Received January 10, 2018, accepted February 26, 2018, date of publication March 7, 2018, date of current version March 19, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2812844

RBAC-SC: Role-Based Access Control Using Smart Contract

JASON PAUL CRUZ[1], (Member, IEEE), YUICHI KAJI[2], (Member, IEEE), AND NAOTO YANAI[1]
[1]Graduate School of Information Science and Technology, Osaka University, Suita 565-0871, Japan
[2]Information Strategy Office, Nagoya University, Nagoya 464-8601, Japan

Corresponding author: Jason Paul Cruz (jpmcruz@ymail.com)

ABSTRACT The role-based access control (RBAC) framework is a mechanism that describes the access control principle. As a common interaction, an organization provides a service to a user who owns a certain role that was issued by a different organization. Such trans-organizational RBAC is common in face-to-face communication but not in a computer network, because it is difficult to establish both the security that prohibits the malicious impersonation of roles and the flexibility that allows small organizations to participate and users to fully control their own roles. In this paper, we present an RBAC using smart contract (RBAC-SC), a platform that makes use of Ethereum's smart contract technology to realize a trans-organizational utilization of roles. Ethereum is an open blockchain platform that is designed to be secure, adaptable, and flexible. It pioneered smart contracts, which are decentralized applications that serve as "autonomous agents" running exactly as programmed and are deployed on a blockchain. The RBAC-SC uses smart contracts and blockchain technology as versatile infrastructures to represent the trust and endorsement relationship that are essential in the RBAC and to realize a challenge-response authentication protocol that verifies a user's ownership of roles. We describe the RBAC-SC framework, which is composed of two main parts, namely, the smart contract and the challenge-response protocol, and present a performance analysis. A prototype of the smart contract is created and deployed on Ethereum's Testnet blockchain, and the source code is publicly available.

INDEX TERMS Blockchain technology, role-based access control, smart contracts.

I. INTRODUCTION

Roles and titles are often used to distinguish the eligibility of users to access certain services. Such mechanism is modeled as the *role-based access control (RBAC)* [1] framework, which describes the access control relation among users and services. In RBAC, users are associated with roles, and roles are associated with services. Many organizations and companies use such framework in their computer systems to implement their internal access control requirements. For example, programmers in a company have access to both the backend and frontend source codes, whereas quality assurance personnel only have access to the frontend source codes. This access control is commonly used within an organization, but it must be noted that RBAC is a versatile framework; that is, roles are often used in a trans-organizational manner. For example, students are often allowed to purchase books at discounted prices. In this example, the "student" role that is issued by an organization (university) is used by another organization (book shop) to determine if a user is eligible to receive a certain service (discounted price). Such trans-organizational use of roles is common in face-to-face communication, but it **is not obvious in a computer network**.

To realize a trans-organizational RBAC mechanism in a computer network, users **should not be able to disguise their roles**; that is, they should not be able to use roles they do not own. In face-to-face communication, this role disguise is prevented with the use of physical certificates, such as identification (ID) cards and passports, which are expected to be difficult to forge or alter (especially that the photo in the certificates can be matched with the person's face). In a computer network, however, role disguise is not a trivial problem. Even if a user has a certain role (student role) that is issued by an organization (university), he/she has no systematic way of convincing another organization (book shop) that he/she really owns that role. An online book shop may, for example, ask for a scanned copy of a physical certificate (student ID card) as proof of the asserted student role, but this process can be insecure, time consuming, and can easily be falsified. Digital certificates [2] can be utilized as an analogue of physical certificates, but the use of digital certificates is not favorable because it requires a secure public-key infrastructure (PKI). PKI-based systems are costly to setup and maintain, and they can be insecure (e.g., DigiNotar [3]). Another less sophisticated approach to the security problem is for a service-providing organization (book shop) to contact a role-issuing organization (university) about the user-role assignments. This approach works in some cases [4], but it requires the agreed beneficial relationship among organizations. Consequently, a third-party organization will not be able to join the partnership if it does not provide any benefit to the other organizations involved, severely restricting the trans-organizational utilization of roles.

Therefore, in this paper, we ask the following questions: **How can we effectively realize an RBAC mechanism in a trans-organizational manner?** and **How can we verify the user-role assignments of organizations in a secure manner?** Following these questions, how can we provide flexibility to users to fully control their roles and to organizations to effectively manage their user-role assignments? What other benefits can such trans-organizational RBAC mechanism provide?

We propose a role-based access control using smart contract (*RBAC-SC*), which realizes a trans-organizational RBAC mechanism using *blockchain technology* and *smart contracts*. Blockchain technology, which was pioneered by Bitcoin [5], was originally intended to enable a payment system and complete digital money that is secure and decentralized; that is, it is a peer-to-peer network powered by its users and with no central authority. As Bitcoin and its blockchain technology began attracting attention, alternative blockchain-based cryptocurrencies have emerged, the biggest of which are Ethereum [6], Litecoin [7], and Dogecoin [8]. In this paper, we have implemented a prototype of the smart contract of RBAC-SC using Ethereum, an open blockchain platform that allows the deployment of smart contracts, which are self-executing scripts triggered by messages and transactions. The smart contract of RBAC-SC was written in the Solidity programming language [9] and was deployed and tested on the Ropsten Testnet of Ethereum [10]. See Appendix for details about the deployed smart contract of RBAC-SC.

In this paper, we aim to realize a trans-organizational RBAC mechanism that is secure (users cannot disguise roles and only authorized entities can execute functions), user-oriented (users can disclose their roles to any organization), verifiable (anyone can verify if a user has a certain role that is managed and issued by another organization), extendable (users can endorse other users), and manageable (roles can be modified and revoked). The key ideas are to publish all relevant information of user-role assignments in a smart contract deployed on a blockchain and to employ a challenge-response authentication protocol for verifying if a user owns an asserted role. Ethereum's protocols and cryptography make the proposed system suitable for the trans-organizational utilization and authentication of roles.

The contributions of this paper are as follows:

1. We present the design and framework of the RBAC-SC and provide a performance analysis.
2. We implement a smart contract prototype of the RBAC-SC, which is deployed on the Testnet blockchain of Ethereum, and publish the source code on GitHub (https://github.com/jpmcruz/RBAC-SC).
3. We discuss other related mechanisms based on PKI, smart contracts, and the Bitcoin blockchain and compare them with the RBAC-SC.

The rest of the paper is organized as follows: The models for RBAC, research goals, desired properties, and adversarial model are presented in Section II. A background of the Ethereum protocols is presented in Section III. The proposed RBAC-SC is discussed in Section IV. Analysis and evaluation of the prototype of RBAC-SC are presented in Section V. Related works on RBAC are discussed in Section VI. Finally, the conclusion and future direction are presented in Section VII.

II. MODELS FOR RBAC AND PROBLEM DEFINITION

In the simplest model of RBAC [1], the access structure is defined by three sets (set U of *users*, set R of *roles*, and set S of *services*) and two relations (*user-role assignment* UA $\subset U \times R$ and *role-service assignment* SA $\subset R \times S$). A user u is eligible to access a service s if and only if there is a role r such that $(u, r) \in$ UA and $(r, s) \in$ SA. Roles assigned to users can be used to access services in a trans-organizational manner. A *service-providing entity* can consult a *role-providing entity* about a role it issued to determine if a service should be given to an unknown guest. The role-providing entity is not always concerned about the service-providing entities, that is, a service-providing entity is not always allowed access to the user-role assignments of the role-providing entity. Therefore, the service-providing entity needs to use an alternative means to confirm if an unknown guest legitimately owns a certain role or not. In such framework, we extend the basic model of RBAC by introducing a set of organizations.

