IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received April 26, 2021, accepted May 21, 2021, date of publication June 3, 2021, date of current version June 11, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3085831

A Novel Access Control Method Via Smart Contracts for Internet-Based Service Provisioning

FARIBA GHAFFARI[1,2], EMMANUEL BERTIN[1,2], (Senior Member, IEEE), NOEL CRESPI[2], (Senior Member, IEEE), SHANAY BEHRAD[3], AND JULIEN HATIN[1]

[1]Orange Innovation, 14000 Caen, France
[2]IMT, Telecom SudParis, Institut Polytechnique de Paris, 91764 Palaiseau, France
[3]Institute of Research and Technology b-com, 35510 Cesson-Sévigné, France

Corresponding author: Fariba Ghaffari (fariba.ghaffari@orange.com)

ABSTRACT The dramatic rise in internet-based service provisioning has highlighted the importance of deploying scalable access control methods, facilitating service authorization for eligible users. Existing centralized methods suffer from single-point-of-failure, low scalability, and high computational overhead. In addition, in these methods, users pay for the service provider as well as the network provider independently for a specific service, imposing extra cost for the user. New business models are needed to resolve such shortcomings. The realization of these models calls for sophisticated access control methods which consider the requirements of all parties who want to: 1) access a service; 2) provide that service; and 3) provide the network connection. Blockchain is an enabling technology that provides unprecedented opportunities to novel distributed access control methods for new business models. We propose an Attribute-based access control solution by leveraging Blockchain to share network providers' and service providers' resources. Our solution offers access flexibility based on the requirements of the parties while fulfilling reliability, accountability, and immutability. Besides, it decreases the overall service cost which is beneficial for each party. Our solution makes it possible for service providers to outsource their access control procedures without requiring a trusted third party. The experiments confirm that our solution can provide a fast, comprehensive, and scalable access control mechanism.

INDEX TERMS ABAC, blockchain, ethereum, flexible access control, network, service provisioning, smart contract, trusted payment, 5G.

I. INTRODUCTION

The dramatic increase in network-based services (e.g., video/audio streaming and internet calls) has led both network providers and service providers to implement a variety of techniques to meet the growing need for network security. Given that service providers offer their services to users (e.g., websites similar to Netflix for video streaming) and network providers provide the network infrastructure, Access Control (AC) is one of the vital aspects in the effort to assess users' eligibility to access services. AC can be implemented by deploying one or more different solutions. The majority of these solutions are deployed in centralized systems, in which a user's request is sent to a Central Authority (CA) and the CA then decides about their eligibility to access the system based on certain rules.

As a typical scenario of current internet-based service provisioning, when a user wants to use an online service (e.g., video streaming), he/she first subscribes to both a network provider and a service provider and then requests the service provider to access the service. The service provider then authorizes the user via its central authority. If the user is eligible, she can watch the video. While watching the video, the user is consuming not only the already paid service, but also her available internet. In the other word, user is paying for the service provider as well as the network provider. This scenario undergoes several drawbacks from both business and technical perspectives. From the technical viewpoint, the central authority in the access control procedure is a single point of failure. So, it causes low scalability, management difficulties (e.g., in the maintenance of the server), lack of automation

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

in the access control process, and low fault tolerance. From a business model perspective, another shortcoming is that the user has to do extra payment for the service. Regarding these parameters, any solution bringing flexibility in access management, rules' immutability, distributed management of requests, fault tolerance, possibility to implement a new business model (to reduce extra payments), and eliminates the need for a trusted third party can be a promising candidate as a new access control mechanism.

Blockchain [1] was introduced in 2008 as a distributed technology. In 2014, the first extension of this technology emerged as smart contracts [2]. These technologies are cryptographically secure, and all changes in the system require the consensus of all the eligible nodes. After the consensus of the nodes, the updates are applied in the distributed ledger stored in each node of the network. Thanks to their features, Blockchain, and smart contracts are changing many aspects of business models, management, and operations in a range of fields. Indeed, Blockchain can be a game-changing technology in access control, because: Its distributed nature can remove the single point of failure and increase the availability of the system;

- The consensus among nodes provides a more reliable access management process;
- It can offer the immutability of rules, due to its cryptographically secure nature;
- Smart contracts provide the ability of access flexibility using agreements among parties, based on their needs;
- Signing the transactions can support non-repudiation;
- Removing the need for a CA can reduce the access management load of the service provider side, as well as decrease the maintenance cost.

In this paper, we propose a solution, based on smart contracts to provide flexible access control in a new business model. In this system we aim to 1) remove the central authority to decrease the maintenance cost and remove the inherited threats of centralized systems, and 2) eliminate the user's extra payment for the service. It is important to mention that in our proposed method, we do not remove the user's payment to the network provider. We assume that the user is subscribed to the network provider, but, while using the specific service through our platform, the user is exempt from payment (e.g., does not use the available internet capacity). To achieve the first goal, the service provider outsources the authorization process to a Blockchain-based platform, and for the second target, after user authorization, the service provider is the party who pays the network provider on behalf of the user.

In this setting, there are three main parties (user, service provider, and network provider) that do not trust each other. For instance, the network provider does not trust the service provider to pay the network cost. In this paper, we aim to resolve the above-mentioned challenges in centralized systems and to provide a trusted connection among the three mentioned parties. Also, to deploy a new business model using the proposed access control solution. The main contributions of this paper are:

1) *Outsourcing the access control* process to an entity without requiring any trusted third party. This feature has two advantages, first, it can decrease the processing load of the service provider (i.e., by simplifying the user registration, access control, etc.), and second, there is no need for any other organization as a trusted third party (i.e., decreases the operation cost).
2) Providing *flexible access control solution*. Smart contracts can be designed to provide authorization possibilities based on the requirements of all parties. These requirements can consider the needs of the network provider (e.g., how much the network provider will be paid), service providers (e.g., which service will be provided at which cost), and users (e.g., which service at which cost and how long is available).
3) Providing *automation* in the access control process. Using smart contracts to handle the access control, there is no need for a central authority. Smart contracts also can make this process completely secure, immutable, and automatic.
4) Make it possible to implement *new business models* for network providers and service providers. Instead of paying twice for a specific service, in our proposed method a user will pay the service provider for its service, and in the next steps the service provider is the entity that pays the network provider on behalf of the user.

The rest of this paper is organized as follows: Section II provides a brief background, followed by a summary of the state of the art in Section III. Section IV outlines the problems of the existing methods and presents our proposed solution. The detailed design and construction of our proposed flexible access control solution to support the new business model is provided in Section V, followed by the experiment and threat analysis in section VI. Section VII provides our conclusions about the proposed method as well as some future research directions.

II. PRELIMINARIES

A. BLOCKCHAIN AND SMART CONTRACT

Blockchain is introduced by Satoshi Nakamoto in 2008 and implemented in 2009 by Bitcoin [1]. Blockchain is a peer-to-peer distributed ledger, immutable, cryptographically secure, permanent, traceable, and transparent technology that is updateable only via consensus among the majority of the nodes that existed on the network [3], [4].

Blockchain is implemented in the structure of linked list, in which, each block is connected to the previous block via its hash. As shown in Fig. 1, using the hash of the preceding block makes it difficult to change the data in a block. Changing a data in one transaction results to change in the hash of that block, and consequently it must be changed in the next block. To change this value, the hash of all next blocks

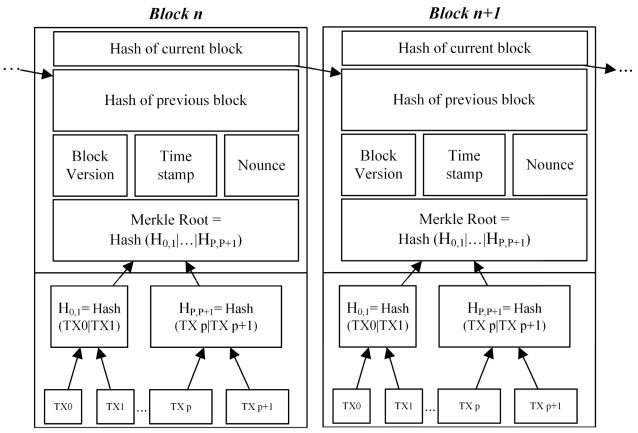

FIGURE 1. The structure of blocks in the blockchain.

must be calculated again. Due to this feature, Blockchain is immutability; it means that any confirmed transaction or data cannot be altered. Blockchain's permanency means that all data can be available at any time, and nothing may be removed from the network [5], [6].

