computers

MDPI

Article

An Oracle-Based On-Chain Privacy

Yu-Jen Chen, Ja-Ling Wu, Yung-Chen Hsieh * and Chih-Wen Hsueh *

Computer Science & Information Engineering, National Taiwan University, Taipei 106216, Taiwan; r04922113@ntu.edu.tw (Y.-J.C.); wjl@cmlab.csie.ntu.edu.tw (J.-L.W.)

* Correspondence: will@cmlab.csie.ntu.edu.tw (Y.-C.H.); cwhsueh@csie.ntu.edu.tw (C.-W.H.)

Received: 26 May 2020; Accepted: 12 August 2020; Published: 29 August 2020

Abstract: In this work, we demonstrate how the blockchain and the off-chain storage interact via Oracle-based mechanisms, which build an effective connection between a distributed database and real assets. For demonstration purposes, smart contracts were drawn up to deal with two different applications. Due to the characteristics of the blockchain, we may still encounter severe privacy issues, since the data stored on the blockchain are exposed to the public. The proposed scheme provides a general solution for resolving the above-mentioned privacy issue; that is, we try to protect the on-chain privacy of the sensitive data by using homomorphic encryption techniques. Specifically, we constructed a secure comparison protocol that can check the correctness of a logic function directly in the encrypted domain. By using the proposed access control contract and the secure comparison protocol, one can carry out sensitive data-dependent smart contract operations without revealing the data themselves.

Keywords: blockchain; smart contract; privacy; homomorphic encryption

1. Introduction

Recently, the blockchain has aroused intensive discussions and has attracted much attention across a wide span of industries, including finance [1], healthcare [2], medical records [3], real estate [4], etc. One of the important reasons for this trend is that, with the aid of blockchain-based technology, many valuable transactions can now be operated in a fully decentralized and asynchronized manner i.e., without the need of a central authority, the transaction and/or exchange of valuable assets can still be done safely between two mutually distrusted parties. The absence of a central authority induces faster communications among the transacting parties and reduces the overhead caused by intermediaries. Using cryptographic primitives (such as digital signatures and secure Hash functions) is one of the key characteristics of the blockchain that allows for achieving authority in a distributed network. Certain self-executing scripts called "smart contracts" reside on the blockchain, allowing workflows to be executed properly, distributively and automatically. We leverage these concepts to implement our system: an autonomous stock management system working in the domain of the Internet of Things (IoTs) [5], where the on-chain access control scheme is secured. It is well-known that not every application is transferable to a decentralized network; therefore, a blockchain-based network may not fulfill the requirements of all applications. The blockchain might bring many advantages, but also carry certain concerns. One of the major concerns for developing blockchain- and/or smart contract-based applications is the privacy issue. Since all nodes on the network have rights to access all on-chain data, transactions are transparent to all participants who were exploring the blockchain; clearly, it is hard to maintain data privacy in this situation. Fortunately, there are some cryptographical techniques, such as multiparty computation [6], zero-knowledge proof [7] and homomorphic encryption [8], that might be applied to address this challenging issue. In general, they tackle the privacy issue by restricting the transaction's inputs and outputs visible only to senders and receivers while still permitting all other

nodes of the network to verify the transaction. In this work, an Oracle-based mechanism is proposed to integrate the above-mentioned superiorities of smart contracts with the desired on-chain privacy protection concerns. The contributions of this work include the following: (1) We demonstrate how the blockchain and off-chain storage properly interact with an Oracle to establish a connection between distributed databases and real assets. (2) We construct a blockchain-based access control scheme focusing on the on-chain privacy protection. Although this seems contradictory to the original purpose of the blockchain, it is necessary when we need to keep sensitive data on-chain. This paper is organized as follows. Some preliminary background, such as the definitions of transactions, the blockchain, smart contracts and the Paillier cryptosystem, is briefly addressed in Section 2.1 and some related works are reviewed in Section 2.2. The proposed mechanism is introduced in Section 3 and the concerned security issues are analyzed in Section 4. Some comparisons of our work with related works are given in Sections 5 and 6 concludes this work.

2. Background Review and Related Work

2.1. Background Review

2.1.1. Blockchain

Blockchain can be viewed as a distributed and globally shared database. It is a public ledger of transactions and digital events that have been executed and shared among participating parties. Blockchain was first introduced in a Bitcoin-related writeup [9] to solve the "double spending" issue [10] of digital currencies. Even if participants are not deliberately behaving maliciously, this issue still possibly exists due to the propagation delays in peer-to-peer networks. Blockchain can also be considered as a log file split into many time-stamped blocks. Each block contains the hash of the previous block, which guarantees that no previous block or the associated contents have been changed without being noticed; therefore, blockchain can be regarded as a trustworthy platform for providing correctness and availability.

2.1.2. Smart Contract

The smart contract concept was first introduced in 1994 by Nick Szabo [11], where he defined a smart contract as "a computerized transaction protocol that executes the terms of a contract." He believed that we would be starting to leverage smart contracts for real property in the future by implementing contractual clauses into the contract that are self-executing and self-enforcing. A smart contract ensures minimizing the occurrence of malicious intervention, the appearance of accidental exceptions and the transaction cost of the intermediaries.

Bitcoin has a pretty similar mechanism by offering a limited programmability through a scripting language. The main problem is its non-Turing-completeness and not being user friendly. Of course, there are also some smart contract-like applications built on top of Bitcoin, e.g., lottery [12], micro-payments [13] and verifiable computation [14]. These applications reduced the difficulty of leveraging the limited script on Bitcoin; it then motivates people to develop a Turing complete and user-friendly smart contract language.

To be more specific, a smart contract is an event-driven program with states that runs and takes custody over assets on a replicated, shared ledger. We consider this as the best explanation of the smart contract. In summary, a smart contract is a replicated state machine that receives inputs, executes logic operations on the states according to the input data and then provides an output depends on its function. Smart contracts are scripts stored on the blockchain, allowing us to have general-purpose computation occurring within the chain. It is powerful enough to implement some business logic through the contract beyond "Bob paying money to Alice."

