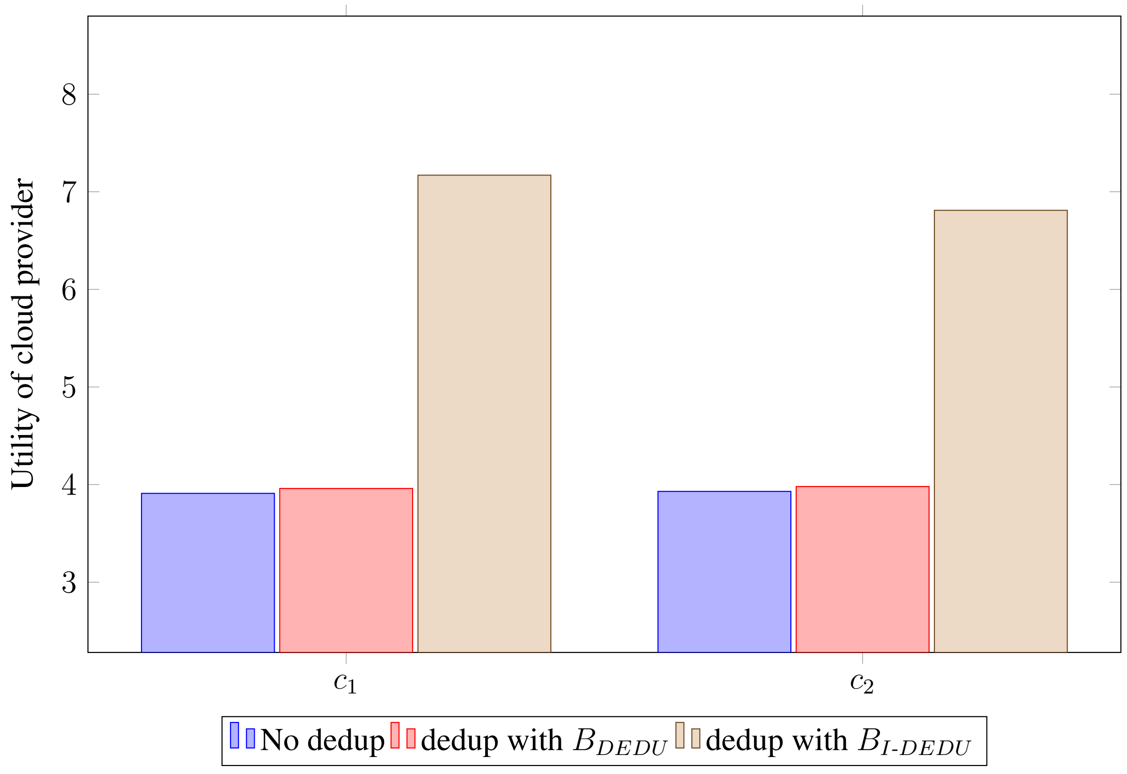

Figure 6.6: Utility of cloud providers with public dataset

in Wang *et al.* [93], the authors have considered fair payments, their method does not support fair payments in all the cases defined in Theorem 6.3.2.

Scheme	**Category**	**Incentive**	**Feature of Incentives**	**Correctness of de-duplication rate**	**Fairness**
[221]	PC	✓	IR	✓	x
[92]	BC	x	x	x	x
[225]	PC	✓	IR, IC	x	x
[93]	UC	✓	-	x	✓
Proposed method	BC	✓	IR, IC	✓	✓

Table 6.4: Comparison with existing data de-duplication works. PC - provider controlled, UC - user controlled, BC - Blockchain controlled

6.7 Summary

de-duplication techniques save storage costs of a cloud storage provider. However, adoption of de-duplication techniques by cloud users require strong incentives and a fair payment platform. In this Chapter, our contributions are two-fold: first, we have designed a new incentive mechanism, and second, we have designed a Blockchain-based de-duplication protocol. Experimental results show that our proposed incentive mechanism is individually rational and incentive compatible for both cloud provider and users. The proposed dedup protocol solves correctness, uniform payments and fair payments in de-duplication of cloud data without a trusted intermediary. The designed smart contracts in the proposed protocol are implemented in the Ethereum network, and the costs of interacting with the smart contract are presented.

Chapter 7

Fair Payment Protocol for Microservices-based software deployed in cloud under Software-as-a-Service

7.1 Introduction

With the advances in containerization technology [227], the traditional monolithic applications are being decomposed into a suite of small services known as microservices [228], each running in its process and communicating with lightweight mechanisms. Microservices now are a new trend in software architecture, emphasizing the design and development of highly maintainable and scalable software [229]. Industry giants like Amazon [230], Netflix [231], Linkedin [232] and Uber [233] are adopting and enhancing the microservice architecture. On the other hand, many individuals and enterprises prefer the cloud to deploy their applications that are delivered as services over the Internet. The adoption of microservice architecture in cloud-hosted software reduces infrastructure and maintenance costs [234]. The traditional and microservice-based software deployment in the cloud is shown in Figure 7.1. In Figure 7.1(a), users interact with a front-end application, which redirects user requests to multiple instances of the software hosted within a container. In Figure 7.1(b), the application is split into multiple components hosted in

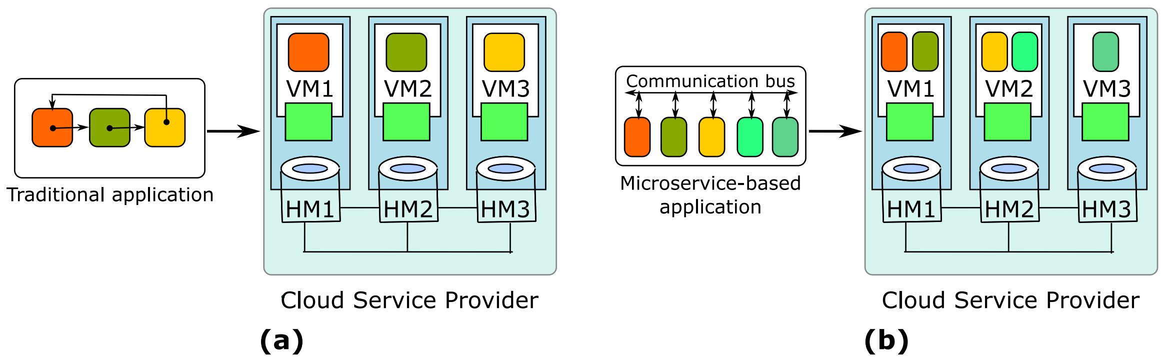

Figure 7.1: Software deployment in cloud [235]. (a) Traditional (b) Microservice-based

multiple containers and replicated at the user's convenience. Microservices architecture provides benefits like isolation, scalability, productivity, flexibility, and faster application development. Although there are some advantages of development and deployment of cloud applications using microservices, there are some limitations:

(a) **Trusted registry, and service discovery**: Microservice architecture by design appears to be fully decentralized. However, it includes centralized components like a trusted registry and a service discovery module (ZooKeeper [236], Eureka [237]). These two components are required to publish and discover microservices by a user interface or another microservice. These centralized components may become a single point of failure and require additional security methods to avoid attacks like denial of service, microservice integrity breach etc. A distributed immutable storage is required to avoid the attacks.

(b) **Microservice communication**: Since every microservice is independent, communication between them is complex. Microservices are designed to trust each other completely; the compromise of one microservice jeopardises the entire application. Several attacks like Man-In-The-Middle (MITM), service identity spoofing etc., can occur during communications between microservices. A reliable communication platform is required to mitigate the attacks.

(c) **Rating and billing**: As microservices consume cloud resources dynamically, real-time rating and billing models are required [238]. However, existing models do not consider transparency, and in most cases, the user ends up paying for more resources than consumed. Also, the cloud being rational may tamper with the usage records of a user to increase bill. Hence, a transparent and immutable rating and billing platform are required where a user can verify his consumption and trust the billing process.

(d) **Double charging**: Most of the times, cloud resources are shared by multiple microservices simultaneously. For example, a microservice only spends five minutes per hour doing any processing. However, it is charged for the entire hour. During the ideal time, the resources are allocated to other microservice, and the cloud provider benefits from charging both the microservices. In many cases, the users are unaware of double charging. A publicly available, transparent and immutable resource usage record log is required to detect double charging or ideal time charging.

The recent progress in Blockchain and smart contract technologies can reduce the previously discussed limitations. This Chapter proposes a new Blockchain-based rating, charging and billing system for microservices deployed on a cloud platform.
We summarize the contributions of this Chapter as follows:

1. We design a Blockchain-based registry to publish microservices and a Blockchain-based tamper-proof communication platform for microservices.

2. We design a new cost computation model based on real-time usage and dynamic pricing of resources with respect to the state of the cloud operating environment.

3. To the best of our knowledge, we are the first to propose a Blockchain-based rating charging and billing platform for microservices deployed on a cloud platform.

4. We have implemented the proposed system using Solidity [24] and presented the transactional and financial costs of the proposed system.

7.2 Microservice rating, charging and billing (RCB) architecture

A microservice RCB architecture is presented in Figure 7.2. The architecture consists of three core services and two supporting services. The three core services are (1) Usage

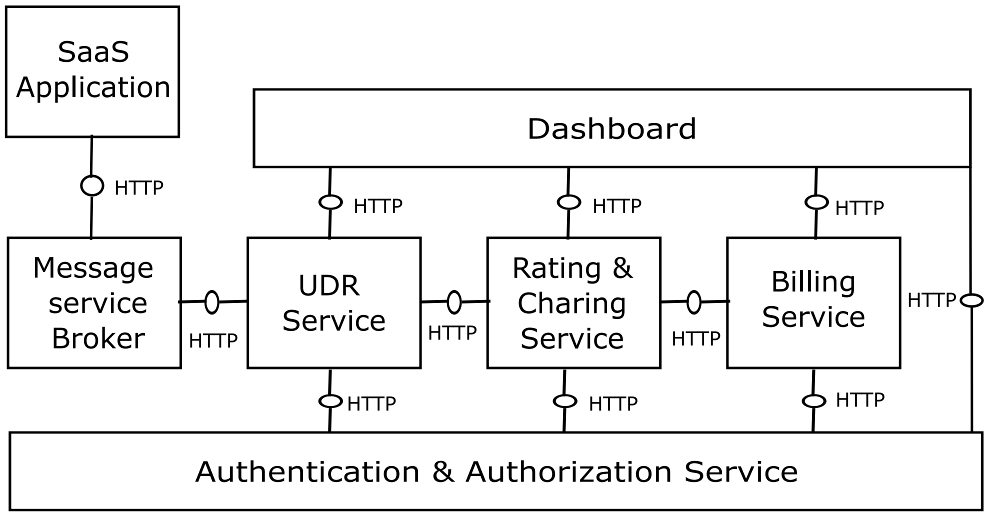

Figure 7.2: Traditional microservice rating, charging and billing system

data record (UDR) service, (2) Rating and charging service and (3) Billing service. The two supporting services are (1) Authentication and authorization service and (2) Message broker service.

The message broker service fetches the usage data records from an external SaaS application (a SaaS application is a collection of several microservices). These usage records are used in the calculation of bill. The RCB services interact with the authentication and authorization service to validate the service request's authenticity. The cloud user uses the dashboard to interact with the RCB platform.

Definition 7.2.1. *A fair rating, charging, and billing platform for microservices must provide the following guarantees:*

(a) ***Decentralized service discovery****: Decentralized service discovery is available all the times, and no party can control the service discovery.*

(b) ***Tamper-proof communication platform****: The messages exchanged between microservices are tamper-proofed, and the messages are stored securely for future auditing.*

(c) ***Fair Rating and charging platform****: A platform is said to be a fair rating and charging platform if the rating and charging values depend on the environment variables and no other party can influence the rating and charging variables.*

(d) ***Fair Billing platform****: A platform is said to be a fair billing platform if the billing solely depends on the user consumption of the resources and the prices of the resources, and no other party can influence the billing process.*

7.3 Blockchain-based Microservice Rating, Charging and Billing (RCB) System

In this section, we first present the architecture of the Blockchain-based RCB system. Then, we build a mathematical model for reputation-based rating, charging and billing system. We later construct protocols for the Blockchain-based RCB platform and conclude the section with the proofs for the goals described in Definition 7.2.1.

A Blockchain-based rating charging and billing platform is shown in Figure 7.3. The architecture consists of several smart contracts, which are discussed briefly in the following subsection. The architecture also contains a dashboard similar to Figure 7.2. The message broker service from Figure 7.2 is divided into two different broker services: (1) Server broker service - fetches the usage data records from external SaaS application. (2) Operating environment broker service - fetches the state of the operating environment.

7.3.1 Smart contracts for RCB system

The RCB system consists of a total seven contracts: (1) Service discovery contract (SDC), (2) Message exchange contract (MEC), (3) Usage data contract (UDC), (4) Rating and charging contract (RCC), (5) Billing contract (BIC), (6) Registration and reputation contract (RRC), and (7) Error data contract (EDC).

Service discovery contract (SDC): A cloud provider deploys a set of microservices at the cloud and lists the microservices' details in the SDC contract. The SDC contract contains details such as a service name, available endpoints, API specification, current load etc. The contract also contains service chains specifying the order of microservice execution. The structure of data records stored at the SDC is shown in Figure 7.4.

