symmetry

MDPI

Article

Oblivious Access for Decentralized Database Systems: A New Asymmetric Framework from Smart Contracts †

Zhong-Yi Guo [1], Yu-Chi Chen [1,*] and Hsiu-Ping Lin [2]

[1] Department of Computer Science and Engineering, Yuan Ze University, Taoyuan 32003, Taiwan; s1076026@mail.yzu.edu.tw
[2] imToken Labs, Singapore 038989, Singapore; nic@token.im
* Correspondence: wycchen@saturn.yzu.edu.tw; Tel.: +886-3-4638800 (ext. 2358)

† This paper is an extended version of our paper published in AsiaCCS 2020 (Zhong-Yi Guo, Yu-Chi Chen, and Hsiu-Ping Lin, "POSTER: Oblivious Access System on Decentralized Database over Parallel Smart Contract Model", ACM ASIA Conference on Computer and Communications Security, 2020).

Abstract: With the rapid development of cloud servers, storing data on cloud servers has become a popular option. However, cloud servers are centralized. Storing data on centralized cloud servers may involve some risks. For example, the data access pattern may be revealed when accessing data on cloud servers. Therefore, protecting a user's patterns has become a crucial concern. Oblivious RAM (ORAM) is a candidate solution to hide the data access pattern. However, it inherently induces some overhead of accessing data, and many blockchain-based applications also do not consider the access pattern leakage issues. In this paper, we address these issues above by proposing a decentralized database system with oblivious access in a (parallel) smart contract model. The interactions of oblivious access are asymmetric where the smart contract side is expected to put much effort into computation. The proposed system slightly reduces the overhead of ORAM and overcomes the issues stemming from the centralization of servers. The main techniques are to use the garbled circuits to reduce the cost of communication and to combine with the parallel smart contract model to (conceptually) improve the performance of smart contract execution on the blockchain.

Keywords: smart contract; blockchain; oblivious RAM; garbled circuits

Citation: Guo, Z.-Y.; Chen, Y.-C.; Lin, H.-P. Oblivious Access for Decentralized Database Systems: A New Asymmetric Framework from Smart Contracts . *Symmetry* **2022**, *14*, 680. https://doi.org/10.3390/sym14040680

Academic Editor: José Carlos R. Alcantud

Received: 19 February 2022
Accepted: 15 March 2022
Published: 25 March 2022

1. Introduction

With the increasing popularity of the Internet of Things (IoT), the number of novel IoT applications is increasing [1]. Contemporary IoT systems usually consist of low-power and lightweight devices with sensors. The devices are responsible for collecting data from the surrounding environment and may exchange data with other devices or servers. In traditional IoT systems, the collected data are stored on centralized cloud servers. Hence, users have to trust that the centralized cloud servers will protect their sensitive and private data, which are normally unencrypted. In fact, centralized IoT systems might face some challenges. Stored data are unencrypted, so they face the risk of being modified or deleted. Furthermore, if centralized cloud servers stop working, the entire system might get paralyzed.

Blockchain, first proposed by Nakamoto and Bitcoin [2], is a distributed timestamp ledger of blocks that is utilized to share and store data. It may be a promising technique to mitigate the issues faced by centralized IoT systems because of its superior properties (decentralization, tamper-resistance, anonymity, security, and privacy). Smart contracts [3], which refer to executable code on the blockchain, can help blockchain be thought of as a platform for developing distributed applications. Smart contracts can be regarded as decentralized cloud servers. Zhou et al. [4] proposed the BeeKeeper, which is a decentralized database system. Although this system addresses the concerns posed by centralized systems, it fails to protect the data access pattern. Imagine that the data are encrypted

by users before uploading to the smart contract. If users access identical data multiple times, the smart contract will learn that the data are important to users. Thus, although the data are encrypted, attackers can still infer sensitive information based on the data access pattern. Thus, in this case, the data access pattern is the information leaked by the program's series of accesses to the data storage.

To address this issue, oblivious random access machine (ORAM) protocol, first introduced by Goldreich and Ostrovsky [5], is a candidate method for hiding the data access pattern. ORAM allows clients to store data on an untrusted server and to access them as if they were stored locally. The protocol guarantees that the server learns nothing about the client's access pattern or data. In recent years, several ORAM frameworks (e.g., [6,7]) have been introduced to improve efficiency. However, when these frameworks are used, users must maintain a position map on the local side. Thus, ORAM actually induces some communication and memory storage overhead related to data access. If we directly apply such ORAM to smart contracts (without considering any technical issues with the combination), a disadvantage is the communication overhead that induces the transaction cost on the smart contract. In Table 1, we show the comparisons with ORAM and conceptual ORAM on smart contract. Here we do not construct the conceptual ORAM on smart contract, so the results in Table 1 are conjecture.

Table 1. Security and efficiency comparisons (n is the database size).

	ORAM	**ORAM on SC**	**Proposal**
Round	$O(\log n)$	$O(\log n)$	$O(1)$
Payload	$O(\log n)$	$O(\log n)$	$O(1)$
Client storage	$O(\log n)$	$O(\log n)$	$O(1)$
Security	Centralized	Decentralized	Decentralized

1.1. Contributions

This paper aims to address the issues faced by centralized cloud servers and to protect the data access pattern. Our main results are conceptually simple solutions that use smart contracts to implement a decentralized database and oblivious access within this system. The main contributions of this paper are two-fold.

- We propose a decentralized database system with oblivious access in a (parallel) smart contract model, aiming to provide the service of decentralized cloud storage that hides the data access pattern and overcomes the overhead associated with ORAM.
- In our construction, we combine smart contracts with ORAM and garbled circuits [8] to ensure that this system enables more secure and confidential personal data protection. Hence, the user of this system can access and upload their personal data as frequently as they wish without any privacy concerns.
- The abstraction design of our construction could be easily applied to the parallel smart contract (PSC) model that is proposed by Yu et al. [9]. It uses multi-thread technology [10] to execute smart contracts in parallel. Using this new model to process transactions could reduce the average time cost and improve performance.

In general, the proposed system inherits advantages from blockchain and ORAM. We roughly summarize a few properties, *Decentralization, Tamper-Resistance, and Obliviousness*, as follows. There is no centralized party controlling the system, but the data are maintained by all participating nodes instead. Therefore, the user has higher ownership of their own data. In addition, all transactions executed on the smart contract are completely recorded and can be viewed and checked by any entity. The user does not need to worry about the risk of data loss or or unintentional data modification. Finally, the user accesses the data on the smart contract through the index. However, the index will not be learned by the smart contract, nor will the smart contract learn which data the user accesses.

Let us briefly highlight the techniques in our system. The smart contract not only enables oblivious access and data storage functions but also maintains low cost. Therefore, the user does not incur additional server maintenance overhead. In our oblivious structure, we adopt the tree-based ORAMs [6,7,11,12] that store data in the binary tree and rely on the position map to access the data. Notably, the position map is not held by users; instead, it is stored on the smart contract. Furthermore, we garble some fields on the position map and use garbled circuits to return the data. Through these techniques, the costs of communication and memory storage are reduced. In the proposed system, it is not trivial to combine the two existing techniques, blockchain and ORAM. The innovation of this system is adopting garbled circuits over smart contracts to simplify the immediate level between the decentrailzed storage and ORAM structure without losing any security. Its practical applications can provide a decentralized database (e.g., multiple NASs) where the outside attacker does not know which components are accessed. (Note that encryption can provide confidentiality, but cannot hide access pattern.) For example, if a location of storage is frequently touched, it may be very important. Let us consider the binary search tree on the storage. We can have side information that the frequently touched location is the root.

1.2. Organization

The remainder of this paper is organized as follows. Section 2 lists the related work. Section 3 briefly reviews background techniques of the decentralized database system with oblivious access in the (parallel) smart contract model. Section 4 presents the system framework and security definition of this system. Section 5 introduces the specific structure and security proof of the proposed system. Specific experimental implementations and discussion are presented in Section 6. Finally, conclusions for this paper are given in Section 7.

2. Related Work

2.1. Oblivious Random Access Machine (ORAM)

The ORAM framework, which was first introduced by Goldreich and Ostrovsky [5,13,14], aims to hide the user's pattern of access to server data storage. In the past few years, numerous ORAM schemes have been proposed. Based on their data lookup technique, we can roughly divide them into two categories. One is hierarchical [15–19], motivated by the work of Goldreich and Ostrovsky. The other is tree-based [6,7,11,12], motivated by the work of Shi et al. [20]. The technical details of the two kinds of ORAM are as follows, and the structure is shown in Figure 1.

- Hierarchical ORAMs: This kind of ORAM uses hash functions for data lookup and it requires stashes and buckets to deal with hash collisions. To the best of our knowledge, the balanced ORAM [17] achieves high communication efficiency among hierarchical ORAMs.
- Tree-based ORAMs: This kind of ORAM uses an index for data lookup and it requires the user-side storage to maintain the index. However, if the number of data blocks is quite large, this may be infeasible. If the user cannot store the index on user-side storage, it can outsource the index to the server in a manner similar to storing real data blocks. This may increase the cost of communication. The path ORAM proposed by Stefanov et al. [6] is a typical scheme in this category.

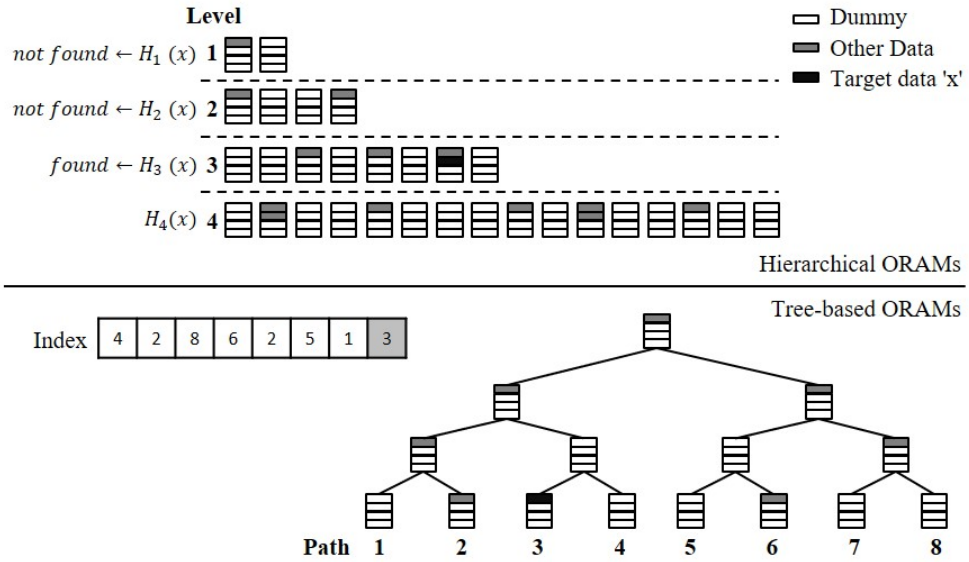

Figure 1. Hierarchical ORAMs and tree-based ORAMs.

