2018 42nd IEEE International Conference on Computer Software & Applications

Storage Protocol for Securing Blockchain Transparency

Ryosuke Abe
Graduate School of Media and Governance
Keio University
Fujisawa, Kanagawa, Japan
Email: chike@sfc.wide.ad.jp

Hiroki Watanabe, Shigenori Ohashi,
Shigeru Fujimura, Atsushi Nakadaira
NTT Service Evolution Labs., NTT Corp.
Yokosuka, Kanagawa, Japan
Email: hiroki.watanabe.eh@hco.ntt.co.jp

***Abstract*—There are several studies that propose identity management that utilizes blockchain technology. In Ethereum, the main programmable blockchain system, data on a blockchain are saved as encoded binaries for executing automatic verification. To decode a binary, users need to know the application binary interfaces (ABI) that describes the data structure of the registered information. However, the manner by which ABI are shared in the current blockchain protocol is opaque. To resolve this problem, we describe a new protocol for embedding the ABI on a blockchain transaction when a registrant registers information. Our method enables users to read registered data with information on the blockchain alone, and it guarantees transparency without requiring users to trust third parties.**

I. Introduction

Blockchain was invented as a core technology for a peer-to-peer digital payment system known as bitcoin [1] by "Satoshi Nakamoto"[1] in 2009. Bitcoin is a peer-to-peer system that makes a distributed public ledger. This ledger is called a blockchain. Blockchains are characterized by their high resistance to tampering without requiring a trusted third party.

There are several studies that propose identity management that utilizes blockchain technology (e.g., personal data access [2], IoT [3], PKI [4], and medical data access [5]). These studies adapt programmable blockchain systems mainly as access controllers for data. For example, in one study on medical data access, a doctor registers information about a patient on a server, and the patient and pharmacy can read the information in accordance with permission information registered on a blockchain. Some information (e.g., psychotherapy notes) on the server should not be accessible to the pharmacy. Whenever users (e.g., patients or a pharmacy) want to check whether the access permission is set correctly, they need to be able to freely read the information on the blockchain. That is, there must be nothing restricting patients from knowing who can access their identity information. Therefore, it is important to secure the transparency of the data on the blockchain.

In Ethereum [6], the main programmable blockchain system, data on the blockchain are saved as encoded binaries for executing automatic verification. The binary information is not readable to human users, and guessing it is difficult. To decode a binary, users need to know the data structure of the registered information. This results in a transparency problem because information on blockchains is not easily readable for users.

[1]Anonymous individuals or groups who go by the name "Satoshi Nakamoto".

The data-structure information is generated as an application binary interface (ABI) when the code of a smart contract is compiled. As a result, only the person who registers the information knows the ABI. To read information in the blockchain, users need to get the ABI or an information-browsing application that includes the ABI from the registrant. If the registrant is malicious, or the application has tampered or has injected while distributed, there is a high probability that the application response information is false. For users to trust the information on a blockchain, they need to be able to trust the application.

In this paper, we describe a method for embedding the data-structure information on the blockchain when a registrant registers information. Information on a blockchain is not rewritten after being registered. Therefore, our method enables users to read information with information on the blockchain alone. Users are thus able to trust the information.

Our work contributes to enabling secure, transparent access to data on a blockchain. When users (e.g., patients or a pharmacy) access data on the blockchain by current methods, they need registrants (e.g., a doctor) to tell give information to decode the data. Therefore, the trustworthiness of a blockchain's readability is contingent on the trustworthiness of a registrant or a browsing application. In studies of blockchain-based applications, it is important that there is no required trusted third party. With our method, it is possible to verify information got via browsing application by referring data on the blockchain and to structure systems without requiring trusted third parties for readability.

II. Background

A. Blockchain Technology

Blockchain was invented as a core technology of a peer-to-peer digital payment system known as bitcoin. In bitcoin, blockchains are used to detect the "double-spending of money [1]." The data recording that someone paid a bitcoin to another person is saved in the bitcoin blockchain. This data is called a transaction. When a node gets a new transaction, a different node verifies that the transaction is not a "double-spending of money" by referencing the blockchain. A transaction includes a digital signature, so each node can verify who made each transaction with that transaction's signature.