Message exchange contract (MEC): Microservices communicate with each other via

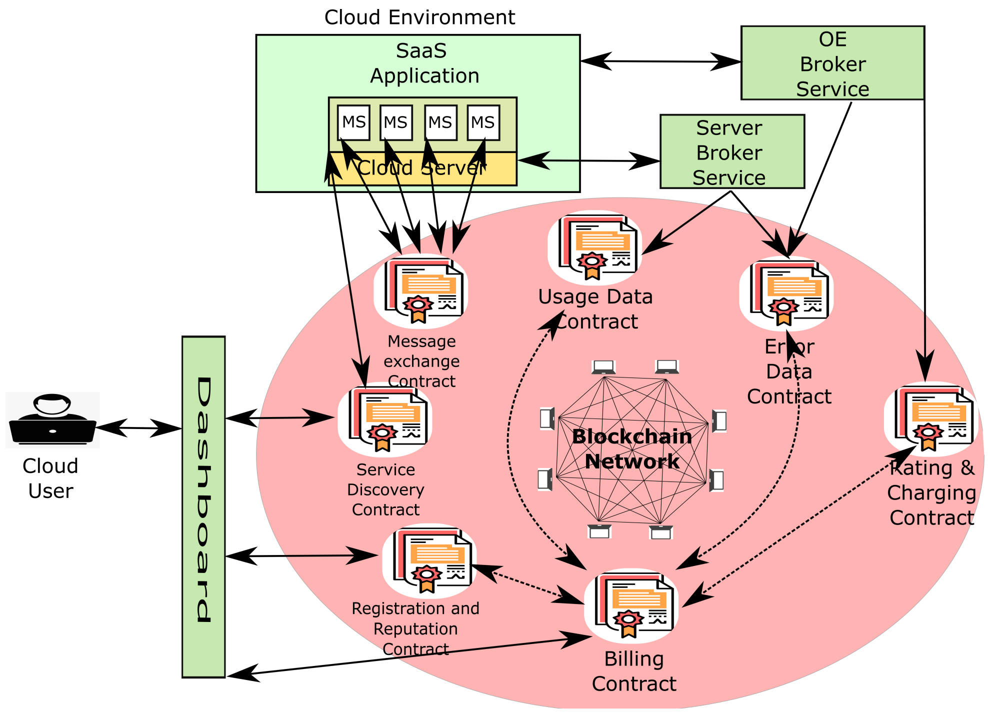

Figure 7.3: Blockchain-based microservice RCB platform

Service Name	List of end-points	Hash of API desc	List of Inputs	List of Outputs	Current Load

Service Chain ID	Start Service Name	End Service Name	Sequence of Intermidiate Services

Figure 7.4: Structure of the data records stored in SDC contract.

MEC contract. The contract generates an event whenever a message arrives. Microservices listen to events and processes the messages intended for them. The contract also ensures that only the authorized microservices can send and receive messages. The structure of the data records stored at MEC is given in Figure 7.5.

Caller Service	Caller End-point	Callee Service	Callee End-point	List of Inputs

Figure 7.5: Structure of the data records stored in MEC contract.

Usage data contract (UDC): The contract stores the usage statistics of all the users of the SaaS application. A server broker service collects the usage information from the cloud server and sends it to the UDC contract. However, a cloud provider may influence the broker service from fetching the correct usage statistics. To avoid this attack, we assume that the server broker service runs in a secure enclave such as Intel SGX. The structure of the data records stored in the UDC is shown in Figure 7.6.

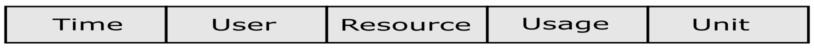

Figure 7.6: Structure of the data records stored in UDC contract.

Rating and charging contract (RCC): An operating environment broker service fetches a number of parameters of the cloud environment and sends them to RCC. We assume this broker service also runs in a secure enclave to avoid malicious cloud provider affecting resources' price. The parameters include power rate, operating load, number of active customers and other similar parameters. The RCC computes the prices of the resource based on the received parameters' value. However, the cloud provider has to code the rules into RCC for determining the resource prices with respect to operating environment parameters. The structure of the records stored in the RCC is shown in Figure 7.7.

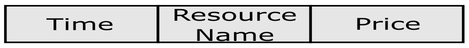

Figure 7.7: Structure of the data records stored in RCC contract.

Error data contract (EDC): The operating environment service broker also logs the errors during service provisioning in EDC. The structure of the records stored in EDC is shown in Figure 7.8.

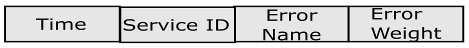

Figure 7.8: Structure of the data records stored in EDC contract.

Registration and reputation contract (RRC): A cloud user has to register with RRC to use services provided by a cloud provider. Initially, the default reputation values of cloud user and provider are set to 1. Later, the reputation of cloud user is updated according to his interactions with the system and the reputation of the cloud provider is updated according to the quality of the service provided.

Billing contract (BIC): The cloud user calls BIC to deposit the estimated pay before the start of execution of the service. After completing the service, BIC receives usage data from UDC, rating data from RCC, reputation data from RRC and generates the final payment the user has to pay. The cloud provider also inputs the discounts offered to the user during service level agreement negotiations.

7.3.2 Cost computation model

Let CU_i be the set of cloud users and CP be a cloud provider. Let $\mathcal{R}$ be the set of resources provided by CP, $\mathcal{O}$ be the set of operating environment parameters, and $\mathcal{E}$ be the set of errors that could occur during service provisioning. The cost to be paid by a cloud user for using cloud resources is computed from the following four factors: (1) Resource consumption of the user, (2) Reputation of the user, (3) Reputation of the cloud provider, and (4) State of the operating environment.

7.3.2.1 Resource consumption of cloud user

Let a cloud user $CU_i \in CU$ consume resources in the time between τ_s and τ_e. Then the usage vector of the user CU_i measured between τ_s and τ_e is represented as:

$$\vec{U}(CU_i, \tau_s, \tau_e) = \sum_{\tau_s \leq \tau \leq \tau_e} \vec{U}(CU_i, \tau) \tag{7.1}$$

Let r_j be the meter value measuring the resource consumption of j^{th} resource in $\mathcal{R}$. Let $n = |\mathcal{R}|$ be the total number of resource types provided by a cloud provider. Then, the usage vector at time τ is expressed as:

$$\vec{U}(CU_i, \tau) = \langle (CU_i, r_1, \tau), (CU_i, r_2, \tau), ..., (CU_i, r_n, \tau) \rangle \tag{7.2}$$

7.3.2.2 Reputation of cloud user

The reputation of a cloud user R_{CU_i} depends on his behavior with the system. If CU_i has paid the fee without any delays, his reputation is not affected; otherwise decreased. The decrease in the reputation value is directly proportional to the amount the user defaulted. Let f be the amount defaulted by CU_i, then the reputation $R_{CU_i}(CP_\tau)$ for defaulting f

computed as:

$$R_{CU_i}(CP_\tau) = \begin{cases} V_1 & \text{if } f > A_1 \\ V_2 & \text{if } f > A_2 \\ \dots & \dots\dots\dots\dots \\ V_n & \text{if } f > A_n \end{cases}$$

Where $V_1, V_2, ..., V_n$ are the values defined by CP for categories $A_1, A_2, ..., A_n$ respectively. The reputation is calculated as

$$R_{CU_i} = \begin{cases} R_{CU_i} & \text{if } f = 0 \\ \beta * R_{CU_i}(CP_\tau) + (1-\beta) * R_{CU_i} & \text{otherwise} \end{cases} \tag{7.3}$$

Where $\beta \in [0, 1]$ is smoothing factor.

7.3.2.3 Reputation of cloud provider

A cloud provider's reputation R_{CP} depends on the errors that occur during the service provision. Let $R_{CP}(CU_\tau)$ be the reputation of CP while serving CU_i at time τ. Let $\mathcal{E}$ be the set of errors that can occur during the service provision, and w_e is the weight associated with the error $e \in E$. The value of $R_{CP}(CU_\tau)$ can be computed as:

$$R_{CP}(CU_\tau) = \frac{1}{\sum_{e=1}^{\mathcal{E}} w_e * X_e} \tag{7.4}$$

where X_e is the number of times an error $e \in E$ occurs during the service provisioning. The reputation of a cloud provider is updated as

$$R_{CP} = \alpha * R_{CP}(CU_\tau) + (1-\alpha) * R_{CP} \tag{7.5}$$

where $0 < \alpha < 1$ is a smoothing factor.

7.3.2.4 Operating environment of the cloud

The state of the cloud operating environment $S(\tau)$ is expressed as a set of variables representing different parameters like the number of active users, operating load, power rate etc., Let there are m number of operating system parameters, and w_k be the weight given to parameter p_k. The state of the cloud operating environment $S(\tau)$ is defined as:

$$S(\tau) = w_0 * p_0 + w_1 * p_1, ..., +w_m * p_m \tag{7.6}$$

such that

$$\sum_{k=0}^{m} w_k = 1 \tag{7.7}$$

Let the state space of S is denoted by Z such that $S(\tau) \in Z$. Therefore the price of a resource varies depending on the state of the operating environment.

$$P_{r_i}(\tau) = \begin{cases} P_1 & \text{if } S(\tau) \in Z_1 \\ P_2 & \text{if } S(\tau) \in Z_2 \\ \cdots & \cdots\cdots\cdots\cdots \\ P_n & \text{if } S(\tau) \in Z_n \end{cases}$$

such that $P_i \in \mathbb{Z}$ and $\bigcup\limits_{i=1}^{n} Z_i = Z$. The price vector for resources at time τ is given as

$$\vec{P}(\tau) = \langle P_{r_1}(\tau), P_{r_2}(\tau), ..., P_{r_{n(\tau)}} \rangle \tag{7.8}$$

Now the cost incurred to a cloud user CU_i at time τ is computed as

$$P_{CU_i}(\tau_s, \tau_e) = \sum_{\tau_s \le \tau \le \tau_e} \vec{U}(CU_i, \tau) \cdot \vec{P}(\tau) \tag{7.9}$$

The cost payable by the cloud user at time τ is computed as

$$C_{CU_i}(\tau_s, \tau_e) = (P_{CU_i}(\tau_s, \tau_e) * R_{CP} * \frac{1}{R_{CU_i}}) - D_{CU_i} \tag{7.10}$$

where D_{CU_i} is the discount offered to CU_i during service layer agreement negotiations

7.3.3 High-level overview of the RCB protocol

The protocol consists of three phases: (1) Initialization phase (IP) (2) Service provisioning phase (SAP) (3) Rating, charing and billing phase (RBP).

Initialization phase

(i) During this phase, a cloud provider CP prepares and deploys a series of smart contracts on a public Blockchain platform and publishes their addresses on a public platform.

(ii) CP also develops a dashboard where users can login and monitor their resource usage, resources rating and billing information.

Service provisioning phase (SAP)

(iii) A cloud user CU_i requests a service from CP through a dashboard. The request is received by a service discovery contract which returns the approximate cost $\$p_{es}$ of the service execution.

(iv) The request is executed as soon as the CU_i sends $\$p_{es}$ to the billing contract. The request may initiate a series of microservices execution. The microservices communicate among themselves through a message exchange contract.

(v) A server broker service SBS fetches the users' resource usage information and sends this information to the usage data contractd.

(vi) An operating environment broker service $OEBS$ fetches the operating environment variables values and sends this information to the rating and charging contract. Also, the $OEBS$ logs the errors during service provisioning in the error data contract.

Billing Phase

(vii) CU_i request a bill by sending a request to the billing contract. The billing contract fetches the usage records from UDC, charging records from RCC, the reputation of CP and CU_i from RRC.

(viii) Then, BIC computes the final bill and the amount to be paid by CU_i after deducting $\$p_{es}$. If $\$p_{es}$ is greater than the computed amount, then BIC will refund the excess amount.

(ix) The final bill may also include user-specific discounts set by the cloud provider.

(x) If CU_i fails to send the excess amount, then his reputation is decreased. BIC also calculates CP's reputation based on the error data records, and a new reputation value is updated in the reputation contract.

7.3.4 RCB Protocol

7.3.4.1 Initialization phase

During this phase, a cloud service provider CP deploys a set of smart contracts described in section 7.3.1 on a public Blockchain network and publishes their address publicly. CP deploys the microservices at a cloud operated by it and sends the details of deployed microservices to the SDC contract. This a one-time setup for a single SaaS application [1]. The registration contains details like application name, service name, service endpoints, the hash value of API, and estimated cost. CP also registers the microservice chains with SDC contract. Although microservices are distributed and independent, they are executed in a sequence to accomplish a task. The sequence is called microservice chaining. The service chaining also helps in predicting the execution time and cost of microservices. After receiving the details of microservice or microservice chains, the SDC contract stores them in contract storage. To access the application, a cloud user has to send a registration request and a self-generated public key to RRC. The protocol for the initialization phase is given in Figure 7.9.

Protocol: Initialization phase

For Cloud provider CP

1. Send **trans**$_{deploy}^{CP} = (SDC, MEC, UDC, RCC, RRC, EDC, BIC)$ to Blockchain BC. After receiving the addresses of the deployed contracts, publish the addresses publicly.

[1] Even though our framework does not explicitly mention multiple instantiations and load balancing of microservices, it can effectively incorporate those concepts.

2. Host the microservices at cloud and send **trans**$^{CP}_{mService}$ = $(sAppName$, $sName$, $sEndPoints$, $APIHash$, $sInputs$, $sOutputs$, $Cost_s)$ to SDC.
3. To register service chains, send **trans**$^{CP}_{sChain}$= $(sAppName$, $sChainID$, $sName_s$,..., $sName_e$, $Cost_c)$ to SDC.