The *trans-organizational RBAC* is defined similarly to the usual RBAC, but a set O of *organizations* is defined in addition to the sets of users, roles, and services. Furthermore, the set R of roles is partitioned into several subsets, with each subset of R associated with an element in O, that is, $R = R_{o_1} \bigcup \ldots \bigcup R_{o_n}$, where $o_1, \ldots, o_n \in O$ and $R_{o_i} \bigcap R_{o_j} = \phi$ if $i \neq j$. To make the relation among roles and organizations explicit, a role r in R_{o_1} is written as $o_1.r$. Similarly, the user-role assignment UA is partitioned into disjoint subsets; $\mathrm{UA} = \mathrm{UA}_{o_1} \bigcup \ldots \bigcup \mathrm{UA}_{o_n}$, where $\mathrm{UA}_{o_i} \subset U \times R_{o_i}$. Obviously, $o_1.r \in R_{o_i}$ means that the role $o_1.r$ is managed by the organization o_i and the assignment of users to $o_1.r$ is controlled by that organization o_i.

In the trans-organizational RBAC, a user u requests a service s by asserting the role $o_1.r \in R_{o_i}$ that has been issued by a role-providing entity (organization) o_1. The service-providing organization provides the service to the user if and only if $(u, o_i.r) \in \mathrm{UA}_{o_i}$ and $(o_i.r, s) \in$ SA. Note that the service-providing organization can easily verify $(o_i.r, s) \in \mathrm{SA}$ because it defined SA. On the other hand, it cannot trivially verify $(u, o_i.r) \in \mathrm{UA}_{o_i}$, which is sometimes called an *authentication*, because another organization defined UA_{o_i}.

A. PROBLEM STATEMENT

In a trans-organizational RBAC, the verification of the authenticity of roles in a user-role assignment is important. For example, a service-providing organization should be able to verify if a user is the rightful owner of the asserted role (and that the role was issued by the corresponding role-issuing organization). Without this verification process, the RBAC will be insecure and result in unfair and unreliable access control. In face-to-face communication, the verification problem is naturally handled with the use of physical certificates, such as ID cards and passports. However, these physical certificates cannot be easily imported to the digitalized world over a computer network. Digital certificates have been studied for the replacement of physical certificates [2], but they are not widely implemented because of the cost issues for acquiring these certificates, keeping related keys secure, and maintaining a PKI [11], [12] that should be secure and always available. A less sophisticated but simpler approach is for the role-providing organizations and service-providing organizations to create a partnership or mutual agreement. However, such a framework will be semi-closed and only include organizations that share reciprocal benefits. For example, different universities may create a partnership, but a restaurant will not be able to join such partnership because it does not provide any benefit to the universities.

In this paper, we mainly focus on the realization of an RBAC mechanism in a trans-organizational scenario and on the secure verification of a user's ownership of roles. To achieve these goals, the proposed system should have the following properties:

1. **Issuance**: Role-issuing organizations can issue roles and other related information, such as expiration dates and personalizations if needed, to users.
2. **Management**: Role-issuing organizations can manage and modify information as needed in a transparent manner.
3. **Revocation**: Role-issuing organizations can revoke the roles issued to users if needed.
4. **Endorsement**: Users can endorse other users.
5. **Verification**: Any entity can verify the user-role assignment through a challenge-response protocol.
6. **Transparency**: All actions (functions executed) performed in the smart contract are recorded and any entity can audit these actions.
7. **Restriction**: An entity can only perform specific actions and cannot perform actions on behalf of other entities or as other entities.

B. ADVERSARIAL MODEL

An adversary can try to act as a role-issuing organization or a user. Acting as a role-issuing organization, the adversary's goal is to issue, modify, and/or revoke any role/information it wants to a user on behalf of another organization. As a user, the adversary's goal is to disguise its role, if it has any; that is, it asserts and tries to prove ownership of a role it does not own. We assume that the adversary cannot break standard cryptographic primitives used in the proposed system, such as finding hash collisions or forging digital signatures. Moreover, the adversary cannot compromise the private keys of the entities in the system (unless the entities knowingly leak them to the adversary). Finally, given that the proposed system is based on the blockchain technology, the adversary is assumed to not control a majority of the hashing power participating in the blockchain network to prevent the double-spending attack [13] and the 51% attack [14].

III. BLOCKCHAIN TECHNOLOGY AND ETHEREUM

Blockchain technology was introduced to the world by Bitcoin, which is a decentralized global currency cryptosystem that has increased in value and popularity since its inception by Satoshi Nakamoto in 2008 [5]. In the case of Bitcoin, the blockchain enables a payment system and complete digital money that is secure and decentralized; that is, it is a peer-to-peer network powered by its users and with no central authority. This allows a distributed computing architecture where the transactions are publicly announced and the participants agree on a single history of these transactions (or some kind of ledger). The transactions are grouped into blocks, given timestamps, and then published. The hash of each block includes the hash of the previous block to form a chain, making published blocks difficult to alter.

As Bitcoin began attracting attention, developers have taken advantage of the features of blockchain technology as an infrastructure to create their own platforms (aside from the main use of blockchain in facilitating the transfer of digital currency in Bitcoin). On the one hand, some platforms use the Bitcoin network as infrastructure for notarization or proof of existence of digital files, crowdfunding, dispute mediation, and spam control, among others. On the other hand, some platforms have emerged and took the form of "alt coins", which are alternative blockchain-based cryptocurrencies that aim to improve the capabilities of Bitcoin (or lack thereof) by implementing their own features and capabilities. The "improvements" can come in the form of a different proof-of-work algorithm (to shorten the verification time of transactions) or different hashing algorithm. There are almost a 1,000 alt coins, but the biggest ones that have attracted a following and attention are Ethereum [6], Litecoin [7], and Dogecoin [8]. In this paper, we are going to use Ethereum.

A. ETHEREUM

In 2013, Ethereum was proposed by Vitalik Buterin to create a blockchain-based distributed computing platform with the capability of building and running decentralized applications or *smart contracts* [6], [15]. The development of Ethereum was successfully funded by an online crowdsale in mid-2014, and the platform went live in 2015. Since then, Ethereum has received significant attention and is a pioneer in blockchain 2.0, which is the next-generation cryptoledger space. As of October 2017, Ethereum has a market capitalization of 28.3 billion USD, market price per ether (ETH) of approximately 300 USD, and on average, 300,000 transactions daily [16]. As a blockchain-based cryptocurrency, it offers the same features as Bitcoin of easy mobile payments, reliability, full control of one's own money, high availability, fast international payments, zero or low fees, protected identity, and privacy. Ethereum, however, offers more than enabling online transfer of digital money; **it enables its users to build and deploy smart contracts**.