Each smart contract has a unique address and its own states to manage the corresponding assets on the blockchain. The correctness of the contract has to be inspected before deploying it on the network. A smart contract is triggered by sending a transaction with the contract's address as the receiver. Then, the contract is executed automatically and independently on every node in the network. Every node is running the smart contract on a virtual machine (VM), which makes the whole blockchain network act as a distributed VM; we provide a more comprehensive explanation in the next section. Since the whole network gets cryptographically verifiable traces of the contract's operations, the whole structure offers the merest possibility of controversies on the processes. If the destination of the transaction is another External Owned Account (EOA), then the transaction may transfer some digital currencies (e.g., ethers) nothing than other else.

An essential feature of a smart contract is that it is always deterministic; the same input should always give the same output. If it was non-deterministic, there would be many results after different nodes executed the contract, preventing the blockchain network from reaching consensus. In practical realization, there is no random behavior function. Thus, non-deterministic features should be avoided; therefore, if one attempts to induce random behavior in a smart contract, the blockchain network will not accept the contract; however, in practice, it is hard to ensure that a contract could cover all possible conditions of the desired outcome, even if it is a well written one.

2.1.3. Oracle

Blockchains and smart contracts cannot access data directly from the outside of their network. To know where the data are and what contents are provided, a smart contract often needs to access information from the physical world relevant to the contractual agreement in the form of electronic data, also referred to as "Oracles." These Oracles are services that send and verify real-world occurrences and submit the related information to smart contracts, triggering state changes on the blockchain. In general, Oracles link the outside world and the smart contract by transferring, processing and sometimes storing the needed information and data between them to fulfill the objective of smart contract autonomously; however, new system security holes may be created if the integration of the Oracles and the blockchain network has not been carefully conducted. Although the decentralized nature of blockchain ensures the privacy of peers' identity and the use of hash-chaining structure makes the on-chain data even more protected, there are still issues with the overall system security. For example, since blockchain techniques do not have thorough security measures yet, hackers and inner-circle scammers can steal information and tokens of the outside world through the bridging of the Oracles. A proper security and privacy protection mechanism is a must for building effective blockchain-Oracle based applications. In this work, we focus on providing a general solution to the challenge using homomorphic encryption techniques.

2.1.4. Homomorphic Encryption Algorithms

Homomorphic encryption is a class of encryption algorithms in which certain operations can be directly carried out on ciphertexts to generate the corresponding encrypted results. After decrypting the encrypted result, it will match the result of the same operations performed on the corresponding plaintexts. For example, m denotes a plaintext message and $[m]$ denotes the corresponding ciphertext. As a result, $[m] = E_{pk}(m)$ means m is encrypted with a public key pk, and $m = D_{sk}([m])$ means $[m]$ can be decrypted with the secret key sk. A homomorphic encryption scheme that works well with both additions and multiplications is denoted as a fully homomorphic encryption (FHE) scheme. Other methods that work with only one of the two operations are called partial homomorphic encryption (PHE) schemes. The Paillier cryptosystem [15] is one of the PHE algorithms that provides a complete additive feature and a multiplicative feature with constraint. In addition to being an additive homomorphism, the Paillier scheme can be computed rapidly; therefore, it is widely used in the fields of data security and privacy preservation applications.

2.2. Related Work

To preserve the on-chain privacy, lots of users applied the so-called "one-time keys" to Bitcoin blockchain, which became ubiquitous thanks to Pieter Wuille's open-sourced "hierarchical deterministic wallets" [16]. A variety of practical blockchain solutions have also adopted this approach; however, one-time keys only provide privacy of identity, but do not address the privacy of amounts and the privacy of history. In other words, observers can analyze patterns in the remaining data to de-anonymize and trace the history of certain transfers on the chain, compromising even the privacy of identity. Maxwell et al. developed "Confidential Transactions" [17], a protocol for encrypting the input and output amounts of a transaction to allow on-chain members to validate the balances of the transactions. Notably, the Confidential Transactions scheme is independent of any cryptographic assumptions other than those used for the Elliptic Curve digital signatures, which are already used in Bitcoin and other blockchain protocols. Another concern about on-chain privacy is that the inputs used in a transaction can be traced back to its previous transactions that created them. This backward linkage reveals sensitive information about where the asset transfers originated. Protocols such as CoinJoin [18], TumbleBit [19], MimbleWimble [20], CryptoNote [21] and Zerocash [22] (a zero knowledge-based method and had implemented in Zcash) have been proposed to deal with this problem. The three most-related works to our approach are addressed in this sub-section. One of them was also chosen as one of the benchmarks for a comparison.

Zyskind et al. [23] developed a personal data management system based on a decentralized peer-to-peer network, enabling different "compound identities" to store and process computations on data jointly. A compound consists of a user and a server, while the user's identity has the authority to modify the access rights of the server. They proposed to store data off-chain because the required heavy computation and privacy protection cannot be realized on the current blockchain directly. As a result, an off-chain solution for data storage is more desirable and functional. The data are encrypted in the database, and only the data pointers are logged on the chain.

