IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received March 24, 2020, accepted April 6, 2020, date of publication April 13, 2020, date of current version April 29, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2987608

A Blockchain-Assisted Trust Access Authentication System for Solid

TING CAI[1,2], ZETAO YANG[1], WUHUI CHEN[1], (Member, IEEE), ZIBIN ZHENG[1], (Senior Member, IEEE), AND YANG YU[1], (Member, IEEE)

[1]School of Data and Computer Science, Sun Yat-sen University, Guangzhou 510006, China
[2]College of Mobile Telecommunications, Chongqing University of Posts and Telecommunications, Chongqing 401520, China

Corresponding author: Wuhui Chen (chenwuh@mail.sysu.edu.cn)

This work was supported in part by the National Key Research and Development Plan under Grant 2018YFB1003803, in part by the National Natural Science Foundation of China under Grant 61802450 and Grant 61722214, in part by the Natural Science Foundation of Guangdong under Grant 2018A030313005, in part by the Program for Guangdong Introducing Innovative and Entrepreneurial Teams under Grant 2017ZT07X355, and in part by the Science and Technology Research Program of Chongqing Municipal Education Commission under Grant KJZD-K201802401.

ABSTRACT The Solid (Social Linked Data) project focuses on data sharing and privacy security and aims to build a decentralized ecosystem that radically changes the way web applications work today. Our goal is to introduce a ''trust access authentication system'' to achieve secure authentication and fine-grained access control, thereby promoting the implementation of Solid. Blockchain, equipped with multiple security properties and authentication functions, is a crucial technology. In this paper, we present a blockchain-assisted system for secure authentication in Solid and for implementation of fine-grained access control policies. Specifically, we explore to integrate threshold RSA signatures in a permissioned blockchain system to enable a fault-tolerant distributed signature scheme, thereby enhancing the resilience and robustness of authentication system. Moreover, we utilize smart contract to control transaction flows and manage access control policies automatically. Experimental results show that our proposed trust access authentication system enhances security, scales well, and is efficient and economically feasible.

INDEX TERMS Solid, blockchain, authentication, access control, smart contract, threshold signatures.

I. INTRODUCTION

Solid (Social Linked Data) is a web re-decentralization project led by Tim Berners-Lee, the inventor of the World Wide Web [1]. This project aims to build a decentralized ecosystem for current social web applications, where the data are completely owned by users themselves who can allow external applications how to use it for certain purposes such as social media [2]. This may response to our current pressing issues on social web applications, such as Twitter, Facebook, Wikipedia, Doodle, and many other online platforms that usually store user data in centralized repositories, consequently preventing users from controlling their data. For example, they cannot reuse their data across different storage services, or specify trusted authentication and personal access control mechanisms [3]. Thus, many centralization problems have attracted proposals for re-decentralizing the web; some examples are WebBox, Diaspora, and Musubi, but none of them has been widely adopted [4]. In this context, efforts to promote Solid as a high-potential project are of great significance.

The Solid ecosystem consists of four main layers, namely, *Linked Data Resources*, *Decentralized Web*, *Pod Servers*, and *Solid Applications*, as shown in Fig. 1. The *Linked Data Resources* are owned by users and can be stored in personal online datastores (pods) via the *Decentralized Web*. To access *Linked Data Resources*, the *Solid Applications* need to obtain an identity profile from the *Pod Servers*, and then access data in the user's pod. The users control their data in pods through using a resource description framework (RDF) profile stored in the pod [5], [6].

Data sharing and privacy security is one of most important concerns associated with the implementation of Solid [7]. Blockchain, as a recent technology applied in academia and industry, has attracted increasing attention to provide secure authentication and privacy for distributed systems [8], [9]. To this end, we propose the introduction of blockchains to assist Solid in developing a trust access authentication system. In a blockchain-assisted Solid application scenario, for example,

The associate editor coordinating the review of this manuscript and approving it for publication was Wen Sun.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

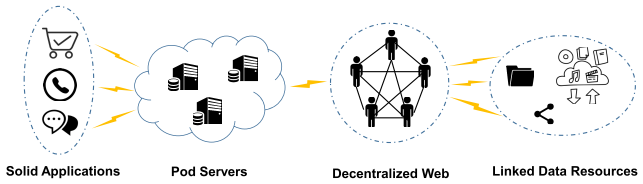


FIGURE 1. The illustration of a Solid ecosystem.

when a Solid application requests linked data resources from the pod servers, the blockchain provides secure authentication for the requestor and grants access to the corresponding resource after validation. The blockchain brings the following advantages:

- *Trustworthy Transactions:* In the blockchain, a group of participants work cooperately to validate and record transactions and thus maintain a public ledger [10]. This non-tampering property enables the Solid entity act in trusted environments.
- *Fine-grained Access Control:* Solid uses an RDF-based access control mechanism to achieve resource authorisation [2]. Whereas, blockchain can utilize smart contracts to gain a more fine-grained effect to manage access control policies in a secure and automatic manner.
- *Traceability and Auditability:* Activities such as the Solid application's access history, instead of being stored in pod servers, are recorded in the blockchain where all access records are traceable and auditable.

Although applying the blockchain into a Solid ecosystem is a promising development, difficulties and challenges still remain. In this paper, we only focus on how to exploit the blockchain to ensure secure authentication with fine-grained access control that is suitable for a Solid system. *Digital signatures*, as a key component of a blockchain system, can be integrated to provide secure authentication [11], [12]. Different from mainstream signatures such as the elliptic curve digital signature algorithm (ECDSA) adopted in the Bitcoin and Ethereum blockchains, our proposal is to introduce a threshold signature scheme in blockchain systems. *Threshold signatures* can enhance the robustness and resilience of an applied system while maintaining the decentralized nature of blockchains, which is why a threshold signature scheme is necessary [13]. Furthermore, threshold signatures eliminate the high cost of transmission arise from an increasing number of signatures that occurs in a multi-signature scheme. Our goal is to integrate threshold signatures with the blockchain to enhance the security performance of Solid, i.e., provide distributed yet fault-tolerant signatures to authenticate Solid entities.

Extending the previous conference version [14] that has already reported our preliminary works, we make advances and new contributions in this journal version, including: 1) adjust our centers around the ''trust access authentication'' to conduct more in-depth work; 2) a better representative system model is proposed, which is closer to a real Solid application scenario; 3) a new digital signature scheme is applied in blockchain systems; 4) two signature reconstruction algorithms are considered, thereby making our solutions available to many more potential applications in Solid; and 5) based on the theory and assessment in our previous version, this work implements the system designs such as developing and deploying smart contracts, integrating signature schemes in the transaction validation of permissioned blockchains, and implementing our solutions to evaluate them with benchmarks. Our main contributions are listed as follows:

- We propose a conceptual blockchain-assisted system that provides trust access authentication, suited for the web re-decentralization project Solid.
- We integrate threshold RSA signatures and the permissioned blockchain to enhance the resilience and robustness of the proposed authentication system, which can tolerate failed or corrupted peers.
- We leverage smart contracts to implement a fine-grained access control mechanism in which the access permissions on resources can be adaptively granted or revoked. We also evaluate the performance of our solutions in real-world scenarios.

The rest of this paper is organized as follows. In Section II, we present our system model. In Section III, we introduce the relevant building blocks used in our system. In Section IV, we describe how to design our proposed system in detail and provide the concrete implementation. In Section V, we present the security analysis of our system, measure the cost of smart contracts, and evaluate the performance on threshold RSA signatures. Finally, Section VI concludes the paper.

II. SYSTEM MODEL

In this section, we present the system model and the relevant functional descriptions. Fig. 2 shows the participants involved in our system, including *Solid Applications*, *Blockchain Network*, *Pod Servers*, *Decentralized Web*, and *Linked Data Resources*.