Ethereum is composed of most of the protocols that other cryptocurrencies, like Bitcoin, also use. For example, Ethereum also includes a peer-to-peer protocol for the blockchain, and the blockchain is managed and kept secure by nodes in the network. In addition to these protocols, the main modification and innovation of Ethereum is being a programmable blockchain, i.e., it allows its users to create, deploy, and run decentralized applications on the blockchain.

B. ETHEREUM VIRTUAL MACHINE

At the center of Ethereum is the *Ethereum Virtual Machine (EVM)*, which can execute codes of arbitrary algorithmic complexity. Therefore, applications that are created using known programming languages, such as Javascript, can be run on the EVM. To facilitate the execution of codes in the blockchain and to maintain consensus, the nodes of the network run the EVM and execute the same instructions. Computations in the EVM are payed in *ether* (ETH), which is the currency used in Ethereum.

C. ETHEREUM ACCOUNTS

Ethereum's basic unit is the *account*. Ethereum uses two types of accounts: Externally Owned Account (EOA) and Contract Account. An EOA is controlled by a corresponding private key, has an ether balance, can send transactions (transfer ether to another account or trigger a contract code), and does not have an associated code. Similar to a Bitcoin address, an EOA is in the form of random numbers and letters, e.g., 0x1d7073d23eF7490974aCb7C7F3bBD634dB3417a0, and therefore looks anonymous and can be shared publicly. A contract account (or simply called contract) has an ether balance and has an associated code. All actions in the blockchain are set in motion by the transactions created by EOAs. This means that the code in a contract is executed when it receives a transaction from an EOA, where the input parameters for the code execution are included in the transaction. Therefore, contracts can be considered as "autonomous agents" inside the EVM that execute a specific piece of code when "poked" by a transaction. Code execution in a contract can also be triggered by messages from other contracts (see the next subsection for detailed explanation on transactions and messages). In contrast to Bitcoin's script, a contract performs Turing-complete computations and is typically written using some high-level language, such as Solidity, Serpent, and Lisp Like Language. A contract's behavior is fully dependent on the its code and on the transactions sent to it and therefore offers the possibility for creating decentralized and trusted systems.

D. TRANSACTIONS AND MESSAGES

An Ethereum *transaction* is a signed data package that stores a message from an EOA to another account on the blockchain. A transaction contains the Ethereum address of the recipient, a signature that identifies the sender, the amount of ether being transferred, an optional data field, and startGas and gasPrice values. The startGas limits the maximum amount of gas the code execution triggered by the message can incur and the gasPrice is the amount in ether to be paid for one unit of gas consumed (see the next subsection for detailed explanation on gas). When users send transactions, they pay a small transaction fee in ether to the network. This fee protects the blockchain from malicious computational tasks, such as distributed denial-of-service (DDoS) attacks and infinite loops.

A *message* is a virtual object that can only be sent by a contract to another contract. A message contains the identity of the sender, the identity of the recipient, the amount of ether being transferred, input data, and a startGas value. Similar to a transaction, a message leads to the recipient account running its code. Therefore, contracts can have relationships with other contracts in exactly the same way an EOA can.

E. ETHER AND GAS

Ether (ETH) is Ethereum's native value token and is the currency of the network. The sender of a transaction needs to pay for the code it wants to execute, including computation and data storage. When a code in a contract is executed as a result of being triggered by a message or transaction, every node in the network executes this code. The cost of this execution is expressed in *gas*. Gas is purchased for ether from the miners that execute the code (miners are the nodes in the Ethereum network that receive, propagate, verify, and execute transactions). Gas and ether are decoupled because gas is supposed to be constant cost of network utilization, whereas ether, and currencies in general, is volatile. Therefore, even if the price of ether increases, the gas price in terms of ether of executing a function in a contract remains constant.

Every computational step that is executed in a contract or transaction requires gas, and each transaction includes a gas limit and a fee that it is willing to pay per gas. The price of the gas is decided by the miners, and miners have the choice of including the transaction and collecting the fee or not (similar to the transaction fee in Bitcoin, wherein miners can decide to get the fee or not). Ethereum clients automatically purchase gas for the ether specified by the sender as maximum expenditure for a transaction, and the excess gas not used by the transaction execution is returned to the sender in ether. Therefore, overspending on the gas is not an issue because the user will only be charged for the gas consumed by a transaction. Readers can refer to [17] to read more about gas.

F. MINING AND PROOF-OF-WORK

Transactions are grouped together in blocks, which are then added to the blockchain through the process called *mining*. The mining process uses a proof-of-work (PoW) system wherein miners all around the world use special software to solve mathematical problems. Blocks are connected and linked together to form a blockchain, where a new block is added to the block that came before it. Every block contains the hash of the previous block, and thus, creating a chain that connects the first block (genesis block) to the current block. The miner who solves a block is rewarded with ether (currently at 5 ETH), the cost of the gas used in the transactions that are mined, and an extra reward of 1/32 per uncle. Uncles are stale blocks with parents that are ancestors of the including block. Valid uncles are rewarded to increase the security of the network by neutralizing the effect of network lag on the dispersion of mining rewards.

The PoW algorithm used in Ethereum is called *Ethash* (a modified version of the Dagger-Hashimoto algorithm) and requires a ‘‘brute force’’ solution; that is, miners scan and test for a nonce to find a solution that is below a certain difficulty threshold. The difficulty is adjusted accordingly so that it takes approximately 15 seconds to find a valid nonce. The Ethash PoW is a memory-hard computational problem, that is, it is application-specific integrated circuit (ASIC) resistant and allows a more decentralized distribution of security (as compared to specialized hardware used by many mining pools that dominate the mining in Bitcoin).

The security of the blockchain relies on this PoW system, which inherently means that a block cannot be modified without redoing the work spent on it, including the work spent on blocks chained after it. Therefore, an attacker will be outpaced by honest miners as long as majority of the overall computation power participating in the Ethereum network are controlled by honest miners. In this case, a block recorded in the blockchain is almost impossible to modify.

IV. ROLE-BASED ACCESS CONTROL USING SMART CONTRACT (RBAC-SC)

A. OVERVIEW

In this paper, we investigate how to effectively realize an RBAC mechanism in a trans-organizational manner and verify the user-role assignments of organizations in a secure manner. To achieve these goals, we present RBAC-SC, a smart contract-based authentication mechanism that is suitable for the trans-organizational utilization of roles. Fig. 1 shows the overall structure of the proposed system. The RBAC-SC is composed of two main parts, a **smart contract** and a **challenge-response protocol**.

The smart contract (SC) is used for the creation of the user-role assignments, which are then published on the blockchain. The SC provides different functions for the effective, efficient, and secure creation of user-role assignments. The use of the blockchain allows for transparency in the created roles, while maintaining the anonymity of the users, and serves as the synchronization point for service-providing organizations to check the asserted roles. The SC is an efficient and secure programmable asset that runs exactly as programmed. All actions (executed functions) performed using the SC are published on the blockchain as well. The SC has the following features: 1) allow role-issuing organizations to issue roles (and other related information) to users, 2) allow role-issuing organizations to manage and modify information as needed in a transparent manner, 3) allow role-issuing organizations to revoke the roles issued to users if needed, and 4) allow users to endorse other users (and remove the endorsement as well).