The authors of MedRec [3] built their scheme based on the idea mentioned above. They developed a smart contract-based permission scheme that supports data retrieving and sharing on medical records. They are the first team to introduce blockchain technology to the application of medical record storing, which empowered individuals with record authenticity and data sharing. They saved an assortment of data pointers and associated access permissions on a contract. Each pointer consists of a query string that returns a subset of data when executed on the database. To enable users to share records with others, they implemented a dictionary (a hash table) that maps the viewers' addresses to a list of additional query strings. Each string can specify a portion of the user's data to which third party viewers are allowed to access. Delgado-Mohatar et al. explored practical tradeoff in blockchain-based biometric template storage [24]. They had recently experimentally studied the fundamental tradeoffs involved in the integration of Biometric (such as face and handwritten signatures) template storage with the Ethereum blockchain [25], including latency, processing time, gas cost and recognition rate. Experiments were conducted based on three different implementation schemes (i.e., full on-chain storage, data hashing and Merkle trees). They reported in popular biometrics research benchmarks, including deep approaches and databases captured in the wild. As a result, the authors found that straightforward schemes for data storage in the blockchain may be prohibitive for biometric template storage using state-of-the-art biometric methods. Still, the system based on Merkle trees had an excellent cost-performance tradeoff; however, no specific security and privacy protection scheme other than digital signature and hashing has been considered in [24]. In this work, we try to provide a general security and privacy-preserving approach to blockchain-Oracle combined applications by using homomorphic encryption techniques.

3. The Proposed Mechanisms

We offer two application scenarios that involve leveraging the Ethereum blockchain to show the effectiveness of the proposed mechanism. Before presenting these two applications, the execution environment of the adopted Ethereum blockchain is first introduced.

3.1. Execution Environment

An Ethereum Virtual Machine (EVM) is designed to serve as a runtime environment for smart contracts based on Ethereum. It can read and execute the bytecodes that are compiled from the completed contracts. All the nodes execute contracts with their EVMs. Actual computations in an EVM are conducted through a stake-based bytecode language. Note that EVM is a closure performing environment, and every executing processes are done in EVM. In other words, contracts cannot get any data outside the EVM environment. This characteristic critically limits EVM in the development of decentralized applications (DApps). For instance, there is no random number generator in the contract language, so dealing with applications based on a random variable in contracts is impossible and prohibited. Oracle is a solution to solve the limitation mentioned above. The idea of including a distributed Oracle network on-chain has been presented for a long time. Oracle behaves like a third party listening to the events that occur on the blockchain and responds to them. For example, you can interact with an Oracle when you need some data, including those you cannot fetch yourself. Leveraging Oracle, interestingly, one can complete an N-out-of-M multi-signature transaction by giving only one private key to each of the receivers. The transaction will be validated once N-out-of-M Oracles have a consensus on signing the transaction. More generally, it not only provides a signature on the transaction once some conditions are satisfied, but also provides the data one requested and then triggers the succeeding transactions or state changes. In this way, contracts can interact with the off-chain world, sending external data to contracts that the contract could not initially access.

As illustrated in Figure 1, the Oracle listens for a particular event and receives the query from some contract re-sites on an EVM. After receiving the query, the Oracle reaches the external database or web-APIs to acquire the desired data. After getting the desired data, the Oracle calls the callback function in the contract to return the data that the contract requested. This process is the primary interaction workflow of the whole proposed mechanism.

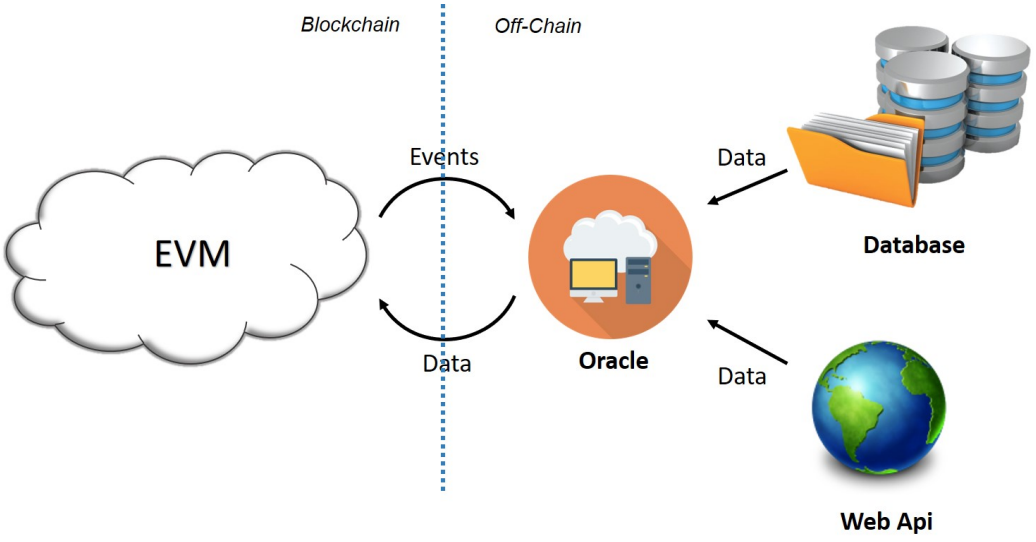

Figure 1. A blockchain diagram of the proposed scheme, where an Ethereum Virtual Machine (EVM) is connecting with an Oracle.

3.2. Application Scenario 1

In this section, we first demonstrate how the proposed mechanism works in our application scenario 1: the automation of a stock management system (AoSMS). Since all the data and records are stored on the blockchain, tampering or modifying them is prevented. Also, the whole process can be viewed as being executing by a trusted black-box. As shown in Figure 2, there are five entities involved in this application including: the user (which is a customer), the service provider (which is a logistics company), the oracle (which has the escrow behavior), suppliers and nodes (which maintain the operation of the blockchain).