2.2. Blockchain and Smart Contract

The concept of blockchain was first proposed by Satoshi Nakamoto in the Bitcoin White Paper [2] in 2008. Blockchain is a peer-to-peer network that consists of all participating nodes. Bitcoin is one of the applications of blockchain. In the past, many scientists have found that blockchain is applicable not only in a decentralized payment system, but also in other fields such as healthcare data management [21], digital rights management [21], supply chain traceability [22], and identity management [23]. There is no centralized party in the blockchain network, and all participating nodes possess the same transcript of a database. A key feature of blockchain is that transactions and communications between untrusted participating nodes can be securely conducted without a trusted centralized party.

The consensus mechanism is a critical blockchain component, and it maintains the entire functionality in blockchain and the consistency of the node. Ever since Bitcoin emerged in 2008, its original consensus mechanism, i.e., Proof-of-Work (PoW) [24,25], has been emulated and iterated due to its limited programming capability. Practical Byzantine Fault Tolerance (PBFT) [26], Proof-of-Stake (PoS) [27], and Delegated-Proof-of-Stake (DPoS) [28] are widely-used consensus mechanisms on various blockchain platforms. Thus, besides Bitcoin, there are many blockchain platforms that are also based on the property of blockchain, such as Ethereum [29,30], EOS.IO [31], and Hyperledger [32].

Etherenum, which was proposed by Vitalik Buterin in 2014, is the representative of blockchain 2.0. It supports the development of decentralized applications in addition to its support of the Ether cryptocurrency. Thus, Ethereum enables the execution of smart contracts. The concept of smart contracts was first proposed by Nick Szabo in 1994 [3]. Simply put, a smart contract is a special protocol used in the development of a contract in the blockchain. It is mainly used to provide verification and execution of the conditions set in the smart contract. Ethereum provides an Ethereum virtual machine (EVM) to process peer-to-peer contracts through its dedicated cryptocurrency, ETH and gas, which depend on the complexity of the internal structure. Ethereum allows developers to run decentralized applications on global public nodes. The process of transactions processing on Etherenum, the so-called smart contract model, is shown in Figure 2.

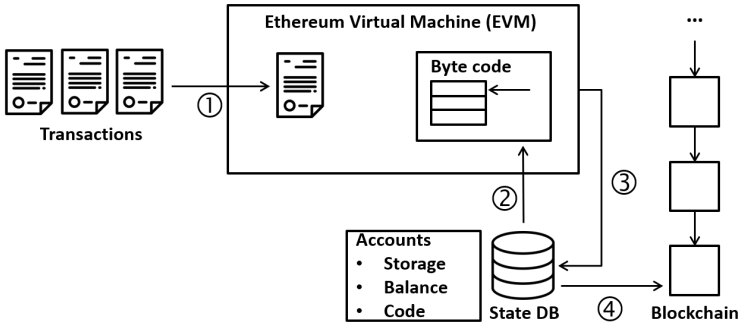

Figure 2. Smart contract model.

2.3. Applications of Blockchain

There are many blockchain-based applications on AI, 5G, IoT, etc. The survey of Salah et al. [33] summarizes emerging blockchain applications for AI. Lu et al. [34] proposed a synchronous federated learning scheme of deep reinforcement learning to address privacy concerns of data providers in IoV. In addition, to achieve the reliability of data sharing, it applies a two-stage verification by integrating learned models into blockchain. Ouaddah et al. [35] proposed a new blockchain-based framework for supporting access control in IoT. They introduced a system, so-called FairAccess, that provides decentralized pseudonymous and privacy preserving authorization management. FairAccess allows users to control their data for accessing. Bera et al. [36] pointed out a few challenges and issues of applying blockchain in 5G-based IoT-enabled Internet of Drones (IoD) environment, and then proposed a blockchain-based secure framework for data management. The work of Zhang et al. [37] overcomes privacy issues in online social networks by using blockchain. The work of [38] mainly studies the security and privacy issues in the transportation system and vehicle IoT environment in the SDN-enabled 5G-VANET. Due to the poor scalability, low capacity, and intermittent connectivity of IEEE 802.11p networks, the main technique is to use 5G network communication to support the application of vehicle IoT. The 5G technology is considered key to providing ultra-reliable and low-latency communication in VANET. It presents a new framework to support vehicle IoT services, based on the decentralization and immutability of the blockchain. The work of [39] presents a modified IPFS solution that allows smart contracts to implement access control for data sharing. The key idea is to maintain the access control list on smart contracts. All files being uploaded, downloaded, or transferred are of interactions with the smart contract. Among these applications, distributed data management has attracted widespread attention from computer science and academic fields. Steichen et al. [39] used blockchain to implement the authority management in traceability systems, and Dai et al. [40]used it to do learning resource authentication in MOOCs. The above blockchain applications involve data storage, but do not consider the security issue, leakage of data access patterns. However, the main results of this paper can be referred to as a solution or remedy (with obliviousness) for improving security of those applications.

3. Preliminaries

3.1. Garbled Circuits

Garbled circuits, first introduced by Yao [8], is a cryptographic notion that can extend to two-party secure computation. It enables two parties to evaluate a function over one's private input without the presence of a trusted third party. In the garbled circuit protocol, the function must be described as a Boolean circuit. Garbled circuits consist of two phases as follows:

- $\mathsf{CreateGC}(1^\lambda, C) \rightarrow (\widetilde{C}, \widetilde{w})$: This algorithm takes as input a security parameter λ and a circuit C, then outputs a garbled circuit $\widetilde{C}$ and $\widetilde{w}$, which is a set of input labels for each input wire of C.

- $\mathsf{EvalGC}(\widetilde{C}, \widetilde{w}_i) \rightarrow y$: This algorithm takes as input a garbled circuit $\widetilde{C}$ and a garbled input $\widetilde{w}_i$, then outputs the evaluated value y.

The formal security definition of garbled circuits is given as follows:

Definition 1. *We require that for all probabilistic polynomial-time adversary $\mathcal{A}$, there is a probabilistic polynomial-time simulator $\mathcal{S}$ such that the following distributions are computationally indistinguishable:*

- $\mathsf{Real}_{\mathcal{A}}(\lambda)$: $\mathcal{A}$ chooses a circuit C. The experiment runs $\mathsf{CreateGC}(1^{\lambda}, C) \rightarrow (\widetilde{C}, \widetilde{w})$ and sends $\tilde{C}$ to $\mathcal{A}$. $\mathcal{A}$ outputs an input i and the experiment outputs $(\widetilde{C}, \widetilde{w}_i)$.
- $\mathsf{Ideal}_{\mathcal{A},\mathcal{S}}(\lambda)$: $\mathcal{A}$ chooses a circuit C. The experiment runs $\mathcal{S}(1^{\lambda}) \rightarrow (\widetilde{C}, \sigma)$ and sends $\tilde{C}$ to $\mathcal{A}$. $\mathcal{A}$ outputs an input i and the experiment runs $\mathcal{S}(1^{\lambda}, \sigma) \rightarrow \widetilde{w}_i$ and outputs $(\widetilde{C}, \widetilde{w}_i)$.

3.2. ORAM

As noted above, ORAM aims to hide the user's patterns of access to the server data storage. We inherit the original security of ORAMs. Intuitively, an ORAM scheme is said to be secure if the server learns nothing about the user's data access pattern. More precisely, we give the formal security definition of ORAM as follows:

Definition 2. *Let $\vec{x} = \langle (op_1, id_1, B_1), \cdots, (op_n, id_n, B_n) \rangle$ denote a private sequence of the user's intended data requests, where each op is either a read or a write operation, id is a data's ID, and B is a data block.*

Let $\mathcal{A}(\vec{x}) = \langle (op'_1, loc_1, B'_1), \cdots, (op'_n, loc_n, B'_n) \rangle$ denote the sequence of the user's access to the remote storage (observed by the adversary), where loc is a location to which data belong, in order to accomplish the user's intended data requests. An ORAM scheme is said to be secure if for any two equal-length private sequences $\vec{x}$ and $\vec{y}$ of the intended data requests, their corresponding observable access sequences $\mathcal{A}(\vec{x})$ and $\mathcal{A}(\vec{y})$ are computationally indistinguishable.

4. System Framework

4.1. Model

In order to mitigate the concerns engendered by centralized cloud servers and protect data access patterns, we propose a decentralized database system with oblivious access in a (parallel) smart contract model. The framework of this system is shown in Figure 3. There are two crucial entities: a user and a smart contract. The smart contract integrates many functions.

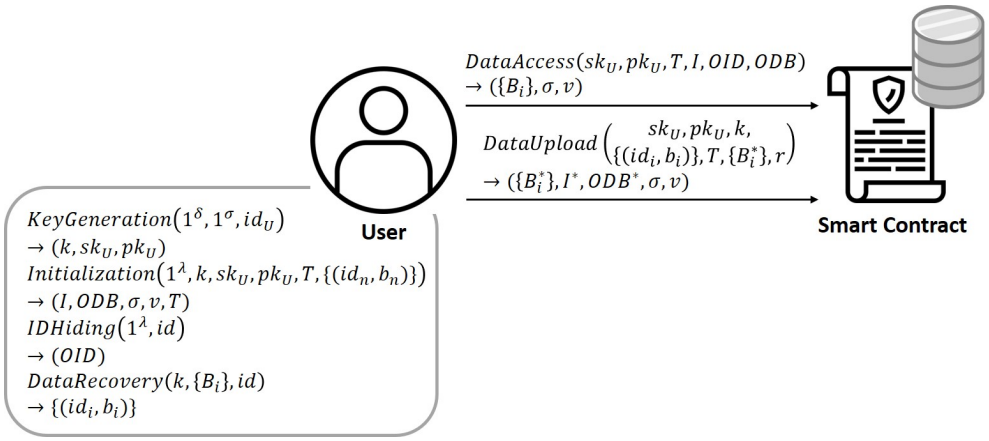

Figure 3. Model framework of the decentralized database system with oblivious access.