The challenge-response protocol is used for the authentication of the ownership of roles and the verification of the user-role assignment. The protocol is composed of the following steps: 1) declaration of a user owning a role, 2) checking of the information related to the declaration, 3) challenging of the user, 4) response of the user to the challenge, and 5) verification of the response of the user. The challenge-response protocol is designed in such a way that different organizations do not need to interact with each other and/or create a partnership for it to function, and it can be performed offline as well.

B. EXAMPLE SCENARIO

Consider that a role-issuing organization, a university (A-university), would like to manage a ‘‘student’’ role for its students. First, it would create the RBAC-SC’s smart contract and publish it on the Ethereum blockchain. Using the smart contract, it can execute a function to add a user (student) into the system. A-university would include the student’s externally owned address (EOA), the role to be issued to the student, and some related information, such as expiration date of the role and personalizations. Then, the user requests a service from a service-providing organization, for example a restaurant, by asserting that he/she possesses the student role from A-university. Based on the assertion, the restaurant checks the smart contract created by A-university and verify all the information it needs from there. After checking all the details, the restaurant can verify through the challenge-response protocol if the unknown user has access to the corresponding EOA that received the role, which finally proves that the unknown user was indeed issued a student role by A-university. Note that the restaurant does not have to know anything about the role beforehand, and it does not have to make any contract or inquiry to A-university that issued the role to the student because the details it needs are published publicly and/or possessed by the user (details below).

C. INITIALIZATION

In the RBAC-SC, we assume that the role-issuing organizations and the endorsers (users who will endorse other users) are Ethereum users, whereas the users who will not endorse other users and the service-providing organizations may or may not be Ethereum users. An Ethereum user means

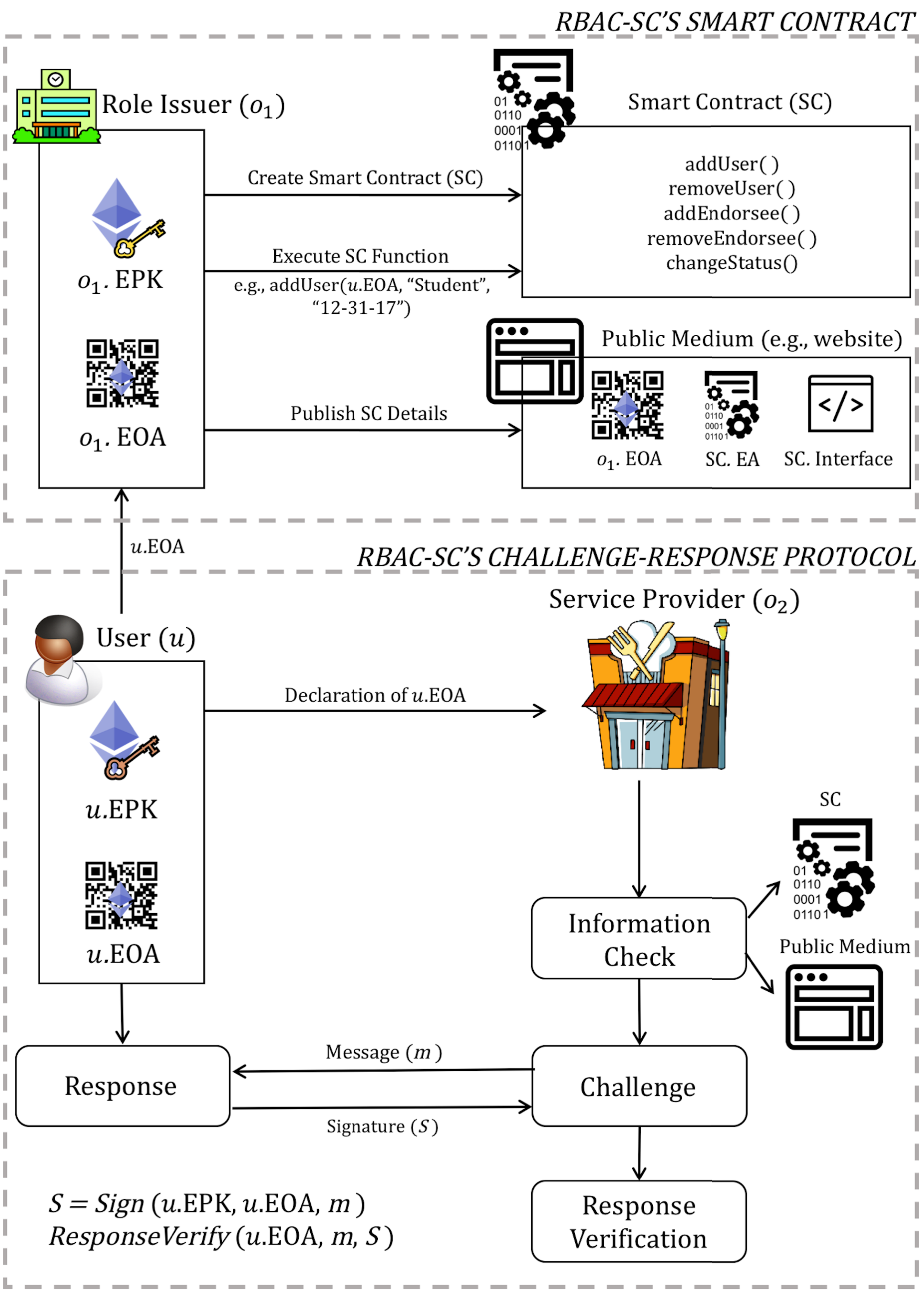


FIGURE 1. Overview of the RBAC-SC framework, which is composed of a smart contract and a challenge-response protocol.

that the entity knows how to create and publish smart contracts, execute the functions of smart contracts, and perform transactions.

To initialize the RBAC-SC, a role-issuing organization (o_1) generates a keypair of private key and the corresponding EOA (public key). This keypair will be used to create the smart contract (SC) and execute the functions of the SC. The creation of the keypair can be accomplished using several options, including Ethereum wallets and online/offline Ethereum address generators. We write o_1.EPK and o_1.EOA for the private key and the EOA of o_1, respectively.

Then, o_1 creates the SC that will handle all of the functions and serve as the decentralized database and synchronization system. After creating the SC, o_1 deploys the SC on the Ethereum blockchain under a corresponding smart contract address (SC.EA). Details about the SC can be accessed by checking the SC.EA in the blockchain, and its interface (SC.Interface) can be generated from the Contract JavaScript Object Notation (JSON) Interface.

Then, o_1 publishes o_1.EOA, SC.EA, and SC.Interface on media (e.g., website or public database) of its choice to make them available to the public. The publication of these information will serve as proof that o_1 owns and manages o_1.EOA and SC.EA (it should be noted that o_1 will not gain any benefit from publishing EOAs and smart contracts that it does not own, and thus, will only publish the EOA/s and smart contract/s it owns).

Similarly, a user (u) generates a keypair of a private key and EOA as u.EPK and u.EOA, respectively. Alternatively, o_1 can generate the (u.EPK, u.EOA) keypair and send it to u through a secure communication channel. Note, however, that it is recommended by the Ethereum community that only the one who created the keypair should be in possession of the keypair because the private key is used for accessing the ether stored in the corresponding EOA.