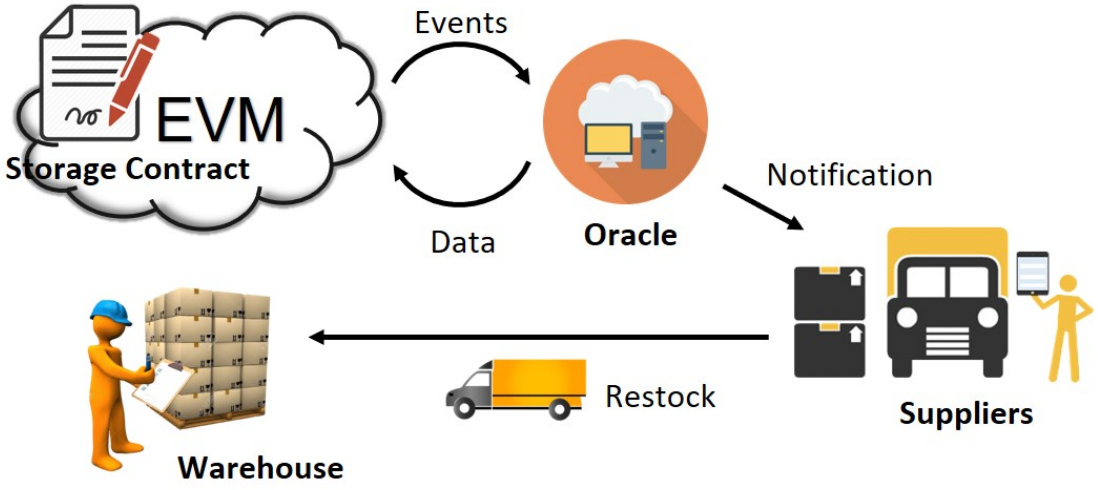

Figure 2. The sketch map of the proposed application scenario 1.

Let us draw up our first contract: The Storage Contract, which is re-sited on an EVM. It contains digital assets listed on the inventories, which keeps track of the assets in the real world. Our goal is to make the management process as autonomous as possible. We can apply this system to any logistics company dealing with her inventory management. First, we introduce some primary functions of this contract.

There are three main functional modules in this contract, *NewRecord*, *Restock* and *Sell*. The *NewRecord* function is for establishing a new item record, like making an archive for a new item. It registers the item name, item type, item cost and the first arrive quantity in-stock. The *Restock* function is an external linking function for our Oracle. The call for this function is valid only for some particular accounts, such as the suppliers' accounts. It adjusts the stock quantity after restocking the assets. The *Sell* function sells the item that was requested, only if payment and storage quantity are both sufficient. We withdraw the item that has the earliest in-stock time by checking the timestamp of every item. Note that customers can call the *Sell* function attached with the currency as the parameters of the transaction.

In our system, the Oracle acts like an escrow. It receives the contract requests when the codes, given in Figure 3, read the events that are highlighted in lines 4, 16, 21, 23 and 26. We now explain the processes among some logistics companies' stocking process in detail.

Suppose a logistics company is leveraging the smart contract, given in Figure 3. The owner of the company first calls the *NewRecord* function by sending transactions to this contract to archive all inventories. The oracle will listen to the event mentioned in line 4 of Figure 3, which works as a notification to the user. The contract allows buyers to send transactions by calling the *Sell* function to request for items. The contract checks the item quantity first, if the stock is insufficient to fulfil the request, the event in line 26 of Figure 3 will be triggered and fetched by the Oracle. The Oracle will notify the suppliers to activate the restock process. After a while, when the suppliers finish restocking, the Oracle automatically sends a transaction by calling the *Restock* function to conduct the

quantity adjustment. The buyers can now resend the transaction requesting items. After checking the item quantity, the contract will review the payment amount from the buyer. This flow completes the whole automation process for stock management. Note that the Oracle plays a critical role, such as handling notifications, negotiating among data transfers and acting like a bridge between on-chain and off-chain worlds.

```
1.contract Storage
2.
3.      function NewRecord(item_name, item_type, cost, quantity)   onlyOwner
4.              event NewRecordAlert
5.              saved all the parameters to a struct
6.              record the timestamp
7.      end function
8.
9.      function Restock(item_name, cost, quantity)   onlySupplier
10.             event RestockAlert
11.             increase the restock amount to the particular item
12.             record the timestamp
13.     end function
14.
15.     function Sell(item_name, quantity)
16.             event SellAlert
17.             if item enough then
18.                     if paying enough then
19.                             subtract the desired item quantity amount
20.                             if occur item quantity shortage then
21.                                     event RestockAlert
22.                     else
23.                             event PaymentShortageAlert
24.                     end if
25.             else
26.                     event RestockAlert
27.             end if
28.     end function
29.
30.end contract
```

Figure 3. The storage contract for supporting the proposed automation of a stock management system (AoSMS).

3.3. Application Scenario 2

In the second application scenario, we assume sensitive and valuable data are stored in the database. On-chain data are always exposed to the public, and an access control mechanism is a must while leveraging the blockchain technology. One way to implement such an access control system (ACS) is to embed an encryption hierarchy on the sensitive data and handle the decryption key distribution issue, which depends on the security level the query user demanded.

As an example, we will now build a smart and secure data access control scheme of a company, which is located on each node of the blockchain network. In our ACS, contracts are made for containing metadata about the human resources records, granting permissions and rules for maintaining data integrity. To successfully navigate through this relatively elaborate scheme, we constructed three types of contracts. Figure 4 illustrates the contract structures and the relationships among the contracts.

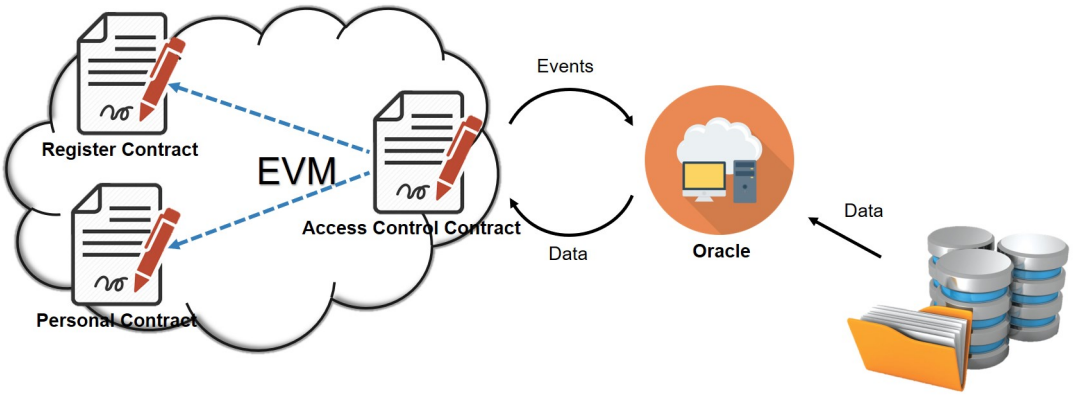

Figure 4. The sketch map of the proposed access control system (ACS) for protecting sensitive databases.