A transaction is saved in a blockchain as follows. Each node can make a piece of data called a block that includes some transactions. As Figure 1 shows, a block holds the hash number of a previous block, forming a sort of chain. Each time a node gets a new block, first, the node verifies it; if it passes verification, the block is then added to the local blockchain. Each block holds the cryptographic hash of the previous block. Hence, by checking all the hashes in each block, one can determine that each block has not been rewritten since the previous block was added. If an attacker wants to rewrite a block generated in the past, the attacker needs to rewrite all the blocks from the one being targeted up to the newest one. In other words, the attacker needs to make blocks before other benign users do. By the consensus mechanism called Nakamoto consensus, this sort of attack would succeed only when the attacker has more than 50% of the computing power in that particular network. This is not realistic, so the blockchain has high tamper resistance.

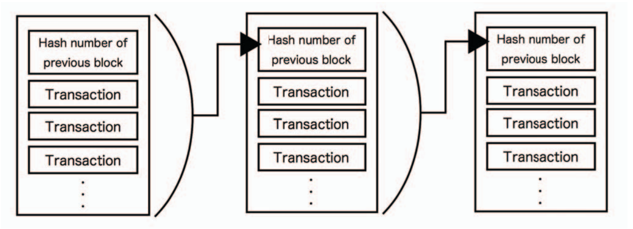

Fig. 1. Data structure of blockchain

B. Ethereum and Smart Contracts

There are blockchain systems with different characteristic features. For example, Ethereum [6] is one blockchain system that is able to execute programs called smart contracts. This program is able to register any data structures and update them. In Ethereum, some specialized programming languages (e.g., Solidity [7]) are used to write smart contracts. A smart contract is deployed in an Ethereum blockchain and can be executed by specifying its address. Information in an Ethereum blockchain is encoded into binary to be recognized by the Ethereum Virtual Machine (EVM). For example, a string is saved as ASCII code, and an integer is saved as a hex number. To read each piece of information, users need to decode them.

III. Access to Information on Ethereum Blockchain

In this section, we describe some methods for accessing the information on the Ethereum blockchain and discuss a problem with the current methods.

An overview of these methods is shown in Figure 2. Ethereum is one blockchain system mainly used in works described in section II. In Ethereum, a registrant is able to define the data structures of information in a smart contract's source code, and that information is dumped as an ABI when the code is compiled. The code is compiled and saved in a blockchain. The compiled source code is binary. When users access information on the blockchain, they need to get the ABI to decode it.

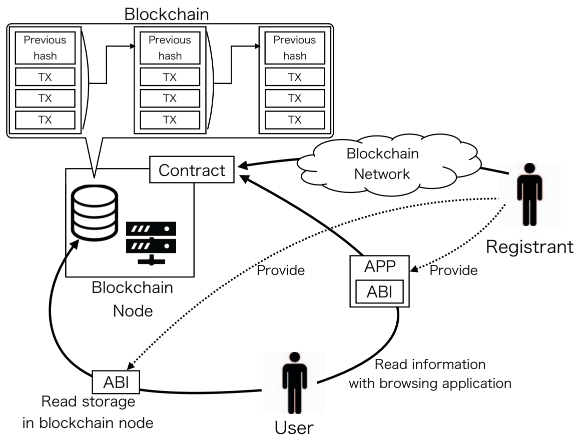

Fig. 2. Current information reading model

A. Browsing Applications

A registrant is able to define the methods for returning registered information in a smart contract. To access the smart contract's methods, users need the ABI that describes what methods and variables are defined in the source code. If the registrant provides a browsing application including the ABI for a particular smart contract, users are able to read information on the blockchain. However, there is no proof that the provided application's response is the correct information. For example, a registrant could implement an application wherein even if there is a different value in the blockchain, the application always responds with a specific value. The same situation happens when the application is tampered while distributed, or is injected. When users use a browsing application provided by a registrant, the users need to trust the registrant or the application.