D. RBAC-SC's SMART CONTRACT

The code inside the SC offers the following functions:

addUser(u.EOA, u.role, u.notes): This function can only be executed by the SC owner or creator to add users in the SC and issue the corresponding roles and related information (Note: The restrictions on which entity can execute the function/s are realized by modifiers in the code, which will be explained in Section V-G). It takes as input the public key of the user (u.EOA), the role to be issued to u (u.role), and notes (u.notes), which could contain any other relevant information such as expiration date and personalizations. This function outputs the inputs, as well as timestamp of when the function was executed, and writes the outputs to the SC. Consequently, the SC is updated.

removeUser(u.EOA): This function can only be executed by the SC owner or creator to remove users from the SC and revoke their roles. It takes as input the pubic key of the user (u.EOA) and removes the user from the SC upon successful execution. Consequently, the SC is updated.

addEndorsee(eu.EOA, eu.notes): This function can only be executed by a user that has already been added in the SC to endorse another user (eu). It takes as input the public key of the endorsee (eu.EOA) and some notes (eu.notes), which could contain any other relevant information such as expiration date and personalizations. This function outputs the inputs, as well as the public key of the endorser (u.EOA) and the timestamp of when the function was executed, and writes the outputs to the SC. Consequently, the SC is updated.

removeEndorsee(eu.EOA): This function can only be executed by a user to remove an endorsee from the SC. An endorsee can only be removed by the user who endorsed him/her. It takes as input the public key of the endorsee (eu.EOA) and removes the endorsee from SC upon successful execution. Consequently, the SC is updated.

changeStatus(): This function can only be executed by the SC owner or creator to deactivate the SC. The SC will remain on the blockchain forever once it is deployed, and thus, the status is important to indicate if the SC is inactive and not supposed to be used anymore.

E. CHALLENGE-RESPONSE PROTOCOL

The challenge-response protocol is performed when a user u visits a service-providing organization (o_2) to request for a certain service corresponding to a role. The protocol is composed of the following steps:

Declaration: User u asserts that he/she owns an EOA that was issued a role by the role-issuing organization o_1. The asserted role corresponds to a service provided by o_2.

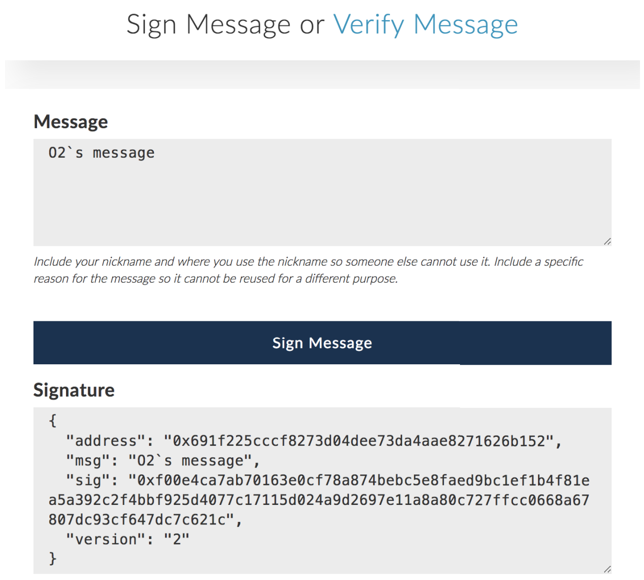
Information Check: Given the assertion of u, o_2 inquires for the EOA, say u.EOA, that was issued a role by o_1. To determine the information/details related to u.EOA, o_2 checks a medium where o_1 published the o_1.EOA, SC.EA, and SC.Interface it owns. Using these data, o_2 will be able to access the SC and check the information related to u.EOA, including u.role and u.notes. Given the data on the SC, o_2 is assured that the role u.role and other related information associated with u.EOA are assigned by o_1 to the owner of u.EOA. Now, o_2 can challenge u to verify if he/she is the genuine owner of u.EOA.

Challenge: Organization o_2 chooses an arbitrary data m and requests u to sign it.

Response: User u signs m together with u.EOA and the private key u.EPK. The signature is defined by $S = Sign(u.\text{EPK}, u.\text{EOA}, m)$, and thus a correct S can only be created if u has u.EPK. User u then sends S back to o_2.

Response Verification: After receiving the signature S from u, o_2 will verify using the function $ResponseVerify(u.\text{EOA}, m, S)$. If the verification is successful, then o_2 can offer the service to u.

Remark that o_2 can confirm if u has access to the role u.role without querying o_1, and that u has little chance to disguise his/her role. The challenge-response protocol can be performed offline, and the signing and verification functions of the Ethereum source code can be used as well. Using the Ethereum source code, a signature can be created by using the eth.sign(address, web3.sha3(message)) function. This allows the user to create a signature for a message without revealing the private key. Examples of the signing and verifying of a message to prove ownership of an EOA are shown in Fig. 2 and Fig. 3, respectively.



Sign Message or Verify Message

Message

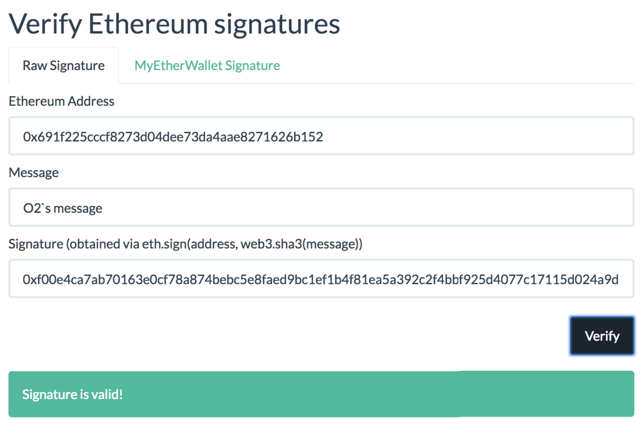
```
02`s message
```

Include your nickname and where you use the nickname so someone else cannot use it. Include a specific reason for the message so it cannot be reused for a different purpose.

Signature

```
{
  "address": "0x691f225cccf8273d04dee73da4aae8271626b152",
  "msg": "02`s message",
  "sig": "0xf00e4ca7ab70163e0cf78a874bebc5e8faed9bc1ef1b4f81e
a5a392c2f4bbf925d4077c17115d024a9d2697e11a8a80c727ffcc0668a67
807dc93cf647dc7c621c",
  "version": "2"
}
```

FIGURE 2. **Example of signing a message.**



Verify Ethereum signatures

Ethereum Address

0x691f225cccf8273d04dee73da4aae8271626b152

Message

O2`s message

Signature (obtained via eth.sign(address, web3.sha3(message))

0xf00e4ca7ab70163e0cf78a874bebc5e8faed9bc1ef1b4f81ea5a392c2f4bbf925d4077c17115d024a9d

Signature is valid!

FIGURE 3. **Example of verifying a signature.**

V. ANALYSIS AND EVALUATION

In this section, we analyze the framework and different features of the RBAC-SC, which provides an efficient and secure RBAC mechanism that can represent the trans-organizational usage of roles. Moreover, the use of blockchain technology offers additional features that are not common in traditional RBAC systems but can be useful.

A. ISSUANCE

In the RBAC-SC, the issuance of roles and other related information to users is performed efficiently and effectively by using the smart contract SC.