3.3.1. Register Contract (RC)

This contract maps the identity strings of the participants (i.e., the employees of the company) to their Ethereum account addresses (which plays a role similar to their public keys) along with some attributes (e.g., job-grade and salary). This contract also maps each identity string to an address on the blockchain, where the so-called "Personal Contract (PC)" resided. Notice that each participant owns only one personal contract (PC) in our RC. For security and privacy consideration, PC can only be accessed by some specific accounts (e.g., the manager or human resources staff).

3.3.2. Personal Contract (PC)

A PC is usually constructed for a particular purpose: it is created immediately when a participant's identity is registered to an RC. A PC holds a data pointer pointing towards one's sensitive personal data stored on the database. The data pointer consists of a query string that matches to one entry of the stored key-value database. That is, when a participant works on the contract with a pointer pointing towards the database, the set of the participant's data (associated with the pointer) will be returned. The query string is affixed with the hash value of the corresponding data set to ensure the returned data have not been altered at the source (database). Also, PC stores the access information to the database where the sensitive data are stored. At the same time, company employees have fine-grained access control to their data records, such as selecting essentially any portion of the records they wish to share, by updating their personal data and the associated data pointer. We demonstrate the effectiveness of our design by leveraging the levelDB database [26], which supports the above-mentioned key-value storage structure.

Note that the data files in the database have been encrypted with the employee's secret key. So even if an adversary gets the personal data set, the leaked data will not reveal any sensitive information. In our system, we do not claim to protect the security of individual databases, as we believe that local administrators are responsible for data protection.

3.3.3. Access Control Contract (ACC)

This particular contract functions as an authorization center: it gives the permit for accessing the resources. In our ACS, the resources indicate the data stored on PCs, RCs and some classified investment plans of the company. We strictly control the access rights to setters and getters of those data through this contract. That is, the only way to access or modify the data stored on PCs and RCs is to send a valid transaction to ACC. To be more specific, only the human resources staff or manager's accounts can register new records on RC and update the attributes of the employees; only the employee themselves can set or update the data pointer stored on their individual PC. Each employee can give access rights to some participants for accessing their sensitive data stored on the database. We also developed some policies for making extra queries; that is, we stipulated for specific information

requisition regulations. For example, if one satisfied the condition of one's salary amount or the level of job grade, they are granted the information about the classified investment plan.

3.4. On-Chain Privacy

We have addressed the issue of on-chain privacy from the viewpoint of the transactions of a smart contract. Consequently, the statuses and contents of contract states are kept private from the public, including all chain participants except those involved in the contract. That is, the only exception is the case where the involved parties voluntarily disclose the information. Unlike Hawk [27], which achieved its security guarantee by compiling the program via a cryptographic protocol, we only provided partial privacy-preservation when compared to Hawk. In other words, our scheme keeps the desired sensitive data private but let the remaining data be publicly accessible.

What does the above-mentioned on-chain privacy mean? Let us first consider the following situation: if the logic or function of a contract depends on some data that are so sensitive that they have to be protected, but at the same time, we want to leverage the benefits of blockchain technology, what should we do? Recall that the blockchain is a distributed public ledger, that is, the on-chain data are expected to be publicly accessible; therefore, it is infeasible to encrypt the data or the logic function directly, because we cannot verify the result from nodes that do not have the decrypt key. What if we had a way to test the truthfulness of the encrypted logic condition? Consider the following example shown in Figure 5.

```
1.contract AccessControl
2.      each employee's salary is encrypted and stored as [[Salary + R]] in this contract
3.      ...
4.
5.      function RequestInvestmentQualification()
6.              event SendEncryptedData
7.      end function
8.
9.      function CheckInvestmentQualification(requestInvestmentQualification result)  onlyHA
10.             if employee's job grade or employee's salary > some amount of currency then
11.                     event ReturnInvestmentPlan
12.             end if
13.     end function
14.     ...
15.end contract
```

Figure 5. An illustrative example of an encrypted data-dependent access control mechanism. Note that the public key encrypts each employee's salary, and R represents a random number that is selected by the employees they themselves.

In line 5 of Figure 5, an employee can request the company's investment plan. The *RequestInvestmentQualification* function sends an event to the Oracle. Let's assume Node A is the Oracle server with the highest authority in this company. It has the highest authority account that only this account can execute the function listed in line 9. Node A then follows the protocol described below to interact with Node B, another node in the same blockchain network. We constructed a secure comparison protocol that allows our contract to have the ability to compare the encrypted numbers directly on the blockchain.

Notions: The parameters with $[\![\cdot]\!]$ show that they are Paillier encrypted, and $[\cdot]$ show that they are QR [28] encrypted. Let *pk1* and *sk1* denote the data owner's public key and private key, respectively; *pk2* and *sk2* indicate Node A's public key and private key. Let **S** denote Salary and **V** denote the number to be compared. Table 1 presents the proposed secure comparison protocol.

Table 1. The proposed secure comparison protocol.