B. Reading Ethereum Node Storage with API

Users are able to set up Ethereum nodes and read information from those nodes' storages. Ethereum nodes have key-value store (KVS) storage. Each key and value is 32 bytes. When a value is stored in the storage, the structure of the key and value is affected by a smart contract's source code. For example, integers are converted to a hexadecimal number and add zero padding so that the total size of the storage becomes 32 bytes. The key is defined by the value that defines the order in the smart contract's source code and by the type of the value. It is difficult to detect the type of the value from a binary key and value. Hence, users need an ABI to read information on the blockchain.

Geth, a widely used Ethereum client, has application programming interfaces (APIs) read self-storage.

1) eth.getStorageAt: The API responds to the value at a particular block number by posting an address of a smart contract, a key in the storage, and a block number. The key is defined by source code, so users need to know the source code or the ABI to use this API.

2) debug.storageRangeAt: The API responds to a set of values in storage at a particular block number by posting the address of a smart contract and a block number. It is different from *eth.getStorageAt* in that it does not need the key in the storage. In this manner, users are able to get information from the storage. However, the values are encoded, and it is difficult to guess the value types. Hence, users need to find out from the ABI what types of values are defined, among other information.

C. Problem with Access to Information on Ethereum Blockchain

The problem with the methods described in this section is that they require users to use information not on the blockchains to read the information on them. If the browsing application returns false information, users have no method to verify. If the ABI information has been tampered with, users cannot access data properly; in the case of identity management, it is possible that tampered ABIs could lead to identity theft.

IV. Proposed Method

To read information with a blockchain alone, we propose a method that embeds the data structures and the values' defining order into transactions. The current access methods described in section III require additional information to read information on the blockchain. In other hands, there is a high probability that additional information is different from information on blockchain. By our method, we can utilize the characteristics of the blockchain to guarantee readability, because it is difficult to rewrite information on the blockchain.

As a proof of concept, we implemented a smart contract that is able to embed data structures, values' defining order, and decoding-system data obtained from the storage of an Ethereum node.

A. Optimizing API for embedding (Modified ABI)

For embedding, we propose a modified ABI that shows only the types of values. From the viewpoint of storage cost, we considered embedding an entire ABI to be excessive for identity management. An ABI includes a function interface and the data type of values. In identity management, users check their permissions that are saved on a blockchain. In other words, they read static information on the blockchain.

In an Ethereum smart contract, a value defined first in the source code is saved as key 0 in the storage. Here, we describe the data structures and the values defining order within the value of key 0.

The sample Solidity source code is shown in Figure 3. The variable *def* defines data structures and the values' defining order. The variable *def* is defined as a 32-byte type, and every 2 bytes from the top byte show type definition and data length. Table I shows mapping data for types and signs. Figure 4 shows an example of a modified ABI for Figure 3. First, the address type is defined; the head of 2 bytes is 3. After that, if there is not requirement to define the address type, then there is zero padding until 2 bytes. Then, the address array is defined. The head of 2 bytes is 6, and after that, the type of the value is defined in the array. In the end, 32 bytes is defined by 4. This method of type definition is able to define 16 variables.

```
pragma solidity ^0.4.16;
contract Protocol {
  bytes32 public def;
  address public land;
  address[] public owner;
  bytes32 public data;
function Protocol(bytes32 _hash,bytes32 _def) {
  def = "0x300063004000000000000000000000..."
```

Fig. 3. Example of embedding data structures and the values defining order

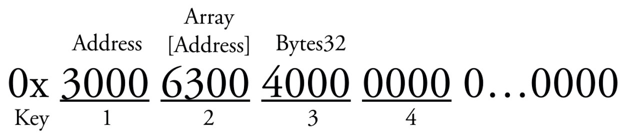

Fig. 4. Modified ABI

TABLE I
DATA TYPES AND SIGNS

data type	signs
Boolean	0
Uint	1
Int	2
Address	3
Bytes	4
String	5
Array	6
Mapping	7