1) PERSONALIZATION

The relation between users and roles is represented by the ownership of the private keys that correspond to the EOAs (public keys) that received a role. This approach opens the possibility for security risks; the leakage and loss of keys (Note that these risks also exist for physical certificates used in face-to-face communication). The RBAC-SC has a personalization feature that deters the leakage of keys. The personalization can be directly attached to the user via the ‘notes’ part of the addUser() function. For example, a role-issuing organization can write in the notes “The role is given to student with ID# 123”. This personalization will make users more careful about leaking their private keys because a consequent investigation of a maliciously used role can be performed. Remember that blockchain technology allows for verifiability and thus presents a natural “traitor tracing” capability that can link accounts to users.

2) ROLE RE-ISSUANCE

If the private keys are lost or forgotten, or if access to the digital wallet is lost or forgotten, then control over the corresponding EOAs is also lost. The ownership of an EOA cannot be verified or proven without the corresponding unique private key. The RBAC-SC has a revocation feature to mitigate the loss of keys, wherein a role-issuing organization can simply delete the compromised EOA (via the removeUser() function) and re-issue the role to the new EOA of the user.

3) EXPIRATION DATES

The timestamp server of Ethereum provides a natural solution to the inclusion of expiration dates or validity of the roles in the proposed system. Role-issuing organizations can include an expiration date to the roles it will issue in the ‘notes’ part of the addUser() function. The service-providing organization can then check the validity of the asserted role according to the timestamp of when the role was issued to the user.

4) ACCOUNTS HANDLED

A role-issuing organization only needs to generate and handle one account/EOA, which is used for creating the smart contract and executing its functions. Similarly, a user only needs to generate and handle one account/EOA, which is used for receiving roles and for endorsing another user (and therefore, an endorsee only needs to generate and handle one account). The service-providing organization do not need to generate and handle an account/EOA.

B. MANAGEMENT

A role-issuing organization can easily update the information related to a user issued a role by using the addUser() function. Similarly, a user can easily update the information related to a user it endorsed using the addEndorsee() function. Even though the information can easily be updated, all the updates can be verified by anyone (see Transparency below) and will therefore be done in a responsible manner.

C. REVOCATION

Revocation can be achieved by executing the removeUser() function that will delete the user from the RBAC-SC. This feature is important in cases when the role of the user is expired. This will prevent the misuse of expired roles.

D. ENDORSEMENT

Endorsement among individuals is useful, and sometimes necessary, in certain scenarios. Semi-closed organizations, such as academic societies and golf clubs, have the tradition or policy that a new member must be referred or endorsed by a current member. Consider for example that Alice (u) is an authorized member of a golf club (o_1). This relationship is realized by the addition of Alice as a user in the *SC* created by o_1. If Alice would like to endorse Bob (eu) to o_1, then she can use the addEndorsee() function in the *SC* to endorse Bob. Then, Bob can go to o_1 and declare eu.EOA. Finally, o_1 can perform the challenge-response protocol to verify the assertion. Note that o_1 did not have to inquire Alice for the verification of the endorsement.

E. VERIFICATION

The RBAC-SC achieves the verification property effectively and efficiently. The challenge-response protocol is designed to securely verify user-role assignments and a user's ownership of roles. The challenge and response verification steps can be performed offline, and the transferring of the message and signature can be done efficiently using current technologies used in mobile devices, such as Quick Response (QR) codes [20] and Near Field Communication (NFC) [21].

F. TRANSPARENCY

The RBAC-SC achieves the transparency property because all functions that are executed in the *SC* are reflected on the events log of the *SC* and on the Ethereum blockchain. Therefore, an entity will not be able to perform a "secret" action without the other entities knowing. Moreover, entities will not be able to deny any action committed (as it is assumed that only the respective entities have control over their own private keys).

G. RESTRICTION

The RBAC-SC achieves the restriction property because the *SC* includes modifiers in its source code that will restrict the different entities to specific functions. For example, the *SC* has the "onlyOwner" modifier that allows only the creator of the contract (i.e., the role-issuing organization) to execute the addUser(), removeUser(), and changeStatus() functions. If a non-owner tries to execute these functions, the execution will fail and the action will not be recorded on the RBAC-SC and on the blockchain. The same restriction rule applies for the "onlyUser" modifier for the execution of the addEndorsee() and removeEndorsee() functions.

H. COSTS AND PRACTICALITY

A prototype of the smart contract of RBAC-SC was compiled and deployed on the Testnet of the Ethereum network. The costs of creation of the smart contract and the execution of the functions were analyzed. During the analysis, on October 2017, 1 ether ≈ 325 USD, and 1 gas ≈ 1 wei (0.000000001 eth) were used. 1 wei is the minimum gas value that can be used for a transaction. At the time of analysis, the average gas value was approximately 0.000000021 ETH. The lower the gas value, the longer the transaction will be validated by miners as miners can choose to ignore transactions with low gas value. However, for our purposes, the validation time is not vital and thus the minimum value is sufficient. Moreover, the difference in the time of confirmation (from 15 secs on average using the average gas value) was insignificant.

TABLE 1. Costs of the Different Functions in the *SC* Based on 1 Ether ≈ 325 USD and 1 gas ≈ 0.000000001 eth Rates

Function	Gas Used	Actual Tx Cost	USD
Create Contract	1813010	0.0.00181301	0.590
addUser (Address Only)	145590	0.00014559	0.047
addUser (Address + 20 bytes)	148152	0.000148152	0.048
addUser (Address + 40 bytes)	149432	0.000149432	0.049
addUser (Address + 60 bytes)	150712	0.000150712	0.049
addEndorsee	153531	0.000153531	0.050
removeUser	101819	0.000101819	0.033
removeEndorsee	39643	0.000039643	0.013

The costs of the execution of the different functions in the *SC* are shown in Table 1. As can be seen in the table, the highest cost corresponds to the creation and deployment of the *SC* on the blockchain at 0.590 USD. This cost, however, is only a one-time cost to setup and initialize the system. All the other functions have relatively low costs, with the costs of the addUser() and addEndorsee() at 0.05 USD on average. The costs of these functions will vary because of the variable input lengths (roles and notes). Based on the AddUser() function, each byte will cost 64 gas (0.00002 USD). The removeUser() and removeEndorsee() will be constant at 0.033 USD and 0.013 USD, respectively, as their inputs have constant size. It should be noted that the costs are based on the prototype deployed on the blockchain, and the costs can be lower with an optimized code. The costs can also be lowered further if the sizes of the inputs of the functions are minimal. Nevertheless, it can be argued that these costs will be significantly lower compared to buying/setting up and maintaining a private database.

I. FLEXIBILITY

The RBAC-SC provides role-issuing organizations with the flexibility to join or leave the system easily. Role-issuing organizations only need to follow the Ethereum protocol of the proposed system (create a smart contract and execute the functions) to participate in the system, and they can leave the system simply by retracting or disabling the media where they published the data related to their smart contract. Alternatively, the role-issuing organizations can deactivate their smart contract by updating the status field to "disabled" via the changeStatus() function.