Fitting the transactions inside a block in an order, and publishing a new block, can be done through solving a consensus puzzle [7]. Higher number of nodes to contribute in the consensus proses, results in the lower the probability that an individual miner can monopolize the ability to alter the order of the transactions [8]. Proof of Work (PoW) [9], Proof of Stake (PoS) [10] and Practical Byzantine Fault Tolerant (PBFT) [11] are three well-known examples of existing consensus models. As one of the extensions of blockchain technology, smart contracts introduced by Szabo in 1998 [2] and implemented by Ethereum in 2015 [12], are defined as computerized transaction protocols that execute the terms of a contract on a Blockchain. The main purposes of smart contracts are to satisfy common contractual conditions, minimize exceptions both malicious and accidental, and minimize the need for trusted intermediaries.

B. ACCESS CONTROL

Access control is a security technique that regulates who or what (i.e., subject) can perform which action on resources (i.e., object) [13]. There are a variety of access control mechanisms that are used for different purposes. Some of the most well-known methods are Capability-based Access Control (CapBAC) [14], [15], Discretionary Access Control (DAC) [16], Role-Based Access Control (RBAC) [17], and Attribute-Based Access Control (ABAC) [18]. Elaboration of all of these methods is not in the scope of this paper, and to be focused on our goal, we only explain the ABAC solution.

Fine-grained access management of ABAC makes this solution a primitive candidate for our method. ABAC is fine-grained because it supports different constraints to define the legitimate user. Also, it provides dynamic and context-specific access which makes the resource owner capable to define the access control policy based on their needs. ABAC generally uses Boolean logic in which the validator can verify the subject's eligibility in 0/1 logic based on different attributes. In the rest of this part, the ABAC method is formulated and explained comprehensively.

The attribute-based access control method has four sets of attributes to define the access policy and manage the subject's access to the object. These sets are Subject Attributes (SA), Object Attributes (OA), Environment Attributes (EA), and Action Attributes (AA). Let define all the attributes (AT) of access policy as equation (1):

$$AT = (SA, OA, EA, AA) \tag{1}$$

each set of attributes are defined in below:

$$SA = \{s_1, s_2, \ldots, s_n\}, OA = \{o_1, o_2, \ldots, o_m\},$$
$$EA = \{e_1, e_2, \ldots, e_p\}, AA = \{a_1, a_2, \ldots, a_q\} \tag{2}$$

where $n = |SA|$, $m = |OA|$, $p = |EA|$ and $q = |AA|$. Each attribute in ABAC is defined as a pair ($attribute\_name$, $value$).

Subject by its identifiers such as username, token, and so on. Object attributes distinguish the resources that the subject wants to access; for instance the file name, the network resource, the service name, etc. Action attributes are the actions that can be performed by the subject (e.g., "read", "write", and "execute"). Finally, Environment attributes describe the context in which access is requested (e.g., the time and location from where access is requested, the type of communication channel, etc.).

The request of the subject u to access a resource can be formulated as equation (3). To shorten the formulation, we avoid expanding each attribute set.

$$Req_u = \{SA_r, OA_r, EA_r, AA_r\} \tag{3}$$

Different validators in a system may need a subset of the attributes to validate the subject to perform a specific action based on the access policy. Let's define the attribute subset for validator v as:

$$AT_v = \{SA_v, OA_v, EA_v, AA_v\} \tag{4}$$

The validation result for each attribute set based on the predefined access policy is as (5):

$$V_{SA} = \begin{cases} 1, & \text{if } SA_v = SA_r, \\ 0, & \text{otherwise} \end{cases} \tag{5}$$

The validation process for OA, EA, and AA sets is the same as (5).

Finally, access control result (AR), based on the policies defined by the owner, which is returned as "allow" or "deny" to the user, can be formulated as (6):

$$AR = \begin{cases} 1(allow), & \text{if } V_{SA} = V_{OA} = V_{EA} = V_{AA} = 1 \\ 0(deny), & \text{otherwise} \end{cases} \tag{6}$$

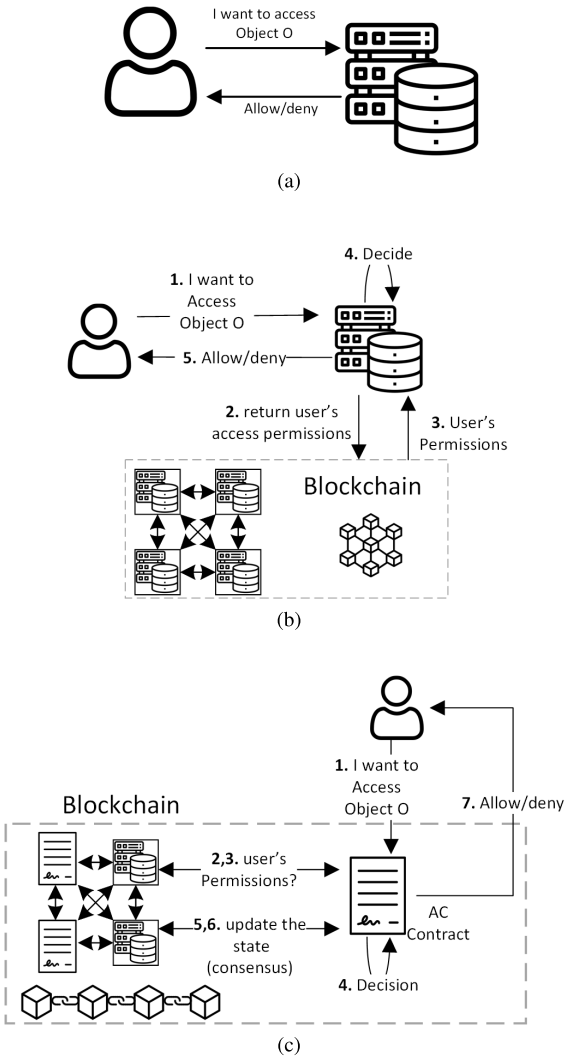

FIGURE 2. Access control approaches. (a) centralized model, (b) using blockchain as a distributed database, (c) using blockchain in all access management procedure.

III. RELATED WORKS

Due to the significance of access control, a variety of solutions are proposed in this field. A considerable part of the literature is dedicated to centralized systems in which there is a trusted central authority that manages users' requests (see Fig. 2 (a)). Despite the low complexity in implementation of these methods and time efficiency of access validation, they suffer from having a single point of failure, low scalability, low availability, and low non-repudiation [19], [20].

Most of these drawbacks can be resolved by a reasonable trade-off between the advantages and disadvantages of using Blockchain technology. As an example, using Blockchain for access control procedure may decrease the time efficiency, while it increases the non-repudiation and removes the single point of failure. Existing Blockchain-based access control methods can be classified into two main categories [21]. In the first category, they use Blockchain as a distributed database to store rules (see Fig. 2 (b)); Even though these methods improve the centralized solutions regarding non-repudiation, rule immutability, etc., they could suffer from a single point of failure. Because in these methods, access decision is made by a central authority outside the Blockchain. In the second category, the Blockchain is used not only as a database, but also as a decision point (see Fig. 2 (c)); it means the rules are stored in the Blockchain, and the access decision can be done using smart contracts. Even though the complexity of implementation of these methods is high, and their time efficiency is marginally lower than the centralized solutions, they can provide high scalability (in terms of the number of users), availability, fault tolerance, the immutability of rule and decision, non-repudiation, and audibility (Table 1).

TABLE 1. General comparison among access control solutions.

AC Model Parameter	Centralized model	Blockchain as DB	Blockchain in whole Process
removing SPoF	Low	Moderate	High
Immutability	Low	High	High
Rule Integrity	Low	High	High
Non-repudiation	Low	Low	High
availability	Low	Moderate	High
time efficiency	High	Moderate	Low
Implementation complexity	Low	Moderate	High
Scalability*	Low	Moderate	High
auditability	Low	Low	High

*Scalability in terms of the number of users. Note that scalability in the number of transaction(s) per second (TPS) in the Blockchain is generally lower than centralized systems [22]

Following, some of the existing Blockchain-based access control solutions are introduced. To the best of our knowledge, using Blockchain for access control in internet-based service provisioning in our proposed business model is not seen in other works. So, in the following we introduce the works that are more related to our work regarding their use-case purpose or implementation.