B. Decoding System

We implemented a value-decoding system that is based on embedded information as a web application. With this system, users are able to decode information with only the rules of the modified ABI. This enables them to implement their own original decoding systems optimized for their purposes. We implemented this system on Ethereum API *debug.storageRangeAt* because users need an ABI or source code to decode data that are read via a browsing application and *eth.getStorageAt.*

Figure 5 shows a sequence of a decoding system. A registrant deploys a smart contract and tells a user its address. Users post the address, and range of block numbers. Then, the application shows decoded values that were obtained from the storage of an Ethereum node with the *debug.storageRangeAt*

API method. The system shows values at each block when each values changes. Thus, users are able to trace the state change of values using only information on the blockchain. By the system, users are able to verify a browsing application given by a registrant return data on a blockchain correctly.

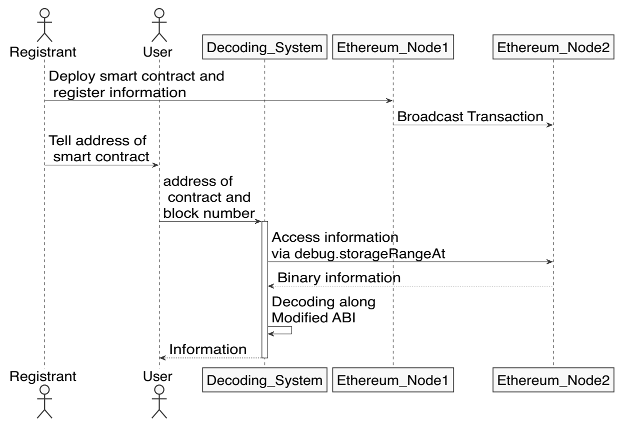
Registrant
User
Decoding_System
Ethereum_Node1
Ethereum_Node2
Deploy smart contract and register information
Broadcast Transaction
Tell address of smart contract
address of contract and block number
Access information via debug.storageRangeAt
Binary information
Decoding along Modified ABI
Information

Fig. 5. Sequence of decoding system

V. Discussion of Future Architecture

In this section, we discuss future blockchain architecture for more secure and open data management. Our prototype demonstrates that it is possible to access blockchain data without requiring a browsing application given by a registrant. This is possible by embedding an ABI into a blockchain transaction. Our method allows users to decode, read, and trace data easily from a smart contract's storage, where there is even less trust between users and registrants. However, if a guarantee of openness and validity of embedded ABIs is required more strictly, its implementation will require a new blockchain architecture. Specifically, the blockchain protocol itself should oblige registrants to embed an ABI into a transaction and include the process of verifying whether or not the registered data is decodable with the embedded ABI. Because all ABI (or, in our work, modified ABI) information is verified and shared on a blockchain protocol layer, anyone may refer to the registered data with a greater degree of trust. Figure 6 shows a conceptual comparison of this new architecture with the existing one.

VI. Related Works

Another approach for allowing users to know the data structure of a compiled smart contract is so-called reverse engineering. Bhargavan et al. [8] proposed a decompiler that translates compiled Solidity code to F* [9]. Amani et al. [10] proposed verification of the behavior of smart contracts using Isabelle/HOL. Porosity [11] is an open source decompiler of Solidity. These proposed methods help users decompile smart contracts from binary source code, but the reconstruction is not perfect. Definitions of values are lost in decompiled code, so it is hard for users to understand completely what registered data means. Therefore, our approach, which shares ABIs, is

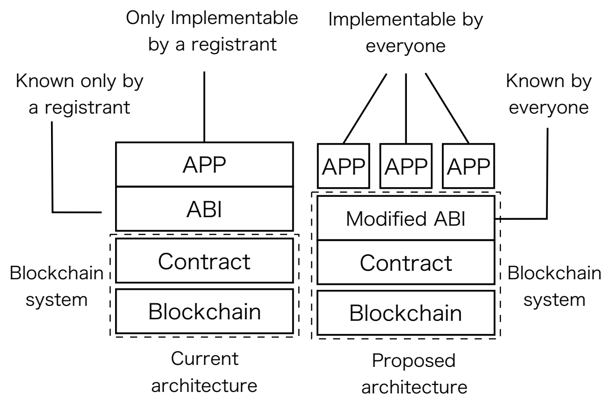

Fig. 6. Layers of application based on blockchain

more practical in terms of accurately securing information transparency.