A. SOLID APPLICATIONS

Solid applications constitute a set of social web applications (e.g., Twitter, Facebook, and Wikipedia) that often request for access or process the linked data resources. When a Solid application is requesting an access, it needs to post a corresponding transaction to the blockchain network.

B. BLOCKCHAIN NETWORK

The blockchain in the proposed system adopts permissioned blockchains such as Hyperledger Fabric. Thus, the peers in a blockchain may have the following roles: *submitting peers* (submitter), who are responsible for submitting transactions by providing a client interface; *endorsing peers* (endorser), who is responsible for simulating and validating transactions with the execution of chaincodes (i.e., smart contract) [15], [16]. In general, any requests from the Solid applications

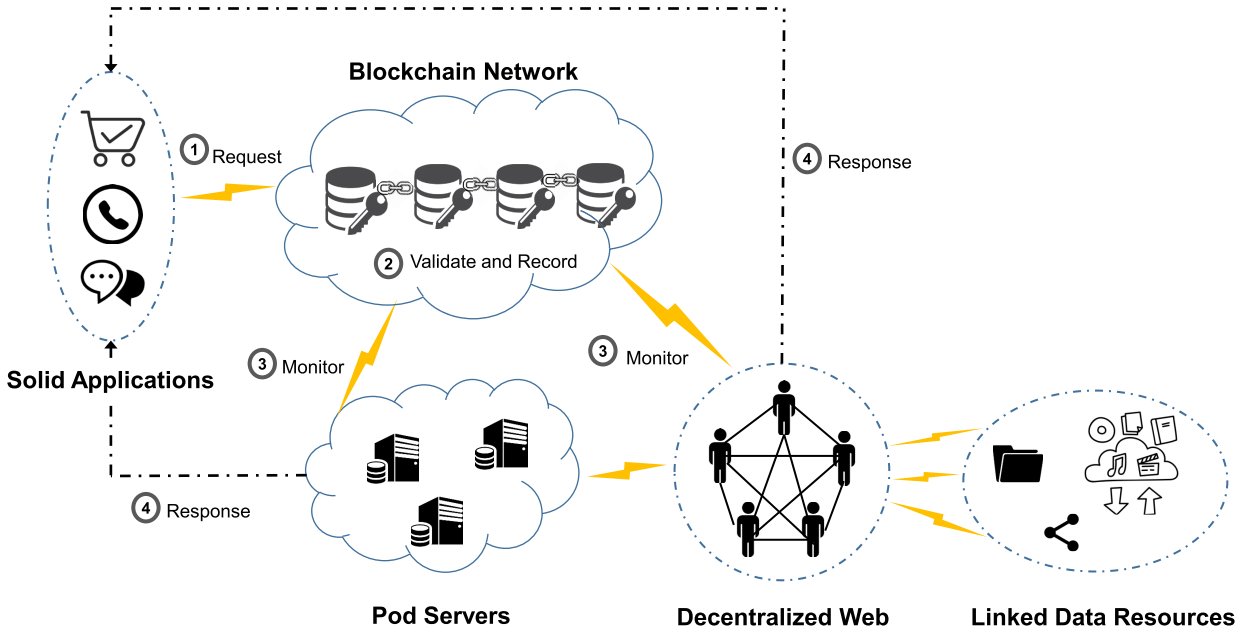


FIGURE 2. Proposed blockchain-assisted trust access authentication system for Solid.

can be recorded in the blockchain after reaching a consensus among peers.

C. POD SERVERS

Pod servers host a number of pods that store and process the linked data resources, and respond to the authorized access requests from Solid applications. A pod server is not controlled by any third party, and it can be from the free providers of commercial entity established by the Solid founding team or the users whose personal computers are installed and run as a node-solid server in the network.

D. DECENTRALIZED WEB

In a Solid ecosystem, the design of pods allow specific Solid applications to access the specific resource through a decentralized network, i.e., decentralized web. In our system, such a decentralized web monitors the blockchain, receives data operation requests, and helps transmit and store the linked data resources to corresponding pods.

E. LINKED DATA RESOURCES

Although differences exist between non-linked data (e.g., image, PDF file) and linked data (e.g., Turtle and JSON-LD formats) in a pod, all the data resources in Solid are grouped into data folders with public or private access rights [17]. Therefore, in our system, we do not consider their distinctions and uniformly represent them as the linked data resources.

III. SYSTEM BUILDING BLOCKS

In this section, we introduce the relevant building blocks used in our system, including the blockchain, distributed hash table for access control, threshold RSA signatures, and transactions.

A. BLOCKCHAIN

Blockchain is the core data structure that consists of key components such as the public ledger, consensus mechanism and smart contract, which enables multiple parties to participate in transactions without trusting each other [18]. Undoubtedly, the permissionless blockchains such as Bitcoin and Ethereum are the most popular blockchain systems where anyone can engage and validate transactions [4], [19]. However, our system adopts a permissioned blockchain (i.e., Hyperledger Fabric) to realize construction. The option is for two reasons: 1) transactions in a Solid ecosystem need to be scalable and fast, which in a permissioned blockchain such as Hyperledger Fabric can process thousands of transactions per second; and 2) a permissioned blockchain requires the transaction validators to be the approved selected participants, which is easy to satisfy in a Solid ecosystem.

Typical consensus mechanisms in blockchains are proof of work, proof of stake, delegated proof of stake, and practical byzantine fault tolerance (PBFT) [20]. The PBFT is used as a representative consensus algorithm for permissioned blockchains. In our design, every participating peer in the blockchain network uses an ordering service with the PBFT consensus protocol to maintain a synchronized ledger. The details on the PBFT consensus protocol are found in *Castro*'s scheme [21].

B. DISTRIBUTED HASH TABLE FOR ACCESS CONTROL

Distributed hash table (DHT), an implementation of storing access control policies that are csimilar to the off-chain

key-value store for blockchains, is adopted to offer privacy-aware fine-grained access control [22]. Let $\mathcal{H}: \{0,1\}^{256} \rightarrow \{0,1\}^n \times \{0,1\}^n$, where $n \gg 256$. That is, *hash_of _predicate* : *predicate* × *policy*, where *predicate* maps to various permission policies corresponding to different access *user_ID*. For static access control rules, the *policy* keeps the same *predicate* for insert, update, delete, and query. In our system, the DHTs can be structured in the pod servers, which can only be updated with smart contract.

C. THRESHOLD RSA SIGNATURES

A blockchain-assisted system for Solid can ensure authenticity, integrity, pseudonymity, and non-repudiation with the help of cryptography, specifically, digital signatures [8], [23]. To achieve a single fault-tolerant signature from a group of participants, we integrate threshold signatures in the blockchain system. As the advantage of applications in Bitcoin discussed by *Goldfeder* et al., the threshold signatures can enhance our system's robustness and resilience while maintaining the decentralized nature of blockchains [24].

However, applying the threshold signatures in blockchain systems has not been well exploited. In these limited studies, most threshold signatures are developed for Bitcoin based on a threshold ECDSA scheme [25]; thus, most of them focus on the improvement of security and performance for permissionless blockchains [26]. In our system, we implement an RSA threshold signature scheme based on *Shoup*'s work [27], in which security is proven in the random oracle model. To describe concrete construction, for example, a threshold signature with (n, t), we assume a key generation center (KGC) $D \notin \mathbf{P}$, the actors are a group of n participants, and $\mathbf{P} = \{P_1, P_2, \ldots, P_n\}$, where t is the number of possible corrupted participants. In detail, we present the five general steps in threshold RSA signatures as follows.