For cloud user CU_i

4. To register with CP for using an application $sAppName$, send **trans**$^{CU_i}_{reg}$ $=$ $(pk_{CU_i}, sAppName)$ to RRC

For Blockchain BC: $list_{ms} \leftarrow \{\}, list_{sc} \leftarrow \{\}, list_{CU} \leftarrow \{\}$

5. On receiving **trans**$^{CP}_{deploy}$, deploy all the received contracts and return the contracts' address.
6. On receiving **trans**$^{CP}_{mService}$, set $list_{ms} \leftarrow list_{ms} \cup (CP$, $sAppName$, $sName$, $sEndPoints$, $APIHash$, $sInputs$, $sOutputs$, $Cost_s)$.
7. On receiving **trans**$^{CP}_{sChain}$, set $list_{sc} \leftarrow list_{sc} \cup (CP$, $sChainID$, $sAppName$, $sName_s$,...,$sName_e$, $Cost_s)$
8. On receiving **trans**$^{CU_i}_{reg}$
   set $list_{CU} \leftarrow list_{CU} \cup (CU_i, sAppName)$

Figure 7.9: Initialization phase

7.3.4.2 Service provisioning phase

To access an application, CU_i has to send a request to SDC along with the application name, start time and end time [2]. The SDC returns the estimated cost ($\$p_{es}$) to CU_i. CU_i has to send the $\$p_{es}$ to BIC before the start time of the application. Estimating cost is unknown a priori and may not be possible in most cases. We assume that the cloud provider sends an exhaustive list of service chains of the service provided, which will be used to estimate the cost. Another option is to compute and feed the estimated cost to SDC manually. However, the cost estimation is beyond the scope of this Chapter, and we assume a proper cost estimation mechanism is already in place. After the receiving the $\$p_{es}$, the execution sequence of microservices begins. Every microservice (except the initial microservice) publish messages to MEC and receive the messages by listening to the events generated in the MEC contract. The server broker service (SMS) periodically collects resource usage records from the cloud servers and sends them to UDC. Similarly, an operating environment broker service ($OEBS$) periodically collects the values of environmental parameters

[2]We note that sometimes specifying the start time and end time is not possible. In that case, the user omits both fields.

and computes the cost of each parameter according to rates set by the cloud provider and sends the result to RCC. The $OEBS$ also collects the error data during service provisioning and sends the data to EDC.

Protocol: Service provisioning phase

For cloud user CU_i:

1. To access a cloud application, send $\textbf{trans}^{CU_i}_{callSer} = (sAppName, \tau_s, \tau_e)$ to SDC.
2. After receiving the estimated cost, send $\textbf{trans}^{CU_i}_{estCost} = (sAppName, \$p_{es})$ to BIC

For a microservice $sName$:

3. Subscribe to the communication events in the MEC.
4. To send message to another microservice, send $\textbf{trans}^{sName}_{call} = (sName_{callee}, endPoint_{callee}, inputs_{callee})$ to MEC
5. If an communication event is generated, process the received message.

For server broker service SBS:

6. Send $\textbf{trans}^{SBS}_{recUsage} = (\tau, CU_i, serviceID, r_j, us, un)$ to UDC for logging usage information.

For operating environment broker $OEBS$:

7. Send $\textbf{trans}^{OEBS}_{recEnv} = (\tau, r_j, price)$ to RCC for logging operating environment information.
8. Send $\textbf{trans}^{OEBS}_{recErr} = (\tau, serviceID, e_{name}, e_{weight})$ to EDC for logging operational error information.

For Blockchain BC: $list_d \leftarrow \{\}$, $list_{recUsage} \leftarrow \{\}$, $list_{recEnv} \leftarrow \{\}$, $list_{recError} \leftarrow \{\}$

9. On receiving $\textbf{trans}^{CU_i}_{callSer}$
   (a) Assert $CU_i \in list_{CU}$
   (b) Assert $(*, sAppName, *, *, *, *, *, *) \in list_{ms}$
   (c) Assert $\tau < \tau_s$
   (d) Compute and return the estimated cost for the requested service.
10. On receiving $\textbf{trans}^{CU_i}_{estCost}$
   (a) Assert $(CU_i, sAppName) \in list_{CU}$
   (b) Assert $ledger[CU_i] \geq \$p_{es}$
   (c) Set $ledger[CU_i] \leftarrow ledger[CU_i] - \$p_{es}$
   (d) Create a new $serviceID$ and return the $serviceID$.
   (e) Set $billPaid := false$, $startTime := \tau_s$, $endTime := \tau_e$, $cost := \$p_{es}$, $billGenTime = \tau_e + k$, $billPayTime = \tau_e + l$, $user = CU_i$.
   (f) Set $list_d \leftarrow list_d \cup (serviceID, user, cost, billPaid, startTime, endTime, billGenTime)$
11. On receiving $\textbf{trans}^{sName}_{call}$
   (a) Assert $sName \in list_{ms}$ and $sName_{callee} \in list_{ms}$

```
        (b) Notify $sName_{callee}$ about the call
    12. On receiving $\textbf{trans}^{SMB}_{recUsage}$
        (a) Set $list_{recUsage}[serviceID].rec[\tau].res[r_j].usage := us$
        (b) Set $list_{recUsage}[serviceID].rec[\tau].res[r_j].units := un$
    13. On receiving $\textbf{trans}^{OESB}_{recEnv}$
        (a) Set $list_{recEnv}.rec[\tau].value := price$
    14. On receiving $\textbf{trans}^{OESB}_{recErr}$
        (a) Set $list_{recError}[serviceID].rec[\tau].error = e_{name}$
        (b) Set $list_{recError}[serviceID].rec[\tau].weight = e_{weight}$
```

Figure 7.10: Service provisioning phase protocol

7.3.4.3 Billing phase

After the application execution, the user must request a bill by sending the $serviceID$ to BIC. The BIC contract fetches the usage details from UDC, rating and charging details from RCC, reputation details from RRC and computes the bill. The reputation of the cloud provider is calculated according to the errors reported during service provisioning. The final bill is generated after deducting the $\$p_{es}$ paid earlier. If the final bill is negative, the billing contract sends the excess amount to the user. Otherwise, depending on the behavior of CU_i, there are two cases as follows:

Case 1: CU_i has paid the final bill before the given time. In this case, the reputation of CU_i is not changed.

Case 2: CU_i has failed to pay the final bill before the given time. In this case, the new reputation value for CU_i is computed. The new value depends on the amount of payment the CU_i has defaulted.

7.4 Simulation Results and Discussions

The simulation environment is discussed in Section 1.2.3. The transactional and financial costs of the proposed RCB platform are shown in Table 7.1. The table consists of each contracts' deployment and its functionalities execution cost. We notice that the deployment

Protocol: Billing phase

```
For cloud user $CU_i$
  (1) To request a bill, send $\textbf{trans}^{CU_i}_{reqBill} = (serviceID)$ to $BIC$.
  (2) To pay the final cost, send $\textbf{trans}^{CU_i}_{finalBill} = (serviceID, \$pay)$ to $BIC$
For Blockchain $BC$
  (3) On receiving $\textbf{trans}^{CU_i}_{reqBill}$
        Assert $list_d[serviceID].billPaid = false$
        Assert $list_d[serviceID].endTime \leq \tau$
        Assert $list_d[serviceID].billGenTime \geq \tau$
        Assert $list_d[serviceID].user = CU_i$
        $\forall\, i \in [list_d[serviceID].startTime, list_d[serviceID].endTime]$
          $\forall\, j \in Resources$
            Set $Cost_{total} := Cost_{total} +$
            $list_{recUsage}[serviceID].rec[i].resource[j].units *$
            $list_{recEnv}.rec[i].resource[j].value$
          $E_{val} := E_{val} + list_{recError}[serviceID].rec[i].weight$
        Set $E_{val} := \frac{1}{E_{val}}$
        Set $R_{CP} \leftarrow \alpha * E_{val} + (1-\alpha) * R_{CP}$
        Set $Cost_p := Cost_{total} * R_{CP} * \frac{1}{R_{CU}} - D_{CU_i}$
        if $Cost_p \leq list_d[serviceID].cost$
          Set $ledger[CU_i] := ledger[CU_i] + (list_d[serviceID].cost - Cost_p)$
          Set $ledger[CP] := ledger[CP] + list_d[serviceID].cost$
          Set $list_d[serviceID].paid := true$
        else
          Set $Cost_{payable} := Cost_p - list_d[serviceID].cost$
          Set $ledger[CP] := ledger[CP] + list_d[serviceID].cost$
  (4) On receiving $\textbf{trans}^{CU_i}_{finalBill}$
        Assert $\$pay \geq Cost_{payable}$
        Assert $\tau < list_d[serviceID].billPayTime$
        Assert $ledger[CU_i] \geq \$pay$
        Set $ledger[CU_i] := ledger[CU_i] - \$pay$
        Set $ledger[CP] := ledger[CP] + \$pay$
        Set $list_d[serviceID].paid := true$
Timer
    if $list_d[serviceID].paid = false$ and $list_d[serviceID].billPayTime \geq \tau$
        If $cost_{payable} > A_i$ set $R_{val} = V_i$, Set $R_{CU_i} = \beta * R_{val} + (1-\beta) * R_{CU_i}$
```

Figure 7.11: Billing Phase

Contract	**Function**	**Caller**	**Cost in gas**	**Cost in $**
Registration (RRC)	Deployment	CP	241900	49.50
	Add user	CU_i	85751	17.54
Communication (CC)	Deployment	CP	764518	156.46
	Receive message	Microservice	78805	16.12
Service discovery (SDC)	Deployment	CP	832771	170.43
	Add service	CP	90097	18.43
	Add service chain	CP	37137	7.6
Usgae data (UDC)	Deployment	CP	398048	81.46
	Add record	SBS	108921	22.29
Rating (RAC)	Deployment	CP	132747	27.16
	Add record	OEBS	42949	8.79
Billing (BIC)	Deployment	CP	535968	109.69
	Compute bill	CU_i	See Figure 7.12	
	Pay bill	CU_i	40120	8.21
	Update user reputation	CP	32127	6.57

Table 7.1: Cost of interacting with proposed smart contracts

costs of smart contracts are high due to access to large contract storage. However, this a one time process and can be amortized over several interactions. Most of the computations take place in the billing contract, and the number of computations depends on the usage records collected during the service provisioning to a cloud user. The execution cost of the billing contract is presented in Figure 7.12. We observe that the execution cost increases with the increase in the number of usage records.

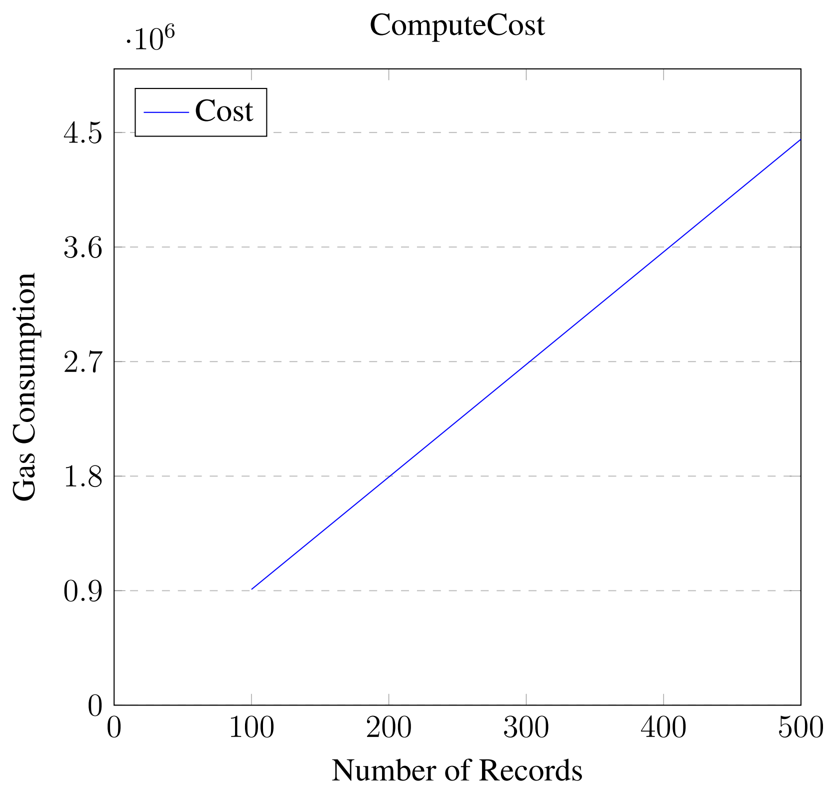

Figure 7.12: Gas Consumption of $computeBill$ functionality

7.5 Summary

In this work, we have designed a Blockchain-based registry and a Blockchain-based communication platform for microservices. We have developed a new cost computation model based on real-time usage and dynamic pricing of resources with respect to the state of the cloud operating environment. We have also considered the reputation of the cloud user and provider during bill generation. We have implemented the proposed system and presented the transactional and financial costs of the proposed system.

Chapter 8

Conclusion and Future Scope

8.1 Conclusions

This thesis has investigated the design of fair payment protocols for cloud services without a trusted intermediary.

In Chapter 2, we have presented a literature survey on existing Blockchain-based cloud services and listed the open issues in Blockchain-based cloud services.

In Chapter 3, we have designed fair payment protocols for proof-based and replication-based verifiable computation. Our theoretical analysis shows that our designed protocols are fair, and our experimental analysis using the Ethereum network shows the feasibility of our protocols. We have achieved fairness by imposing fines on cheating cloud providers and offering bounties to honest cloud providers.