The user who is a leader of this system can upload his personal data to this system and access them through the function that is provided by the smart contract. In order to avoid the leakage of the access pattern when the user accesses his personal data, the user has to

perform some preprocessing on his personal data before uploading them to this system. The smart contract aims to provide a decentralized cloud storage to store databases and secure functions to access the databases. Hence, it has to store the user's personal data in an oblivious structure, and then it executes some transactions for secure access in this system. The property of blockchain ensures the transparency and anonymity of all operations on the smart contract while also satisfying the main needs of the user. The user may securely and efficiently complete data access through the smart contract. Note that according to the model setting, there are only two entities (user and SC), and thus the following theoretical model does not consider the key management issue. However, for practical applications, we can apply smart contracts to handle the key management (refer to [41] with domain name service). If the key management is involved in the system, we must rely on the Ethereum name service (ENS) instead. According to the online introduction, ENS is built on smart contracts. It consists of two principle components, the registry and resolvers, to provide human-readable names for accessing smart contracts. ENS also ensures the target smart contracts are being correctly accessed.

In a nutshell, the smart contract in this system has the following characteristics:

1. It allow users to upload their personal data to the smart contract.
2. It allow users to access their data without revealing data access patterns.

More specifically, the system consists of the following phases: (KeyGeneration, Initialization, IDHiding, DataAccess, DataRecovery, and DataUpload) and the syntax of them is formally described as follows:

- $\mathsf{KeyGeneration}(1^{\delta}, 1^{\sigma}, id_U)$: This algorithm takes as input two security parameters (δ, σ) and the user's ID id_U, then outputs a secret key k, the user's private key sk_U, and the user's public key pk_U.
- $\mathsf{Initialization}(1^{\lambda}, 1^{\gamma}, \mathsf{k}, sk_U, pk_U, T, \{(id_n, b_n)\})$: This algorithm takes as input two security parameters (λ, γ), the secret key k, the user's private key sk_U, the user's public key pk_U, a transaction T, and a set of the user's personal data and corresponding IDs $\{(id_n, b_n)\}$, then outputs an index I, an oblivious database ODB that consists of a set of encrypted data $\{B_n\}$, a signature of the transaction σ, and a verification of the transaction v.
- $\mathsf{IDHiding}(1^{\lambda}, tid)$: This algorithm takes as input the security parameter 1^{λ} and the target data's ID tid, then outputs a hidden ID OID.
- $\mathsf{DataAccess}(sk_U, pk_U, T, \mathsf{OID}, I, \mathsf{ODB})$: This algorithms takes as input the user's private key sk_U and the user's public key pk_U, the target data's ID tid, the oblivious index I, a transaction T, and the oblivious database ODB, then outputs a set of the encrypted data $\{B_i\}$, a signature of the transaction σ, and a verification of the transaction v.
- $\mathsf{DataRecovery}(\mathsf{k}, \{B_i\}, tid)$: This algorithm takes as input the secret key k and a set of the encrypted data $\{B_i\}$, then outputs a set of plaintext data and corresponding IDs $\{(id_i, b_i)\}$.
- $\mathsf{DataUpload}(sk_U, pk_U, \mathsf{k}, \{(id_i, b_i)\}, T, p^*)$: This algorithm takes as input the user's private key sk_U, the user's public key pk_U, the secret key k, a set of plaintext data and corresponding IDs $\{(id_i, b_i)\}$, a transaction T, and a random number p^*, then outputs the re-encrypted data $\{B_i^*\}$, an updated oblivious index I^*, an updated oblivious database ODB^*, a signature of the transaction σ, and a verification of the transaction v.

4.2. Security Definition

We now define the security of a decentralized database system with oblivious access (DDOA) in the presence of *semi-honest* adversaries. There are two entities in this system: user and smart contract. We assume that the smart contract is an honest party and will not be corrupted by any adversary $\mathcal{A}$. In addition, $\mathcal{A}$ follows the rules of not obtaining the source code of the smart contract.

To define the security of this system, we design the following game between a challenger $\mathcal{C}$ and a probabilistic polynomial-time adversary $\mathcal{A}$, as shown in Figure 4. First, $\mathcal{A}$ chooses and sends two IDs of data, id_0 and id_1, to $\mathcal{C}$ for a challenge, and then $\mathcal{C}$ selects

$b \in \{0,1\}$ and sends the corresponding sequence of data blocks $\{B\}$ back to $\mathcal{A}$. Finally, $\mathcal{A}$ outputs $b' \in \{0,1\}$. In the case of $b' = b$, $\mathcal{A}$ wins this game and outputs 1; otherwise, it outputs 0.

The security of the decentralized database system with oblivious access is formally defined as follows:

Definition 3. *The decentralized database system with oblivious access is secure if for any probabilistic polynomial-time adversary $\mathcal{A}$ such that the following distributions are computationally indistinguishable:*

$$\mathsf{View}_{\mathcal{A}}(\mathsf{DDOA}(id_0)) \approx \mathsf{View}_{\mathcal{A}}(\mathsf{DDOA}(id_1))$$

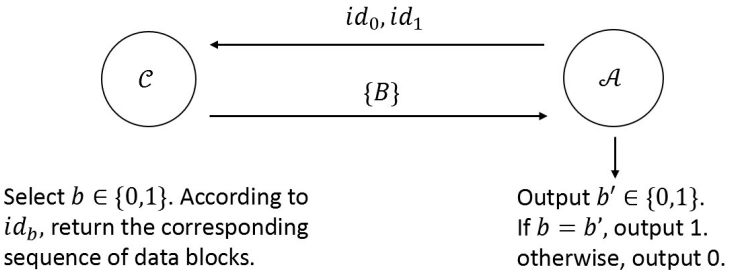

Figure 4. Game-based security of the decentralized database system with oblivious access.

5. The Proposed System

5.1. Building Blocks

5.1.1. Data Block Format

Similar to existing ORAMs, we handle data with blocks as units. A data block can be accessed with read or write operations offered by the user. A plaintext data block consists of the ID of the data block and the content of the data block. Before being uploaded to the service side, the plaintext data block is encrypted with the secret key k by a symmetric encryption. Thus, the encrypted data block is denoted as $B_n = (id_n, b_n)$, where id_n and b_n are the encrypted ID and the content of the data block, respectively.

5.1.2. Service-Side Storage

We follow the tree-based ORAMs structure and take it as our oblivious structure. The service-side storage is organized into a binary tree of height $H = \lceil \log_2 N \rceil + 1$, where N is the number of leaf nodes. Each node in the binary tree is a data bucket that stores up to $O(\log N)$ data blocks, and each bucket may have a dummy block. However, each real data block exists in only one bucket.

In order to access the data from the service side, the user must create a table that is related to the data position. We call this table position map *pos*. It consists of two fields: *ID of the data block* and *path of the data block*. The variables stored in the two fields are plaintext, and the user accesses the data through the ID of the data block. Assume the user accesses the data whose ID is I_5, and the system will find the corresponding path 5. The path means that the target data are stored on the path from the leaf node bucket l to the root bucket. After that, the user will get a sequence of data blocks along 5. The illustration of data access is shown in Figure 5.

For the purpose of information security, the user needs to perform *garbling* on the position map. The user generates garbled labels based on IDs of data blocks using a security parameter λ. The ID field will be replaced by garbled labels, i.e., I_1, ..., I_n are replaced by W_1, ..., W_n. Consequently, the service side cannot identify the ID of the data block the user accessed. After creating and garbling the position map, the user has to upload it to the service side as an initialization.

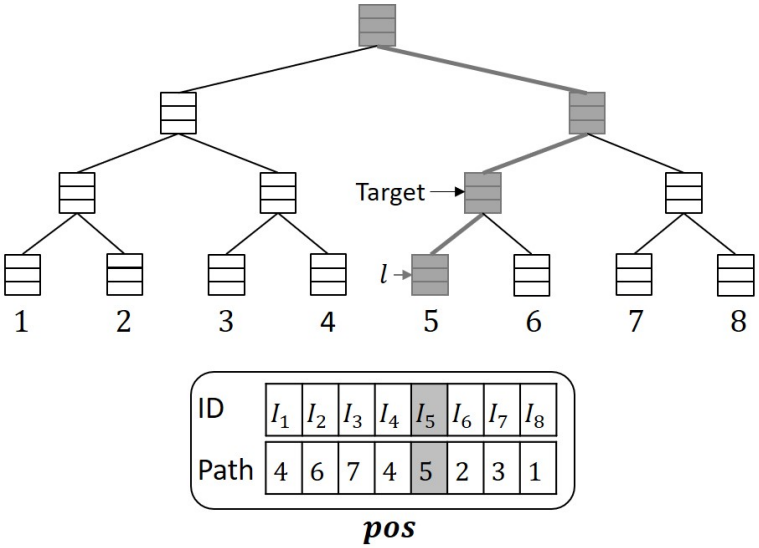

Figure 5. Accessing data from the service-side storage.

5.1.3. User-Side Storage

The user-side storage is organized into two parts: *temporary storage* and *permanent storage*. Temporary storage is used to buffer and process the data blocks that are downloaded from the service side. Permanent storage stores the user's private information, including (1) a secret key k for data encryption and decryption, (2) a pair of the user's private key sk_U and public key pk_U for signing and verifying, respectively, (3) a security parameter λ for generating garbling labels, and (4) a set of the data's ID $\{B_n\}$ for accessing data.