1) GENERATE KEYS

The KGC selects two prime numbers P and Q, where $P = 2p + 1$, $Q = 2q + 1$, p, q are prime numbers, and P, Q are both *safe primes* that are equal in length. Let N represents an RSA modulus, and N is equal to PQ. Then, the KGC selects a prime number e as the public exponent for RSA, and $e > n$; thus, the public key is established as (N, e). Next, the KGC computes a secret key d that satisfies $de \equiv 1 \bmod m$. The KGC then sets $a_0 = d$ and chooses the coefficients $a_i (1 \le i \le tc)$ at random from $\{0, \ldots, m-1\}$ to define a polynomial $f(x) = \sum_{i=0}^{t} a_i X^i \in \mathbb{Z}[X]$. For $1 \le i \le n$, the secret key shares s_i can be caculated as follows:

$$s_i = f(i) \triangle^{-1} \bmod m \tag{1}$$

where $\triangle = n!$. Note that the KGC also generates a global verification key v by selecting a random $v \in Q_n$ and then computes the local verification keys $v_i = v_i^s \in Q_n$, where $Q_n = \mathbb{Z}_m \times \mathbb{Z}_2 \times \mathbb{Z}_2$ and Q_n is the subgroup of squares in $\mathbb{Z}_m^*$ of order m. Note that $\mathbb{Z}_n^* \simeq \mathbb{Z}_m \times \mathbb{Z}_2 \times \mathbb{Z}_2$, a subgroup of elements $u \in \mathbb{Z}_n^*$ with Jacobi symbol $(u|n) = -1$ can be added to v_i.

2) GENERATE SIGNATURE SHARES

Let M be a message, and $u \in \mathbb{Z}_n^*$ be the random element where there is $(u|n) = -1$ (i.e., Jacobi symbol). We have: 1) if $\left(\hat{x} \mid N\right) = -1$, then $x = \mathcal{H}(M) = \hat{x}u^e$, where $\hat{x} = \mathcal{H}(M)$; and 2) if $\left(\hat{x} \mid N\right) = 1$, then x can be hashed as $x = \mathcal{H}(M) = \hat{x}$. Every participant's signature share can be generated as follows:

$$\sigma_i = x^{2s_i} \in Q_n \tag{2}$$

To verify the signature shares, a "proof of correctness" is provided along with the signature share σ_i. For this, we adopt an interactive protocol proposed by *Chaum* and *Pedersen* [28] in which a player randomly selects a $r \in \mathbb{Z}_q$ and then sends $(a, b) = (g^r, x^r)$ to a verifier, where q represents the prime order of a group G_q generated by part of the public key, i.e., $g\ \left(g \in G_q\right)$. Then, verifiers randomly select and send a challenge c to the player; accordingly, the player returns a $z = r + cs$ in which s is the secret key. Follow *Shoup*'s work in [27], a hash function H' that produces a fixed-length output $L_1 = 128$ is adopted. Then, a participant P_i randomly selects a number $r \in \left\{0, \cdots, 2^{L(N)+2L_1}\right\}$ to calculate $v' = v^r$ and $x' = \tilde{x}^r$; thus, the following challenge is obtained:

$$c = H'\left(u, \tilde{x}, v_i, x_i^2, v', x'\right) \tag{3}$$

For a signature share s_i, the proof of correctness is tuple (c, z) calculated as follows:

$$z = s_i c + r \tag{4}$$

3) VERIFY SIGNATURE SHARES

To verify the above generated proof of correctness, the $\log_{\tilde{x}}\left(x_i^2\right) = \log_v(v_i)$ needs to be proved, that is:

$$c = H'\left(u, \tilde{x}, v_i, x_i^2, \frac{v^z}{v_i^c}, \frac{x^z}{x_i^{2c}}\right) \tag{5}$$

We must point out that the exponentiation in (5) is an expensive operation because z is the length of $L(N) + 2L_1$.

4) COMBINE SIGNATURE SHARES

According to an (n, k, t) threshold signature scheme [33], a signature σ is required to be generated by some subset of at least k signers that work together. We assume that k is from a set of participants $\mathcal{S}$, and $\mathcal{S} = \{i_1, i_2, \cdots, i_k\} \subset \{1, 2, \cdots, n\}$. Let $x = H(M) \in \mathbb{Z}_n^*$, we have

$$w = \prod_{j \in \mathcal{S}} \sigma_j^{2\lambda_{0,j}^{\mathcal{S}}} \tag{6}$$

where $\lambda_{i,j}^{\mathcal{S}} = \triangle \frac{\prod_{j' \in \mathcal{S} \setminus \{j\}} (i - j')}{\prod_{j' \in \mathcal{S} \setminus \{j\}} (i - j')} \in \mathbb{Z}$, which are always integers because $\triangle$ is equal to $n!$. According to the Lagrange's interpolation formula, we can compute as follows:

$$w^e = \left(\prod_{j \in \mathcal{S}} x^{2s_j 2\lambda_{0,j}^{\mathcal{S}}}\right)^e = x^{4e \sum_{j \in \mathcal{S}} \lambda_{0,j}^{\mathcal{S}} s_j} = x^4 c \tag{7}$$

The signature σ can be calculated as $\sigma^e = x$. Furthermore, $gcd\ (4, e) = 1$, and a, b exist such that $4a + eb = 1$; thus, $\sigma = w^a x^b$ is the combined signature.

5) VERIFY SIGNATURE

The receiver can verify the signature σ simply as similar to verifying a standard RSA signature. That is, a message M that is hashed by a hash function $\mathcal{H}$ must be checked to satisfy $\sigma^e = \mathcal{H}(M)$.

To make the algorithm easy to describe, we generally denote the above steps as five main methods, i.e., *GenKeys*(), *GenSigShares*(), *VerSigShares*(), *ComSigShares*(), and *VerSig*(). To integrate such an RSA threshold signature scheme in our system, we implement a **ThresholdSinger** interface including the following two functions:

- *Sign*(): Taking a massage M, this method returns a signature share σ_i based on the secret key share s_i.
- *Verify*(): Taking a massage M and a signature σ, this method returns *true* if σ is valid.

D. TRANSACTION

We leverage the blockchain to validate and record transactions, thereby providing a trust access authentication system for Solid. Therefore, unlike a cryptocurrency in the blockchain systems such as Bitcoin and Ethereum, ''transactions'' in our system mainly refer to various instructions. Specifically, the blockchain in our system accepts two types of transactions such as T_{data} and T_{access}, as described in the following:

1) RESOURCE REQUEST TRANSACTION

A resource request transaction, denoted as T_{data}, is a request for linked data resources such as a data retrieval from the Solid applications; or a storage request on the linked data resources. The following lists the possible items contained in a transaction T_{data}. Therefore, we can formally define a T_{data} as follows: $T_{data} = Sign(from||to||type||timestamp||pk||res\_info||chain\_ID||tx\_ID)$.

- *from*: a requestor's address, perhaps a unique identity *requestor_ID* of the requestor, which is a necessary item;
- *to*: a receiver's address, e.g., the address of submitting peer in the blockchain network;
- *type*: especially designs for data request transactions, which includes ''storage'' and ''retrieval'';
- *timestamp*: a sequence of characters that identifies when the transaction occurred;
- *pk*: a requestor's public key that can be used to encrypt messages when it communicates with the pod servers or decentralized web nodes;
- *res_info*: the identity of the resource owner who controls the ownership of requested linked data;
- *chain_ID*: the parameter or functions that are used to trigger the related smart contracts;
- *tx_ID*: the unique identity of a transaction, which is generally a hash value of the transaction;
- *sig*: the signature of a transaction signing with the requestor's private key using the function *Sign*().

2) PERMISSION MANAGEMENT TRANSACTION

A transaction for permission management, denoted as T_{access}, is used to manage access control over linked data resources. Specifically, we can grant or invoke permissions by updating the access policies related to different predicates. Based on the aforementioned items, we can define a T_{access} as $T_{access} = Sign(from||to||timestamp||pk||predicate||policy||chian\_ID||tx\_ID)$. Note that only the resource owner can issue a transaction T_{access}.