In Chapter 4, we have proposed two fair payment protocols for the privacy-preserving aggregation of mobile crowdsensing data. Our protocols show that the untrusted data aggregator in traditional privacy-preserving aggregation (PPA) methods can be replaced by a smart contract running on a public Blockchain network. Unlike traditional PPA methods, we have achieved fairness without any additional cryptographic operations or trusted intermediaries. We have tested the protocols for the MotionSense dataset and presented the transactional and financial costs of interacting with the smart contracts. Our methods allow the data aggregator to know the dataset properties before buying the data without losing privacy.

In Chapter 5, a fair payment protocol for cloud resource allocation is proposed. Our protocol shows that the resource allocator in a traditional online auction can be replaced by a smart contract running on a public Blockchain network. Modeling an online auction algorithm as a smart contract also guarantees auction correctness. Our theoretical analysis shows that the proposed protocol is fair without any trusted intermediaries. We have tested the smart contract with real-world online auction configuration and presented the transactional and financial costs. We have also deployed the designed smart contract in the Ropsten test network and listed the transaction's addresses.

In Chapter 6, a fair payment protocol for cloud data de-duplication is proposed. We have designed a new incentive model for data de-duplication that is individually rational and incentive compatible. We have also designed a Blockchain-based data de-duplication protocol that satisfies correctness, uniform payments and fairness properties without a trusted intermediary. We have tested the designed smart contracts for the Debian dataset and shows that the Blockchain-based data de-duplication generates more profits for both cloud users and cloud providers when compared to user-controlled and provider-controlled data de-duplication. When compared to existing de-duplication methods, our method provides correctness of data de-duplication rate and financial fairness.

In Chapter 7, we have designed a fair rating, charging and billing (RCB) platform for microservices deployed in the cloud. We have designed a new cost computation model for microservices usage based on four factors: resource consumption of the cloud user, the reputation of the cloud user, the reputation of the cloud provider and the state of the operating environment. We have designed the RCB platform as a set of smart contracts running on a public Blockchain network. Our experiment analysis shows the transactional and financial costs of interacting with the designed smart contracts.

8.2 Future Scope

The following research directions are suggested for the future:

(1) Although we have considered several cloud services and developed fair payment protocols, they are still many services like Function-as-a-service, Security-as-a-service,

Virtual Network Function (VNF)-as-a-service, Benchmarking-as-a-service etc., for which fair payment protocols have to be developed.

(2) In most of our protocols, if both cloud user and provider are honest and follow the protocol correctly, then the overhead due to Blockchain is negligible. However, the execution of fair payment protocols in case of disputes is costly. Efficient off-chain dispute resolution techniques with little / no cost have to be developed.

(3) There are many automated security testing frameworks available for performing security analysis of a protocol. However, an automated fairness testing framework is not available. In future, an automated fairness testing framework may be developed.

(4) In this thesis, we have designed different fair payment protocols for different cloud services. However, having a generalized fair payment protocol for all the cloud services is beneficial to both cloud users and providers.

(5) The high write latency of public Blockchain systems makes most cloud applications unsuitable for real-time usage. Hence, research efforts are to be made to develop low latency real-time cloud applications using Blockchain.

(6) Different cloud providers are integrating their cloud infrastructure, giving rise to federated cloud computing. Fair payment protocols for federated cloud computing have to be developed.

Author's Publications

Journals:

1. Mallikarjun Reddy Dorsala, V. N. Sastry, and Sudhakar Chapram. "Fair payments for verifiable cloud services using smart contracts." *Computers & Security (Elsevier)*, 90:101712, March 2020.
   DOI: https://doi.org/10.1016/j.cose.2019.101712 (**Accepted & Published**)
2. Mallikarjun Reddy Dorsala, V. N. Sastry, and Sudhakar Chapram. "Fair payments for privacy-preserving aggregation of mobile crowdsensing data." *Journal of King Saud University - Computer and Information Sciences (Elsevier)*, 2021. DOI: https://doi.org/10.1016/j.jksuci.2021.01.009 (**Accepted & Published**)
3. Mallikarjun Reddy Dorsala, V. N. Sastry, and Sudhakar Chapram. "Blockchain-based solutions for cloud computing: a survey." *Journal of Networks and Computer Applications (Elsevier)*, 2021. DOI: https://doi.org/10.1016/j.jnca.2021.103246 (**Accepted & Published**)
4. Mallikarjun Reddy Dorsala, V. N. Sastry, and Sudhakar Chapram. "Blockchain-based online auction scheme for resource allocation in cloud computing with fair payments." *Journal of Ambient Intelligence and Humanized Computing (Springer)*. (**Under review**)
5. Mallikarjun Reddy Dorsala, V. N. Sastry, and Sudhakar Chapram. "Fair payments for secure cloud data deduplication using smart contracts." *Journal of Computer Science and Technology (Springer)*. (**Under review**)

Conferences:

1. Mallikarjun Reddy Dorsala, V. N. Sastry, and Sudhakar Chapram. "Fair Protocols for Verifiable Computations Using Bitcoin and Ethereum." *In Proceedings of the 2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, San Francisco, U.S.A, pp. 786-793, 2018.

Bibliography

[1] Peter Mell and Timothy Grance. The NIST Definition of Cloud Computing (Draft). https://www.nist.gov/system/files/documents/itl/cloud/NIST$_S$P-500-291$_V$ersion-2$_2$013$_J$une18$_F$INAL.pdf. Online; accessed 11 November 2020.

[2] Nadarajah Asokan. Fairness in electronic commerce. 1998. PhD. thesis, University of Waterloo.

[3] Henning Pagnia and Felix C Gärtner. On the impossibility of fair exchange without a trusted third party. Technical report, Technical Report TUD-BS-1999-02, Darmstadt University of Technology, 1999.

[4] Henning Pagnia, Holger Vogt, and Felix C Gärtner. Fair exchange. *The Computer Journal*, 46(1):55–75, 2003.

[5] Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System. https://git.dhimmel.com/bitcoin-whitepaper/. Online; accessed 11 November 2020.

[6] Meni Rosenfeld. Overview of Colored Coins. https://bitcoil.co.il/BitcoinX.pdf. Online; accessed 11 November 2020.

[7] S. Wang, L. Ouyang, Y. Yuan, X. Ni, X. Han, and F. Wang. Blockchain-Enabled Smart Contracts: Architecture, Applications, and Future Trends. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 49(11):2266–2277, November 2019.

[8] M. Conoscenti, A. Vetrò, and J. C. De Martin. Blockchain for the Internet of Things: A systematic literature review. In *2016 IEEE/ACS 13th International Conference of Computer Systems and Applications (AICCSA), Agadir, Morocco*, pages 1–6. IEEE, November 2016.

[9] Victoria Louise Lemieux. Trusting records: Is Blockchain technology the answer? *Records Management Journal*, 26(2):110–139, July 2016.

[10] Keke Gai, Jinnan Guo, Liehuang Zhu, and Shui Yu. Blockchain Meets Cloud Computing: A Survey. *IEEE Communications Surveys & Tutorials*, 22(3):2009–2030, 23.

[11] Charles Noyes. BitAV: Fast Anti-Malware by Distributed Blockchain Consensus and Feedforward Scanning. https://arxiv.org/pdf/1601.01405.pdf, January 2016. Online; accessed 11 November 2020.

[12] Mike Sharples and John Domingue. The Blockchain and Kudos: A Distributed System for Educational Record, Reputation and Reward. In *Proceedings of Adaptive and Adaptable Learning, Lyon, France*, pages 490–496. Springer, 2016.

[13] Dinh C. Nguyen, Pubudu N. Pathirana, Ming Ding, and Aruna Seneviratne. Blockchain for 5G and beyond networks: A state of the art survey. *Journal of Network and Computer Applications*, 166:102693, September 2020.

[14] Marco Iansiti and Karim R Lakhani. The truth about blockchain harvard business review. https://hbr.org/2017/01/the-truth-about-blockchain. Online; accessed 11 November 2020.

[15] Changyu Dong, Yilei Wang, Amjad Aldweesh, Patrick McCorry, and Aad van Moorsel. 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, Dallas, Texas, USA*, pages 211–227. ACM, October 2017.

[16] Massimo Bartoletti and Livio Pompianu. An empirical analysis of smart contracts: Platforms, applications, and design patterns. In *Proceedings of the 2017 International conference on Financial Cryptography and Data Security, Sliema, Malta*, pages 494–509. Springer, April, 2017.

[17] Xiaomin Bai, Zijing Cheng, Zhangbo Duan, and Kai Hu. Formal modeling and verification of smart contracts. In *Proceedings of the 2018 7th International Conference on Software and Computer Applications,Kuantan, Malaysia*, pages 322–326, February 2018.

[18] Faheem Zafar, Abid Khan, Saif Ur Rehman Malik, Mansoor Ahmed, Adeel Anjum, Majid Iqbal Khan, Nadeem Javed, Masoom Alam, and Fuzel Jamil. A survey of cloud computing data integrity schemes: Design challenges, taxonomy and future trends. *Computers & Security*, 65:29–49, March 2017.

[19] S. Wang, K. Liang, J. K. Liu, J. Chen, J. Yu, and W. Xie. Attribute-Based Data Sharing Scheme Revisited in Cloud Computing. *IEEE Transactions on Information Forensics and Security*, 11(8):1661–1673, August 2016.

[20] Qi Feng, Debiao He, Sherali Zeadally, Muhammad Khurram Khan, and Neeraj Kumar. A survey on privacy protection in blockchain system. *Journal of Network and Computer Applications*, 126:45–58, January 2019.

[21] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou. Hawk: The Blockchain Model of Cryptography and Privacy-Preserving Smart Contracts. In *2016 IEEE Symposium on Security and Privacy (SP), San Jose, CA, USA*, pages 839–858. IEEE, May 2016.

[22] Ari Juels, Ahmed Kosba, and Elaine Shi. The Ring of Gyges: Investigating the Future of Criminal Smart Contracts. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria*, pages 283–295. ACM, October 2016.

[23] Gavin Wood et al. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151(2014):1–32, 2014.

[24] Solidity Documentation. https://readthedocs.org/projects/solidity/downloads/pdf/v0.5.8/. Online; accessed Dec 2020.

[25] Truffle suite. https://trufflesuite.com/docs/index. Online; accessed Dec 2020.

[26] F. Tschorsch and B. Scheuermann. Bitcoin and Beyond: A Technical Survey on Decentralized Digital Currencies. *IEEE Communications Surveys Tutorials*, 18(3):2084–2123, thirdquarter 2016.

[27] Wenbo Wang, Dinh Thai Hoang, Peizhao Hu, Zehui Xiong, Dusit Niyato, Ping Wang, Yonggang Wen, and Dong In Kim. A Survey on Consensus Mechanisms and Mining Strategy Management in Blockchain Networks. *IEEE Access*, 7:22328–22370, 2019.

[28] Hong-Ning Dai, Zibin Zheng, and Yan Zhang. Blockchain for Internet of Things: A Survey. *IEEE Internet of Things Journal*, 6(5):8076–8094, October 2019.

[29] Muhammad Salek Ali, Massimo Vecchio, Miguel Pincheira, Koustabh Dolui, Fabio Antonelli, and Mubashir Husain Rehmani. Applications of Blockchains in the Internet of Things: A Comprehensive Survey. *IEEE Communications Surveys Tutorials*, 21(2):1676–1717, Secondquarter 2019.

[30] Shaoan Xie, Zibin Zheng, Weili Chen, Jiajing Wu, Hong-Ning Dai, and Muhammad Imran. Blockchain for cloud exchange: A survey. *Computers & Electrical Engineering*, 81:106526, January 2020.

[31] Ruizhe Yang, F. Richard Yu, Pengbo Si, Zhaoxin Yang, and Yanhua Zhang. Integrated Blockchain and Edge Computing Systems: A Survey, Some Research Issues and Challenges. *IEEE Communications Surveys & Tutorials*, 21(2):1508–1532, 2019.

[32] Jesse Yli-Huumo, Deokyoon Ko, Sujin Choi, Sooyong Park, and Kari Smolander. Where Is Current Research on Blockchain Technology?—A Systematic Review. *PLOS ONE*, 11(10):e0163477, October 2016.

[33] J. Li and B. Li. Erasure coding for cloud storage systems: A survey. *Tsinghua Science and Technology*, 18(3):259–272, June 2013.

[34] Guy Zyskind, Oz Nathan, and Alex 'Sandy' Pentland. Decentralizing Privacy: Using Blockchain to Protect Personal Data. In *2015 IEEE Security and Privacy Workshops, San Jose, CA, USA*, pages 180–184. IEEE, May, 2015.

[35] Hossein Shafagh, Lukas Burkhalter, Anwar Hithnawi, and Simon Duquennoy. Towards Blockchain-based Auditable Storage and Sharing of IoT Data. In *Proceedings of the 2017 on Cloud Computing Security Workshop - CCSW '17, Dallas, Texas, USA*, pages 45–50. ACM Press, November 2017.

[36] Xueping Liang, Juan Zhao, Sachin Shetty, and Danyi Li. Towards data assurance and resilience in IoT using blockchain. In *MILCOM 2017 - 2017 IEEE Military Communications Conference (MILCOM), Baltimore, MD, USA*, pages 261–266. IEEE, October 2017.

[37] Edoardo Gaetani, Leonardo Aniello, Roberto Baldoni, Federico Lombardi, Andrea Margheri, and Vladimiro Sassone. Blockchain-based Database to Ensure Data Integrity in Cloud Computing Environments. In *Italian Conference on Cybersecurity, Venice, Italy*, January 2017.