A. USING BLOCKCHAIN AS A DATABASE FOR RULES

Shafeeq *et al.* [23] proposed an ABAC mechanism in which the object's owner can define the access rules and store them in the Blockchain. In the access request, the owner sends the authorization token to the requester only if the requester meets the conditions defined in the access control policy. Another ABAC method for cloud computing is proposed by Qin *et al.* [24] in which the Central Authority (CA) is responsible for managing the security of the whole system. First, the CA issues an attribute key to the user and adds the validity period of the key in the smart contract. Blend-CAC [19] is a CapBAC mechanism in which, smart contracts are used for storing the access control matrix. Each node interacts with the smart contract through the provided contract address and the Remote Procedure Call interface. Another CapBAC scheme is proposed by Tan *et al.* [25], that Blockchain stores capability set and access logs of the users. Wang *et al.* [26] proposed a fine-grained access control

method using attribute-based encryption (ABE) scheme [27]. At first, the owner encrypts the system's master key and saves it to the Blockchain, and then deploys a smart contract. The user sends the registration request to the owner; the owner manages the secret key for the user and saves it on the Blockchain and sends transaction ID and smart contract's address to the user through a secure channel. These data will be used for the next connections. Moreover, Guo *et al.* [28] proposed a traceable attribute-based encryption method named as TABE-DAC to provide the capability of sharing private data in cloud. This system uses the ABE method. In TABE-DAC the Blockchain is used to store the encrypted key and policies. Ling *et al.* [29], [30] proposed an ABAC model using Blockchain Radio Access Network (BRAN) in which the user and network provider reach an agreement on some parameters such as payment and digitized spectrum assets, written in the smart contract. After the validation of the smart contract concerning the user's balance and network's spectrum assets, the user will be granted time-limited access to the resource, and the access point will automatically receive the payment for the access.

B. USING BLOCKCHAIN IN ACCESS MANAGEMENT PROCESS

Yang *et al.* [31] proposed AuthPrivacyChain in which, the policies and access logs are stored in Blockchain and access control is done by the smart contract. This system is designed to handle the user's request to access the data in the cloud and supports all access control models. RBAC-SC [32] is an RBAC mechanism that consists of a smart contract and a challenge-response protocol. The smart contract is used for the creation, changing, and revoking of the user's role assignments, while the challenge-response protocol is for the authentication of the owner. Fabric-IoT [33] is an ABAC method that uses three kinds of smart contracts to: 1) store the URL of the data that is produced by devices, 2) store ABAC policies, and 3) implement access control methods. The main problem of this method is its low scalability in the context of IoT. Zhang *et al.* [34] proposed a smart contract-based ABAC framework in which multiple contracts are used for access management. The environment attributes in this method are limited to the time attributes. Another similar system for data sharing in IoT is proposed by Sultana *et al.* [35], [36]. In this system, the user sends an access request to a central server. This server redirects the request to the access control contract. If the user's history is clear, the user's permission level is verified, and an access decision is made. The main problem is that both systems have a central point that can be a single point of failure.

IV. PROBLEM STATEMENT AND PROPOSED SOLUTION

To explain the problem of existing internet-based service provisioning, we start with an example depicted in Fig. 3. Assume that user (u_a) wants to watch a video from the website of a service provider (sp_a). The scenario is described as follows:

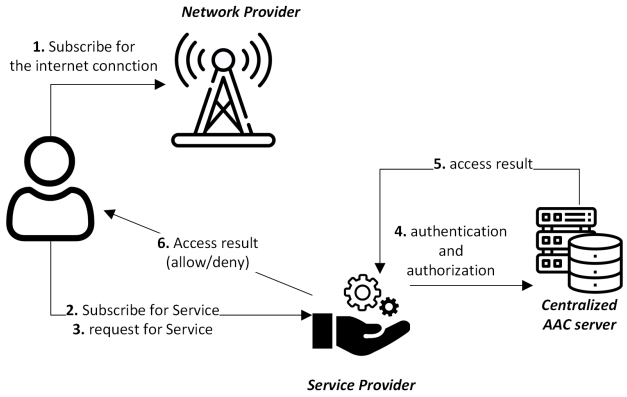

FIGURE 3. An example of a current scenario for service provisioning.

1) u_a subscribes to network provider np_a for the internet connection. In this step, the user's information is stored in the authorization database of np_a for further connections.
2) u_a subscribes to sp_a and pays for the service based on her needs. In this step, the authorization information of u_a is stored in the database of sp_a for the next connections.
3) u_a connects to the website of sp_a using her internet provided by np_a and requests to watch the video.
4) sp_a sends u_a's credentials and identifiers to its centralized access control server.
5) The access control server returns the authorization result to sp_a, which specifies the user's permission to use the service.
6) If u_a is eligible to use the service, she can watch the video. In this scenario, not only the user is subscribed in sp_a, but using the service consumes her share of internet access from np_a.

We have identified several drawbacks in this scenario. First, there is a central authorization server for sp_a, which can be a single point of failure. Moreover, u_a must pay twice to watch a video. The processing loads of sp_a and np_a are high, because they need to authorize the u_a separately for each connection. Finally, there are several general issues in centralized access control solutions which this scenario has inherited. For example, the risk of losing the user's data in a centralized server, denial of an action done by a malicious user (low non-repudiation), the possibility of an attacker changing a user's permissions, and the high maintenance cost of the centralized server.

Addressing these constraints, we propose a novel flexible access control solution to share network and service with users. This solution provides a trustful payment capability to pay network provider based on Blockchain technology, without the need for a central authority. Moreover, this system can eliminate users' extra payments for a service, which is a new business model for both service and network providers. A schematic of the method is presented in Fig. 4.

TABLE 2. Key advantages of the proposed method.

Feature	Description	Advantages
Access control outsourcing	The ability to outsource the access control procedure on the service provider side	• the processing load and maintenance cost
No need to Trusted Third- Party	Thanks to using smart contracts for payments, no trusted third party is needed.	• Reduces the outsourcing and payment costs
Supports a new business model	Provides a new business model for service providers and network providers which can bring new business opportunities.	• Avoids double payment by the user
Trusted payment	After the use of the service has been terminated by the user, the platform sends a trigger to the Blockchain. The Blockchain then pays the network provider from the service provider's account.	• Provides trustful payment in trustless environments.
Access flexibility	The access control attributes are based on the agreement between the user and the service provider.	• Meets the user's, the service provider's and the network provider's needs
Removing Single point of Failure	All of this access control procedure is done via distributed and fault-tolerant smart contracts; there is no central authority for access control.	• High availability • High fault tolerance
Immutability	Thanks to checking the block hash reference in the Blockchain, and the requirement of consensus for any change in the system, the rules, permissions, and user's service level agreement cannot be altered.	• Reduces the probability of misuse • Improves the level of trust in the whole system
Accountability	Traceability of access requests increases the accountability of the system. Also, the users' signatures on contracts and transactions remove the possibility of access or request denial.	• Non-repudiation • Traceability

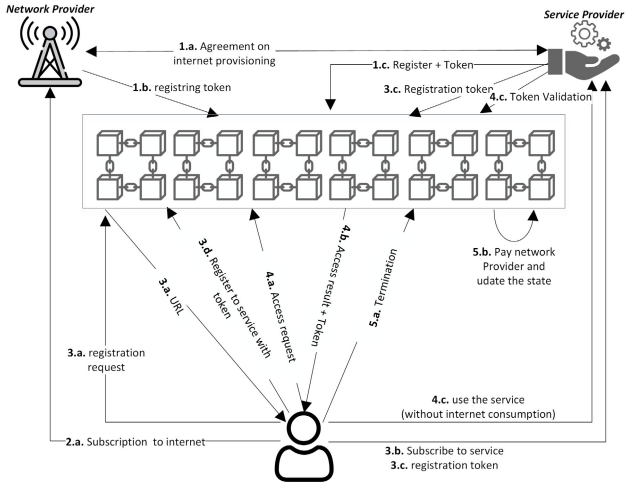

FIGURE 4. Proposed scenario for Internet-based service provisioning via blockchain-based access control solution.