IV. SYSTEM DESIGN AND IMPLEMENTATION

In this section, we explain how to integrate the threshold RSA signatures in a permissioned blockchain, i.e., HyperLedger Fabric. Furthermore, we use a Solid application scenario to describe our system design in detail.

A. IMPLEMENTING THRESHOLD RSA SIGNATURES IN THE BLOCKCHAIN

An RSA threshold signature scheme enables a group of participants to generate a single yet fault-tolerant signature in a cooperative distributed way [29], [30]. As a powerful tool, the threshold RSA signatures may have many potential applications for blockchain systems. For example, every peer in the Hyperledger Fabric is identified with a certificate signed by one certificate authority (CA), instead of which a group of CAs can be replaced using the threshold signature scheme [31]. In this paper, our goal is to integrate the threshold RSA signatures into Hyperledger Fabric and focus on the transaction flow working with the threshold signatures.

1) SIGNATURE SHARE COMBINATION ALGORITHMS

Signature shares enable robustness in a threshold signature scheme; thus, how to combine such shares into a signature plays an important role in our system design [32]. Assuming this scenario, we find a (n, k, t) threshold signature scheme, where n represents the signing servers $S_1, S_2, \ldots, S_n$. A client $\mathbb{C}$ wants to sign a message M with such a threshold signature [33]. To this end, client $\mathbb{C}$ needs to request at least k $(t < k \leq n - t)$ valid signature shares from the singing servers and then reconstruct a signature $\sigma_{\mathbb{C}}$ using an equipped function *ComSigShares*(). Assuming that a reliable and synchronized network transmits the signature shares to client $\mathbb{C}$, we present two solutions to achieve the aforementioned purpose.

Solution 1: $\mathbb{C}$ calls the function *ComSigShares*() after it receives k validated signature shares, and thereby reconstructs the final signature $\sigma_{\mathbb{C}}$ as shown in Algorithm 1.

Solution 2: Algorithm 2 takes the first received k signature shares as the arguments of function *ComSigShares*() to reconstruct an assumed signature σ_t, and then validates if σ_t

Algorithm 1 Unoptimistic Signature Reconstruction

Input: $s_1, s_2, \ldots, s_n$
Output: $\sigma_{\mathbb{C}}$

```
function UnCombineSignature(s)
    validShares ← [ ]
    length ← 0
    for s ∈ signatureShares do
        if VerSigShares(s) then
            validShares.add(s)
            length ← length + 1
        end if
    end for
    if length > k then
        σ_C ← ComSigShares(validShares)
        return σ_C
    end if
end function
```

Algorithm 2 Optimistic Signature Reconstruction

Input: $s_1, s_2, \ldots, s_n$
Output: $\sigma_{\mathbb{C}}$

```
function OpCombineSignature(s)
    invalidCombinations ← [ ]
    for i ∈ 0, 1, ..., n − k do
        validCombinations ← {s_1, s_2, ..., s_{k+i}}
        for c ∈ validCombinations do
            if c ∉ invalidCombinations then
                σ_C ← ComSigShares(c)
                if Verify(σ_C) then
                    return σ_C
                else
                    invalidCombinations.add(c)
                end if
            end if
        end for
    end for
end function
```

is valid. If it is valid, then σ_t is the correct signature $\sigma_{\mathbb{C}}$. If it fails, then the algorithm continues to receive signature shares and tries other different combinations of subsets of $k+i$ shares until a final signature is obtained.

2) TRANSACTION FLOW WITH THRESHOLD SIGNATURES

In this subsection, we explain how to exploit the threshold signatures to validate transactions in the proposed system. Fig. 3 illustrates the transaction flow with adopted threshold signatures. In our design, we accelerate the transaction validation by integrating the threshold signatures in transaction endorsement, so that we do not need to verify every one of the endorsements. In detail, our solution consists of the following phases:

Phase 0: When a transaction is created and delivered to a submitting peer via a client, the submitter needs to verify the signature of the client. If the verification is successful, then the transaction is wrapped as a *proposal* message and the *proposal* is sent to a group of endorsers.

Phase 1: Once the endorsers receive the *proposal*, they start to simulate the transaction after successfully verifying the signature and then return the *proposal response* messages (i.e., endorsements) to the submitter together with their signatures. The submitter, upon receiving the required number of endorsements, utilizes these endorsers' signature shares to reconstruct a signature, thereby adding only one *endorsement* message signed with the signature to the transaction, which is submitted to the ordering service.

Phase 2: After receiving the endorsed transaction, the ordering service orders the transaction, packages it in a block, and broadcasts it to all peers in the blockchain network.

Phase 3: The peers engage with the endorsement validation in which they only need to verify only one signature because of our threshold signature scheme in the endorsement procedure, as described in Phase 1. Thereafter, the peers chains the transaction in local ledger and modify the world state.

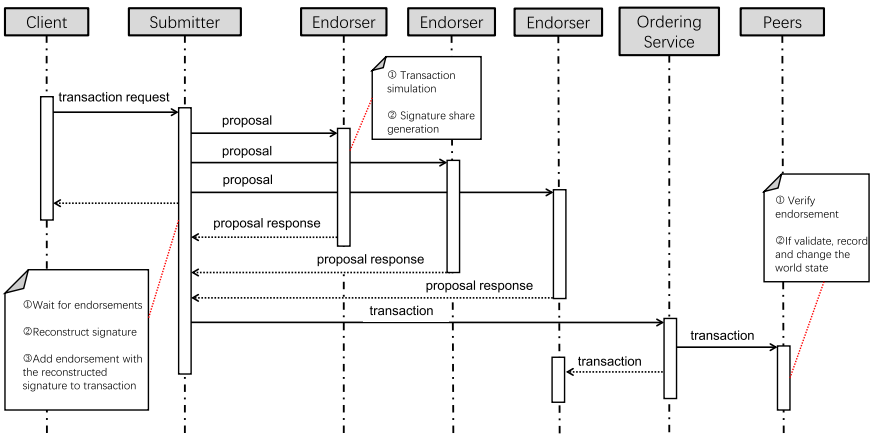


FIGURE 3. Transaction flow with the proposed threshold RSA signatures.

Algorithm 3 Smart Contract for Access Control

```
procedure AcContract
    linked_data_requestor ← a_policy, a_pre
    policyCheck ← false
    procedure Create( )
        dataOwner_ID ← sender.pk
        t ← [ ]
        length ← 0
    end procedure
    procedure UpdateDHT(pk, predicate, policy)
        if sender.pk is dataOwner then
            if t[i].hash = H(predicate) is exist then
                if policy = NULL then
                    Delete(t[i])
                    length ← length − 1
                else
                    policy ← t[i].policy
                end if
            else
                if policy! = NULL then
                    length ← length + 1
                    t[length].hash ← H(predicate)
                    t[length].policy ← policy
                end if
            end if
        end if
    end procedure
    procedure CheckPolicy(a_pre, a_policy)
        if (t[i].hash = H(a_pre) is exist) and (a_policy ∈
    t[i].policy) then
            policyCheck ← true
        else
            policyCheck ← false
        end if
    end procedure
end procedure
```

Algorithm 4 Smart Contract for Transactions

```
procedure TxContract
    procedure Create( )
        submitter[ ] = {sp_1, sp_2, ··· , sp_n}
        vTx ← [ ]
        vTxLen ← 0
        uvTx ← [ ]
        uvTxLen ← 0
    end procedure
    procedure UploadTx(pk, Tx, k)          ▷ run by peers
        if k = 1 then
            if pk ∈ submitter then
                vTxLen ← vTxLen + 1
                vTx[vTxLen] ← Tx
            end if
        else
            uvTxLen ← uvTxLen + 1
            uvTx[uvTxLen] ← Tx
        end if
    end procedure
    procedure getTx(k)          ▷ run by submitting peers
        if k = 0 then
            Tx ← uvTx[uvTxLen]
            uvTx[uvTxLen] ← 0
            uvTxLen ← uvTxLen − 1
        else
            Tx ← vTx[vTxLen]
            vTx[vTxLen] ← 0
            vTxLen ← vTxLen − 1
        end if
    end procedure
end procedure
```

B. SYSTEM DESIGN

1) INITIALIZATION

The proposed system needs to initialize related system parameters, which mainly comprises the following two phases:

- **Signature Initialization:** Select the required arguments such as the RSA modulus size N, number of signing nodes n, and number of secret shares t based on the Solid application scenario. The proposed system generates the public key pair (N, e), the secret key shares s_i, and the group of verification keys v_i by invoking the method *GenKeys*(). When a Solid application first registers through a client, a signing key σ combined by n signature shares can be generated using the method *ComSigShare*().
- **Contract Deployment:** Code the designed smart contracts such as Algorithm 3 and Algorithm 4, compile and deploy them on the blockchain. Smart contracts deployed on blockchains are successful transactions, which means they have passed the validation process and been recorded into blocks. Each smart contract on the blockchain has one unique address, for example, we use *A_ID* to stand for *AcContractc* and *T_ID* for *TxContract*.