Node A (holding decryption key sk2)	Node B (holding decryption key sk1)
1: receive $[\![S+R]\!]_{pk1}$	
2: get $[\![R]\!]_{pk1}$ from data owner $[\![S]\!]_{pk1} \leftarrow [\![S+R]\!]_{pk1} \cdot [\![R]\!]_{pk1}^{-1}$	
3: $[\![V]\!]_{pk1} \xleftarrow{\text{encrypt}} V$	
4: Set $[\![S]\!]_{pk1}$, $[\![V]\!]_{pk1}$ with same bit length l, execute secure comparison protocol	
5: ***Secure Comparison Protocol*** [29] **Input**: $[\![S]\!]_{pk1}$ and $[\![V]\!]_{pk1}$ **Result**: **Node A gets a bit** t' **(where** $t' = 1$ **if** $a \leq b$**)**	
6: $[\![X]\!]_{pk1} \leftarrow [\![V]\!]_{pk1} \cdot [\![2^l]\!]_{pk1} \cdot [\![S]\!]_{pk1}^{-1} \bmod N^2$ $(X \leftarrow V + 2^l - S)$	
7: $[\![Z]\!]_{pk1} \leftarrow [\![X]\!]_{pk1} \cdot [\![R]\!]_{pk1} \bmod N^2$ $(Z \leftarrow X + R)$	
8: $\xrightarrow{[\![Z]\!]_{pk1}}$	
9:	$Z \xleftarrow{\text{decrypt}} [\![Z]\!]_{pk1}$
10: $c \leftarrow R \bmod 2^l$	$d \leftarrow Z \bmod 2^l$
11: Note A and Note B privately compute the encrypted bit [t] such that $\{ (t = 1) \equiv (d < c) \}$	
12: $[R_{l+1}]_{pk2} \xleftarrow{\text{encrypt}} R_{l+1}$	$[Z_{l+1}]_{pk2} \xleftarrow{\text{encrypt}} Z_{l+1}$
13: $\xrightarrow{[R_{l+1}]_{pk2}}$	
14:	$[t']_{pk2} \leftarrow [t]_{pk2} \cdot [Z_{l+1}]_{pk2} \cdot [R_{l+1}]_{pk2}$ $(t' \oplus Z_{l+1} \oplus R_{l+1})$
15: $\xleftarrow{[t']_{pk2}}$	
16: $t' \xleftarrow{\text{decrypt}} [t']_{pk2}$	

3.5. The Applicability of the Proposed Scheme

Besides the time required to upload data to and download data from the Ethereum blockchain, the most time-consuming operations occur at the proposed secure comparison protocol because encryption, decryption and comparison in the ciphertext domain are conducted at this stage. Fortunately, as there has been significant progress in cryptography, the involved partial homomorphic encryption schemes (Paillier and OR) can be realized in a sub-second time extent on a personal computer. To illustrate the processes of the discussed application scenarios, an operational demo video of the system operation is provided at the following link: https://www.dropbox.com/s/nm3c65lm4uctl2l/App_demo_with_sub.mp4?dl=0.

In this proposed scheme demo, we set up our private Ethereum-based blockchain with the web3 API, an open-source API. We implemented our Registrar, Personal and Access Control smart contracts with Solidity, an object-oriented and high-level language, to implement the smart contracts and deployed the private blockchain. We also set up three Oracles with Node.js and named them

Node A, Node B and DataServer, respectively. Node A is for listening to the event from the Access Control contract, waiting to execute the secure comparison protocol with Node B requested from the received event. Node B has the single purpose of being the execution party of the secure protocol. The DataServer returns the decrypted investment plan if the protocol returns the right result.

We created two accounts in the Registrar contract with different job grades and salary bases to provide two examples, one that was successful and one that failed. The encrypted investment plan was stored in our levelDB, key-value database, previously. We handled the request by sending our blockchain transactions, which will trigger our contract code to execute. We used an account to send the transaction to the Access Control contract. After calling the Access Control contract, it automatically sent an event to Node A, starting the secure comparison protocol with Node B. The final result decides whether we can get the decrypted data in the DataServer. The whole process was done in a few seconds, which is a reasonable timeframe, especially if a company has a lot more oracles and can handle more requests.

4. Security Analysis

Like all the previous works in designing smart contract applications on an underlying decentralized blockchain, we rely on blockchain being tamper-free. We assume the blockchain consensus protocol is secured under the assumption that the adversary does not own a large enough computation power to dominate the whole network, since the Ethereum blockchain is also based on the proof-of-work consensus protocol. We indicated some security situations that our system may encounter as follows:

4.1. What If Some Adversaries Send Fake Transactions and Try to Tamper with the System?

We assume the involved parties have the intention to protect the blockchain network and correct the result. In application scenario 1, we set strict limits to the contract, where only the company sieved accounts can call the *NewRecord* function. Only the supplier accounts can call the *Restock* function if necessary. In application scenario 2, any access or modification of the involved data is controlled by sending transactions to ACC and then interacting with RC and PC. By binding ACC's account to the other two contracts, there is no way to access both contracts' data; therefore, in general, we can prevent data-tampering caused by fake transactions.

4.2. What If an Adversary Can Read the Variable of the Contract Code?

Although all the contract codes are stored in the bytecode form, a participant can still read and even comprehend the codes once he knew how to transfer the contract codes into bytecodes; therefore, we assume all the contract codes are transparent to any participant involved in the blockchain. In application scenario 1, since our purpose is to show all the stock information on-chain publicly, it is no harm to allow anyone reading the states of the contract or other information. In application scenario 2, one can learn a participant's publicly accessible personal information like name, account and job-grade. One may rely on the protection job in the hash function used in PC; however, that data hash is used only to validate the data integrity. In other words, the hash is only used to ensure that the data are not tampered with after being stored on the database. Fortunately, we kept the security of our on-chain sensitive data by encrypting them with a random number. It can prevent any other node, even if it has the decryption key, from decrypting the data directly. Recall that we kept the random number and the decryption key separately, so the on-chain sensitive data are safe and can still be verified by the blockchain network.