Assume that user u_b wants to watch a video from the website of the service provider sp_b and she uses the network provider np_b. The procedures to follow in this system are outlined below (Fig. 4):

1) Service provider registration in the system:
   a) sp_b and np_b agree on the price of network media.
   b) After reaching a compromise, registration token is sent to Blockchain by np_b.
   c) sp_b is registered in the system using the one-time token.
2) Subscription for the internet:
   a) u_b subscribes to np_b for the internet connection. This process is off-chain. After the user's subscription in np_b, her next connections, is authenticated by np_b in the first step, and then the user can use the system.
3) User subscription and registration for the service:
   a) u_b sends the registration request to the Blockchain and gets the URL of sp_b for registration.
   b) The user is redirected to sp_b for subscription to the service and getting the one-time registration token.
   c) After payment, sp_b sends a one-time registration token to the u_b and also stores it in the Blockchain for further process.
   d) u_b sends the registration request accompanied by the token. If the token is valid and available, the agreement of u_b and sp_b can be added to the Blockchain as a smart contract.
4) User authorization (access control) procedure:
   a) After registration, u_b send the access request to the Blockchain a request for using the service.
   b) The access request result is returned to u_b, and if she is validated, the URL of sp_b accompanied by a one-time access token is sent to the user.
   c) u_b connects to the URL and sends the token. Then, sp_b validates the token stored in the Blockchain. If the request is valid, sp_b can have access to the service and in our example scenario, while u_b watching the video, her internet will not be consumed.
5) Termination and payment:
   a) When u_b finished using the service, the termination trigger will be sent to the Blockchain.
   b) In Blockchain, after receiving the trigger, the payment of np_b will be performed through the account of sp_b.

TABLE 3. Symbols and their descriptions.

Symbol	Description
$Attr_u$	User access attributes
$Attr_{sp}$	The attributes and identifiers of service provider
$Addr_{admin}$	Address of System admin
$Addr_{KAddr}$	$Known\_Addresses$ contract address
$Addr_{sp}$	The address of service provider
$Addr_u$	User's Blockchain address
$Addr_{SPDB}$	$Service\_Provider\_DB$ contract address
$Addr_{ACM}$	$AC\_Manager$ contract address
$Addr_{Reg}$	$Registration$ contract address
$Addr_{User\_Contract}$	User and service provider's contract
$Addr_{Contract_{sp,np}}$	Service provider and network provider's contract
$Addr_{UCDB}$	$User\_Contracts\_DB$ contract
$Addr_{NP}$	Network provider
$CAddr$	All addresses stored in the contract $Addr_{KAddr}$
$Code_{sp}$	The unique code of sp to retrieve its information
$URL_{reg_{sp}}$	The URL for registration in sp
$URL_{con_{sp}}$	The URL for connection to the sp
$Hash(M)$	Representation of hash function ($Keccak256$)

Considering the drawbacks of the current methods, the advantages of this system are listed in Table 2. Before focusing on a detailed presentation of the system, it is important to mention that in this method we don't eliminate the user's subscription fee in the network provider. Rather, we want to remove the user's extra payment for using that specific service.

V. SYSTEM DESIGN

Our proposed method consists of four main steps: 1) setup; 2) registration; 3) access control, and 4) payment. In the setup, we deploy all the unique contracts that are needed for the further steps. Next, in the registration step, the service provider and the user register in the system. After registration, in the access control step, the user requests authorization to use the service without internet consumption, based on her desired connection level and service plan. Finally, after using the service, the termination feedback is sent to the Blockchain to pay the network provider. All symbols used, are listed in Table 3. Our general assumptions for deploying the system are listed below:

- The connections between the user, Blockchain, service provider, and network provider are secure;
- There is only one network provider in the system and the user is subscribed to the network provider;
- User is authenticated to use the Blockchain (as mentioned in step 2.a in Fig. 4);
- User is limited to use the service at the granularity of their contract (e.g., just watch one episode of a series, or have a call lasting 1 hour, etc.);

Before describing the steps, we first introduce the smart contracts used in the on-chain part of our system.

A. DESIGN SMART CONTRACTS

Following, are the designed smart contracts:

1) *Known_Addresses*: six contracts in our system are each deployed once. *Known_Addresses* stores the addresses of these single contracts, enabling them to collaborate safely. In this contract, the names of contracts are mapped to their addresses:
$$CAddr \xleftarrow{name_{contract}} Addr_{contract}$$
$name_{contract}$ is in string format and has a set of predefined values (Table 4 in the setup step). It is important to mention that we designed this contract to 1) avoid using hardcoded addresses and resolve the maintainability [37] defects of smart contracts, and 2) having a list of predefined addresses that let us state-specific requirements per functions.
2) *SP_NP_Contract*: A unique contract between the network provider and a service provider. Since the service provider must pay for the user's connection, the Internet price per service unit is declared in this contract. Also, the amount of network providers owe from the service provider is indicated in *Owe*.
3) *Service_Provider_DB* is a distributed database to store the attributes of all registered *sp* as a mapping of $Code_{sp}$ to a tuple of five parameters:
$$Attr_{sp} \xleftarrow{Code_{sp}} (Addr_{sp}, URL_{reg_{sp}}, URL_{con_{sp}} \\ Addr_{Contract_{sp,np}}, Balance) \quad (7)$$
$Code_{sp}$ is a unique identifier to retrieve the $Attr_{sp}$. We use an incremental function to generate $Code_{sp}$ (i.e., $Code_{sp} = i$, where i is the number of registered *sp*s). two URLs are the web addresses for user's registration or connection, and $Addr_{Contract_{sp,np}}$ is the address of *SP_NP_Contract* between *sp* and *np*.
4) *User_Contract*: this is a unique contract between a user and a specific service provider to which that the user is subscribed to. These contracts store the user's access attributes, which are used to check the user's eligibility to use the service. The stored attributes of user u in these smart contracts are:
$$Attr_u \xleftarrow{Addr_u} (ExpT, Balance, Price_{service})$$
$Addr_u$ is the address of u in the Blockchain. $ExpT$ is the expired time of the subscription of u in sp. In the first registration $ExpT$ is:
$$ExpT = block\_time + (t \times 86400)$$
where block time is the Unix timestamp of the contract deployment and t is the number of subscription days. *Balance* indicates the remaining balance of u to use the service in sp, and $Price_{service}$ is the price of using a predefined unit of the service (e.g., the price of each video available on the website of sp). In this contract, there is a flag as $Active_{User\_Contract}$ which indicates if the user is currently using the service or not.
5) *User_Contracts_DB* stores the $Addr_{User\_Contract_{u,sp}}$ for each pair (u,sp). To store these data, we defined a structure for a user's contract as a tuple of two parameters:
$$UC\_str = (Addr_{User\_Contract_{u,sp}}, Code_{sp}).$$
$Addr_{User\_Contract_{u,sp}}$ is the address of the contract deployed between u and sp. A user may have registered with several service providers, and so to have all the contracts of u, we map the $Addr_u$ to the set of UC_str:

TABLE 4. Initial parameters in known_addresses.

Identifier	Parameter	Address of
'Owner'	$Addr_{admin}$	System admin
'KAddr'	$Addr_{KAddr}$	$Known\_Addresses$ contract
'SPDB'	$Addr_{SPDB}$	$Service\_Provider\_DB$ contract
'ACM'	$Addr_{ACM}$	$AC\_Manager$ contract
'REG'	$Addr_{Reg}$	$Registration$ contract
'UCDB'	$Addr_{UCDB}$	$User\_Contracts\_DB$ contract
'NP'	$Addr_{NP}$	Network provider

$$All\_UC = \xleftarrow{Addr_u} \{UC\_str_{u,sp_1}, \ldots UC\_str_{u,sp_n}\}$$

where n is the number of service providers to which the user is registered.

6) *Registration* aims to register the users and service providers in the system. To do so, after the registration of *sp*, its data is added to the *Service_Provider_DB* contract, and after registration of *u* in service *s*, the address of deployed *User_Contract* will be added to the *User_Contracts_DB*.
7) *AC_Manager* handles the access control and the payment. When the user requests access to a service, this contract validates the request based on the user's access control attributes stored in $User\_Contract_{u,sp}$. Also, after the user terminates their use of the service, this contract manages the payment to *np*.

B. SETUP STEP

The first step in the deployment of our system is the setup. This is where we deploy the smart contracts, store their addresses in *Known_Addresses*, a tamper-proof safe contract, and initiate the system by the required variables. First of all, the system admin deploys the *Known_Addresses* contract. Initially, the contract stores $Addr_{admin}$ as the owner. After deploying the *Known_Addresses*, the procedure for the deployment of other single contracts is as follows. The deployment of *Service_Provider_DB*, is given as example. To deploy *Service_Provider_DB*, the admin uses the $Addr_{KAddr}$ as the initial parameter. The constructor of the *Service_Provider_DB* calls a function of $Addr_{KAddr}$ by sending a dedicated name (i.e., $'SPDB'$) to add its address. *Known_Addresses* checks the input and if there is no other address stored with that name in *CAddr*, it stores the new address. After deployment of all single contracts, the stored values in the *Known_Addresses* contract are as Table 4. Note that, the $Addr_{NP}$ is a predefined address that is added by the admin to the system and cannot be changed.