2) REQUEST ISSUANCE

We assume that $\mathcal{A}$ is a Solid application that wants to publish a request on linked data resource. It needs to execute the following steps:

- **Create Transaction:** Prepare all components required in a request transaction as defined in Section III(D) to construct a transaction T_{data}. Specifically, $T_{data} = Sign_{sk_{\mathcal{A}}}(\mathcal{A}ID||sp_i||timestamp||pk_{\mathcal{A}}||TID, UploadTx\left(pk_{sp_i}, \mathcal{H}(nonce), 0\right)||\mathcal{H}(nonce))$, where *from* is the unique identity $\mathcal{A}\_ID$ of $\mathcal{A}$; *to* refers to the target receiver, a submitting peer sp_i in the blockchain network; *chian_ID* instantiates as the function *UploadTx*() with parameters $pk = pk_{sp_i}$, $tx = \mathcal{H}(nonce)$, and $s = 0$, which means that transaction T_{data} will trigger

the smart contract as shown in Algorithm 4 and thereby be added to invalid transactions $uvTx[\ ]$; and Tx_ID is the unique identity $\mathcal{H}(nonce)$ of transaction T_{access}. In addition, the public and private key pair $(pk_{\mathcal{A}}, sk_{\mathcal{A}})$ used in T_{data} are generated by the CA when registered for initialization.

- **Simulate Transaction:** After the transaction invocation, the transaction T_{data} is sent to the endorsing peers in the blockchain network to simulate the transaction. Notably, the simulation of T_{data} is executed by the transaction chaincodes (i.e., smart contracts *TxContract* and *AcContract*), and then the result of a simulation *proposal response* returns to the submitting peer sp_i together with the signature σ_i of each endorsing peer. As illustrated in Fig. 3, the submitting peer sp_i waits to collect required t signature shares of σ_i $(1 \leq i \leq n)$ to reconstruct the signature σ (see algorithms in Section IV(A)), and adds only one endorsement result appended with the signature σ to the transaction T_{data}.

3) CHAIN TRANSACTION

This phase needs to validate transactions, record and chain the verified transactions on blockchains, and modify the world state in the blockchain network.

- **Ordering Transactions:** The endorsed transaction T_{data} is signed by the submitting peer sp_i and is submitted to the ordering service. In the ordering service, the transaction T_{data} is ordered into a block with other endorsed transactions in $uvTx[\ ]$ using a consensus procedure (i.e., PBFT). Thereafter, all peers in the blockchain network receive the broadcasted transaction T_{data} from the ordering service.
- **Validate and Record Transaction:** To validate the transaction T_{data}, the peers must check if the predicate of T_{data} is the permission granted by the resource owner by invoking the *CheckPolicy*() in *AcContract*, and if the signature σ is valid using the function *Verify*(). Here, the peers utilize the PBFT consensus mechanism to make an agreement. After validation, the transaction T_{data} is recorded in the blockchain ledger. Notably, a validated T_{data} can first be obtained from invalid transactions $uvTx[\ ]$ by invoking a $getTx(0)$ and then set as a valid transaction in $vTx[\ ]$ by invoking the function $UploadTx(pk_{\mathcal{A}}, uvTx\,[i]\,, 1)$, as illustrated in Algorithm 4.

4) RESPONSE REQUEST

Pod servers and decentralized web gateways continue communicating with the blockchain to make instant request response. A new valid request that is validated and recorded by blockchains can be processed by the corresponding entity, as explained in the following:

- **Pod Servers:** The pod servers are responsible for responding to validated transactions such as T_{data} which requests for data storage or retrieval. For instance, the *type* in T_{data} is retrieval, and then the final response is $(En_{pk_{\mathcal{A}}}(data)\,, SHA256\left(En_{pk_{\mathcal{A}}}(data)\,, pk_{pod_i}\right))$, where $En_{pk_{\mathcal{A}}}(data)$ is the encrypted request resources with the public key of requestor $\mathcal{A}$, pk_{pod_i} represents the responding pod server, and $SHA256()$ ensures the confidentiality and authentication of messages.
- **Decentralized Web:** A validated transaction for permission management such as T_{access} is handled by the decentralized web. For example, we assume that T_{access} is a validated request to change the permissions on some data resources granted to a Solid application. The new set of permissions specified by the data owner can be functioned by revoking the function *UpdateDHT*() in smart contract *AcContract* via the decentralized web.

V. EVALUATION

A. SECURITY ANALYSIS

The application of blockchains establishes a secure access environment for multiple entities in Solid without mutual trust. We integrate threshold RSA signatures into the transaction validation process and utilize smart contracts to achieve the following security performances:

1) *All Solid entity behaviors in the system are trustworthy:* Each valid operation on the data resource is formalized as one transaction, which is verified and then recorded on chains. This immutable records on the blockchain enables a trustworthy data sharing ecosystem for Solid.
2) *Fault tolerant:* Due to the threshold RSA signatures, our system allows a group of peers in the blockchain network to engage in the signature reconstruction. All peers create signature shares, which are combined into a single fault-tolerant signature.
3) *Security authentication:* Different from blockchain systems such as Bitcoin and Ethereum, our system is built on a permissioned blockchain (i.e., HyperLedger Fabric). In this manner, other than digital signatures, a Fabric CA is used to help issue and validate certificates.
4) *Fine-grained access control:* In a smart contract, the resource owner can predefine the mappings of predicate and policy. Furthermore, during the transaction, the permission of predicates can also be modified by revoking the function *UpdateDHT*().
5) *Privacy protection:* In our proposed system, the linked data resource can be sent for storage in the pods in encrypted form so that eavesdroppers in a network cannot obtain it.

B. PERFORMANCE OF SMART CONTRACT

We consider the performance of our proposed smart contracts because the cost affects the system overhead. To this end, we exploited the Remix IDE[1] to develop, deploy, and test the smart contracts such as *AcContract* and *TxContract*. In Table 1, we list the gas cost of all functions included in the two smart contracts with 100 resource requestors. Gas, a

[1] http://remix.ethereum.org/

TABLE 1. Cost of functions in smart contracts

Type	Function	Execution Cost in Gas	Other Cost in Gas	Total Cost in Gas	Total cost in $
	Create	1768831	586008	2354839	0.463903283
	AddDHT	8789	21878	30667	0.006041399
AcContract	DeleteDHT	10246	23442	33688	0.006636536
	CheckPolicy	6208	17809	24098	0.004747306
	Resource Owner Total	19035	45320	64355	0.012677935
	Create	1542208	435974	1978182	0.389701854
TxContract	UploadTx	27832	21622	49454	0.009742438
	GetTx	20466	22704	43170	0.008504490
	Resource Requestor Total	91127	105716	196843	0.038778071

special unit in the Ethereum blockchain, is used as a meter to measure the resource consumption of the smart contract. The execution cost shows the gas costs of executing the contract instructions, whereas the other cost indicates the consumed gas on transactions calling the function. As shown in Table 1, the total gas for *AcContract* is 64355 gas, which enables a permission management transaction that includes **AddDHT** and **DeleteDHT**. According to the current exchange rate, i.e., 1 *Ether* $\approx$ 197$, and 1 *Wei* $= 10^{-18}$ *Ether*; thus, only 0.012677935$ is required for a resource owner to manage the DHT. As shown in *TxContract*, the total cost for a resource requestor is 0.038778071$, which includes calling the functions **AddDHT**, **CheckPolicy**, and **GetTx** once, and running the **UploadTx** function twice. The total cost for function **Create** seems somewhat a little high, but it only runs once at the first time when the smart contract is created. From the results, we can conclude that the cost of smart contracts could be economically feasible for our system.