VII. Conclusion

In this paper, we described a method for embedding the data structures and the values' defining order into transactions. In current blockchain-based systems, users must get information that is known only by a registrant to trace the data on the blockchain. The registrant tells users the information, but there is the possibility that the registrant has malicious intent or the browsing application given by the registrant is tampered while distributed. Identity-management systems must guarantee transparency without requiring users to trust a third party and must be able to verify that information is correct. With our proposed method, users of blockchain-based identity-management systems are able to trace data using only information on the blockchain. Information on the blockchain is difficult to overwrite. Thus, users can trust information in this trace process.

In our proposed method, the data size for a smart contract is larger. In a blockchain system, each node needs to hold the entirety of the blockchain. As a result, our proposed method makes the storage size for each node larger. As a subject for future study, we need to discuss the trade-off between transparency and scalability.

References

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[2] G. Zyskind, O. Nathan, and A. Pentland, "Decentralizing Privacy: Using Blockchain to Protect Personal Data," in *2015 IEEE Security and Privacy Workshops*. IEEE, may 2015, pp. 180–184.

[3] R. Rivera, J. G. Robledo, V. M. Larios, and J. M. Avalos, "How digital identity on blockchain can contribute in a smart city environment," in *2017 International Smart Cities Conference (ISC2)*. IEEE, sep 2017, pp. 1–4.

[4] M. Al-Bassam, "SCPKI: A Smart Contract-based PKI and Identity System," in *Proceedings of the ACM Workshop on Blockchain, Cryptocurrencies and Contracts - BCC '17*. New York, New York, USA: ACM Press, 2017, pp. 35–40.

[5] A. Azaria, A. Ekblaw, T. Vieira, and A. Lippman, “MedRec: Using Blockchain for Medical Data Access and Permission Management,” in *2016 2nd International Conference on Open and Big Data (OBD)*. IEEE, aug 2016, pp. 25–30.

[6] G. Wood, “Ethereum: A secure decentralised generalised transaction ledger,” pp. 1–32, 2014. [Online]. Available: http://www.cryptopapers.net/papers/ethereum-yellowpaper.pdf

[7] “Solidity.” [Online]. Available: http://solidity.readthedocs.io/en/latest/

[8] B. Karthikeyan, N. Swamy, S. Zanella-Béguelin, A. Delignat-Lavaud, and et el., “Formal Verification of Smart Contracts,” in *Proceedings of the 2016 ACM Workshop on Programming Languages and Analysis for Security - PLAS’16*. New York, New York, USA: ACM Press, 2016, pp. 91–96.

[9] N. Swamy, M. Kohlweiss, J.-K. Zinzindohoue, S. Zanella-Béguelin, and et el., “Dependent types and multi-monadic effects in F*,” in *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages - POPL 2016*, vol. 51, no. 1. New York, New York, USA: ACM Press, 2016, pp. 256–270.

[10] S. Amani, M. Bégel, M. Bortin, and M. Staples, “Towards verifying ethereum smart contract bytecode in Isabelle/HOL,” in *Proceedings of the 7th ACM SIGPLAN International Conference on Certified Programs and Proofs - CPP 2018*. New York, New York, USA: ACM Press, 2018, pp. 66–77.

[11] M. Suiche, “Porosity: A decompiler for blockchain-based smart contracts bytecode,” 2017. [Online]. Available: https://www.comae.io/reports/dc25-msuiche-Porosity-Decompiling-Ethereum-Smart-Contracts-wp.pdf