5.1.4. Algorithms on Service Side

Here, the smart contract is the service side. The user has the following list of transactions ($T_{initialize}$, T_{eval}, T_{read}, T_{write}, $T_{replace}$ and T_{update}) that are used to communicate and interact with the SC. The specific functions are formally described as follows:

- $T_{initialize}(\{B_n\}, \widetilde{pos}, \widetilde{C}_{eval}, \widetilde{C}_{replace})$: This algorithms takes as input a set of ciphertext data blocks $\{B_n\}$, the garbled position map $\widetilde{pos}$, a garbled circuit $\widetilde{C}_{eval}$ for evaluation, and a garbled circuit $\widetilde{C}_{replace}$ for replacing labels, then outputs a transaction T_I and an oblivious database ODB.
- $T_{eval}(\widetilde{w}_{id}, \widetilde{pos}, \widetilde{C}_{eval})$: This algorithm takes as input a garbled label of the target data's ID $\widetilde{w}_{id}$, the garbled position map $\widetilde{pos}$, and a garbled circuit $\widetilde{C}_{eval}$ for evaluation, then outputs a transaction T_E and loc_{id}, which is the location of the target data's ID on the position map.
- $T_{read}(loc_{id})$: This algorithm takes as input the location of the target data's ID on the position map loc_{id}, then outputs a transaction T_R and a set of the ciphertext data blocks $\{B_i\}$ based on the path on the position map.
- $T_{write}(\{B_i^*\}, p)$: This algorithm takes as input re-encrypted data blocks $\{B_i^*\}$ and a random path p, then outputs a transaction T_W and an updated oblivious database ODB*.
- $T_{replace}(\widetilde{pos}, \widetilde{C}_{replace})$: This algorithm takes as input the garbled position map $\widetilde{pos}$ and a garbled circuit $\widetilde{C}_{replace}$ for replacing the label, then outputs a transaction T_{RP} and $\widetilde{pos}^*$ as the garbled position map to the next round.
- $T_{update}(\widetilde{C}^*_{eval}, \widetilde{C}^*_{replace})$: This algorithm takes as input a garbled circuit $\widetilde{C}^*_{eval}$ for updating the old one and a garbled circuit $\widetilde{C}^*_{replace}$ for updating the old one, then outputs a transaction T_U.

5.1.5. Algorithms on User Side

The user has two kinds of circuits. One is the evaluation circuit C_{eval}, which is shown in Algorithm 1 for evaluating the location of the target data's ID. The other is the replacing

circuit $C_{replace}$, which is shown in Algorithm 2 for replacing the label on the position map. The specific functions are formally described as follows:

- $\mathsf{KeyGen}(1^{\delta})$: This algorithm takes as input a security parameter 1^{δ}, then outputs a secret key k.
- $\mathsf{DataRec}(\mathsf{k}, \{B_n\}, tid)$: This algorithm takes as input a secret key k, a set of ciphertext data blocks $\{B_n\}$, and the target data's ID tid, then outputs a set of the user's personal data and the corresponding IDs $\{(id_n, b_n)\}$.
- $\mathsf{DataEnc}(\mathsf{k}, \{(id_n, b_n)\})$: This algorithm takes as input a secret key k, a set of plaintext data blocks and corresponding IDs $\{(id_n, b_n)\}$, then outputs a set of ciphertext data blocks $\{B_n\}$.
- $\mathsf{CreateGC}(1^{\lambda}, C)$: This algorithm takes as input a security parameter λ and a circuit C, then outputs a garbled circuit $\widetilde{C}$ and $\widetilde{w}$, which is a set of input labels for each input wire of C.
- $\mathsf{IDGarble}(tid, \widetilde{w})$: This algorithm takes as input the target data's ID tid and a set of labels $\widetilde{w}$, then outputs a garbled ID $\widetilde{w}_{id}$.
- $\mathsf{PosRefresh}(1^{\gamma}, n)$: This algorithm takes as input a security parameter γ and the number n of path, then outputs a set of random paths $\{p_n\}$ where $1 \leq \{p_n\} \leq N$ and N is the number of the leaf nodes.
- $\mathsf{PosCreate}(\{id_n\}, \{p_n\})$: This algorithm takes as input the IDs of the user's data $\{id_n\}$ and a set of random paths $\{p_n\}$, then outputs a position map pos.
- $\mathsf{PosGarble}(pos, \widetilde{w})$: This algorithm takes as input the position map pos and a set of labels $\widetilde{w}$, then outputs the garbled position map $\widetilde{pos}$.

Algorithm 1: Evaluation circuit

Input: $\widetilde{pos}[\,] = \{\widetilde{w}_i\}, \widetilde{w}_{id}$
Output: loc_{id}
for $i = 0; i \leq size(\widetilde{pos}[\,])$; **do**
  if $\widetilde{w}_{id} = \widetilde{w}_i$ **then**
    return loc_{id};

Algorithm 2: Replacing circuit

Input: $\widetilde{pos}[\,] = \{\widetilde{w}_i\}, \widetilde{w}^*_{eval} = \{(\widetilde{w}^*_i)_0, (\widetilde{w}^*_i)_1\}, x^0_i, x^1_i$
Output: $\widetilde{pos}^*[\,] = \{\widetilde{w}^*_i\}$
for $i = 0; i \leq size(\widetilde{pos}[\,])$; **do**
  if $\widetilde{w}_i = x^0_i$ **then**
    $\widetilde{pos}^*[i].id = (\widetilde{w}^*_i)_0$;
  if $\widetilde{w}_i = x^1_i$ **then**
    $\widetilde{pos}^*[i].id = (\widetilde{w}^*_i)_1$;
return $\widetilde{pos}^*[\,]$;

5.2. Full Construction

Combining the techniques above, we propose a decentralized database system with oblivious access in a (parallel) smart contract model. The system consists of the following functions: KeyGeneration refers to generation of keys that are used to encrypt the data, verify the transaction, and sign on the transaction; Initialization refers to initialization of the entire system; IDHiding refers to hiding of the target data's ID; DataAccess refers to accessing of the user's data; DataRecovery refers to recovery of the user's data; DataUpload refers to uploading of the user's data. The construction of this system is shown in Figure 6. The details of the proposed system are elaborated as follows:

- KeyGeneration:

1. The user invokes KeyGen with the security parameter 1^{δ} to generate the secret key k for encryption and decryption of the data blocks.

$$\mathsf{KeyGen}(1^{\delta}) \rightarrow \mathsf{k}$$

- Initialization:
  1. The user invokes DataEnc with the secret key k to encrypt the user's data and the corresponding IDs $\{(id_n, b_n)\}$ and generates the ciphertext data blocks $\{B_n\}$. Following this, the ciphertext data blocks $\{B_n\}$ are ready to upload to the smart contract.

$$\mathsf{DataEnc}(\mathsf{k}, \{(id_n, b_n)\}) \rightarrow \{B_n\}$$

  2. After generating the ciphertext data blocks, the user has to build the position map. First, the user invokes PosRefresh with the security parameter 1^{γ} to generate a set of the random paths $\{p_n\}$ based on the number of the ciphertext data blocks $\{B_n\}$.

$$\mathsf{PosRefresh}(1^{\gamma}, n) \rightarrow \{p_n\}$$

  Second, the user invokes PosCreate with the data's ID and a set of random paths $\{p_n\}$ to create the position map pos.

$$\mathsf{PosCreate}(\{id_n\}, \{p_n\}) \rightarrow pos$$

  3. In order to use the scheme of garbled circuits on the smart contract, the user has to perform some preprocessing locally. The user invokes CreateGC with the security parameter 1^{λ} and the evaluation circuit C_{eval} to generate the garbled circuit $\widetilde{C}_{eval}$ for evaluation and the corresponding garbled labels $\widetilde{w}_{eval}$. The design of the evaluation circuit is shown in Figure 7.

$$\mathsf{CreateGC}(1^{\lambda}, C_{eval}) \rightarrow (\widetilde{C}_{eval}, \widetilde{w}_{eval})$$

  4. After generating the garbled labels $\widetilde{w}_{eval}$, the user invokes PosGarble to garbled the position map pos. The ID field of pos will be replaced by $\widetilde{w}_{eval}$ and the position map pos will become the garbled position map $\widetilde{pos}$. Thus, the data's ID will not be learned by the smart contract.

$$\mathsf{PosGarble}(pos, \widetilde{w}_{eval}) \rightarrow \widetilde{pos}$$

  5. As the garbled circuit can only be used once, the user has to update the labels on the garbled position map $\widetilde{pos}$. The user invokes CreateGC with the security parameter 1^{λ} and the replacing circuit $C_{replace}$ to generate the garbled circuit $\widetilde{C}_{replace}$ for replacing labels and the corresponding garbled labels $\widetilde{w}_{replace}$. The design of the replacing circuit is shown in Figure 8.

$$\mathsf{CreateGC}(1^{\lambda}, C_{replace}) \rightarrow (\widetilde{C}_{replace}, \widetilde{w}_{replace})$$

  6. The user builds the oblivious structure on the smart contract and stores the ciphertext data blocks $\{B_n\}$ in it based on the position map pos. The user deploys the smart contract to the Ethereum blockchain by sending the transaction $T_{initialize}$, which contains the ciphertext data blocks $\{B_n\}$, the garbled position

map $\widetilde{pos}$, and the garbled circuits $\widetilde{C}_{eval}$ and $\widetilde{C}_{replace}$. Next, it will generate the corresponding transaction T_I and the oblivious database ODB.

$$T_{initialize}(\{B_n\}, \widetilde{pos}, \widetilde{C}_{eval}, \widetilde{C}_{replace}) \rightarrow (T_I, \mathsf{ODB})$$

- IDHiding:
  1. After the oblivious database ODB is built on the smart contract, the user is ready to access the data. Before that, the user must invoke IDGarble with $\widetilde{w}_{eval}$ to garble the target data's ID tid into $\widetilde{w}_{id}$. Therefore, the target data's ID tid will not be learned by the smart contract.

$$\mathsf{IDGarble}(tid, \widetilde{w}_{eval}) \rightarrow \widetilde{w}_{id}$$

- DataAccess:
  1. As the garbled position map $\widetilde{pos}$ and garbled circuit $\widetilde{C}_{eval}$ have been uploaded to the smart contract, the user invokes T_{eval} with the target data's garbled ID $\widetilde{w}_{id}$ to generate the corresponding transaction T_E and the target data's location loc_{id} on the garbled position map $\widetilde{pos}$.