C. REGISTRATION STEP

In this step, the user or service provider is registered in the system. The registration process for the user and also for the service provider is given below.

1) SERVICE PROVIDER REGISTRATION

In this step, a service provider *sp* will be registered in the system. Since the *sp* should pay the network provider (*np*) on behalf of the user, registration of service providers is done after the agreement of *np* and *sp* for the internet and medium price. Following is the process of registration (see Fig. 5).

Algorithm 1 *Check_Caller*

Input name
Output Boolean

1: $Address \leftarrow$ instance of $Addr_{KAddr}$
2: value $\leftarrow Address.getAddress(name)$
3: **return** $(msg.sender == value)$

1) *sp* connect to *np* and ask for the registration token. In this process, *sp* and *np* agree on a price (off-chain) and they define the URLs which are free to connect.
2) *np* generates a one-time registration token for the *sp*, using the agreed price and the address of both parties. The token (T) is:

$$T = Hash\left(Addr_{NP} \parallel Addr_{sp} \parallel price_{internet}\right) \quad (8)$$

the hash function is *Keccak*256 [38].
3) *np* sends the generated token to *Registration* contract. This contract checks the sender of the token (Algorithm 1) and if it is $Addr_{NP}$, the token is added to the list of valid tokens for service provider registration.
4) *sp* calls *AddNewSP*() from the *Registration* (Algorithm 2). As shown in Algorithm 2, the *Registration* checks the validity of the token by generating another token with the same algorithm (Algorithm 2, L:2) and compares it with the token which is sent by the *np* in step 3. If both tokens are the same and the token is valid, firstly it revokes the token and generates $Code_{sp}$, the unique identifier for *sp*.
5) *Registration* contract deploys a *SP_NP_Contract*, and stores the agreement price of *sp* and *np* in this contract. Note that, because this contract is deployed after checking the validity of the token, and the $price_{internet}$ is a parameter of the token, then it cannot be changed by *s*p. To deploy the *SP_NP_Contract*, the $Addr_{KAddr}$ is sent to the contract. If the caller of the contract is *Registration*, the *SP_NP_Contract* is deployed and its address is sent to the *Registration*.
6) *Registration* gets the $Addr_{SPDB}$ and calls *AddSP*() from *Service_Provider_DB* to insert service provider's attribute in the list of registered service providers (Algorithm 2, L:13, Algorithm 3).
7) As shown in lines 1 to 6 of Algorithm 3, there are two requirements to execute *AddSP*(). First, this function must be called by *Registration*. To check this, *Check_Caller* (Algorithm 1) is called. Second, another service provider with $Addr_{sp}$ must not exist (Algorithm 3, L: 4-8). If both of the requirements were satisfied, *sp* will be added to the system (Algorithm 3, L: 9)

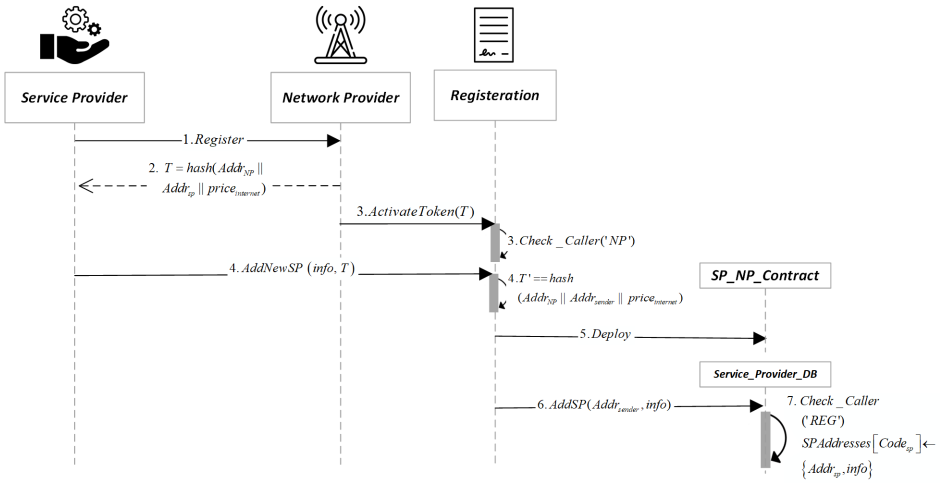

FIGURE 5. Service provider registration procedure.

Algorithm 2 *AddNewSP*

Input $Name$,$URL_{reg_{sp}}$, $URL_{con_{sp}}$,$price_{internet}$, $Token$
Output Boolean

1: $Addr_{NP} \leftarrow Known\_Addresses.getAddress('NP')$
2: $T' = hash(URL_{reg_{sp}} \parallel msg.sender \parallel price_{internet})$
3: **if** $T' \neq Token$
4: **throw**
5: **end if**
6: **if** $!validSPtoken[Token]$
7: **throw**
8: **end if**
9: $validSPtoken[Token] \leftarrow false$
10: $Addr_{Contract_{sp,np}} \leftarrow deployNewContract(price_{internet})$
11: $Code_{sp} = counter_{registered}$
12: $counter_{registered} \mathrel{+}= 1$
13: $Service\_Provider\_DB.AddSP\ (msg.sender, URL_{reg_{sp}}, URL_{con_{sp}}, Code_{sp}, Addr_{Contract_{sp,np}})$

Algorithm 3 *AddSP*

Input $Addr_{sp}$,$URL_{con_{sp}}$,$URL_{reg_{sp}}$,$Code_{sp}$,$Addr_{Contract_{sp,np}}$
Output Boolean

1: **if** $!Check\_Caller('REG')$
2: **throw**
3: **end if**
4: **while** $(i \leq Code_{sp})$
5: **if** $SPAddresses[i].address \neq null$
6: **throw**
7: **end if**
8: **end while**
9: $SPAddresses[Code_{sp}] \xleftarrow{\text{add}} (Addr_{sp}, URL_{reg_{sp}}, URL_{con_{sp}}, Addr_{Contract_{sp,np}})$

2) USER REGISTRATION

User registration in the system is done by registering for a service of the specific service provider (*sp*). To register the user *u*, after selecting the *sp*, *u* asks *Registration* to get $URL_{reg_{sp}}$. *Registration* sends $URL_{reg_{sp}}$ to the user, and then *u* can subscribe to a service and get the one-time registration token to store a unique contract in the Blockchain. The user registration steps (Fig. 6) are given next.

1) *u* sends the subscription request to $URL_{reg_{sp}}$ accompanied by $Addr_u$. *u* Then, selects her preferred subscription plan and performs an off-chain payment. This plan defines $ExpT$, $Price_{service}$, and the user's payment indicates initial $Balance$.
2) *sp* generates a one-time registration token and sends it to *u*. The token (T) is the hash amount of the agreement of *u* and *sp* on access parameters as well as $Addr_{sp}$ and $Addr_u$. the token is generated as:

$$T = Hash\begin{pmatrix} ExpT \parallel Balance \parallel Addr_u \parallel \\ Price_{service} \parallel Code_{sp} \parallel Addr_{sp} \end{pmatrix} \quad (9)$$

3) *sp* sends the T to *Registration* contract to add it to the valid registration tokens. After checking the existence of the service provider, *Registration* adds the token to valid registration tokens. Note that, adding the token in Blockchain, prevent the user from misbehavior to add arbitrary data in the system.
4) The function *RegisterNewUser*() is called by *u* from *Registration* with the agreed parameters for access accompanied by $Code_{sp}$ and T (Algorithm 4).
5) *Registration* generates a hash amount (T') based on received data and with the same algorithm of *sp*. The generated hash is calculated as Algorithm 4, L:3. It is important to mention that all addresses (e.g., $Addr_{SPDB}$) are fetched from *Known_Addresses*, but to make the figures simple, we avoid mentioning the unnecessary details.