[38] Shangping Wang, Yinglong Zhang, and Yaling Zhang. A Blockchain-Based Framework for Data Sharing With Fine-Grained Access Control in Decentralized Storage Systems. *IEEE Access*, 6:38437–38450, 2018.

[39] Chunhua Li, Jiaqi Hu, Ke Zhou, Yuanzhang Wang, and Hongyu Deng. Using Blockchain for Data Auditing in Cloud Storage. In *Cloud Computing and Security*, volume 11065, pages 335–345. Springer International Publishing, August 2018.

[40] Liehuang Zhu, Yulu Wu, Keke Gai, and Kim-Kwang Raymond Choo. Controllable and trustworthy blockchain-based cloud data management. *Future Generation Computer Systems*, 91:527–535, February 2019.

[41] Jingting Xue, Chunxiang Xu, Jining Zhao, and Jianfeng Ma. Identity-based public auditing for cloud storage systems against malicious auditors via blockchain. *Science China Information Sciences*, 62(3):32104, March 2019.

[42] Yang Xu, Ju Ren, Yan Zhang, Cheng Zhang, Bo Shen, and Yaoxue Zhang. Blockchain Empowered Arbitrable Data Auditing Scheme for Network Storage as a Service. *IEEE Transactions on Services Computing*, 13(2):289–300, March 2020.

[43] Pei Huang, Kai Fan, Hanzhe Yang, Kuan Zhang, Hui Li, and Yintang Yang. A Collaborative Auditing Blockchain for Trustworthy Data Integrity in Cloud Storage System. *IEEE Access*, 8:94780–94794, November 2020.

[44] Qi Xia, Emmanuel Sifah, Abla Smahi, Sandro Amofa, and Xiaosong Zhang. BBDS: Blockchain-Based Data Sharing for Electronic Medical Records in Cloud Environments. *Information*, 8(2):44, April 2017.

[45] Qi Xia, Emmanuel Boateng Sifah, Kwame Omono Asamoah, Jianbin Gao, Xiaojiang Du, and Mohsen Guizani. MeDShare: Trust-Less Medical Data Sharing Among Cloud Service Providers via Blockchain. *IEEE Access*, 5:14757–14767, July 2017.

[46] Hongyu Li, Liehuang Zhu, Meng Shen, Feng Gao, Xiaoling Tao, and Sheng Liu. Blockchain-Based Data Preservation System for Medical Data. *Journal of Medical Systems*, 42(8):141, August 2018.

[47] Dinh C. Nguyen, Pubudu N. Pathirana, Ming Ding, and Aruna Seneviratne. Blockchain for Secure EHRs Sharing of Mobile Cloud Based E-Health Systems. *IEEE Access*, 7:66792–66806, May 2019.

[48] Sheng Cao, Gexiang Zhang, Pengfei Liu, Xiaosong Zhang, and Ferrante Neri. Cloud-assisted secure eHealth systems for tamper-proofing EHR via blockchain. *Information Sciences*, 485:427–440, June 2019.

[49] T. Benil and J. Jasper. Cloud based security on outsourcing using blockchain in E-health systems. *Computer Networks*, 178:107344, September 2020.

[50] Haiping Huang, Xiang Sun, Fu Xiao, Peng Zhu, and Wenming Wang. Blockchain-based eHealth system for auditable EHRs manipulation in cloud environments. *Journal of Parallel and Distributed Computing*, 148:46–57, February 2021.

[51] Longxia Huang, Gongxuan Zhang, Shui Yu, Anmin Fu, and John Yearwood. SeShare: Secure cloud data sharing based on blockchain and public auditing. *Concurrency and Computation: Practice and Experience*, 31(22), September 2017.

[52] Aravind Ramachandran and Dr Murat Kantarcioglu. Using Blockchain and smart contracts for secure data provenance management. https://arxiv.org/pdf/1709.10000.pdf, September 2017. Online; accesed on 14 may 2021.

[53] Yuan Zhang, Xiaodong Lin, and Chunxiang Xu. Blockchain-Based Secure Data Provenance for Cloud Storage. In *Information and Communications Security*, volume 11149, pages 3–19. Springer International Publishing, Cham, 2018.

[54] Thomas Renner, Johannes Muller, and Odej Kao. Endolith: A Blockchain-Based Framework to Enhance Data Retention in Cloud Storages. In *2018 26th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP),Cambridge, UK*, pages 627–634. IEEE, March 2018.

[55] Xiaodong Yang, Xizhen Pei, Meiding Wang, Ting Li, and Caifen Wang. Multi-Replica and Multi-Cloud Data Public Audit Scheme Based on Blockchain. *IEEE Access*, 8:144809–144822, July 2020.

[56] Yuan Zhang, Chunxiang Xu, Nan Cheng, Hongwei Li, Haomiao Yang, and Xuemin Sherman Shen. Chronos+: An Accurate Blockchain-based Time-stamping Scheme for Cloud Storage. *IEEE Transactions on Services Computing*, 13(2):216–229, March 2020.

[57] Gabriel Estevam, Lucas M. Palma, Luan R. Silva, Jean E. Martina, and Martín Vigil. Accurate and decentralized timestamping using smart contracts on the Ethereum blockchain. 58(3), 2021.

[58] Stephen Kirkman and Richard Newman. A Cloud Data Movement Policy Architecture Based on Smart Contracts and the Ethereum Blockchain. In *2018 IEEE International Conference on Cloud Engineering (IC2E), Orlando, FL,USA*, pages 371–377. IEEE, April 2018.

[59] Haochen Li, Keke Gai, Zhengkang Fang, Liehuang Zhu, Lei Xu, and Peng Jiang. Blockchain-enabled Data Provenance in Cloud Datacenter Reengineering. In *Proceedings of the 2019 ACM International Symposium on Blockchain and Secure Critical Infrastructure - BSCI '19, Auckland, New Zealand*, pages 47–55. ACM Press, August 2019.

[60] Changsong Yang, Xiaofeng Chen, and Yang Xiang. Blockchain-based publicly verifiable data deletion scheme for cloud storage. *Journal of Network and Computer Applications*, 103:185–193, February 2018.

[61] Kai Fan, Yanhui Ren, Yue Wang, Hui Li, and Yingtang Yang. Blockchain-based efficient privacy preserving and data sharing scheme of content-centric network in 5G. *IET Communications*, 12(5):527–532, March 2018.

[62] Xiaochen Zheng, Raghava Rao Mukkamala, Ravi Vatrapu, and Joaqun Ordieres-Mere. Blockchain-based Personal Health Data Sharing System Using Cloud Storage. In *2018 IEEE 20th International Conference on E-Health Networking, Applications and Services (Healthcom), Ostrava, Czech Republic*, pages 1–6. IEEE, September 2018.

[63] Mu Yang, Andrea Margheri, Runshan Hu, and Vladimiro Sassone. Differentially Private Data Sharing in a Cloud Federation with Blockchain. *IEEE Cloud Computing*, 5(6):69–79, November 2018.

[64] Jingting Xue, Chunxiang Xu, Yuan Zhang, and Lanhua Bai. DStore: A Distributed Cloud Storage System Based on Smart Contracts and Blockchain. In *Algorithms and Architectures for Parallel Processing,Guangzhou, China*, volume 11336, pages 385–401. Springer International Publishing, November 2018.

[65] Longxia Huang, Junlong Zhou, Gongxuan Zhang, Jin Sun, Tongquan Wei, Shui Yu, and Shiyan Hu. IPANM: Incentive Public Auditing Scheme for Non-Manager Groups in Clouds. *IEEE Transactions on Dependable and Secure Computing*, pages 1–1, June 2020.

[66] Shorouq Alansari, Federica Paci, and Vladimiro Sassone. A Distributed Access Control System for Cloud Federations. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS),Atlanta, GA, USA*, pages 2131–2136. IEEE, June 2017.

[67] Maryline Laurent, Nesrine Kaaniche, Christian Le, and Mathieu Vander Plaetse. A Blockchain based Access Control Scheme. In *Proceedings of the 15th International Joint Conference on E-Business and Telecommunications, Porto, Portugal*, pages 168–176. SCITEPRESS - Science and Technology Publications, 2018.

[68] Jason Paul Cruz, Yuichi Kaji, and Naoto Yanai. RBAC-SC: Role-Based Access Control Using Smart Contract. *IEEE Access*, 6:12240–12251, 2018.

[69] YongJoo Lee and Keon Myung Lee. Blockchain-based RBAC for user authentication with anonymity. In *Proceedings of the Conference on Research in Adaptive and Convergent Systems,Chongqing, China*, pages 289–294. ACM, September 2019.

[70] Arnab Chatterjee, Yash Pitroda, and Manojkumar Parmar. Dynamic Role-Based Access Control for Decentralized Applications. In *International conference on Blockchain - ICBC, Honolulu, HI, USA*.

[71] Ronghua Xu, Yu Chen, Erik Blasch, and Genshe Chen. BlendCAC: A Smart Contract Enabled Decentralized Capability-Based Access Control Mechanism for the IoT. *Computers*, 7(3):39, July 2018.

[72] Damiano Di Francesco Maesa, Paolo Mori, and Laura Ricci. A blockchain based approach for the definition of auditable Access Control systems. *Computers & Security*, 84:93–119, July 2019.

[73] Hao Guo, Ehsan Meamari, and Chien-Chung Shen. Multi-Authority Attribute-Based Access Control with Smart Contract. In *Proceedings of the 2019 International Conference on Blockchain Technology - ICBCT, Honolulu, HI, USA*, pages 6–11. ACM Press, 2019.

[74] Yuanyu Zhang, Shoji Kasahara, Yulong Shen, Xiaohong Jiang, and Jianxiong Wan. Smart Contract-Based Access Control for the Internet of Things. *IEEE Internet of Things Journal*, 6(2):1594–1605, April 2019.

[75] Caixia Yang, Liang Tan, Na Shi, Bolei Xu, Yang Cao, and Keping Yu. AuthPrivacyChain: A Blockchain-Based Access Control Framework With Privacy Protection in Cloud. *IEEE Access*, 8:70604–70615, 2020.

[76] L. Guo, X. Yang, and W.-C. Yau. TABE-DAC: Efficient Traceable Attribute-Based Encryption Scheme With Dynamic Access Control Based on Blockchain. *IEEE Access*, 9:8479–8490, 2021.

[77] Shengshan Hu, Chengjun Cai, Qian Wang, Cong Wang, Xiangyang Luo, and Kui Ren. Searching an Encrypted Cloud Meets Blockchain: A Decentralized, Reliable and Fair Realization. In *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications, Honolulu, HI, USA*, pages 792–800. IEEE, April 2018.

[78] Lanxiang Chen, Wai-Kong Lee, Chin-Chen Chang, Kim-Kwang Raymond Choo, and Nan Zhang. Blockchain based searchable encryption for electronic health record sharing. *Future Generation Computer Systems*, 95:420–429, June 2019.

[79] Yinghui Zhang, Robert H. Deng, Jiangang Shu, Kan Yang, and Dong Zheng. TKSE: Trustworthy Keyword Search Over Encrypted Data With Two-Side Verifiability via Blockchain. *IEEE Access*, 6:31077–31087, June 2018.

[80] Shunrong Jiang, Jianqing Liu, Liangmin Wang, and Seong-Moo Yoo. Verifiable Search Meets Blockchain: A Privacy-Preserving Framework for Outsourced Encrypted Data. In *ICC 2019 - 2019 IEEE International Conference on Communications (ICC), Shanghai, China*, pages 1–6. IEEE, May 2019.

[81] Shan Jiang, Jiannong Cao, Julie A. McCann, Yanni Yang, Yang Liu, Xiaoqing Wang, and Yuming Deng. Privacy-Preserving and Efficient Multi-Keyword Search over Encrypted Data on Blockchain. In *2019 IEEE International Conference on Blockchain (Blockchain),Atlanta, GA, USA*, pages 405–410. IEEE, July 2019.

[82] Agipa Aigissinova, Hieu Hanh Le, and Haruo Yokota. Evaluation of the performance of secure keyword search using bit-string signatures in the blockchain. https://db-event.jpn.org/deim2020/post/proceedings/papers/E1-5.pdf, 2020. Online; accessed on 14 May 2021.

[83] Chengjun Cai, Jian Weng, Xingliang Yuan, and Cong Wang. Enabling Reliable Keyword Search in Encrypted Decentralized Storage with Fairness. 18(1):131–144, January 2021.

[84] Yang Yang, Hongrui Lin, Ximeng Liu, Wenzhong Guo, Xianghan Zheng, and Zhiquan Liu. Blockchain-Based Verifiable Multi-Keyword Ranked Search on Encrypted Cloud With Fair Payment. *IEEE Access*, 7:140818–140832, September 2019.

[85] Chao Zhang, Shaojing Fu, and Weijun Ao. A blockchain based searchable encryption scheme for multiple cloud storage. In Jaideep Vaidya, Xiao Zhang, and Jin Li, editors, *Cyberspace Safety and Security, Guangzhou, China*, pages 585–600. Springer International Publishing, December 2019.

[86] Shaojing Fu, Chao Zhang, and Weijun Ao. Searchable encryption scheme for multiple cloud storage using double-layer blockchain. *Concurrency and Computation: Practice and Experience*, April 2020.

[87] Shufen Niu, Lixia Chen, Jinfeng Wang, and Fei Yu. Electronic Health Record Sharing Scheme With Searchable Attribute-Based Encryption on Blockchain. *IEEE Access*, 8:7195–7204, June 2019.