4.3. Is an Adversary Still Be Able to Tamper with Our System without Going through the Register Account?

In application scenario 1, although the unregistered account can send transactions successfully, the contract statuses remain unchanged. We limited the access rights of the working function to some specific accounts. The *Restock* function will only be automatically executed through the Oracle,

which holds the supplier's signature key. This is why we assume the Oracle was set up on the supplier's server; therefore, the unregistered accounts should not be able to do anything. In application scenario 2, the reasons are pretty much the same as the above. The proposed ACC will block the unregistered accounts from accessing anything about the data. That is, ACC made our system tamper-proofed to the unregistered accounts.

4.4. What If the Oracle Is Not Trustworthy?

Using a single Oracle does induce a trust issue, as expected, because asking a decentralized contract to trust a single outside data-source is a dangerous and unacceptable choice. This issue can be somewhat mitigated by using multiple Oracles independently located on different nodes. They respond to the same queries, and reach a common consensus. On the other hand, one may also wonder about the feasibility of leveraging multi-signature Oracles to verify transactions to achieve the purpose of access control. Since an elliptic curve-based verification process takes longer execution time than running a few secure hash functions, we believe that the blockchain-based approach is easier and faster. The Oracle's best role is in dealing with the external data and the protocol or computation related to the privacy matters. So, we choose to leverage Oracles to achieve our system goal.

In short, the advantages of the proposed Oracle-based on-chain privacy-preserving mechanism can be summarized in Table 2.

Table 2. The security merits of the proposed scheme.

	Adversary Tamper Resistance	**Sensitive Data Protection**	**Access Control Mechanism**	**Distributed System Consensus**
our system	✓	✓	✓	✓

5. Discussion

Comparisons of the proposed system to some related works are presented in this section, where the merits of leveraging blockchain technology are also provided.

In Table 3, we compare our access control scheme with two related works [23] and *MedRec* [3]. There are four simple remarks about the comparison: (i) Both *MedRec* and our system leveraged the benefits of smart contracts while [23] was implemented on the UTXO-based blockchain. (ii) We apply the off-chain storage method to store the sensitive data while only keeping the data pointer on-chain to examine the data correctness. (iii) We can all achieve data sharing through the adjustment of access control. (iv) Only the proposed system supports on-chain privacy, allowing the encrypted sensitive data storage on-chain while the execution on other nodes are not affected.

Table 3. The comparison between the proposed access control scheme and two related works.

	Model	**Off-Chain Storage**	**Data Sharing Feature**	**On-Chain Privacy**
[23]	UTXO-based	✓	x	x
MedRec	Account-based	✓	✓	x
Proposed	Account-based	✓	✓	✓

Building the proposed systems on a blockchain has the following advantages: (i) The records on the blockchain will not be tampered with. There is no room for tampering with the data without leaving a trace. Consequently, the storage list of our AoSMS is correctly maintained, and the access right of our ACS is appropriately managed. (ii) The system services will not crash or disappear even if encountering attacks. A few nodes crashing will not affect our systems' regular operations due to their decentralized nature. (iii) The self-execution property of smart contracts will ensure the execution

of logic codes from the beginning to the end, autonomously producing the correctness of the results. (iv) Only the signature of an account is revealed as an identification in the blockchain. A signature in the account-based model is hard to imitate; that is, it is difficult for an adversary to disguise themselves as someone else. This fact is critical for the accounts that have high authority or a considerable amount of wealth.

Of course, there are some disadvantages to blockchain technology in practical usage. For example, in a traditional stock management system, when we purchase items, the deal is done after paying the cash; however, for leveraging blockchain, the agreement does not count until the transaction has been successfully collected into a block. The buyer has to wait for a block miner to deploy the block that contains his transactions. If there is a transaction traffic, the waiting time may be too long to tolerate.

6. Conclusions and Future Work

We take advantage of the blockchain record's immutability and decentralized nature and the smart contract's autonomy to illustrate how Oracles work to establish the bridge between on-chain and off-chain storages. This construction can be easily extended to any item-based management scheme. A blockchain-based access control scheme with the specific on-chain privacy feature is also presented. We conquered the on-chain privacy challenge by applying the time-efficient partial homomorphic encryption scheme to encrypt the sensitive data. For demonstration, we constructed an on-chain secure comparison protocol so that we can check the truthfulness of a logic function in the encryption domain, directly. With the aids of the proposed ACC and the secure comparison protocol, we can carry out sensitive data-dependent smart contract operations without revealing the data themselves.

In the future, we will continue to leverage the idea of privacy protection to a distributed database. The goal is to increase the security of both on-chain and off-chain storage. Having only the secure comparison scheme in the encryption domain is far beyond the need to provide privacy-preserving smart contracts. We will devote ourselves to developing other kinds of on-chain secure operations, which is, of course, one of our primary research directions.

Author Contributions: Conceptualization: Y.-J.C. and Y.-C.H.; methodology: Y.-J.C. and Y.-C.H.; software: Y.-J.C.; validation: J.-L.W. and C.-W.H.; formal analysis: Y.-J.C.; investigation: Y.-J.C. and C.-W.H.; resources: J.-L.W. and C.-W.H.; data curation: Y.-J.C.; writing—original draft preparation: Y.-J.C.; writing—review and editing: Y.-J.C. and J.-L.W.; visualization: Y.-J.C.; supervision: J.-L.W.; project administration: J.-L.W. and C.-W.H.; funding acquisition: J.-L.W. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