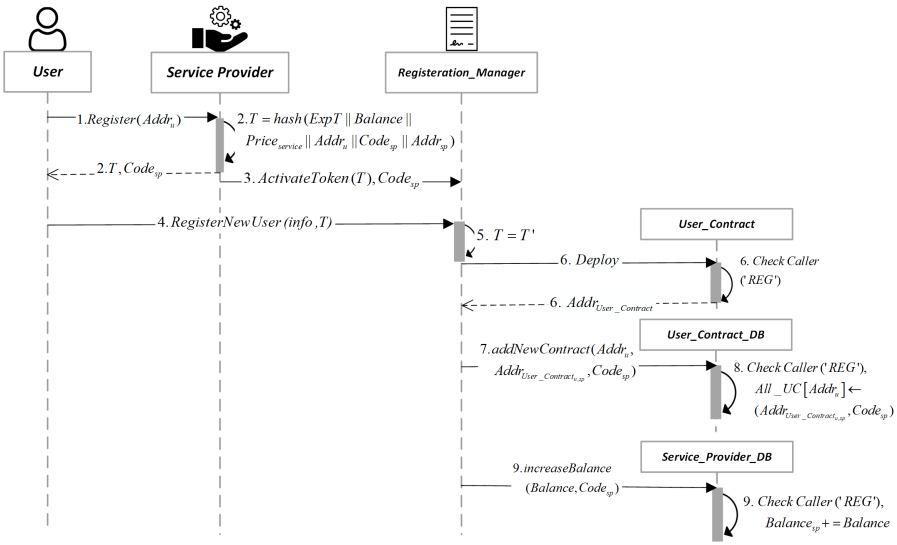

FIGURE 6. User registration procedure.

6) If $T == T'$ and the token exists in valid registration tokens, *Registration* deploys a contract between u and sp ($User\_Contract_{u,sp}$) with pre-mentioned parameters. To do so, decided parameters and $Addr_{KAddr}$ is sent to the constructor of $User\_Contract_{u,sp}$.
7) The newly deployed contract must be added in the *User_Contracts_DB*. So, *AddNewContract*() is called by *Registration*, and $Addr_u$, $Code_{sp}$ and $Addr_{User\_Contract_{u,sp}}$ are sent as inputs.
8) The permission of adding a new contract in the database is limited to *Registration*, therefore $Check\_Caller('REG')$ is called and if this requirement passed, the new contract will be pushed to the list of user's contracts.
9) To finish the registration procedure, the available balance of sp must be increased by the user's payment. To do so, *Registration* calls the *increaseBalance*() from *Service_Provider_DB*. The function, after checking the caller (Algorithm 1) and make sure that it is *Registration*, then increases the balance of sp by *Balance*.

Algorithm 4 *RegisterNewUser*

Input $ExpT$, $Balance$, $Price_{service}$, $Code_{sp}$, $Token$
Output Boolean

1:$Addr_{SPDB} \leftarrow KnownAddress.getAddress('SPDB')$
2.$Addr_{sp} \leftarrow Service\_Provider\_DB.getAddress(Code_{sp})$
3: $T' = Hash(ExpT \parallel Balance \parallel Price_{service} \parallel Addr_{sp} \parallel msg.sender)$
4: **if** $T' \neq Token$
5: **throw**
6: **end if**
7: **if** !$validUsertoken[Token]$
8: **throw**
9: **end if**
10: $validUsertoken[Token] \leftarrow false$
11: $Addr_{User\_Contract_{u,sp}} \leftarrow deployNewContract(ExpT, Balance, Price_{service}, Addr_{KAddr})$
12: $SPDB.increaseBalance(Addr_{sp}, Balance)$
13: $User\_Contracts\_DB.addNewContract(msg.sender, Code_{sp}, Addr_{User\_Contract_{u,sp}})$

D. ACCESS CONTROL

In this step, the registered user u requests for connection to the service of sp. The Attribute-based Access Control solution to share the service with the user without internet consumption is given in the following steps(Fig. 7).

1) by sending $Code_{sp}$, u calls the *AccessToService*() function of *AC_Manager* and a hash amount (Algorithm 5). As explained in Fig. 7, to have a trustful connection between u and sp, after authorization, a token is sent to the user for further connection. To generate the unique and unrecoverable token, in the first step of connection, the user generates a random nonce and calculates its hash with $Keccak256$.
2) As the first requirement, the user must have a deployed contract with sp. So, *AC_Manager* gets $Addr_{User\_Contract_{u,sp}}$ from *User_Contracts_DB* (Algorithm 5, L: 1-4).
3) Using $Code_{sp}$ *AC_Manager* gets the $URL_{con_{sp}}$ and the $Addr_{Contract_{sp,np}}$.
4) One of the environmental attributes to validate the user's access to the service is the balance of sp. This attribute guarantees that as long as np provides the network, the payment is performed safely based on their

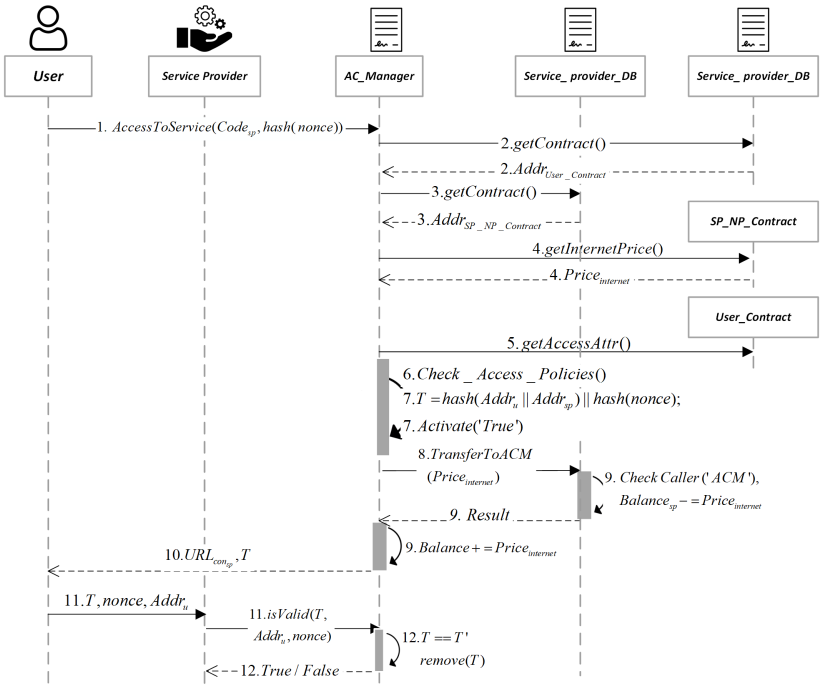

FIGURE 7. The attribute-Based Access control procedure.

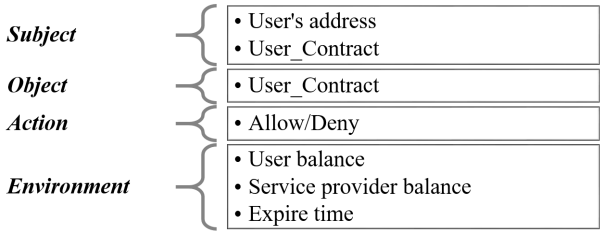

FIGURE 8. Access control attributes.

agreement in *SP_NP_Contract*. So, the $Price_{service}$ is fetched.