[88] Qiang Tang. Towards blockchain-enabled searchable encryption, copenhagen, denmark. In Jianying Zhou, Xiapu Luo, Qingni Shen, and Zhen Xu, editors, *Information and Communications Security,*, pages 482–500. Springer International Publishing, 2020.

[89] Xixi Yan, Xiaohan Yuan, Qing Ye, and Yongli Tang. Blockchain-Based Searchable Encryption Scheme With Fair Payment. *IEEE Access*, 8:109687–109706, June 2020.

[90] Peng Jiang, Fuchun Guo, Kaitai Liang, Jianchang Lai, and Qiaoyan Wen. Searchain: Blockchain-based private keyword search in decentralized storage. *Future Generation Computer Systems*, 107:781–792, June 2020.

[91] Yandong Li, Liehuang Zhu, Meng Shen, Feng Gao, Baokun Zheng, Xiaojiang Du, Sheng Liu, and Shu Yin. CloudShare: Towards a Cost-Efficient and Privacy-Preserving Alliance Cloud Using Permissioned Blockchains. In *Mobile Networks and Management, Melbourne, VIC, Australia*, volume 235, pages 339–352. Springer International Publishing, December 2017.

[92] Jingyi Li, Jigang Wu, Long Chen, and Xi'an China Li, Jiaxing. Deduplication with Blockchain for Secure Cloud Storage. In *Big Data*, volume 945, pages 558–570. Springer Singapore, 2018.

[93] Shangping Wang, Yuying Wang, and Yaling Zhang. Blockchain-Based Fair Payment Protocol for Deduplication Cloud Storage System. *IEEE Access*, 7:127652–127668, 2019.

[94] Sunny King and Scott Nadal. PPCoin: Peer-to-Peer Crypto-Currency with Proof-of-Stake. https://www.chainwhy.com/upload/default/20180619/126a057fef926dc286accb372da46955.pdf. Online; accessed on 14 may 2021.

[95] Intel Corporation. Intel(r) software guard extensions (intel(r) sgx) sdk. https://software.intel.com/en-us/sgx-sdk, 2015. Online; accessed on 14 May 2021.

[96] OASIS Standard. extensible access control markup language (xacml) version 3.0. http://docs.oasis-open.org/xacml/2.0/access$_c$ontrol-xacml-2.0-core-spec-os.pdf, 2013. Online; accessed on 14 May 2021.

[97] Sanjay Jain, Prateek Saxena, Frank Stephan, and Jason Teutsch. How to verify computation with a rational network. https://arxiv.org/pdf/1606.05917.pdf, June 2016. Online; accessed on 14 May 2021.

[98] J. Li, Q. Wang, C. Wang, N. Cao, K. Ren, and W. Lou. Fuzzy Keyword Search over Encrypted Data in Cloud Computing. In *2010 Proceedings IEEE INFOCOM, San Diego, CA, USA*, pages 1–5. IEEE, March 2010.

[99] X. Zhu, Q. Liu, and G. Wang. A Novel Verifiable and Dynamic Fuzzy Keyword Search Scheme over Encrypted Data in Cloud Computing. In *2016 IEEE Trustcom/BigDataSE/ISPA, Tianjin, China*, pages 845–851. IEEE, August 2016.

[100] Wakaha Ogata and Kaoru Kurosawa. Oblivious keyword search. *Journal of Complexity*, 20(2-3):356–371, April 2004.

[101] Dutch T Meyer and William J Bolosky. A study of practical deduplication. *ACM Transactions on Storage*, 7(4):20.

[102] Abdullah Yousafzai, Abdullah Gani, Rafidah Md Noor, Mehdi Sookhak, Hamid Talebian, Muhammad Shiraz, and Muhammad Khurram Khan. Cloud resource allocation schemes: Review, taxonomy, and opportunities. *Knowledge and Information Systems*, 50(2):347–381, February 2017.

[103] Yonggen Gu, Dingding Hou, and Xiaohong Wu. A Cloud Storage Resource Transaction Mechanism Based on Smart Contract. In *Proceedings of the 8th International Conference on Communication and Network Security - ICCNS 2018, Qingdao, China*, pages 134–138. ACM Press, February 2018.

[104] Aleksandr Zavodovski, Suzan Bayhan, Nitinder Mohan, Pengyuan Zhou, Walter Wong, and Jussi Kangasharju. DeCloud: Truthful Decentralized Double Auction for Edge Clouds. In *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS), Dallas, TX, USA*, pages 2157–2167. IEEE, July 2019.

[105] Tonglai Liu, Jigang Wu, Long Chen, Yalan Wu, and Yinan Li. Smart Contract-Based Long-Term Auction for Mobile Blockchain Computation Offloading. *IEEE Access*, 8:36029–36042, February 2020.

[106] Zhili Chen, Wei Ding, Yan Xu, Miaomiao Tian, and Hong Zhong. Fair Auction and Trade Framework for Cloud VM Allocation based on Blockchain. https://arxiv.org/pdf/2001.00771.pdf, January 2020. Online; accessed on 14 May 2021.

[107] Zixuan Xie, Run Wu, Miao Hu, and Haibo Tian. Blockchain-Enabled Computing Resource Trading: A Deep Reinforcement Learning Approach. In *2020 IEEE Wireless Communications and Networking Conference (WCNC),Seoul, Korea (South)*, pages 1–8. IEEE, May 2020.

[108] Sambit Nayak, Nanjangud C Narendra, Anshu Shukla, and James Kempf. Saranyu: Using Smart Contracts and Blockchain for Cloud Tenant Management. In *2018 IEEE 11th International Conference on Cloud Computing (CLOUD),San Francisco, CA, USA*, pages 857–861. IEEE, July 2018.

[109] Vorameth Reantongcome, Vasaka Visoottiviseth, Wudhichart Sawangphol, Assadarat Khurat, Shigeru Kashihara, and Doudou Fall. Securing and Trustworthy Blockchain-based Multi-Tenant Cloud Computing. In *2020 IEEE 10th Symposium on Computer Applications & Industrial Electronics (ISCAIE), Malaysia*, pages 256–261. IEEE, April 2020.

[110] Jianli Pan, Jianyu Wang, Austin Hester, Ismail Alqerm, Yuanni Liu, and Ying Zhao. EdgeChain: An Edge-IoT Framework and Prototype Based on Blockchain and Smart Contracts. *IEEE Internet of Things Journal*, 6(3):4719–4732, October 2018.

[111] Zhen Zhang, Zicong Hong, Wuhui Chen, Zibin Zheng, and Xu Chen. Joint Computation Offloading and Coin Loaning for Blockchain-Empowered Mobile-Edge Computing. *IEEE Internet of Things Journal*, 6(6):9934–9950, December 2019.

[112] Zhenni Li, Zuyuan Yang, Shengli Xie, Wuhui Chen, and Kang Liu. Credit-Based Payments for Fast Computing Resource Trading in Edge-Assisted Internet of Things. *IEEE Internet of Things Journal*, 6(4):6606–6617, August 2019.

[113] Wen Sun, Jiajia Liu, Yanlin Yue, and Peng Wang. Joint Resource Allocation and Incentive Design for Blockchain-Based Mobile Edge Computing. 19(9), 2020.

[114] Lanfranco Zanzi, Antonio Albanese, Vincenzo Sciancalepore, and Xavier Costa-Perez. NSBchain: A Secure Blockchain Framework for Network Slicing Brokerage. In *ICC 2020 - 2020 IEEE International Conference on Communications (ICC), Dublin, Ireland*.

[115] Bo Zhao, Peiru Fan, and Mingtao Ni. Mchain: A Blockchain-Based VM Measurements Secure Storage Approach in IaaS Cloud With Enhanced Integrity and Controllability. *IEEE Access*, 6:43758–43769, January 2018.

[116] Tongchen Wang, Jianwei Liu, Dawei Li, and Qianhong Wu. A Blockchain-Based Resource Supervision Scheme for Edge Devices Under Cloud-Fog-End Computing Models. In Joseph K. Liu and Hui Cui, editors, *Information Security and Privacy, Perth, WA, Australia*, volume 12248, pages 285–305. Springer International Publishing, 2020.

[117] W. Wang, B. Li, and B. Liang. Towards Optimal Capacity Segmentation with Hybrid Cloud Pricing. In *2012 IEEE 32nd International Conference on Distributed Computing Systems, Macau, China*, pages 425–434. IEEE, June 2012.

[118] Weijie Shi, Linquan Zhang, Chuan Wu, Zongpeng Li, and Francis C.M. Lau. An online auction framework for dynamic resource provisioning in cloud computing. In *The 2014 ACM International Conference on Measurement and Modeling of Computer Systems - SIGMETRICS '14, Austin, Texas, USA*, pages 71–83. ACM Press, 2014.

[119] H. Zhang, H. Jiang, B. Li, F. Liu, A. V. Vasilakos, and J. Liu. A Framework for Truthful Online Auctions in Cloud Computing with Heterogeneous User Demands. *IEEE Transactions on Computers*, 65(3):805–818, March 2016.

[120] Hal R. Varian and Christopher Harris. The VCG Auction in Theory and Practice. *American Economic Review*, 104(5):442–445, May 2014.

[121] Tim Roughgarden. Algorithmic game theory. *Communications of the ACM*, 53(7):78–86, July 2010.

[122] Iddo Bentov and Ranjit Kumaresan. How to Use Bitcoin to Design Fair Protocols. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology – CRYPTO, Santa Barbara, CA, USA*, Lecture Notes in Computer Science, pages 421–439, August 2014.

[123] J. P. Morgan Chase. A Permissioned Implementation of Ethereum. https://github.com/ConsenSys/quorum, 2018.

[124] James Kempf, Sambit Nayak, Remi Robert, Jim Feng, Kunal Rajan Deshmukh, Anshu Shukla, Aleksandra Obeso Duque, Nanjangud Narendra, and Johan Sjöberg. The Nubo Virtual Services Marketplace. https://arxiv.org/ftp/arxiv/papers/1909/1909.04934.pdf, November 2019. Online; accessed on 14 May 2021.

[125] Ranjit Kumaresan and Iddo Bentov. How to Use Bitcoin to Incentivize Correct Computations. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security - CCS '14, Scottsdale, Arizona, USA*, pages 30–41. ACM Press, 2014.

[126] Yinghui Zhang, Robert Deng, Ximeng Liu, and Dong Zheng. Outsourcing Service Fair Payment based on Blockchain and its Applications in Cloud Computing. *IEEE Transactions on Services Computing*, pages 1–1, August 2018.

[127] Yinghui Zhang, Robert H. Deng, Ximeng Liu, and Dong Zheng. Blockchain based efficient and robust fair payment for outsourcing services in cloud computing. *Information Sciences*, 462:262–277, September 2018.

[128] Jacob Eberhardt and Stefan Tai. ZoKrates - Scalable Privacy-Preserving Off-Chain Computations. In *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), Halifax, NS, Canada*, pages 1084–1091. IEEE, July 2018.

[129] Mallikarjun Reddy Dorsala, V.N. Sastry, and Sudhakar Chapram. Fair payments for verifiable cloud services using smart contracts. *Computers & Security*, 90:101712, March 2020.

[130] Y. Guan, H. Zheng, J. Shao, R. Lu, and G. Wei. Fair Outsourcing Polynomial Computation Based on the Blockchain. *IEEE Transactions on Services Computing*, pages 1–1, 2021.

[131] Sepideh Avizheh, Mahmudun Nabi, Reihaneh Safavi-Naini, and Muni Venkateswarlu K. Verifiable Computation using Smart Contracts. In *Proceedings of the 2019 ACM SIGSAC Conference on Cloud Computing Security Workshop - CCSW'19, London, United Kingdom*, pages 17–28. ACM Press, November 2019.

[132] Jason Teutsch and Christian Reitwießner. A scalable verification solution for blockchains. https://arxiv.org/pdf/1908.04756.pdf, November 2017. Online; accessed on 14 May 2021.

[133] Dominik Harz and Magnus Boman. The Scalability of Trustless Trust.

[134] Michal Król and Ioannis Psaras. SPOC: Secure Payments for Outsourced Computations. https://arxiv.org/pdf/1807.06462.pdf, July 2018. Online; accessed on 14 May 2021.

[135] Mahmudun Nabi, Sepideh Avizheh, Muni Venkateswarlu Kumaramangalam, and Reihaneh Safavi-Naini. Game-Theoretic Analysis of an Incentivized Verifiable Computation System. In *Financial Cryptography and Data Security, Kota Kinabalu, Malaysia*, volume 11599, pages 50–66. Springer International Publishing, 2020.

[136] Scott Eisele, Taha Eghtesad, Nicholas Troutman, Aron Laszka, and Abhishek Dubey. Mechanisms for Outsourcing Computation via a Decentralized Market. In *Proceedings of the 14th ACM International Conference on Distributed and Event-based Systems, Montreal, Quebec, Canada*, pages 61–72. ACM, July 2020.

[137] Jingzhong Wang, Mengru Li, Yunhua He, Hong Li, Ke Xiao, and Chao Wang. A Blockchain Based Privacy-Preserving Incentive Mechanism in Crowdsensing Applications. *IEEE Access*, 6:17545–17556, 2018.

[138] Chengjun Cai, Yifeng Zheng, and Cong Wang. Leveraging Crowdsensed Data Streams to Discover and Sell Knowledge: A Secure and Efficient Realization. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS), Vienna, Austria*, pages 589–599. IEEE, July 2018.