$$T_{eval}(\widetilde{w}_{id}, \widetilde{pos}, \widetilde{C}_{eval}) \rightarrow (T_E, loc_{id})$$

  2. The user invokes T_{read} with loc_{id} to generate the corresponding transaction T_R and gets a set of ciphertext data blocks $\{B_i\}$ that contains the target data.

$$T_{read}(loc_{id}) \rightarrow (T_R, \{B_i\})$$

  3. After the garbled position map $\widetilde{pos}$ is used by the evaluation circuit, the user has to replace the labels by invoking $T_{replace}$ with $\widetilde{C}_{replace}$. Next, it will generate the corresponding transaction T_{RP} and the updated garbled position map $\widetilde{pos}^*$.

$$T_{replace}(\widetilde{pos}, \widetilde{C}_{replace}) \rightarrow (T_{RP}, \widetilde{pos}^*)$$

- DataRecovery:
  1. After getting a set of ciphertext data blocks $\{B_i\}$, the user invokes DataRec with the secret key k and target data's ID tid to recover the data blocks and gets a set of the user's personal data and corresponding IDs $\{(id_n, b_n)\}$. Next, the user obtains target data based on the target data's ID tid.

$$\mathsf{DataRec}(\mathsf{k}, \{B_i\}, tid) \rightarrow \{(id_i, b_i)\}$$

- DataUpload:
  1. After getting the target data, the user invokes DataEnc with the secret key k to re-encrypt a set of plaintext data blocks and corresponding IDs $\{(id_i, b_i)\}$. Next, it will generate a set of re-encrypted data blocks $\{B_i^*\}$, which is ready to be uploaded to the smart contract.

$$\mathsf{DataEnc}(\mathsf{k}, \{(id_i, b_i)\}) \rightarrow \{B_i^*\}$$

  2. In order to hide the data access pattern, the user has to refresh the path on the position map. Therefore, the user invokes PosRefresh to generate the random path p^*, which is ready to be uploaded to the smart contract.

$$\mathsf{PosRefresh}(1^\gamma, 1) \rightarrow p^*$$

3. After re-encrypting the data and generating the random path, the user invokes T_{write} to upload a set of re-encrypted data blocks $\{B_i^*\}$ and the random path p^*. The target data block will be put in the child node of the previous path and the updated path. Other data blocks will be put back to the original node. It will then generate the corresponding transaction T_W and the updated oblivious database ODB^*.

$$T_{write}(\{B_i^*\}, p^*) \rightarrow (T_W, \mathsf{ODB}^*)$$

4. Because the garbled circuit $\widetilde{C}_{eval}$ on the smart contract has been used by the user, the user has to re-generate the garbled circuit for the next round. Thus, the user invokes CreateGC with the security parameter 1^λ and the evaluation circuit C_{eval} to generate the next round garbled circuit $\widetilde{C}^*_{eval}$ and the corresponding garbled labels $\widetilde{w}^*_{eval}$.

$$\mathsf{CreateGC}(1^\lambda, C_{eval}) \rightarrow (\widetilde{C}^*_{eval}, \widetilde{w}^*_{eval})$$

5. As noted above, because the garbled circuit $\widetilde{C}_{replace}$ on the smart contract has been used by the user, the user has to re-generate the garbled circuit for the next round. Thus, the user invokes CreateGC with the security parameter 1^λ and the circuit $C_{replace}$ to generate the next round garbled circuit $\widetilde{C}^*_{replace}$ and the corresponding garbled labels $\widetilde{w}^*_{replace}$.

$$\mathsf{CreateGC}(1^\lambda, C_{replace}) \rightarrow (\widetilde{C}^*_{replace}, \widetilde{w}^*_{replace})$$

6. After re-generating the garbled circuits for the next round, the user invokes T_{update} to upload garbled circuits $\widetilde{C}^*_{eval}$ and $\widetilde{C}^*_{replace}$. Next, it will generate the corresponding transaction T_U.

$$T_{update}(\widetilde{C}^*_{eval}, \widetilde{C}^*_{replace}) \rightarrow T_U$$

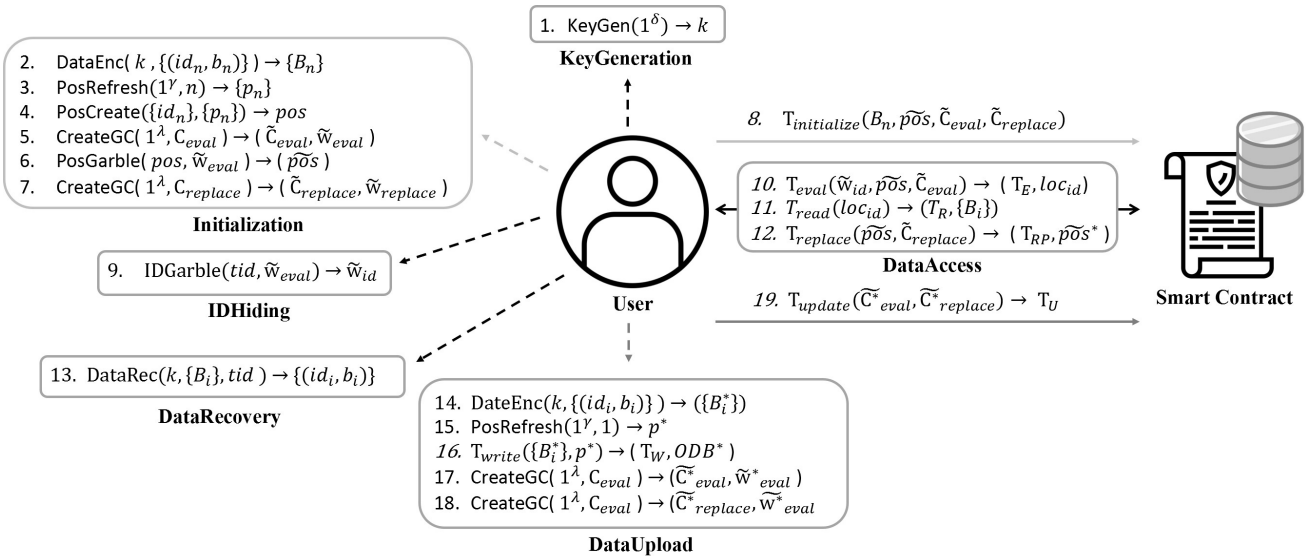

Figure 6. Construction of the decentralized database system with oblivious access.

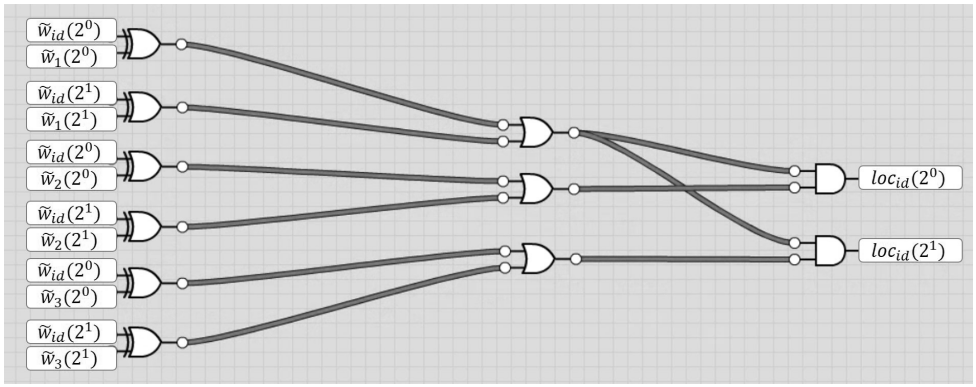

Figure 7. The design of the evaluation circuit with tree height $H = 3$.

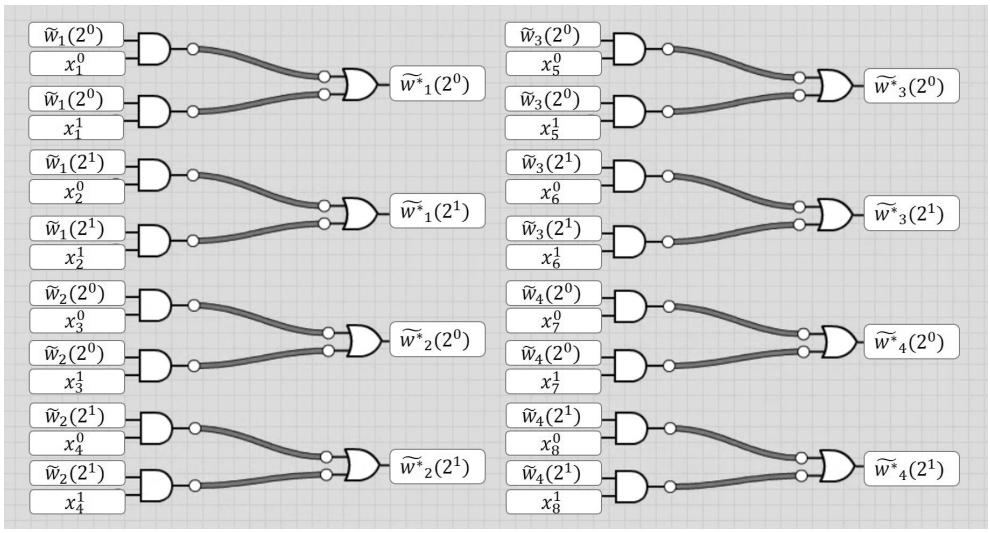

Figure 8. The design of the replacing circuit with tree height $H = 3$.