5) the user's access attributes (i.e., $ExpT$, $Price_{service}$, $Balance$) is fetched from $User\_Contract_{u,sp}$ by *AC_Manager* (Algorithm 5, L: 8).
6) To connect the user to the service, all the subject, object, and environment attributes must be validated. The attributes are shown in Fig. 8. Assume that $V_{SA,OA}$ is the validation result for the subject and object attributes and that V_{EA} is the validation result for the environment attributes. The final validation result based on the predefined policy by the service provider is V as equations (10) - (12). $V_{SA,OA}$ is validated in step 4. Algorithm 5, lines 9-11 ensures that the system can safely pay *np* after using the service, and lines 12-14 ensures that *u* cannot use the service more than allowed by her payment. This limitation removes the possibility of decreasing the $Balance_{sp}$ to less than the sum of the balances of all its active users. This limitation also implicitly guarantees the safe payment of *np*. Another parameter to check is to be sure that the user's contract is not active at the moment (i.e., if it is active, it means that the user is not terminated the previous connection, and the network provider is not paid for the last service (Algorithm 5, L: 18-20)).

$$V_{SA,OA} = \begin{cases} 1, & \text{if } Addr_{User\_Contract} \neq 0, \\ 0, & \text{otherwise} \end{cases} \tag{10}$$

$$V_{EA} = \begin{cases} 1, & \text{if } Balance_{sp} \geqslant Price_{internet}, \\ & Balance_u \geqslant Price_{service} \\ & CurrentTime > ExpT \\ 0, & \text{otherwise} \end{cases} \tag{11}$$

$$V = \begin{cases} 1(allow), & \text{if } V_{SA,OA} = V_{EA} = 1, \\ 0(deny), & \text{otherwise} \end{cases} \tag{12}$$

7) If validation results were 1, *AC_Manager* generates a one-time token for user connection and adds it to the list of valid tokens. The token is the hash amount of user's and service provider's addresses, concatenated with the hash amount of nonce which is sent by the user in the first step of connection. the generation of token T is done as follows:

$$T = Hash(msg.sender \parallel Addr_{sp}) \parallel hash(nonce)$$

In this step also *AC_Manager* calls *Activate*() function with $'true'$ parameter to set the flag *active* of $User\_Contract_{u,sp}$ to indicate that the user is using the service. *Activate*() function is limited to be able to call only by *AC_Manager*. (Algorithm 5, L: 21, 22).

Algorithm 5 *AccessControl*

Input $Addr_u$, $Code_{sp}$, $Hash(nonce)$
Output Boolean

1: $Addr_{User\_Contract_{u,sp}} \leftarrow$ $User\_Contracts\_DB.getContractAddress(Code_{sp}, msg.sender)$
2: **if** $Addr_{User\_Contract_{u,sp}} == 0$
3: **throw**
4: **end if**
5: $Balance_{sp} \leftarrow Service\_Provider\_DB.Balance(Code_{sp})$
6: $SP\_NP\_Contract \leftarrow$ $Service\_Provider\_DB.SP\_NP\_Contract(Code_{sp})$
7: $Price_{internet} \leftarrow SP\_NP\_Contract.getPrice()$
8: $Attr_{UC} \leftarrow User\_Contract_{u,sp}.getAttributes()$
9: **if** $Balance_{sp} < Price_{internet}$
10: **throw**
11: **end if**
12: **if** $Balance_u < Price_{service}$
13: **throw**
14: **end if**
15: **if** $Block.timestamp > ExpT$
16: **throw**
17: **end if**
18:**if** $Addr_{User\_Contract_{u,sp}}.active$
19: **throw**
20: **end if**
21: $Token = Hash(msg.sender \parallel Addr_{sp}) \parallel hash(nonce)$
22. $Addr_{User\_Contract_{u,sp}}.active \leftarrow true$
23: $TransferToACM(Addr_{sp}, Price_{internet})$
24: **return** *true*

8) To be able to pay the *np*, we need to design a payment procedure that is trustful; it means *np* can be sure that its payment will be performed after a successful connection. To satisfy this requirement, after user validation, *AC_Manager* blocks $Price_{internet}$ in the contract, automatically. So, three parties (*u*, *np*, and *sp*) who do not know each other but accept the *AC_Manager*, can safely trust the process. To block the token, *AC_Manager* calls *TransferToACM*() function of *Service_Provider_DB* to virtually transfer $Price_{internet}$ from $Addr_{sp}$ to $Addr_{ACM}$.
9) *TransferToACM*() is limited to be able to call only by *AC_Manager*. To pass this requirement, the $Check\_Caller('ACM')$ is called by the function. If it passed, The $Price_{internet}$ will be decreased from *sp*. Then this amount is increased in the balance on the *AC_Manager* (Algorithm 5, L: 20).
10) The $URL_{con_{sp}}$ and the token are sent to the user.
11) The user sends a connection request to the $URL_{con_{sp}}$ accompanied by $\{T, nonce, Addr_u\}$. Note that, in this step, the user sends the nonce itself (not it's hash). *sp* sends a validation request to the *AC_Manager* via these parameters.
12) To check the validity of the token *AC_Manager* calculates T' as following:

$$H = Hash(nonce)$$
$$T = Hash(Addr_u \parallel msg.sender) \parallel H$$

If the T is equal to T' ($T == T'$), and the token is valid, *AC_Manager* will answer to the *sp* that the token is valid, and also it will remove the token from valid tokens. If the token is valid, the user can access the service. It is important to mention that the token is unrecoverable, because, in step 1, the user sends the hash amount on the nonce, and in step 11, sends the nonce itself. In this solution, the attacker should be aware of the nonce, to be able to act maliciously.

E. PAYMENT

After *u* has used the service and confirmed the termination, the payment to *np* must be performed. The payment procedure (Fig. 9) is listed below:

1) *u* calls the *Termination*() function of *AC_Manager* and sends $Code_{sp}$ as the input.
2) *AC_Manager* gets the $Addr_{User\_Contract_{u,sp}}$ from *User_Contracts_DB*. Also, it gets $Addr_{Contract_{sp,np}}$ from *Service_Provider_DB*.
3) Executing the rest of the payment procedure depends on the trueness of the flag *Active* in $Addr_{User\_Contract_{u,sp}}$. So, this parameter and $Price_{service}$ are fetched from $Addr_{User\_Contract_{u,sp}}$.
4) *AC_Manager* changes the active flag to *'false'* and updates the user's available balance in the contract. To do so, it calls *Activate*() function with *'false'* parameter and *updateBalance*() function. The permission of calling *updateBalance*() function is limited to *AC_Manager*. So, $Addr_{User\_Contract_{u,sp}}$ calls the $Check\_Caller('ACM')$. If the requirement passed, the balance of the user is decreased by $Price_{service}$. It also changes the flag to *false*.
5) To pay the *np* using the blocked money in *AC_Manager*, firstly the contract must know how much to pay. So, it gets the $Price_{internet}$ from *SP_NP_Contract*.
6) Then *AC_Manager* increases the balance of *SP_NP_Contract* by $Price_{internet}$ and decreases this amount from the balance of *AC_Manager*. It is important to mention that the network provider can call the checkout function of *SP_NP_Contract* to withdraw its balance (i.e., *Owe*) from that. The *Checkout*() function checks the caller, and if it is the *np*, it transforms the amount *Owe* from $Addr_{sp}$ to the $Addr_{NP}$.

F. UPDATE

This subsection is not part of the regular procedure of access control. However, we believe that some updates will be needed when using the system. Two scenarios are presented.

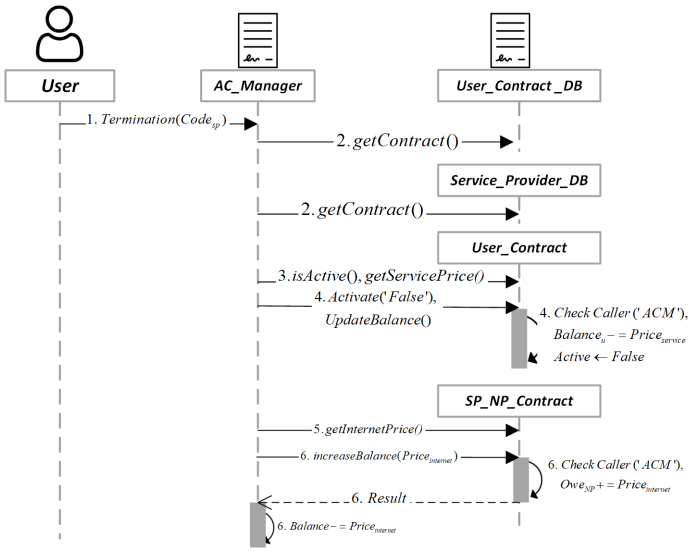

FIGURE 9. Payment procedure.

Algorithm 6 *update*

Input $\_ExpT, \_Balance, \_Price_{service}$
Output Boolean

1: **if** $!Check\_Caller('REG')$
2: **throw**
3: **end if**
4. **if** $ExpT > block.timestamp$
5: $ExpT \leftarrow ExpT + (\_ExpT \times 86400)$
6: **else**
7: $ExpT \leftarrow block.timestamp + (\_ExpT \times 86400)$
8: **end if**
9: $Balance \mathrel{+}= \_Balance$
10:$Price_{service} = \_Price_{service}$
11: **return** *true*

1) UPDATE USER CONTRACT

Assume that user u aims to renew her contract with sp. u is already registered and has $Addr_{User\_Contract_{u,sp}}$. The update procedure is similar to registration and is as follows.