[139] Yuan Lu, Qiang Tang, and Guiling Wang. ZebraLancer: Private and Anonymous Crowdsourcing System atop Open Blockchain. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS), Vienna, Austria*, pages 853–865, July 2018.

[140] Fengrui Shi, Zhijin Qin, Di Wu, and Julie McCann. MPCSToken: Smart Contract Enabled Fault-Tolerant Incentivisation for Mobile P2P Crowd Services. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS), Vienna, Austria*, pages 961–971, July 2018.

[141] Dimitris Chatzopoulos, Sujit Gujar, Boi Faltings, and Pan Hui. Privacy Preserving and Cost Optimal Mobile Crowdsensing using Smart Contracts on Blockchain. In *2018 IEEE 15th International Conference on Mobile Ad Hoc and Sensor Systems (MASS), Chengdu, China*, pages 442 – 450, October 2018.

[142] Mengmeng Yang, Tianqing Zhu, Kaitai Liang, Wanlei Zhou, and Robert H. Deng. A blockchain-based location privacy-preserving crowdsensing system. *Future Generation Computer Systems*, 94:408–418, May 2019.

[143] M. Li, J. Weng, A. Yang, W. Lu, Y. Zhang, L. Hou, J. Liu, Y. Xiang, and R. H. Deng. CrowdBC: A Blockchain-Based Decentralized Framework for Crowdsourcing. *IEEE Transactions on Parallel and Distributed Systems*, 30(6):1251–1266, June 2019.

[144] Wei Feng and Zheng Yan. MCS-Chain: Decentralized and trustworthy mobile crowdsourcing based on blockchain. *Future Generation Computer Systems*, 95:649–666, June 2019.

[145] Junwei Zhang, Wenxuan Cui, Jianfeng Ma, and Chao Yang. Blockchain-based secure and fair crowdsourcing scheme. *International Journal of Distributed Sensor Networks*, 15(7):1550147719864890, July 2019.

[146] Yao Yu, Shumei Liu, Lei Guo, Phee Lep Yeoh, Branka Vucetic, and Yonghui Li. CrowdR-FBC: A Distributed Fog-Blockchains for Mobile Crowdsourcing Reputation Management. *IEEE Internet of Things Journal*, 7(9):8722 – 8735, September 2020.

[147] Jiejun Hu, Kun Yang, Kezhi Wang, and Kai Zhang. A Blockchain-Based Reward Mechanism for Mobile Crowdsensing. *IEEE Transactions on Computational Social Systems*, 7(1):178–191, February 2020.

[148] Maha Kadadha, Hadi Otrok, Rabeb Mizouni, Shakti Singh, and Anis Ouali. SenseChain: A blockchain-based crowdsensing framework for multiple requesters and multiple workers. *Future Generation Computer Systems*, 105:650–664, April 2020.

[149] Saide Zhu, Zhipeng Cai, Huafu Hu, Yingshu Li, and Wei Li. zkCrowd: A Hybrid Blockchain-Based Crowdsourcing Platform. *IEEE Transactions on Industrial Informatics*, 16(6):4196–4205, June 2020.

[150] Shihong Zou, Jinwen Xi, Honggang Wang, and Guoai Xu. CrowdBLPS: A Blockchain-Based Location-Privacy-Preserving Mobile Crowdsensing System. *IEEE Transactions on Industrial Informatics*, 16(6):4206–4218, June 2020.

[151] Junqin Huang, Linghe Kong, Hong-Ning Dai, Weiping Ding, Long Cheng, Guihai Chen, Xi Jin, and Peng Zeng. Blockchain-Based Mobile Crowd Sensing in Industrial Systems. *IEEE Transactions on Industrial Informatics*, 16(10):6553–6563, October 2020.

[152] Michael Walfish and Andrew J. Blumberg. Verifying computations without reexecuting them. *Communications of the ACM*, 58(2):74–84, January 2015.

[153] Mira Belenkiy, Melissa Chase, C. Chris Erway, John Jannotti, Alptekin Küpçü, and Anna Lysyanskaya. Incentivizing outsourced computation. In *Proceedings of the 3rd International Workshop on Economics of Networked Systems - NetEcon '08, Seattle, WA, USA*, page 85. ACM Press, 2008.

[154] Ran Canetti, Ben Riva, and Guy N. Rothblum. Practical delegation of computation using multiple servers. In *Proceedings of the 18th ACM Conference on Computer and Communications Security - CCS '11, Chicago, Illinois, USA*, page 445. ACM Press, 2011.

[155] A. Küpçü. Incentivized Outsourced Computation Resistant to Malicious Contractors. *IEEE Transactions on Dependable and Secure Computing*, 14(6):633–649, November 2017.

[156] Golem whitepaper - whitepaper.io. https://whitepaper.io/document/21/golem-whitepaper, 2014. Online; accessed on 14 May 2021.

[157] G Fedak, H He, M Moca, W Bendella, and E Alves. Blockchain-based decentralized cloud computing. *iExec Blockchain Tech, Tech. Rep.*, 2018.

[158] About SONM — SONM. https://docs.sonm.com/.

[159] Rafael Brundo Uriarte and Rocco DeNicola. Blockchain-Based Decentralized Cloud/Fog Solutions: Challenges, Opportunities, and Standards. *IEEE Communications Standards Magazine*, 2(3):22–28, September 2018.

[160] Prashanth Mohan, Venkata N. Padmanabhan, and Ramachandran Ramjee. Nericell: Rich monitoring of road and traffic conditions using mobile smartphones. In *Proceedings of the 6th ACM Conference on Embedded Network Sensor Systems, New York, NY, USA*, SenSys '08, pages 323–336. Association for Computing Machinery, November 2008.

[161] Arvind Thiagarajan, Lenin Ravindranath, Katrina LaCurts, Samuel Madden, Hari Balakrishnan, Sivan Toledo, and Jakob Eriksson. VTrack: Accurate, energy-aware road traffic delay estimation using mobile phones. In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems, New York, NY, USA*, SenSys '09, pages 85–98. Association for Computing Machinery, November 2009.

[162] Amit Datta, Marc Joye, and Nadia Fawaz. Private Data Aggregation over Selected Subsets of Users. In *International Conference on Cryptology and Network Security, Fuzhou, China*, Lecture Notes in Computer Science, pages 375–391. Springer International Publishing, 2019.

[163] Nicolas Maisonneuve, Matthias Stevens, Maria E. Niessen, and Luc Steels. NoiseTube: Measuring and mapping noise pollution with mobile phones. In *Information Technologies in Environmental Engineering, Thessaloniki, Greece*, Environmental Science and Engineering, pages 215–228. Springer, 2009.

[164] Min Mun, Sasank Reddy, Katie Shilton, Nathan Yau, Jeff Burke, Deborah Estrin, Mark Hansen, Eric Howard, Ruth West, and Péter Boda. PEIR, the personal environmental impact report, as a platform for participatory sensing systems research. In *Proceedings of the 7th International Conference on Mobile Systems, Applications, and Services, Kraków Poland*, MobiSys '09, pages 55–68. ACM, June 2009.

[165] Lijun Wei, Jing Wu, and Chengnian Long. A Blockchain-Based Hybrid Incentive Model for Crowdsensing. *Electronics*, 9(2):215, January 2020.

[166] Heinrich von Stackelberg. *Market Structure and Equilibrium.* Springer Science & Business Media, November 2010.

[167] Alex Biryukov, Dmitry Khovratovich, and Ivan Pustogarov. Deanonymisation of Clients in Bitcoin P2P Network. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, Arizona, USA*, CCS '14, pages 15–29. Association for Computing Machinery, November 2014.

[168] Roberto Tonelli, Maria Ilaria Lunesu, Andrea Pinna, Davide Taibi, and Michele Marchesi. Implementing a Microservices System with Blockchain Smart Contracts. In *2019 IEEE International Workshop on Blockchain Oriented Software Engineering (IWBOSE), Hangzhou, China*, pages 22–31. IEEE, February 2019.

[169] Deeraj Nagothu, Ronghua Xu, Seyed Yahya Nikouei, and Yu Chen. A Microservice-enabled Architecture for Smart Surveillance using Blockchain Technology. In *2018 IEEE International Smart Cities Conference (ISC2), Kansas City, MO, USA*, pages 1–4, July 2018.

[170] Ronghua Xu, Seyed Yahya Nikouei, Yu Chen, Erik Blasch, and Alex Aved. BlendMAS: A BLockchain-ENabled Decentralized Microservices Architecture for Smart Public Safety. In *2019 IEEE International Conference on Blockchain, Atlanta, GA, USA*, pages 564 – 571, February 2019.

[171] Ronghua Xu, Gowri Sankar Ramachandran, Yu Chen, and Bhaskar Krishnamachari. BlendSM-DDM: BLockchain-ENabled Secure Microservices for Decentralized Data Marketplaces. In *2019 IEEE International Smart Cities Conference (ISC2), Casablanca, Morocco*, pages 14 – 17, September 2019.

[172] Ronghua Xu, Yu Chen, Erik Blasch, Alexander Aved, Genshe Chen, and Dan Shen. Hybrid Blockchain-Enabled Secure Microservices Fabric for Decentralized Multi-Domain Avionics Systems. In *Sensors and Systems for Space Applications XIII*, pages 150 – 164. SPIE, April 2020.

[173] Nikola Bozic, Guy Pujolle, and Stefano Secci. Securing virtual machine orchestration with blockchains. In *2017 1st Cyber Security in Networking Conference (CSNet),Rio de Janeiro, Brazil*, pages 1–8. IEEE, October 2017.

[174] Igor D. Alvarenga, Gabriel A. F. Rebello, and Otto Carlos M. B. Duarte. Securing configuration management and migration of virtual network functions using blockchain. In *NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium, Taipei, Taiwan*, pages 1–9. IEEE, April 2018.

[175] Gabriel Antonio F. Rebello, Igor D. Alvarenga, Igor J. Sanz, and Otto Carlos M. B. Duarte. BSec-NFVO: A Blockchain-Based Security for Network Function Virtualization Orchestration. In *ICC 2019 - 2019 IEEE International Conference on Communications (ICC), Shanghai, China*, pages 1–6. IEEE, May 2019.

[176] Gabriel Antonio F. Rebello, Gustavo F. Camilo, Leonardo G. C. Silva, Lucas C. B. Guimarães, Lucas Airam C. de Souza, Igor D. Alvarenga, and Otto Carlos M. B. Duarte. Providing a Sliced, Secure, and Isolated Software Infrastructure

of Virtual Functions Through Blockchain Technology. In *2019 IEEE 20th International Conference on High Performance Switching and Routing (HPSR), Xi'an, China*, pages 1–6. IEEE, May 2019.

[177] Eder J. Scheid, Manuel Keller, Muriel F. Franco, and Burkhard Stiller. BUNKER: A Blockchain-based trUsted VNF pacKagE Repository. In *Economics of Grids, Clouds, Systems, and Services, Leeds, United Kingdom*, volume 11819, pages 188–196. Springer International Publishing, September 2019.

[178] Xiaoyuan Fu, F. Richard Yu, Jingyu Wang, Qi Qi, and Jianxin Liao. Performance Optimization for Blockchain-Enabled Distributed Network Function Virtualization Management and Orchestration. *IEEE Transactions on Vehicular Technology*, 69(6):6670–6679, June 2020.

[179] R. A. Mishra, A. Kalla, K. Shukla, A. Nag, and M. Liyanage. B-vnf: Blockchain-enhanced architecture for vnf orchestration in mec-5g networks. In *2020 IEEE 3rd 5G World Forum (5GWF), Bangalore, India*, pages 229–234, September 2020.

[180] France ETSI Ind. Spec. Group (ISG) Netw. Functions Virtualisation (NFV), Sophia-Antipolis Cedex. ETSI GS NFV 003 V1.2.1: Network Functions Virtualisation (NFV); Terminology for main concepts in NFV. http://www.etsi.org/deliver/etsi$_g$s/NFV/001$_0$99/003/01.02.01$_6$0/gs$_N$FV003v010201p.pdf, Dec 2014. Online; accessed on 14 May 2021.

[181] R. Mijumbi, J. Serrat, J. Gorricho, N. Bouten, F. De Turck, and R. Boutaba. Network Function Virtualization: State-of-the-Art and Research Challenges. *IEEE Communications Surveys Tutorials*, 18(1):236–262, Firstquarter 2016.

[182] S. Lal, A. Kalliola, I. Oliver, K. Ahola, and T. Taleb. Securing VNF communication in NFVI. In *2017 IEEE Conference on Standards for Communications and Networking (CSCN), Helsinki, Finland*, pages 187–192. IEEE, September 2017.

[183] Miguel Castro, Barbara Liskov, et al. Practical byzantine fault tolerance. In *OSDI*, volume 99, pages 173–186, 1999.

[184] B. Parno, J. Howell, C. Gentry, and M. Raykova. Pinocchio: Nearly Practical Verifiable Computation. In *2013 IEEE Symposium on Security and Privacy, Berkeley, CA, USA*, pages 238–252. IEEE, May 2013.

[185] Ralph C. Merkle. A Digital Signature Based on a Conventional Encryption Function. In Carl Pomerance, editor, *Advances in Cryptology — CRYPTO '87, Santa Barbara, CA, USA*, Lecture Notes in Computer Science, pages 369–378. Springer, 1988.

[186] Torben Pryds Pedersen. Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing. In Joan Feigenbaum, editor, *Advances in Cryptology — CRYPTO '91, Santa Barbara, CA, USA*, Lecture Notes in Computer Science, pages 129–140. Springer, 1992.