C. PERFORMANCE EVALUATION OF THRESHOLD RSA SIGNATURES

This section provides a group of experiments to evaluate the proposed threshold signature schemes for a blockchain-assisted Solid system. We conducted the simulations using a laptop on which installed a Linux virtual machine (VMware Workstation Pro 15.5) with Ubuntu OS. The hardware parameters of the laptop were as follows: Intel(R) Core(TM) i7-8550U CPU @ 2.80 GHz, 16 GB RAM, Windows 10 Professional 64 bit. The technical characteristics of the virtual machine are listed: Ubuntu 18.04.3 LTS (64-bit), 12 GB RAM.

To evaluate the performance of our proposals, we measured execution time of a signature reconstruction operation. Specifically, we compared the performance of our proposed two algorithms in Section V(A), i.e., the *unoptimistic* and *optimistic* solutions. To simulate the signature reconstruction, we considered a network of n signing peers among which p signature peers might be faulty. Let $n = 3p+1$ and $k = p+1$, and we adopted the $(3p+1, p+1, p)$ threshold signatures [24]. We conducted comparison experiments under two different presumed scenarios: 1) all received signature shares from the signing peers are valid; and 2) p corrupted signature shares are randomly distributed in n signing peers. For an RSA modulus,

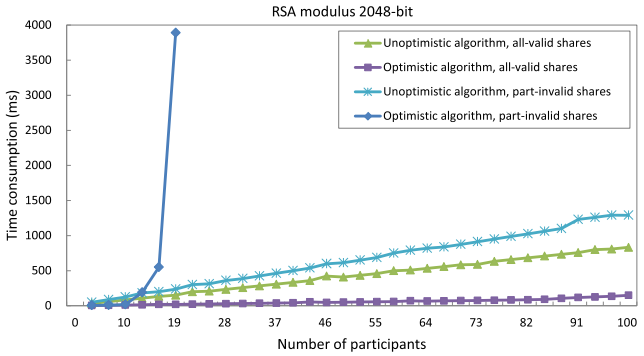


FIGURE 4. Performance of threshold RSA signatures based on signature reconstruction algorithms when the tested modulus for an RSA key is 2048 bits.

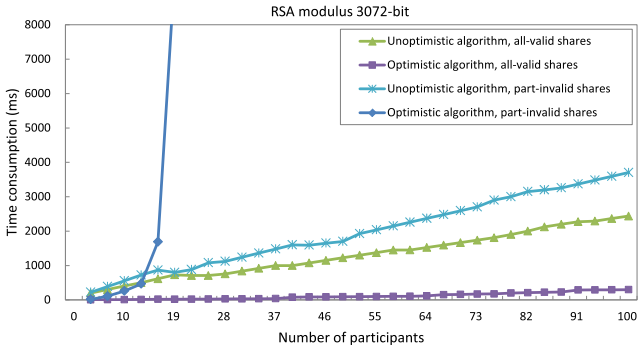


FIGURE 5. Performance of threshold RSA signatures based on signature reconstruction algorithms when the tested modulus for an RSA key is 3072 bits.

we selected the size of 2048 bits and 3072 bits because at least a 2048-bit modulus size in a RSA scheme is considered safe up to now. Fig. 4 and Fig. 5 show the time required for a signature reconstruction operation in the RSA threshold signature scheme with the increasing number of participants under the 2048-bit and 3072-bit RSA modulus, respectively. Fig. 6 compares the proposed two signature reconstruction algorithms in different RSA modulus sizes.

On the other hand, we compared our proposed threshold RSA signatures with RSA multisignatures, an efficient distributed signature scheme. In contrast to the threshold signatures, a multi-signature scheme only has one solution to reconstruct a signature, that is, the reconstruction occurs when a sufficient number of valid signatures are received

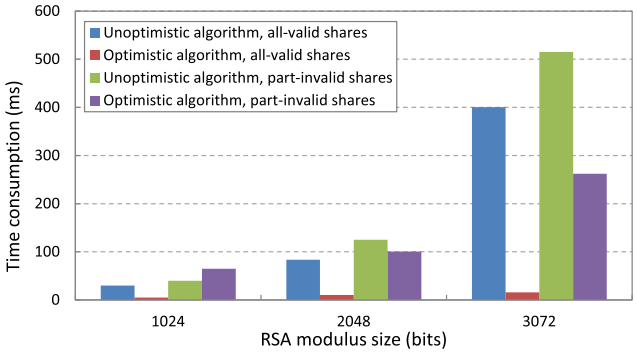


FIGURE 6. Comparison of the optimistic and unoptimistic solutions to the signature reconstruction when the number of participants is 10.

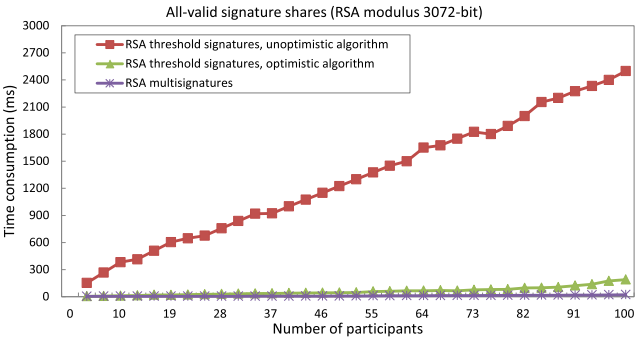


FIGURE 7. Signature reconstruction comparison of two distributed signature schemes in an all-valid signature shares scenario.

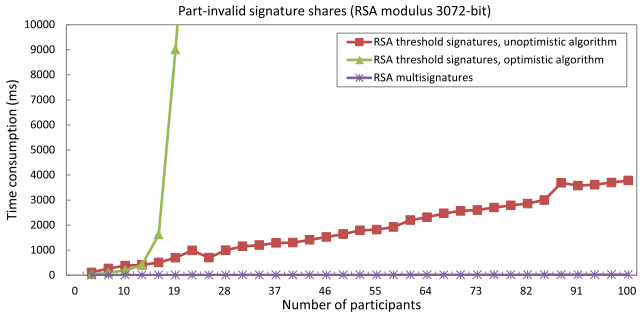


FIGURE 8. Signature reconstruction comparison of two distributed signature schemes in a part-invalid signature shares scenario.