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

References

1. Kar, I. Estonian Citizens Will Soon Have the World's Most Hack-Proof Health-Care Records. 2016. Available online: http://qz.com/628889/this-eastern-european-country-is-moving-its-health-recordsto-the-blockchain/ (accessed on 27 August 2020).
2. Kelly, J.; Williams, A. Forty Big Banks Test Blockchain-Based Bond Trading System. 2016. Available online: https://www.reuters.com/article/banking-blockchain-bonds/forty-big-banks-test-blockchain-based-bond-trading-system-idUSL8N16A30H (accessed on 27 August 2020).
3. Azaria, A.; Ekblaw, A.; Vieira, T.; Lippman, A. MedRec: Using blockchain for medical data access and permission management. In Proceedings of the 2016 2nd International Conference on Open and Big Data (OBD). Institute of Electrical and Electronics Engineers (IEEE), Vienna, Austria, 22–24 August 2016.
4. Lacey, S. The Energy Blockchain: How Bitcoin Could be a Catalyst for the Distributed Grid. 2016. Available online: http://www.greentechmedia.com/articles/read/the-energy-blockchain-could-bitcoin-be-a-catalyst-for-the-distributed-grid (accessed on 27 August 2020).
5. Christidis, K.; Devetsikiotis, M. Blockchains and smart contracts for the internet of things. *IEEE Access* **2016**, *4*, 2292–2303. [CrossRef]

6. Yao, A.C. Protocols for secure computations. In Proceedings of the IEEE 23rd Annual Symposium on Foundations of Computer Science (sfcs 1982), Chicago, IL, USA, 3–5 November 1982.
7. Rackoff, C.; Simon, D.R. Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack. In *Annual International Cryptology Conference*; Springer: Berlin/Heidelberg, Germany, 1991.
8. Armknecht, F.; Boyd, C.; Carr, C.; Gjosteen, K.; Jaschke, A.; Reuter, C.A.; Strand, M. A Guide to Fully Homomorphic Encryption. *IACR Cryptol. ePrint Arch.* **2015**, *2015*, 1192.
9. Nakamoto, S. *Bitcoin: A Peer-to-peer Electronic Cash System*; White Paper; 2008. Available online: https://git.dhimmel.com/bitcoin-whitepaper/ (accessed on 27 August 2020).
10. Double-Spending—Bitcoin WiKi, Mar. 2016. Available online: https://en.bitcoin.it/wiki/Double-spending (accessed on 27 August 2020).
11. Szabo, N. Smart Contracts. 1994. Available online: http://szabo.best.vwh.net/smart.contracts.html (accessed on 27 August 2020).
12. Andrychowicz, M.; Dziembowski, S.; Malinowski, D.; Mazurek, L. Secure Multiparty Computations on Bitcoin. In Proceedings of the 2014 IEEE Symposium on Security and Privacy, San Jose, CA, USA, 18–21 May 2014.
13. Pass, R.; Shelat, A. Micropayments for decentralized currencies. In Proceedings of the ACM 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, 12–16 October 2015.
14. Kumaresan, R.; Bentov, I. How to Use Bitcoin to Incentivize Correct Computations. In Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, 3–7 November 2014.
15. Paillier, P. Public-key crypto systems based on composite degree residuosity classes. *Eurocrypt* **1999**, *99*, 223–238
16. Pieter, W. Hierarchical Deterministic Wallets (BIP:32). Available online: https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki (accessed on 27 August 2020).
17. Maxwell, G. Confidential Transactions. Available online: https://elementsproject.org/features/confidential-transactions (accessed on 27 August 2020).
18. Maxwell, G. CoinJoin. Bitcoin Forum. Available online: https://bitcointalk.org/index.php?topic=279249 (accessed on 27 August 2020).
19. Heilman, E.; Alshenibr, L.; Baldimtsi, F.; Scafuro, A.; Goldberg, S. *Tumblebit: An Untrusted Bitcoin-Compatible Anonymous Payment Hub*; Network and Distributed System Security Symposium: San Diego, CA, USA, 2017.
20. Poelstra, A. Mimblewimble: Non-Interactive CoinJoin and Better Scaling Properties Using Confidential Transactions. 2016. Available online: https://www.reddit.com/r/Bitcoin/comments/4vub3y/\mimblewimble_noninteractive_coinjoin_and_better/d61n7yd/ (accessed on 27 August 2020).
21. Van Saberhagen, N. CrytoNote. 2013. Available online: https://cryptonote.org/whitepaper.pdf (accessed on 27 August 2020).
22. Sasson, E.B.; Chiesa, A.; Garman, C.; Green, M.; Miers, I.; Tromer, E.; Virza, M. Zerocash: Decentralized anonymous payments from bitcoin. In Proceedings of the 2014 IEEE Symposium on Security and Privacy, San Jose, CA, USA, 18–21 May 2014.
23. Zyskindetal, G. Decentralizing privacy: Using blockchain to protect personal data. In Proceedings of the IEEE Security and Privacy Workshops (SPW), San Jose, CA, USA, 21–22 May 2015; pp. 180–184.
24. Delgado-Mohatar, O.; Fierrez, J.; Tolosana, R.; Vera-Rodriguez, R. Biometric Template Storage with Blockchain: A First Look into Cost and Performance Tradeoffs. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops, Long Beach, CA, USA, 16–17 June 2019.
25. White Paper—Ethereum/WiKi, Mar. 2016. Available online: https://github.com/ethereum/wiki/wiki/White-Paper (accessed on 27 August 2020).
26. LevelDB, Google Inc. 2011. Available online: https://github.com/google/leveldb (accessed on 27 August 2020).
27. Kosba, A.; Miller, A.; Shi, E.; Wen, Z.; Papamanthou, C. Hawk: The blockchain model of cryptography and privacy preserving smart contracts. In Proceedings of the 2016 IEEE Symposium on Security and Privacy (SP), San Jose, CA, USA, 22–26 May 2016.

28. Goldwasser, S.; Micali, S. Probabilistic encryption & how to play mental poker keeping secret all partial information. In Proceedings of the ACM Fourteenth Annual ACM Symposium on Theory of Computing, San Francisco, CA, USA, 5–7 May 1982.
29. Bost, R.; Popa, R.A.; Tu, S.; Goldwasser, S. *Machine Learning Classification over Encrypted Data*; NDSS: New York, NY, USA, 2015.