[187] Patrick McCorry, Siamak F. Shahandashti, and Feng Hao. A Smart Contract for Boardroom Voting with Maximum Voter Privacy. In *Financial Cryptography and Data Security, Sliema, Malta*, Lecture Notes in Computer Science, pages 357–375. Springer International Publishing, 2017.

[188] Mohammad Malekzadeh, Richard G. Clegg, Andrea Cavallaro, and Hamed Haddadi. Mobile sensor data anonymization. In *Proceedings of the International Conference on Internet of Things Design and Implementation, Montreal Quebec Canada*, pages 49–58. ACM, April 2019.

[189] Marc Joye and Benoît Libert. A Scalable Scheme for Privacy-Preserving Aggregation of Time-Series Data. In *Financial Cryptography and Data Security, Okinawa, Japan*, volume 7859, pages 111–125. Springer Berlin Heidelberg, 2013.

[190] Philipp Schindler, Aljosha Judmayer, Nicholas Stifter, and Edgar Weippl. ETHDKG: Distributed Key Generation with Ethereum Smart Contracts. https://eprint.iacr.org/2019/985.pdf, 2019. Online; accessed on 14 May 2021.

[191] Qi Li, Yaliang Li, Jing Gao, Lu Su, Bo Zhao, Murat Demirbas, Wei Fan, and Jiawei Han. A confidence-aware approach for truth discovery on long-tail data. *Proceedings of the VLDB Endowment*, 8(4):425–436, December 2014.

[192] S. Yao, M. T. Amin, L. Su, S. Hu, S. Li, S. Wang, Y. Zhao, T. Abdelzaher, L. Kaplan, C. Aggarwal, and A. Yener. Recursive Ground Truth Estimator for Social Data Streams. In *2016 15th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN), Vienna, Austria*, pages 1–12. IEEE, April 2016.

[193] Chuishi Meng, Wenjun Jiang, Yaliang Li, Jing Gao, Lu Su, Hu Ding, and Yun Cheng. Truth Discovery on Crowd Sensing of Correlated Entities. In *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems,Seoul, South Korea*, SenSys '15, pages 169–182. ACM, November 2015.

[194] Yue Liu, Qinghua Lu, Xiwei Xu, Liming Zhu, and Haonan Yao. Applying Design Patterns in Smart Contracts. In *Blockchain – ICBC 2018, Seattle, WA, USA*, Lecture Notes in Computer Science, pages 92–106. Springer International Publishing, 2018.

[195] G. Xu, H. Li, S. Liu, M. Wen, and R. Lu. Efficient and Privacy-Preserving Truth Discovery in Mobile Crowd Sensing Systems. *IEEE Transactions on Vehicular Technology*, 68(4):3854–3865, April 2019.

[196] Gergely Ács and Claude Castelluccia. I have a dream!(differentially private smart metering). In *International Workshop on Information Hiding, Prague, Czech Republic*, pages 118–132. Springer, May 18-20, 2011.

[197] Iraklis Leontiadis, Kaoutar Elkhiyaoui, and Refik Molva. Private and dynamic time-series data aggregation with trust relaxation. In *International Conference on Cryptology and Network Security, Heraklion, Crete, Greece*, pages 305–320. Springer, October 22-24, 2014.

[198] Jianwei Chen and Huadong Ma. Privacy-preserving aggregation for participatory sensing with efficient group management. In *2014 IEEE Global Communications Conference, Austin, TX, USA*, pages 2757–2762. IEEE, 2014.

[199] Iraklis Leontiadis, Kaoutar Elkhiyaoui, Melek Önen, and Refik Molva. Puda–privacy and unforgeability for data aggregation. In *International Conference on Cryptology and Network Security, Marrakesh, Morocco*, pages 3–18. Springer, December 10-12, 2015.

[200] Keita Emura. Privacy-preserving aggregation of time-series data with public verifiability from simple assumptions. In *Information Security and Privacy, Auckland, New Zealand*, pages 193–213. Springer International Publishing, July 3–5,2017.

[201] Elaine Shi, HTH Chan, Eleanor Rieffel, Richard Chow, and Dawn Song. Privacy-preserving aggregation of time-series data. In *Annual Network & Distributed System Security Symposium (NDSS), San Diego, California*, pages 1–17, 2011.

[202] Fabrice Benhamouda, Marc JOYE, and Benoît Libert. A New Framework for Privacy-Preserving Aggregation of Time-Series Data. *ACM Transactions on Information and System Security*, 18(3):21, April 2016.

[203] L. Mashayekhy, M. M. Nejad, D. Grosu, and A. V. Vasilakos. An Online Mechanism for Resource Allocation and Pricing in Clouds. *IEEE Transactions on Computers*, 65(4):1172–1184, April 2016.

[204] Joshua A Kroll, Ian C Davey, and Edward W Felten. The economics of bitcoin mining, or bitcoin in the presence of adversaries. In *Proceedings of WEIS, Washington, D.C., USA*, volume 2013, page 11, June 23-24, 2013.

[205] Hisham S. Galal and Amr M. Youssef. Verifiable Sealed-Bid Auction on the Ethereum Blockchain. In *Financial Cryptography and Data Security, Nieuwpoort, Curaçao*.

[206] A. Hahn, R. Singh, C. Liu, and S. Chen. Smart contract-based campus demonstration of decentralized transactive energy auctions. In *2017 IEEE Power Energy Society Innovative Smart Grid Technologies Conference (ISGT)*, pages 1–5, April 2017.

[207] Y. Chen, S. Chen, and I. Lin. Blockchain based smart contract for bidding system. In *2018 IEEE International Conference on Applied System Invention (ICASI), Chiba, Japan*, pages 208–211. IEEE, April 2018.

[208] Shuangke Wu, Yanjiao Chen, Qian Wang, Minghui Li, Cong Wang, and Xiangyang Luo. CReam: A Smart Contract Enabled Collusion-Resistant e-Auction. *IEEE Transactions on Information Forensics and Security*, 14(7):1687–1701, July 2019.

[209] Chiara Braghin, Stelvio Cimato, Ernesto Damiani, and Michael Baronchelli. Designing Smart-Contract Based Auctions. In *Security with Intelligent Computing and Big-Data Services, New Taipei City, Taiwan*, Advances in Intelligent Systems and Computing, pages 54–64. Springer International Publishing, 2019.

[210] S. Thakur, B. P. Hayes, and J. G. Breslin. Distributed Double Auction for Peer to Peer Energy Trade using Blockchains. In *2018 5th International Symposium on Environment-Friendly Energies and Applications (EFEA),Rome, Italy*, pages 1–8. IEEE, September 2018.

[211] Erik-Oliver Blass and Florian Kerschbaum. Strain: A Secure Auction for Blockchains. In *European Symposium on Research in Computer Security, Barcelona, Spain*, Lecture Notes in Computer Science, pages 87–110. Springer International Publishing, September 2018.

[212] Hisham S. Galal and Amr M. Youssef. Trustee: Full Privacy Preserving Vickrey Auction on Top of Ethereum. In *Financial Cryptography and Data Security, St. Kitts, Saint Kitts and Nevis*, Lecture Notes in Computer Science, pages 190–207. Springer International Publishing, February 2019.

[213] V. Hassija, G. Bansal, V. Chamola, V. Saxena, and B. Sikdar. BlockCom: A Blockchain Based Commerce Model for Smart Communities using Auction Mechanism. In *2019 IEEE International Conference on Communications Workshops (ICC Workshops), Shanghai, China*, pages 1–6. IEEE, May 2019.

[214] J. R. Douceur, A. Adya, W. J. Bolosky, P. Simon, and M. Theimer. Reclaiming space from duplicate files in a serverless distributed file system. In *Proceedings 22nd International Conference on Distributed Computing Systems, Vienna, Austria*, pages 617–624. IEEE, July 2002.

[215] J. Li, Y. K. Li, X. Chen, P. P. C. Lee, and W. Lou. A Hybrid Cloud Approach for Secure Authorized Deduplication. *IEEE Transactions on Parallel and Distributed Systems*, 26(5):1206–1216, May 2015.

[216] Mihir Bellare, Sriram Keelveedhi, and Thomas Ristenpart. Message-Locked Encryption and Secure Deduplication. In *Advances in Cryptology – EUROCRYPT 2013, Athens, Greece*, Lecture Notes in Computer Science, pages 296–312. Springer, 2013.

[217] Mihir Bellare, Sriram Keelveedhi, and Thomas Ristenpart. DupLESS: Server-Aided Encryption for Deduplicated Storage. In *22nd USENIX Security Symposium, Washington, D.C., USA*, pages 179–194. ACM, August 2013.

[218] J. Li, X. Chen, X. Huang, S. Tang, Y. Xiang, M. M. Hassan, and A. Alelaiwi. Secure Distributed Deduplication Systems with Improved Reliability. *IEEE Transactions on Computers*, 64(12):3569–3579, December 2015.

[219] Z. Yan, W. Ding, X. Yu, H. Zhu, and R. H. Deng. Deduplication on Encrypted Big Data in Cloud. *IEEE Transactions on Big Data*, 2(2):138–150, June 2016.

[220] M. Wen, K. Lu, J. Lei, F. Li, and J. Li. BDO-SD: An efficient scheme for big data outsourcing with secure deduplication. In *2015 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), Hong Kong, China*, pages 214–219. IEEE, April 2015.

[221] Meixia Miao, Tao Jiang, and Ilsun You. Payment-based incentive mechanism for secure cloud deduplication. *International Journal of Information Management*, 35(3):379–386, June 2015.

[222] Jian Liu, N. Asokan, and Benny Pinkas. Secure Deduplication of Encrypted Data without Additional Independent Servers. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, Colorado, USA*, CCS '15, pages 874–885. Association for Computing Machinery, October 2015.

[223] Taek-Young Youn and Ku-Young Chang. Necessity of Incentive System for the First Uploader in Client-Side Deduplication. In *Advances in Computer Science and Ubiquitous Computing, Cebu, Philippines*, Lecture Notes in Electrical Engineering, pages 397–402. Springer, December 2015.

[224] Vladimir Rabotka and Mohammad Mannan. An evaluation of recent secure deduplication proposals. *Journal of Information Security and Applications*, 27-28:3–18, April 2016.

[225] X. Liang, Z. Yan, X. Chen, L. T. Yang, W. Lou, and Y. T. Hou. Game Theoretical Analysis on Encrypted Cloud Data Deduplication. *IEEE Transactions on Industrial Informatics*, 15(10):5778–5789, October 2019.

[226] L. Gao, Z. Yan, and L. T. Yang. Game Theoretical Analysis on Acceptance of a Cloud Data Access Control System Based on Reputation. *IEEE Transactions on Cloud Computing*, 8(4):1003–1017, October 2020.

[227] R. Dua, A. R. Raja, and D. Kakadia. Virtualization vs Containerization to Support PaaS. In *2014 IEEE International Conference on Cloud Engineering,Boston, MA, USA*, pages 610–614. IEEE, March 2014.

[228] Microservices. https://martinfowler.com/articles/microservices.html. Online; accessed on 14 May 2021.

[229] Nicola Dragoni, Saverio Giallorenzo, Alberto Lluch Lafuente, Manuel Mazzara, Fabrizio Montesi, Ruslan Mustafin, and Larisa Safina. Microservices: Yesterday, today, and tomorrow. In *Proceeding of Present and Ulterior Software Engineering*, pages 195–216. Springer, April 2017.

[230] What Led Amazon to its Own Microservices Architecture. https://thenewstack.io/led-amazon-microservices-architecture/#:~:text=Amazon's%20approach%20is%20not%20to,can%20be%20scripted%20and%20automated. Online; accessed on 14 May 2021.

[231] Adopting Microservices at Netflix: Lessons for Architectural Design. https://www.nginx.com/blog/microservices-at-netflix-architectural-best-practices/. Online; accessed on 14 May 2021.

[232] Steven Ihde and Karan Parikh. From a monolith to microservices+ rest: the evolution of linkedin's service architecture. https://www.slideshare.net/InfoQ/from-a-monolith-to-microservices-rest-the-evolution-of-linkedins-service-architecture. Online; accessed on 14 May 2021.

[233] Service-Oriented Architecture: Scaling the UBER Engineering Codebase As We Grow. http://zookeeper.apache.org/. Online; accessed on 14 May 2021.

[234] M. Villamizar, O. Garcés, L. Ochoa, H. Castro, L. Salamanca, M. Verano, R. Casallas, S. Gil, C. Valencia, A. Zambrano, and M. Lang. Infrastructure Cost Comparison of Running Web Applications in the Cloud Using AWS Lambda and Monolithic and Microservice Architectures. In *2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid),Cartagena, Colombia*, pages 179–182. IEEE, May 2016.

[235] C. Esposito, A. Castiglione, and K. R. Choo. Challenges in Delivering Software in the Cloud as Microservices. *IEEE Cloud Computing*, 3(5):10–14, September 2016.

[236] Apace zookeeper. https://zookeeper.apache.org/. Online; accessed on 14 May 2021.

[237] Eureka. https://spring.io/guides/gs/service-registration-and-discovery/. Online; accessed on 14 May 2021.

[238] Srikanta Patanjali, Benjamin Truninger, Piyush Harsh, and Thomas Michael Bohnert. CYCLOPS: A micro service based approach for dynamic rating, charging &amp; billing for cloud. In *2015 13th International Conference on Telecommunications (ConTEL), Graz, Austria*, pages 1–8. IEEE, July 2015.