J. ADDITIONAL SECURITY MEASURES

Ethereum users typically create many accounts and use digital wallets to store their public keys and the corresponding private keys. These wallets are the most common target of attacks, but of course, security measures have been implemented and are recommended to minimize such cases. The EOAs serve as the connection of the users to their corresponding roles. Therefore, the users are recommended to only use the EOAs for this purpose. Ultimately, a user only needs to store and secure the private key, which can also be kept offline for added security. An attacker with no prior knowledge of the RBAC-SC and the EOAs involved in the user-role assignments will have no motives or incentives to steal the private keys because the EOAs will not contain large amounts of ether.

VI. RELATED WORK

In this section, we discuss different systems related to RBAC. A couple of related works that make use of blockchain technology are limited, and the system in [18] is the only directly comparable system to RBAC-SC .

A. PKI-BASED SCHEMES

PKI-based and server-based systems are well-recognized frameworks utilized in many applications, such as OpenPGP (based on Pretty Good Privacy) and eXtended Markup Language (XML) [22]–[24]. However, it is often pointed out that, in these frameworks, the setup and service/labor costs become more expensive as the number of users increases. Even though implementation costs vary with the installation, common expenses, such as hardware for servers and backup devices, and personnel expenses for the design, setup, and maintenance of the PKI environment are necessary. For example, a 5,000-user PKI infrastructure can cost approximately 200,000 USD annually, with the initial setup cost amounting to 10,000 USD [25], [26]. These costs also apply for other systems that rely on a central authority, including Identity Providers (IdPs) which authenticate users on the Internet by using security tokens.

In addition to the complexity and costs of PKI-based systems, they are vulnerable to some security problems. In PKI-based systems, a Certificate Authority (CA) typically acts a trusted third party responsible for the distribution of management of digital certificates. The use of a CA creates a single point for failure and is therefore vulnerable to attacks. An example of an attack on CA is the DigiNotar incident [3], wherein the DigiNotar CA issued a rogue certificate for Google. Other attacks are caused by misconfigurations or unpatched software and the eventual attacks using quantum computers.

B. DECENTRALIZED ATTRIBUTE-BASED SCHEMES

To mitigate the risks of PKI-based systems, decentralized systems have been studied. Some of these systems are the decentralized multi-authority systems for attribute-based encryption (MA-ABE) and attribute-based signatures (MA-ABS) [27], [28]. The MA-ABE in [27] is decentralized but requires a trusted setup of common reference parameters. The MA-ABS in [28] is also decentralized and does not require a trusted setup, but it requires the setting of a public parameter for a prime order bilinear group and hash functions. Given these requirements, implementation and interoperability problems may arise if several groups of entities use different parameters. Therefore, the initialization and implementation of such systems require mutual agreement and consensus among all entities that will be involved. Consequently, a scheme for a role authentication mechanism that is secure, practical, and easy to set up has yet to be established. The MA-ABS and MA-ABE are relatively new technologies, and they can possibly be widely accepted in the future. However, as of writing this manuscript, the practical contribution of these techniques to existing network systems and the cost issues of their deployment are yet to be discussed.

C. SMART CONTRACT-BASED PKI

Recently, Al-Bassam created a smart contract-based PKI and Identity System (SCPKI) [29] that aims to detect the issuance of rogue certificates. Matsumoto and Reischuk presented IKP [30], which is a smart-contract based platform that automates responses to unauthorized certificates. The IKP prevents Certificate Authority (CA) misbehavior by receiving reports of unauthorized certificates and by incentivizing CAs for behaving correctly. The IKP was also deployed on the Ethereum blockchain to improve the Transport Layer Security (TLS) of PKI. The SCPKI and IKP provide solutions to the centralization problem of PKI-based system but are mainly aimed at preventing the issuance of rogue/unauthorized certificates.

D. BLOCKCHAIN-BASED RBAC

The authors did a preliminary work on using the Bitcoin network as an infrastructure to realize a trans-organizational RBAC mechanism. As an overview, the previous system in [18], [19] aims to provide an irrefutable proof of the role of a user (issued by an organization) by verifying the connection of the user to the organization through the Bitcoin blockchain. The connection is realized through the payment in Bitcoin, which is used to represent a relation of trust. The roles and other related information are included in the Bitcoin transactions, where the organization uses its own public Bitcoin address/es as input/s and the corresponding users' public Bitcoin address/es as output/s. Upon request for a service from an unknown user who asserts that he/she possesses a role from the said organization, a service-providing organization will verify the Bitcoin transaction containing the Bitcoin addresses of the organization and the user. After establishing the connection, the service-providing organization can verify through a challenge-response protocol if the unknown user has access to the output address in the transaction, which finally connects the role issued by the role-issuing organization to the unknown user. Compared with the RBAC-SC, this system is less efficient and has several disadvantages as follows:

1. A role-issuing organization is required to generate n Bitcoin addresses, where n is the number of roles that it wants to manage.

2. A role-issuing organization needs to create separate Bitcoin transactions to update the data of a user and include the overwritten data in a revocation list to ensure these will not be used in a malicious manner. The same process is used for the re-issuance of roles and deletion of users.

3. A role-issuing organization performs personalization by including unique identifiers to the data it will publish publicly, requiring it to generate a unique Bitcoin address for each personalization.

4. To endorse another user, a user creates a Bitcoin transaction to link their addresses. The service-providing organization verifies this endorsement by checking such connection, and then following the trail of connections back to the role-issuing organization. This can be inefficient and confusing to trace, especially if the addresses involved are included in many transactions. In RBAC-SC, such ''follow the trail'' is unnecessary.

5. A role-issuing organization can include expiration dates or validity of the roles it manages in the information it publishes publicly. In this way, a service-providing organization can verify the validity of a role by investigating the timestamp of the block where the transaction was included in the blockchain and comparing it with the details published by the role-issuing organization. Similar to the personalization of roles, multiple Bitcoin addresses are needed for different expiration dates, particularly if the dates are specific.

6. A role-issuing organization needs to publish *n* Bitcoin addresses, the corresponding roles, and other information related to the roles. These data can easily increase with the number of users, roles, expiration dates, and personalizations. In RBAC-SC, a role-issuing organization only needs to publish one EOA, the address of the smart contract, and the interface of the smart contract.

7. Performing functions via Bitcoin transactions is costly. Currently, the minimum Bitcoin transaction typically costs around 0.2 USD, and even this fee may not be accepted immediately, if at all (based on the rate provided by blockchain.info wallet for more than an hour of confirmation time). This can be caused by the increasing number of transactions per block, and therefore limited space. Consequently, miners are choosing transactions with higher fees.

VII. CONCLUSION AND FUTURE WORK

In this paper, we proposed the RBAC-SC, an RBAC mechanism that uses smart contract and blockchain technologies to realize the trans-organizational utilization of roles. The RBAC-SC provides a secure and efficient mechanism for the creation of user-role assignments and for the verification of a user's ownership of a role. Moreover, many collaborative rights management, such as personalization and endorsement, are naturally included. We described the RBAC-SC framework and presented a performance analysis. A prototype of the smart contract was created and deployed on the Testnet of Ethereum, and the source code was uploaded on GitHub. As future work, we will investigate a formal approach for the security analysis of RBAC-SC and create an optimized prototype to make the functions more efficient and minimize the costs.

APPENDIX

RBAC-SC's SMART CONTRACT

The complete code and the JSON interface of RBAC-SC's smart contract *SC* can be found here:

https://github.com/jpmcruz/RBAC-SC.