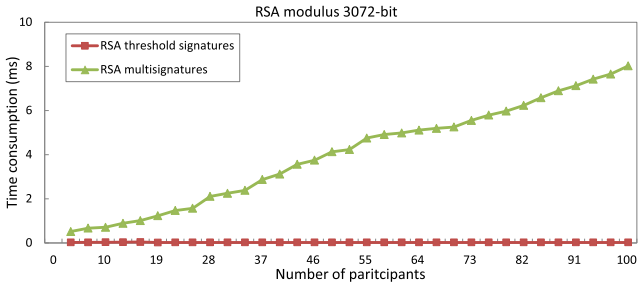


FIGURE 9. Performance comparison of two distributed signature schemes on signature verification.

from signers, which means that each signature needs to be verified until k valid signatures are collected. Such procedures of the RSA multi-signature scheme are equal to the *unoptimistic* solutions to the RSA threshold signature scheme. In this part, we first compared the time cost for a signature reconstruction operation in two RSA signature schemes, and then tested the verification time of their reconstructed signatures. The experiments also considered an all-valid share scenario and a part-invalid share scenario where the experimental settings were $n = 3p + 1$ and $k = p + 1$, and a 3075-bit RSA modulus was adopted. Fig. 7 and Fig. 8 show the comparison of time costs in signature reconstruction operations under two scenarios such as all-invalid signature shares and part-invalid signature shares. Fig. 9 shows the comparison of time costs in signature verification operation of the two RSA signature schemes.

From Fig. 4 and Fig. 5, we can observe that the *optimistic* algorithm (i.e., Algorithm 2) is more efficient than the *unoptimistic* algorithm (i.e., Algorithm 1) in a network without invalid signature shares. The reason is that the *unoptimistic* algorithm has to execute *VerSigShares*() many times, and such a function *VerSigShares*() is more expensive than the function *Verify*() performed in the *optimistic* algorithm. However, when corrupted signature shares exist, the time costs in signature reconstruction operation for the *optimistic* algorithm increase exponentially and quickly outpaced the *upoptimistic* algorithm. Moreover, a comparison of Fig. 4 and Fig. 5 show that the time costs in the signature reconstruction operation are different with an RSA modulus, i.e., the threshold signatures with a 3072-bit RSA modulus are more expensive but have higher security. Fig. 6 also illustrates this trend that the *optimistic* algorithm performs better than the *unoptimstic* algorithm with an increasing RSA modulus size, even in cases where signature shares are partly corrupt or invalid. Therefore, we suggest that the choice between the *unoptimistic* and *optimistic* algorithms depends on the real application. The *optimistic* algorithm is more efficient and suitable for a small and reliable network that does not have too many corrupted signature shares. Besides, the combination of such two algorithms in a signature reconstruction operation can also be a good choice.

In Fig. 7 and Fig. 8, the RSA multisignature scheme is more efficient than the RSA threshold signatures for a signature reconstruction operation. We must point out that the operation costs for the RSA multisignatures are less than 22 *ms*. The high costs in the RSA threshold signatures are due to the expensive verification of signature shares. Moreover, compared with the multisignatures, the threshold signatures have extra operation costs of the polynomial interpolation. In Fig. 9, we can observe that the verification time of the RSA threshold signatures keeps an average value of 0.025 ms/op because only one signature is to be verified in a threshold signature scheme. However, the overhead costs for RSA multisignatures increase linearly as the number of verification operations grows. To summarize, for a signature reconstruction operation, the RSA threshold signature scheme is more

expensive than that of the RSA multisignatures. By contrast, in a signature verification operation, the threshold RSA signatures require a fixed cost of time and can scale better in a large network with numerous participants. Furthermore, the signature size of the threshold RSA signatures is 384 bytes, whereas that of the RSA multisignatures is $384 * k$ bytes, thereby validating the proof that the RSA threshold signature scheme has better scalability.

VI. CONCLUSION

Solid is not a concept just out of reach, but an ongoing project that promises a new working approach for the existing web. To promote the implementation of such a web decentralized project, this paper focused on how to exploit the permissioned blockchain to implement secure authentication with fine-grained access control system suited for a Solid ecosystem. Specifically, the proposed system applies threshold RSA signatures in the blockchain to authenticate Solid applications, and uses smart contract to manage access control policies. We analyzed the security of the system and evaluated the performance of our solutions.

How to leverage blockchain technologies to empower the Solid project is still an open issue. In a future research, we will explore how to use threshold signatures to implement a group of CAs in a permissioned blockchain. Furthermore, a Solid ecosystem requires fast and efficient transactions, so we are motivated to improve the blockchain performance.

REFERENCES

[1] J. Werbrouck, P. Pauwels, J. Beetz, and L. van Berlo, "Towards a decentralised common data environment using linked building data and the solid ecosystem," in *Proc. 36th CIB W*, Newcastle, U.K., 2019, pp. 113–123.

[2] E. Mansour, A. V. Sambra, S. Hawke, M. Zereba, S. Capadisli, A. Ghanem, A. Aboulnaga, and T. Berners-Lee, "A demonstration of the solid platform for social Web applications," in *Proc. 25th Int. Conf. Companion World Wide Web (WWW Companion)*, Montreal, QC, Canada, 2016, pp. 223–226.

[3] W. Liang, K. Li, J. Long, X. Kui, and A. Zomaya, "An industrial network intrusion detection algorithm based on multi-characteristic data clustering optimization model," *IEEE Trans. Ind. Informat.*, vol. 16, no. 3, pp. 2063–2071, Mar. 2020.

[4] W. Liang, M. Tang, J. Long, X. Peng, J. Xu, and K.-C. Li, "A secure FaBric blockchain-based data transmission technique for industrial Internet-of-Things," *IEEE Trans. Ind. Informat.*, vol. 15, no. 6, pp. 3582–3592, Jun. 2019.

[5] G. Zyskind, O. Nathan, and A. Pentland, "Decentralizing privacy: Using blockchain to protect personal data," in *Proc. IEEE Secur. Privacy Workshops*, San Jose, CA, USA, May 2015, pp. 180–184.

[6] A. Sambra, A. Guy, S. Capadisli, and N. Greco, "Building decentralized applications for the social Web," in *Proc. 25th Int. Conf. Companion World Wide Web (WWW Companion)*, Montreal, QC, Canada, 2016, pp. 1033–1034.

[7] A. V. Sambra, E. Mansour, S. Hawke, M. Zereba, N. Greco, A. Ghanem, D. Zagidulin, A. Aboulnaga, and T. Berners-Lee, "Solid: A platform for decentralized social applications based on linked data," MIT CSAIL QCRI, Cambridge, MA, USA, Tech. Rep. MIT-QCRI-2016, 2016.

[8] H.-N. Dai, Z. Zheng, and Y. Zhang, "Blockchain for Internet of Things: A survey," *IEEE Internet Things J.*, vol. 6, no. 5, pp. 8076–8094, Oct. 2019.

[9] K. Liu, W. Chen, Z. Zheng, Z. Li, and W. Liang, "A novel debt-credit mechanism for blockchain-based data-trading in Internet of vehicles," *IEEE Internet Things J.*, vol. 6, no. 5, pp. 9098–9111, Oct. 2019.

[10] W. Chen, Z. Zhang, Z. Hong, C. Chen, J. Wu, S. Maharjan, Z. Zheng, and Y. Zhang, "Cooperative and distributed computation offloading for blockchain-empowered industrial Internet of Things," *IEEE Internet Things J.*, vol. 6, no. 5, pp. 8433–8446, Oct. 2019.

[11] Y. Yao, X. Chang, J. Misic, V. B. Misic, and L. Li, "BLA: Blockchain-assisted lightweight anonymous authentication for distributed vehicular fog services," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 3775–3784, Apr. 2019.

[12] H. Wang, Z. Zheng, S. Xie, H. N. Dai, and X. Chen, "Blockchain challenges and opportunities: A survey," *Int. J. Web Grid Services*, vol. 14, no. 4, pp. 352–375, Oct. 2018.

[13] J. Doerner, Y. Kondi, E. Lee, and A. Shelat, "Threshold ECDSA from ECDSA assumptions: The multiparty case," in *Proc. IEEE Symp. Secur. Privacy (SP)*, San Francisco, CA, USA, May 2019, pp. 1051–1066.