5.3. Security Proof

In this section, we will show the security proof of the decentralized database system with oblivious access (DDOA) based on the formal security definition proposed in the previous section. In Definition 3, we convert the view to the following distributions:

$$\mathsf{View}_{\mathcal{A}}(\mathsf{DDOA}(id_0)) \equiv \langle \widetilde{w}_{id}, \widetilde{pos}, \widetilde{C}_{eval}, \widetilde{C}_{replace}, loc_{id}, \{B_i\} \rangle$$

$$\mathsf{View}_{\mathcal{A}}(\mathsf{DDOA}(id_1)) \equiv \langle \tilde{w}_{id'}, \widetilde{pos}, \widetilde{C}_{eval}, \widetilde{C}_{replace}, loc_{id'}, \{B'_i\} \rangle$$

Therefore, our goal is to prove the following distributions are computationally indistinguishable for any probabilistic polynomial-time adversary $\mathcal{A}$:

$$\langle \widetilde{w}_{id}, \widetilde{pos}, \widetilde{C}_{eval}, \widetilde{C}_{replace}, loc_{id}, \{B_i\} \rangle \approx \langle \tilde{w}_{id'}, \widetilde{pos}, \widetilde{C}_{eval}, \widetilde{C}_{replace}, loc_{id'}, \{B'_i\} \rangle$$

Subsequently, we propose the following lemmas to prove the security of the system:

Lemma 1. *The DDOA system is secure if for any probabilistic polynomial-time adversary $\mathcal{A}$ such that the following distributions are computationally indistinguishable:*

$$\langle \widetilde{w}_{id}, \widetilde{pos}, \widetilde{C}_{eval}, \widetilde{C}_{replace}, loc_{id}, \{B_i\} \rangle \approx \langle \widetilde{w}^{\mathcal{S}}_{id}, \widetilde{pos}^{\mathcal{S}}, \widetilde{C}^{\mathcal{S}}_{eval}, \widetilde{C}^{\mathcal{S}}_{replace}, loc_{id}, \{B_i\} \rangle$$

Proof. Based on the security of garbled circuit we defined earlier, we know that for any adversary $\mathcal{A}$, there is a probabilistic polynomial-time simulator $\mathcal{S}$ such that the following distributions are computationally indistinguishable:

- $\mathsf{Real}_{\mathcal{A}}(\lambda)$: In the phase of Initialization, $\mathcal{A}$ chooses circuit C_{eval} and $C_{replace}$ to generate the garbled circuit. The experiment runs the following algorithms and sends $\widetilde{C}_{eval}$ and $\widetilde{C}_{replace}$ to $\mathcal{A}$.

$$\mathsf{CreateGC}(1^{\lambda}, C_{eval}) \rightarrow (\widetilde{C}_{eval}, \widetilde{w}_{eval}) \; \mathsf{CreateGC}(1^{\lambda}, C_{replace}) \rightarrow (\widetilde{C}_{replace}, \widetilde{w}_{replace})$$

In order to hide the data's ID on the position map, $\mathcal{A}$ outputs pos as input. The experiment then runs the following algorithm to generate $\widetilde{pos}$.

$$\mathsf{PosGarble}(pos, \widetilde{w}_{eval}) \rightarrow \widetilde{pos}$$

In the phase of IDHiding, in order to hide the target data's ID, $\mathcal{A}$ outputs tid as input. The experiment then runs the following algorithm to generate $\widetilde{w}_{id}$.

$$\mathsf{IDGarble}(tid, \widetilde{w}_{eval}) \rightarrow \widetilde{w}_{id}$$

In the phase of DataAccess, the experiments runs the following algorithms to get the target data from the smart contract.

$$T_{eval}(\widetilde{w}_{id}, \widetilde{pos}, \widetilde{C}_{eval}) \rightarrow (T_E, loc_{id})$$
$$T_{read}(loc_{id}) \rightarrow (T_R, \{B_i\})$$
$$T_{replace}(\widetilde{pos}, \widetilde{C}_{replace}) \rightarrow (T_{RP}, \widetilde{pos}^{*})$$

- $\mathsf{Ideal}_{\mathcal{A},\mathcal{S}}(\lambda)$: In the phase of Initialization, $\mathcal{A}$ chooses circuit C_{eval} and $C_{replace}$ to generate the garbled circuit. The experiment runs the following algorithms and sends $\widetilde{C}_{eval}$ and $\widetilde{C}_{replace}$ to $\mathcal{A}$.

$$\mathcal{S}(1^{\lambda}) \rightarrow (\widetilde{C}^{\mathcal{S}}_{eval}, \widetilde{w}^{\mathcal{S}}_{eval})$$
$$\mathcal{S}(1^{\lambda}) \rightarrow (\widetilde{C}^{\mathcal{S}}_{replace}, \widetilde{w}^{\mathcal{S}}_{replace})$$

In order to hide the data's ID on the position map, $\mathcal{A}$ outputs pos as input. The experiment then runs the following algorithm to generate $\widetilde{pos}$.

$$\mathsf{PosGarble}(pos, \widetilde{w}^{\mathcal{S}}_{eval}) \rightarrow \widetilde{pos}^{\mathcal{S}}$$

In the phase of IDHiding, in order to hide the target data's ID, $\mathcal{A}$ outputs tid as input. The experiment the runs the following algorithm to generate $\widetilde{w}_{id}$.

$$\mathsf{IDGarble}(tid, \widetilde{w}^{\mathcal{S}}_{eval}) \rightarrow \widetilde{w}^{\mathcal{S}}_{id}$$

In the phase of DataAccess, the experiments runs the following algorithms to get the target data from the smart contract.

$$T_{eval}(\widetilde{w}^{\mathcal{S}}_{id}, \widetilde{pos}^{\mathcal{S}}, \widetilde{C}^{\mathcal{S}}_{eval}) \rightarrow (T_E, loc_{id})$$
$$T_{read}(loc_{id}) \rightarrow (T_R, \{B_i\})$$
$$T_{replace}(\widetilde{pos}, \widetilde{C}^{\mathcal{S}}_{replace}) \rightarrow (T_{RP}, \widetilde{pos}^{*})$$

Hence, according to the security of garbled circuit, the above distributions are computationally indistinguishable for any probabilistic polynomial-time adversary $\mathcal{A}$. Finally, the proof of this lemma is done. □

Lemma 2. *The DDOA system is secure if for any probabilistic polynomial-time adversary $\mathcal{A}$ such that the following distributions are computationally indistinguishable:*

$$\langle \widetilde{w}^{\mathcal{S}}_{id}, \widetilde{pos}^{\mathcal{S}}, \widetilde{C}^{\mathcal{S}}_{eval}, \widetilde{C}^{\mathcal{S}}_{replace}, loc_{id}, \{B_i\} \rangle \approx \langle \widetilde{w}^{\mathcal{S}}_{id'}, \widetilde{pos}^{\mathcal{S}}, \widetilde{C}^{\mathcal{S}}_{eval}, \widetilde{C}^{\mathcal{S}}_{replace}, loc_{id'}, \{B'_i\} \rangle$$

Proof. Based on the security of ORAM we defined earlier, we know that for any two equal-length private sequences of the intended data requests, their corresponding observable access sequences are computationally indistinguishable. Therefore, accessing data with id and id', respectively, their corresponding observable access sequences $\{B_i\}$, $\{B'_i\}$ are computationally indistinguishable. Finally, the proof of this lemma is done. □

Lemma 3. *The DDOA system is secure if for any probabilistic polynomial-time adversary $\mathcal{A}$ such that the following distributions are computationally indistinguishable:*

$$\langle \tilde{w}^{\mathcal{S}}_{id'}, \widetilde{pos}^{\mathcal{S}}, \widetilde{C}^{\mathcal{S}}_{eval}, \widetilde{C}^{\mathcal{S}}_{replace}, loc_{id'}, \{B'_i\} \rangle \approx \langle \tilde{w}_{id'}, \widetilde{pos}, \widetilde{C}_{eval}, \widetilde{C}_{replace}, loc_{id'}, \{B'_i\} \rangle$$

Proof. The proof process is identical to Lemma 1. By the security of garbled circuit, the above distributions are computationally indistinguishable for any probabilistic polynomial-time adversary $\mathcal{A}$. Finally, the proof of this lemma is done. □

Consequently, the following distributions are computationally indistinguishable for any probabilistic polynomial-time adversary $\mathcal{A}$:

$$\langle \tilde{w}_{id}, \widetilde{pos}, \widetilde{C}_{eval}, \widetilde{C}_{replace}, loc_{id}, \{B_i\} \rangle \approx \langle \tilde{w}_{id'}, \widetilde{pos}, \widetilde{C}_{eval}, \widetilde{C}_{replace}, loc_{id'}, \{B'_i\} \rangle$$

In conclusion, the DDOA system is secure if the garbled circuit and ORAM are secure.

6. Implementations and Discussion

6.1. Garbled Circuit Implementations

In this section, we will demonstrate the design of garbled circuits. We have two kinds of garbled circuits in this paper. One is evaluation circuit $\widetilde{C}_{eval}$ for evaluating the location of the target data's ID; the other is replacing circuit $\widetilde{C}_{replace}$ for replacing the label on the position map. The evaluation circuit $\widetilde{C}_{eval}$ with tree height $H = 3$ is shown in Figure 7. Moreover, the input of the evaluation circuit is the target data's garbled ID $\tilde{w}_{id}$ and garbled position map $\widetilde{pos}$, and the output of the evaluation circuit is the location loc_{id} to which ID belongs. The replacing circuit $\widetilde{C}_{replace}$ with tree height $H = 3$ is shown in Figure 8. Furthermore, the input of the evaluation circuit is $\widetilde{pos}$,x^0_n and x^1_n, and the output of the evaluation circuit is the next round of garbled position map $\widetilde{pos}^*$. Significantly, we put the next round of garbled labels in the replacing circuit. Therefore, the user may update the labels on the service side without retrieving the position map back. The components of the garbled circuit are relevant to different types of transactions, and thus we show some experiments in the next subsection.

6.2. Smart Contract Implementation and Experiments

In this section, we will demonstrate the performance analysis of the smart contract. There are many blockchain networks that can support smart contracts, such as Ethereum, EOS, Hyperledger, RSK RBTC, etc. The smart contract on RSK RBTC and Ethereum are mainly programmed with solidity language, that on EOS is with C++ language, and that on Hyperledger is with Go language. Although they are programmed by different program languages, the logic of the smart contracts remains the same. As long as the program syntax is converted, the smart contract can be deployed on different blockchain networks intuitively. In this implementation, we deploy the smart contract on the Ethereum blockchain. Ganache is chosen to simulate the blockchain environment and JavaScript is chosen to implement experiments, with Intel i5 8500 3.0GHz and 8GB of memory running on Windows 10.