The *SC* was deployed on the Ropsten Testnet of Ethereum with the following address:

0xE8E6De0Fb1a3891BbDaB7554dc5076F407FeC0cc.

Using this address, the transactions can be seen at:

https://ropsten.etherscan.io/.

REFERENCES

[1] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman, ''Role-based access control models,'' *Computer*, vol. 29, no. 2, pp. 38–47, 1996.

[2] S. Farrell and R. Housley, *An Internet Attribute Certificate Profile for Authorization*, document RFC 5755, 2002.

[3] R. Charette, ''DigiNotar certificate authority breach crashes e-government in The Netherlands,'' *IEEE Spectr.*, 2011, accessed: Nov. 28, 2017. [Online]. Available: https://spectrum.ieee.org/riskfactor/telecom/security/diginotar-certificate-authority-breach-crashes-egovernment-in-the-netherlands

[4] Shibboleth. *Internet2*. Accessed: Nov. 28, 2017. [Online]. Available: http://shibboleth.internet2.edu

[5] S. Nakamoto. (2008). *Bitcoin: A Peer-to-Peer Electronic Cash System*. Accessed: Nov. 28, 2017. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[6] Ethereum. *Blockchain App Platform*. Accessed: Nov. 28, 2017. [Online]. Available: https://ethereum.org/

[7] *Litecoin*. Accessed: Nov. 28, 2017. [Online]. Available: https://litecoin.org/

[8] *Dogecoin*. Accessed: Nov. 28, 2017. [Online]. Available: http://dogecoin.com/

[9] *Solidity*. Accessed: Nov. 28, 2017. [Online]. Available: https://solidity.readthedocs.io/en/develop/

[10] Etherscan. *The Ethereum Block Explorer: ROPSTEN (Revival) TESTNET*. Accessed: Nov. 28, 2017. [Online]. Available: https://ropsten.etherscan.io/

[11] C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, and T. Ylonen, *SPKI Certificate Theory*, document RFC 2693, 1999.

[12] P. Gutmann, ''Simplifying public key management,'' *Computer*, vol. 37, no. 2, pp. 101–103, Feb. 2004.

[13] G. O. Karame, E. Androulaki, and S. Capkun, ''Double-spending fast payments in bitcoin,'' in *Proc. ACM Conf. Comput. Commun. Secur. (CCS)*, 2012, pp. 906–917.

[14] *51% Attack, Majority Hash Rate Attack*. Accessed: Nov. 28, 2017. [Online]. Available: https://bitcoin.org/en/glossary/51-percent-attack

[15] G. Wood. Ethereum: A Secure Decentralised Generalised Transaction Ledger. Yellow Paper. Accessed: Nov. 28, 2017. [Online]. Available: https://ethereum.github.io/yellowpaper/paper.pdf

[16] Etherscan. *Ethereum Charts & Statistics*. Accessed: Nov. 28, 2017. [Online]. Available: https://etherscan.io/charts

[17] *Ethereum Homestead Documentation*. Accessed: Nov. 28, 2017. [Online]. Available: https://ethereum-homestead.readthedocs.io/en/latest/index.html

[18] J. P. Cruz and Y. Kaji, ''The bitcoin network as platform for trans-organizational attribute authentication,'' *IPSJ Trans. Math. Model. Appl.*, vol. 9, no. 2, pp. 41–48, 2016.

[19] J. P. Cruz and Y. Kaji, ''The bitcoin network as platform for trans-organizational attribute authentication,'' in *Proc. 3rd Int. Conf. Building Exploring Web Based Environ. (WEB)*, 2015, pp. 29–36.

[20] *QRCode.Com*. Accessed: Nov. 28, 2017. [Online]. Available: http://www.qrcode.com/en/

[21] NFC. *Near Field Communication*. Accessed: Nov. 28, 2017. [Online]. Available: http://nearfieldcommunication.org/

[22] R. Perlman, ''An overview of PKI trust models,'' *IEEE Netw.*, vol. 13, no. 6, pp. 38–43, Nov. 1999.

[23] *OpenPGP*. Accessed: Nov. 28, 2017. [Online]. Available: http://www.openpgp.org/

[24] W3C. *XML Signature Syntax and Processing Version 1.1*. Accessed: Nov. 28, 2017. [Online]. Available: https://www.w3.org/TR/xmldsig-core/
[25] VeriSign. *Total Cost of Ownership for Public Key Infrastructure*. Accessed: Nov. 28, 2017. [Online]. Available: http://www.imaginar.org/sites/ecommerce/index_archivos/guias/G_tco.pdf
[26] A TC TrustCenter Whitepaper. *The Costs of Managed PKI*. Accessed: Nov. 28, 2017. [Online]. Available: https://azslide.com/the-costs-of-managed-pki_59892e9b1723dda4299be236.html
[27] A. Lewko and B. Waters, "Decentralizing attribute-based encryption," in *Proc. 30th Annu. Int. Conf. Theory Appl. Cryptograph. Techn., Adv. Cryptol. (EUROCRYPT)*, 2011, pp. 568–588.
[28] T. Okamoto and K. Takashima, "Decentralized attribute-based signatures," in *Public-Key Cryptography—PKC*. Berlin, Germany: Springer, 2013, pp. 125–142.
[29] M. Al-Bassam, "SCPKI: A smart contract-based PKI and identity system," in *Proc. ACM Workshop Blockchain, Cryptocurrencies Contracts (BCC)*, 2017, pp. 35–40.
[30] S. Matsumoto and R. M. Reischuk, "IKP: Turning a PKI around with decentralized automated incentives," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2017, pp. 410–426.

JASON PAUL CRUZ (M'13) was born in Manila, Philippines, in 1988. He received the B.S. degree in electronics and communications engineering and the M.S. degree in electronics engineering from Ateneo de Manila University, Quezon City, Philippines, in 2009 and 2011, respectively, and the Ph.D. degree in engineering from the Graduate School of Information Science, Nara Institute of Science and Technology, Nara, Japan, in 2017.

He is currently a specially appointed Assistant Professor at Osaka University, Suita, Japan. His current research interests include role-based access control, blockchain technology (Bitcoin and Ethereum), hash functions and algorithms, and Android programming.

YUICHI KAJI (M'97) was born in Osaka, Japan, in 1968. He received the B.E., M.E., and Ph.D. degrees in information and computer sciences from Osaka University, Suita, Japan, in 1991, 1992, and 1994, respectively.

In 1994, he joined the Graduate School of Information Science, Nara Institute of Science and Technology, Nara, Japan. In 2003 and 2004, he visited the University of California at Davis and the University of Hawaii at Manoa as a Visiting Researcher. He joined Nagoya University, Aichi, Japan, in 2016. His current research interests include the theory of error correcting codes, fundamental techniques for information security, and the theory of automata and rewriting systems. He is a member of IPSJ.

NAOTO YANAI received the B.Eng. degree from The National Institution of Academic Degrees and University Evaluation, Japan, in 2009, and the M.S.Eng. and Dr.Eng. degrees from the Graduate School of Systems and Information and Engineering, University of Tsukuba, in 2011 and 2014, respectively.

He is currently an Assistant Professor at Osaka University, Suita, Japan. His research interests are in the areas of cryptography and information security.