[14] T. Cai, W. Chen, and Y. Yu, "Bcsolid: A blockchain-based decentralized data storage and authentication scheme for solid," in *Proc. BlockSys*. Guangzhou, China: Springer, 2019, pp. 676–689.

[15] K. Gai, Y. Wu, L. Zhu, L. Xu, and Y. Zhang, "Permissioned blockchain and edge computing empowered privacy-preserving smart grid networks," *IEEE Internet Things J.*, vol. 6, no. 5, pp. 7992–8004, Oct. 2019.

[16] R. Cheng, F. Zhang, J. Kos, W. He, N. Hynes, N. Johnson, A. Juels, A. Miller, and D. Song, "Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contracts," in *Proc. IEEE Eur. Symp. Secur. Privacy (EuroS P)*, Stockholm, Sweden, Jun. 2019, pp. 185–200.

[17] N. M. Novak and A. M. Tjoa, "Towards a business value framework for linked enterprise data," in *Proc. IEEE-RIVF Int. Conf. Comput. Commun. Technol. (RIVF)*, Danang, Vietnam, Mar. 2019, pp. 1–6.

[18] Z. Li, Z. Yang, and S. Xie, "Computing resource trading for edge-cloud-assisted Internet of Things," *IEEE Trans. Ind. Informat.*, vol. 15, no. 6, pp. 3661–3669, Jun. 2019.

[19] Z. Li, Z. Yang, S. Xie, W. Chen, and K. Liu, "Credit-based payments for fast computing resource trading in edge-assisted Internet of Things," *IEEE Internet Things J.*, vol. 6, no. 4, pp. 6606–6617, Aug. 2019.

[20] X. Qiu, L. Liu, W. Chen, Z. Hong, and Z. Zheng, "Online deep reinforcement learning for computation offloading in blockchain-empowered mobile edge computing," *IEEE Trans. Veh. Technol.*, vol. 68, no. 8, pp. 8050–8062, Aug. 2019.

[21] M. Castro and B. Liskov, "Practical Byzantine fault tolerance," in *Proc. OSDI*, New Orleans, Louisiana, 1999, pp. 173–186.

[22] J. Pan, J. Wang, A. Hester, I. Alqerm, Y. Liu, and Y. Zhao, "EdgeChain: An edge-IoT framework and prototype based on blockchain and smart contracts," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4719–4732, Jun. 2019.

[23] Z. Zheng, S. Xie, H.-N. Dai, W. Chen, X. Chen, J. Weng, and M. Imran, "An overview on smart contracts: Challenges, advances and platforms," *Future Gener. Comput. Syst.*, vol. 105, pp. 475–491, Apr. 2020.

[24] R. Gennaro, S. Goldfeder, and A. Narayanan, "Threshold-optimal dsa/ecdsa signatures and an application to bitcoin wallet security," in *Proc. ACNS*, Bogota, Colombia, 2016, pp. 156–174.

[25] P. Dikshit and K. Singh, "Efficient weighted threshold ECDSA for securing bitcoin wallet," in *Proc. ISEA Asia Secur. Privacy (ISEASP)*, Surat, India, Jan. 2017, pp. 1–9.

[26] S. Guo, X. Hu, Z. Zhou, X. Wang, F. Qi, and L. Gao, "Trust access authentication in vehicular network based on blockchain," *China Commun.*, vol. 16, no. 6, pp. 18–30, Jun. 2019.

[27] V. Shoup, "Practical threshold signatures," in *Proc. EUROCRYPT*, Bruges, Belgium, 2000, pp. 207–220.

[28] D. Chaum and T. P. Pedersen, "Wallet databases with observers," in *Proc. CRYPTO*, Santa Barbara, CA, USA, 1992, pp. 89–105.

[29] K. Huang, X. Zhang, Y. Mu, X. Wang, G. Yang, X. Du, F. Rezaeibagha, Q. Xia, and M. Guizani, "Building redactable consortium blockchain for industrial Internet-of-Things," *IEEE Trans. Ind. Informat.*, vol. 15, no. 6, pp. 3670–3679, Jun. 2019.

[30] Z. Hong, W. Chen, H. Huang, S. Guo, and Z. Zheng, "Multi-hop cooperative computation offloading for industrial IoT–edge–cloud computing environments," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 12, pp. 2759–2774, Dec. 2019.

[31] H. Sukhwani, J. M. Martinez, X. Chang, K. S. Trivedi, and A. Rindos, "Performance modeling of PBFT consensus process for permissioned blockchain network (Hyperledger Fabric)," in *Proc. IEEE 36th Symp. Reliable Distrib. Syst. (SRDS)*, Hong Kong, Sep. 2017, pp. 253–255.

[32] P. Rastegari, M. Dakhilalian, M. Berenjkoub, and W. Susilo, "Multi-designated verifiers signature schemes with threshold verifiability: Generic pattern and a concrete scheme in the standard model," *IET Inf. Secur.*, vol. 13, no. 5, pp. 459–468, Sep. 2019.

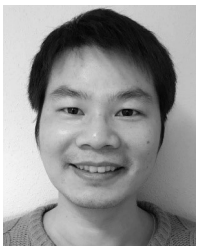
[33] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, pp. 612–613, Jun. 1979.

TING CAI is currently pursuing the Ph.D. degree in software engineering with the School of Data and Computer Science, Sun Yat-sen University, Guangzhou, China.

From 2012 to 2018, she was a Lecturer with the College of Mobile Telecom, Chongqing University of Posts and Telecom, Chongqing, China. Her current research interests include the blockchain, access control, the Internet of Things security, and edge/cloud computing.

ZETAO YANG is currently pursuing the B.Eng. degree in software engineering with the School of Data and Computer Science, Sun Yat-sen University, Guangzhou, China.

His current research interests include the game theory, the resource allocation for edge computing, and the blockchain.



WUHUI CHEN (Member, IEEE) received the bachelor's degree from Northeast University, Shengyang, China, in 2008, and the master's and Ph.D. degrees from the University of Aizu, Aizuwakamatsu, Japan, in 2011 and 2014, respectively.

From 2014 to 2016, he was a Research Fellow with the Japan Society for the Promotion of Science, Tokyo, Japan. From 2016 to 2017, he was a Researcher with the University of Aizu. He is currently an Associate Professor with Sun Yat-sen University, Guangzhou, China. His current research interests include edge/cloud computing, cloud robotics, and blockchain.

ZIBIN ZHENG (Senior Member, IEEE) received the Ph.D. degree from The Chinese University of Hong Kong, in 2011.

He is currently a Professor with the School of Data and Computer Science, Sun Yat-sen University, China. He serves as the Chairman of the Software Engineering Department. He has published over 120 international journal and conference papers, including 3 ESI highly cited articles. According to Google Scholar, his articles have more than 7000 citations, with an H-index of 42. His research interests include blockchain, services computing, software engineering, and financial big data. He was a recipient of several awards, including the Top 50 Influential Papers in Blockchain of 2018, the ACM SIGSOFT Distinguished Paper Award at ICSE2010, and the Best Student Paper Award at ICWS2010. He has served as the BlockSys'19 and the CollaborateCom'16 General Co-Chair, SC2'19, ICIOT'18, and the IoV'14 PC Co-Chair.

YANG YU (Member, IEEE) received the bachelor's and master's degrees in computer science from the Huazhong University of Science and Technology (HUST), Wuhan, China, in 1988 and 1991, respectively, and the Ph.D. degree in computer science from Sun Yat-sen University (SYSU), Guangzhou, China, in 2007.

From 1991 to 2002, he was a Senior Engineer and the CTO with a software company. In 2003, he became a Vice Professor at the School of Information Science and Technology, SYSU, where he has been a Full Professor with the School of Data and Computer Science, since 2011. He has published more than 60 articles and has been a reviewer for several prestigious international conferences and journals. His research interests include workflow, service computing, cloud computing, and software engineering. He is a member of ACM and a Senior Member of CCF.