As we noted above, every transaction sent in Ethereum requires its dedicated cryptocurrency: ETH and gas. Therefore, when executing any operation on the Ethereum blockchain, we must pay a transaction fee that depends on the complexity of the internal structure. The transaction fee is calculated by gas and is paid by ETH. In June 2020, 1 ETH $\approx$ 235.92 USD, and 1 gas $\approx$ 1 wei (0.000000001 ETH). The costs of executing transactions in this system with tree height $H = 3$ are shown in Table 2. As can be seen from Table 2, $T_{initialize}$, which is the deployment of the smart contract, is the most expensive, but this is a one-off that differs from other transactions. The transactions T_{read} and T_{write} are related to the downloading and uploading of data blocks, respectively. Moreover, T_{eval} and $T_{replace}$ are related to the decryption of garbled circuits, and T_{update} is related to the update of garbled circuits. However, our smart contract codes follow some secure programming tutorials carefully. (We refer to the works

of [42,43] and the following on-line resources to develop the secure codes of smart contracts: https://consensys.github.io/smart-contract-best-practices/, https://ethernaut.openzeppelin.com/, https://docs.soliditylang.org/en/develop/security-considerations.html, https://ethereum.org/en/developers/docs/smart-contracts/security/, accessed on 7 March 2022.)

Table 2. Costs of transactions in this system with tree height $H = 3$.

Transaction	Gas Used	Transaction Fees (ETH)	Execution Time (ms)
$T_{initialize}$	4,607,129	0.0741	125
T_{eval}	483,250	0.0077	329
$T_{replace}$	483,250	0.0077	79
T_{read}	295,445	0.0047	69
T_{write}	295,445	0.0047	70
T_{update}	2,877,953	0.0463	518

6.3. Discussion of the Proposed System

In this section, we will give some analysis and discussion of the oblivious access system. According to the experiments, our observation includes a few results. The consumption of gas is mainly proportional to the number of logic gates in circuits. In addition, the transactions sent first will consume more gas, which is more than three times that of other sequences, as shown in Figure 9. In practice, the tutorial from the Ethereum Yellow Paper [30] has indicated such implementation scenarios. If the storage value of the smart contract is written for the first time (the value changes from zero to non-zero), it will charge 20,000 gas; if the storage value of the smart contract is modified (the value changes from non-zero to non-zero), it will charge 5000 gas. Finally, we implemented the garbled circuit in the smart contract, so the limitation is one-time use only. Hence, the user must update the garbled circuit by T_{update} after access. This transaction requires the highest gas, except for initialization. With the price of Ethereum mainnet, the cost of using the proposed system may not be practical. However, we suggest that the system can be abstracted and converted to the other smart contract-enabled blockchains with lower prices and also can be applied in private blockchain to avoid the cost concern.

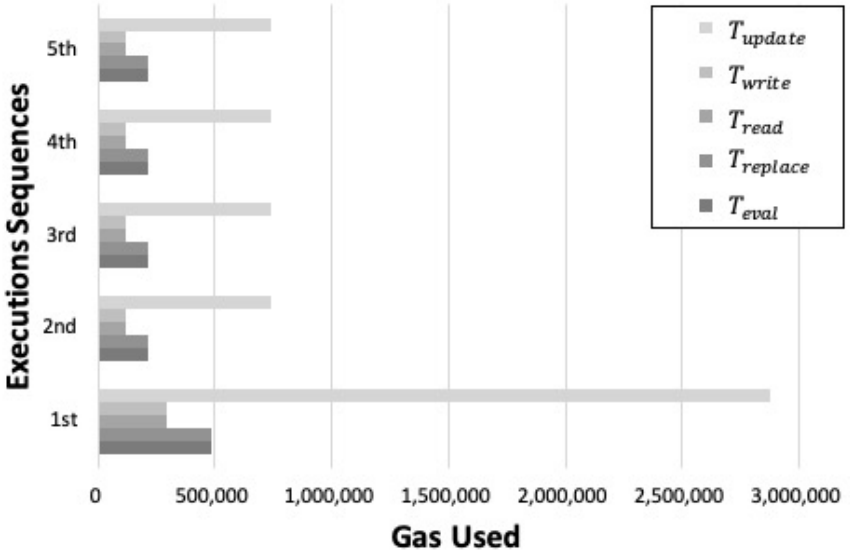

Figure 9. The relationship between execution sequences and gas used.

6.4. Extension on Parallel Smart Contract Model (If Possible)

The parallel smart contract model has a better performance in transaction processing. It uses multi-thread technology to implement the proposed model where transactions are executed in parallel. The differences between the original smart contract model and the parallel smart contract model are shown in Figure 10. This model consists of two phases as follows.

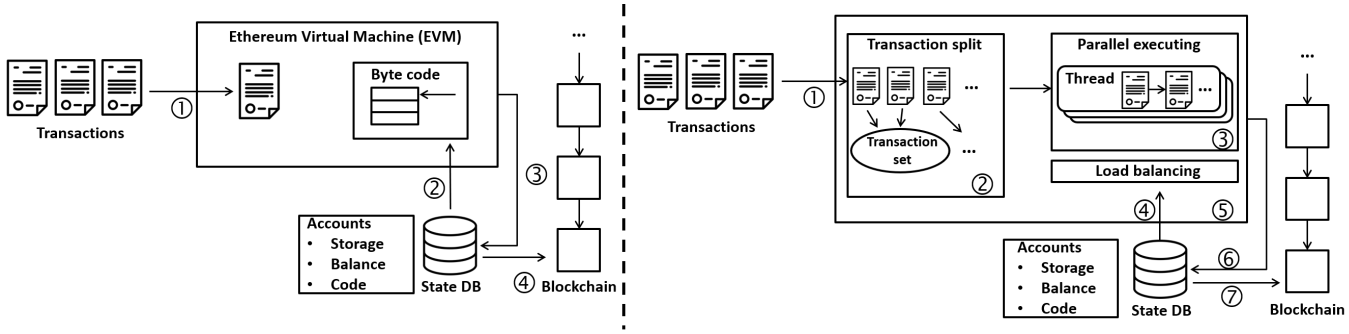

Figure 10. Original smart contract model vs. parallel smart contract model.

- $\mathsf{TransSplit}(T_1, \ldots, T_n)$: This algorithm groups the transactions and leaves no shared variables for each transaction group. It takes as input the transactions $T_1, \ldots, T_n$, then outputs the transaction sets $T_{Set_1}, \ldots, T_{Set_p}$.
- $\mathsf{MultiProcess}(T_{Set_1}, \ldots, T_{Set_p})$: This algorithm takes as input the transaction sets $T_{Set_1}, \ldots, T_{Set_p}$, then outputs the threads $th_1, \ldots, th_p$ to execute the transaction sets in parallel.

This model invokes TransSplit to generate the transaction sets $T_{Set_1}, \ldots, T_{Set_p}$ and generates the threads $th_1, \ldots, th_p$ to execute $T_{Set_1}, \ldots, T_{Set_p}$ in parallel by MultiProcess. If each transaction on the blockchain is compiled with the PSC model, it will improve the performance of processing transactions and reduce the average time cost. The decentralized database system with oblivious access in the parallel smart contract model is shown in Figure 11.

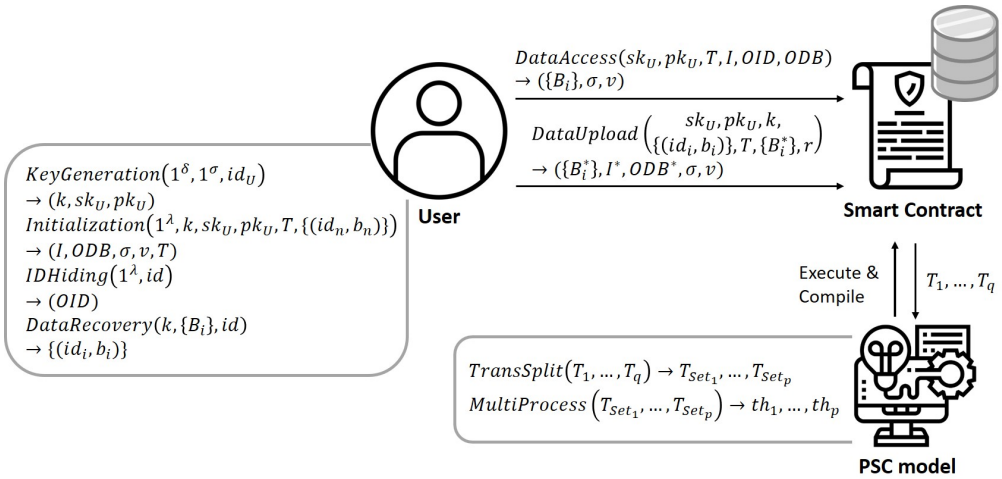

Figure 11. Model framework of the decentralized database system with oblivious access.

7. Conclusions

In this paper, we propose an oblivious access system on a decentralized database over a (parallel) smart contract model to address the problems on centralized cloud servers. This system combines smart contracts with garbled circuits and ORAM to provide a decentralized cloud storage service. More importantly, its security by the formal proof can hide the data access pattern. Finally, we deploy the proposed system on the blockchain,

and it avoids the leakage of data access patterns. We expect the system can be applied with the parallel smart contract model in the future, which can significantly improve the efficiency. Also, it might be able to keep privacy about query over blockchain; for example, a document is frequently accessed, and then the attacker considers that it is very crucial. In our future work, our next goal is to break the one-time restriction and minimize the gas used.

Author Contributions: Conceptualization, Z.-Y.G., Y.-C.C. and H.-P.L.; Formal analysis, Z.-Y.G. and H.-P.L.; Investigation, H.-P.L.; Methodology, Z.-Y.G. and Y.-C.C.; Software, Z.-Y.G. and H.-P.L.; Validation, Y.-C.C.; Writing—original draft, Z.-Y.G.; Writing—review & editing, Y.-C.C. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by Ministry of Science and Technology of Taiwan, grant numbers 109-2628-E-155-001-MY3 and 110-2218-E-004-001-MBK.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Acknowledgments: The authors would like to thank the anonymous reviewers for their comments that helped to improve the presentation of the paper, and acknowledge Wordvice for editing this manuscript.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Dorri, A.; Kanhere, S.S.; Jurdak, R. Towards an optimized blockchain for IoT. In Proceedings of the 2017 IEEE/ACM Second International Conference on Internet-of-Things Design and Implementation (IoTDI), Pittsburgh, PA, USA, 18–21 April 2017; pp. 173–178.
2. Nakamoto, S.; Bitcoin, A. A Peer-to-Peer Electronic Cash System. 2008. Available online: https://bitcoin.org/bitcoin.pdf (accessed on 17 March 2022).
3. Szabo, N. Formalizing and securing relationships on public networks. *First Monday* **1997**, *2*. [CrossRef]
4. Zhou, L.; Wang, L.; Sun, Y.; Lv, P. Beekeeper: A blockchain-based iot system with secure storage and homomorphic computation. *IEEE Access* **2018**, *6*, 43472–43488. [CrossRef]
5. Goldreich, O.; Ostrovsky, R. Software protection and simulation on oblivious RAMs. *J. ACM (JACM)* **1996**, *43*, 431–473. [CrossRef]
6. Stefanov, E.; Van Dijk, M.; Shi, E.; Fletcher, C.; Ren, L.; Yu, X.; Devadas, S. Path ORAM: An extremely simple oblivious RAM protocol. In Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security, Berlin, Germany, 4–8 November 2013; pp. 299–310.
7. Wang, X.; Chan, H.; Shi, E. Circuit oram: On tightness of the goldreich-ostrovsky lower bound. In Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, 12–16 October 2015; pp. 850–861.
8. Yao, A.C. Protocols for secure computations. In Proceedings of the 23rd Annual Symposium on Foundations of Computer Science (SFCS 1982), Chicago, IL, USA, 3–5 November 1982; pp. 160–164.
9. Yu, W.; Luo, K.; Ding, Y.; You, G.; Hu, K. A Parallel Smart Contract Model. In Proceedings of the 2018 International Conference on Machine Learning and Machine Intelligence, Stockholm, Sweden, 10–15 July 2018; pp. 72–77.
10. Arora, N.S.; Blumofe, R.D.; Plaxton, C.G. Thread scheduling for multiprogrammed multiprocessors. *Theory Comput. Syst.* **2001**, *34*, 115–144. [CrossRef]
11. Devadas, S.; van Dijk, M.; Fletcher, C.W.; Ren, L.; Shi, E.; Wichs, D. Onion ORAM: A constant bandwidth blowup oblivious RAM. In Proceedings of the Theory of Cryptography Conference, Beijing, China, 1–3 November 2016; Springer: Berlin/Heidelberg, Germany, 2016; pp. 145–174.
12. Moataz, T.; Mayberry, T.; Blass, E.O. Constant communication ORAM with small blocksize. In Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, 12–16 October 2015; pp. 862–873.
13. Goldreich, O. Towards a theory of software protection and simulation by oblivious RAMs. In Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing, New York, NY, USA, 25–27 May 1987; pp. 182–194.
14. Ostrovsky, R. Efficient computation on oblivious RAMs. In Proceedings of the Twenty-Second Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, 14–16 May 1990; pp. 514–523.
15. Goodrich, M.T.; Mitzenmacher, M. Privacy-preserving access of outsourced data via oblivious RAM simulation. In Proceedings of the International Colloquium on Automata, Languages, and Programming, Zurich, Switzerland, 4–8 July 2011; Springer: Berlin/Heidelberg, Germany, 2011; pp. 576–587.
16. Goodrich, M.T.; Mitzenmacher, M.; Ohrimenko, O.; Tamassia, R. Privacy-preserving group data access via stateless oblivious RAM simulation. In Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, Kyoto, Japan, 17–19 January 2012; pp. 157–167.

17. Kushilevitz, E.; Lu, S.; Ostrovsky, R. On the (in) security of hash-based oblivious RAM and a new balancing scheme. In Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, Kyoto, Japan, 17–19 January 2012; pp. 143–156.
18. Lu, S.; Ostrovsky, R. Distributed oblivious RAM for secure two-party computation. In Proceedings of the Theory of Cryptography Conference, Tokyo, Japan, 3–6 March 2013; Springer: Berlin/Heidelberg, Germany, 2013; pp. 377–396.
19. Stefanov, E.; Shi, E.; Song, D. Towards practical oblivious RAM. *arXiv* **2011**, arXiv:1106.3652.
20. Shi, E.; Chan, T.H.H.; Stefanov, E.; Li, M. Oblivious RAM with O ((logN) 3) worst-case cost. In Proceedings of the International Conference on The Theory and Application of Cryptology and Information Security, Seoul, Korea, 4–8 December 2011; Springer: Berlin/Heidelberg, Germany, 2011; pp. 197–214.
21. Dubovitskaya, A.; Xu, Z.; Ryu, S.; Schumacher, M.; Wang, F. Secure and trustable electronic medical records sharing using blockchain. In Proceedings of the AMIA Annual Symposium Proceedings, Washington, DC, USA, 4–8 November 2017; American Medical Informatics Association: Washington, DC, USA, 2017; Volume 2017, p. 650.
22. Tian, F. An agri-food supply chain traceability system for China based on RFID & blockchain technology. In Proceedings of the 2016 13th International Conference on Service Systems and Service Management (ICSSSM), Kunming, China, 24–26 June 2016; pp. 1–6.
23. Raju, S.; Boddepalli, S.; Gampa, S.; Yan, Q.; Deogun, J.S. Identity management using blockchain for cognitive cellular networks. In Proceedings of the 2017 IEEE International Conference on Communications (ICC), Paris, France, 21–25 May 2017; pp. 1–6.
24. Dwork, C.; Naor, M. Pricing via processing or combatting junk mail. In Proceedings of the Annual International Cryptology Conference, Santa Barbara, CA, USA, 16–20 August 1992; Springer: Berlin/Heidelberg, Germany, 1992; pp. 139–147.
25. Jakobsson, M.; Juels, A. Proofs of work and bread pudding protocols. In *Secure Information Networks*; Springer: Berlin/Heidelberg, Germany, 1999; pp. 258–272.
26. Castro, M.; Liskov, B. Practical Byzantine fault tolerance. In Proceedings of the Third USENIX Symposium on Operating Systems Design and Implementation (OSDI), New Orleans, LA, USA, 22–25 February 1999; Volume 99; pp. 173–186.
27. King, S.; Nadal, S. Ppcoin: Peer-to-peer crypto-currency with proof-of-stake. *Self-Publ. Pap. August* **2012**, *19*. Available online: https://bitcoin.peryaudo.org/vendor/peercoin-paper.pdf (accessed on 6 March 2022).
28. Larimer, D. Delegated proof-of-stake (dpos). *Bitshare Whitepaper* **2014**, *81*, 85.
29. Buterin, V. A next-generation smart contract and decentralized application platform. *White Paper* **2014**, *3*. Available online: https://blockchainlab.com/pdf/Ethereum_white_paper-a_next_generation_smart_contract_and_decentralized_application_platform-vitalik-buterin.pdf (accessed on 6 March 2022).
30. Wood, G. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum Proj. Yellow Pap.* **2014**, *151*, 1–32.
31. Daniel Larimer, B.B. EOS.IO's White Paper. 2017. Available online: https://github.com/EOSIO/Documentation/blob/master/TechnicalWhitePaper.md (accessed on 6 March 2022).
32. Cachin, C. Architecture of the hyperledger blockchain fabric. In Proceedings of the Workshop on Distributed Cryptocurrencies and Consensus Ledgers, Chicago, IL, USA, 25 July 2016; Volume 310, p. 4.
33. Salah, K.; Rehman, M.H.U.; Nizamuddin, N.; Al-Fuqaha, A. Blockchain for AI: Review and open research challenges. *IEEE Access* **2019**, *7*, 10127–10149. [CrossRef]
34. Lu, Y.; Huang, X.; Zhang, K.; Maharjan, S.; Zhang, Y. Blockchain empowered asynchronous federated learning for secure data sharing in internet of vehicles. *IEEE Trans. Veh. Technol.* **2020**, *69*, 4298–4311. [CrossRef]
35. Ouaddah, A.; Abou Elkalam, A.; Ait Ouahman, A. FairAccess: A new Blockchain-based access control framework for the Internet of Things. *Secur. Commun. Netw.* **2016**, *9*, 5943–5964. [CrossRef]
36. Bera, B.; Saha, S.; Das, A.K.; Kumar, N.; Lorenz, P.; Alazab, M. Blockchain-envisioned secure data delivery and collection scheme for 5G-based IoT-enabled Internet of drones environment. *IEEE Trans. Veh. Technol.* **2020**, *69*, 9097–9111. [CrossRef]
37. Zhang, S.; Yao, T.; Arthur Sandor, V.K.; Weng, T.H.; Liang, W.; Su, J. A novel blockchain-based privacy-preserving framework for online social networks. *Connect. Sci.* **2021**, *33*, 555–575. [CrossRef]
38. Xie, L.; Ding, Y.; Yang, H.; Wang, X. Blockchain-based secure and trustworthy Internet of Things in SDN-enabled 5G-VANETs. *IEEE Access* **2019**, *7*, 56656–56666. [CrossRef]
39. Steichen, M.; Fiz, B.; Norvill, R.; Shbair, W.; State, R. Blockchain-based, decentralized access control for IPFS. In Proceedings of the 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, 30 July–3 August 2018; pp. 1499–1506.
40. Dai, Y.; Li, G.; Xu, B. Study on learning resource authentication in MOOCs based on blockchain. *Int. J. Comput. Sci. Eng.* **2019**, *18*, 314–320. [CrossRef]
41. Lou, J.; Zhang, Q.; Qi, Z.; Lei, K. A blockchain-based key management scheme for named data networking. In Proceedings of the 2018 1st IEEE International Conference on Hot Information-Centric Networking (HotICN), Shenzhen, China, 17–19 August 2018; pp. 141–146.
42. Pierro, G.A.; Tonelli, R.; Marchesi, M. An organized repository of ethereum smart contracts' source codes and metrics. *Future Internet* **2020**, *12*, 197. [CrossRef]
43. Pierro, G.A.; Tonelli, R. Paso: A web-based parser for solidity language analysis. In Proceedings of the 2020 IEEE International Workshop on Blockchain Oriented Software Engineering (IWBOSE), London, ON, Canada, 18 February 2020; pp. 16–21.