First, u sends the update request to *Registration* contract. *Registration* checks the existence of a contract between sp and u. if the contract exists, $URL_{reg_{sp}}$ will be sent to the user. The next steps are the same as Fig. 6, unless in two steps. 1) step 6 of user registration; In the update, instead of deploying the contract, the *update*() function is called by the Registration (Algorithm 6), and 2) Steps 7, 8 of Fig. 6 is not needed. As shown in Algorithm 6, to update the expiration time, we check the user's current expiration time, and based on that, the $ExpT$ is updated. Balance will increase by the new balance and $Price_{service}$ is also updated to new values.

2) UPDATE THE ADDRESSES

As mentioned before, one of the well-known defects of smart contracts is maintainability [37]. To overcome this problem,

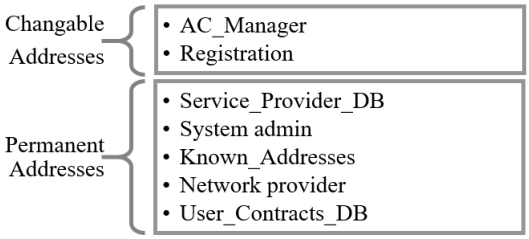

FIGURE 10. Permanent and changeable addresses.

we provided a solution to replace an old contract with a new one. The changes can be used to upgrade the system functionality, fix some of the bugs, or make it compatible with new needs. Fig. 10, lists the contracts with the addresses that can be changed and those that cannot. Contracts that are used as distributed databases are permanent and cannot be changed because they store the user's and the service provider's data. *AC_Manager* and *Registration* are changeable and the replacement procedure is as follows:

1) Admin calls the function *UpdateAddress*() from *Known_Addresses*.
2) *UpdateAddress*() checks the caller of the function, and if it is the admin, it verifies that the request is to change the address of *AC_Manager* or *Registration*. If both requirements are passed, it adds a new variable in *CAddr* (*REG* is given as example):

$$\begin{aligned} CAddr &\xleftarrow{'Temp'} CAddr['REG'] \\ CAddr &\xleftarrow{'REG'} 0 \times 0 \end{aligned} \tag{13}$$

Having a temporal address in the list helps the system to keep the old address in case of any further problem, and writing the address of zero on the existing parameter allows the admin to generate other contracts of

<*Contract_name*>. Admin can deploy the new contract. As mentioned before, the constructor of the contract will add itself to the *Known_Addresses*.

3) When the updated system has proved to function, the admin can remove the temporal address.

VI. EXPERIMENTS

To evaluate the feasibility of the proposed method, we designed a use case in which the user uses her mobile connectivity (i.e,. cellular network) to use the system. As mentioned earlier, we assume that the user's authentication is done by the network provider. To do so we simulated a cellular network environment that the user connects. In this section, we describe the testbed implementation, the architecture of the network, and the implementation of smart contracts. Then we present the performance of the proposed method. We also analyze the security of the system through several thread scenarios.

A. EXPERIMENT ENVIRONMENT

To simulate the complete procedure of the user connection, access control, and termination of the connection, we need to deploy an environment that supports the connection of the user to the cellular network as well as a Blockchain.

Table 5 lists the hardware and software specifications of the test environment. It is important to mention that, as stated in step 2.a of Fig.4, the user's connection to the system is done after the authentication of the network provider. The user authentication is needed to guarantee that the user is already subscribed to the network. Due to the widespread usage of 5G networks [39], we choose this technology as our testbed.

The cellular networks consist of two main parts: Radio Access Network (RAN) and Core Network (CN). To simulate the 5G network, we used OAI (Open Air Interface) consisting of RAN and CN [40]. OAI is open-source software that implements cellular network functions of the RAN named by OAI-RAN and the core named by OAI-CN. To build the RAN part (i.e., the network provider's base station), the OAI-RAN was executed on a PC that is connected to an SDR (software-defined radio) board for radio communications through a USB3 interface. To implement our proposed method, we focused on the CN and implemented a gateway that will send the request from the CN to the Blockchain and authenticate the user. In this system, the user's request is sent to the 5G RAN (Fig. 11) and will be authenticated by the gateway (Fig. 11, 5G Core).

Authentication of the user's equipment and the network provider is done by the AKA procedure (i.e., the existing authentication model in the cellular networks [40]). Once the user has been authenticated, CN sends the user's request to the Blockchain through the gateway (5G core in Fig. 11).

In the Blockchain, we utilized a private Ethereum that allows us to assess the system's performance using a variety of parameters. The smart contracts are written in Solidity language [41]. Solidity supports complex variables such as the mapping of structures, and it provides the capability of defining different requirements per function; also, it is a Turing-complete language (i.e., can be used to simulate any Turing machine).

TABLE 5. Environment specifications.

Part	Parameter	Specification
	Hardware	
5G RAN and Core	CPU	Intel quad-core at 2.9 GHz
	RAM	16 GB
	SDR board	USRP B210
Blockchain	CPU	Intel i7 Dual-core 1.6GHz
	RAM	6 GB
	Hard Disk	128GB SSD
	Software	
5G RAN	OAI-RAN	master branch release v1.1.0
	OS	Ubuntu 16.04-low latency kernel
Blockchain	OS	Xubuntu
	Ganache-cli	6.12.2
	Ganache-core	2.13.2
	Web3j	1.4.1
	Solc	0.8.2

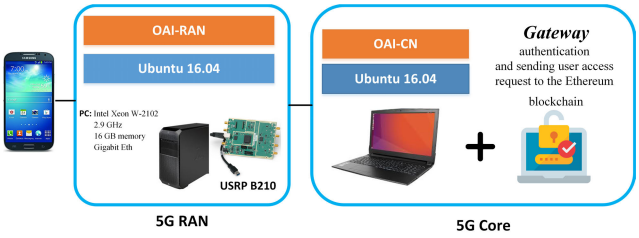

FIGURE 11. The architecture of the testbed.

B. PERFORMANCE ANALYSIS

The performance analysis of the proposed method is done in two parts: 1) By assessing the execution and transaction costs of the transactions and processes, and 2) By evaluating the scalability of the system in terms of the increasing number of concurrent connection requests.

1) TRANSACTION COST

For the first analysis, we calculated the *GAS* price of the processes. The *GAS* is the fee that must be paid by the sender to submit transactions to the Ethereum network. Calculation of the *GAS* is formulated in [42]. The cost that is mentioned in this part is the cost of sending a transaction of a contract to the Ethereum blockchain (i.e., transaction cost) [42]. The *GAS* cost is defined in *Gwei* (i.e., as $10^{(-9)}ETH$). Table 6 shows the *GAS* cost in different processes (i.e., in setup, user registration, service provider registration, access control, and payment). It is important to mention that in private or consortium Blockchains, no currency is required to process or validation of transactions [43]. To make the costs in the public network more tangible, we calculated the price in *USD* (on 8/04/2021 the average price over 24H was $1,933.91 for each *ETH*).

TABLE 6. GAS price of processes and transactions of the proposed method.

Process		Transactions	Tx cost	Price ($)
Registration	Service Provider Registration	Add valid token	21374	0.04
		Registration and deployment of contract	369490	0.73
	User Registration	add valid token	28079	0.05
		Registration and deployment of contract	485536	0.96
Access Control	Access validation	ABAC and token generation	104227	0.20
		Token checking by SP	14661	0.02
Payment		Pay the network provider	81228	0.16

2) SCALABILITY

To measure scalability, we assess the throughput, and the latency [44], [45]. Throughput can be calculated as:

$$Throughput = \frac{|Tx|}{t} \tag{14}$$

where Tx is the set of all transactions, $|Tx|$ is the number of transactions, and t is the required time to handle them. latency is another measure similar to throughput which clarifies the average required time to handle one transaction.

$$Latency = \frac{t_f - t_s}{|Tx|} \tag{15}$$

where t_f is the finishing time of simulation and, t_s is the time in which we started the simulation. It is important to mention that simulation here means the time of starting to send concurrent requests and finishing to receive the transaction receipt of all of them.

The scalability of the system can be defined as changes in throughput or latency when altering a parameter [46]. Different parameters are used to measure the throughput and latency (see Table 7). It is important to mention that adding the one-time registration token for the service provider cannot be executed concurrently, because only the network provider is eligible to add the token in the Blockchain, and we designed one network provider in the system. One node cannot send concurrent requests to the system without waiting for the receipt of the previous transaction.

Fig. 12 (a-e) depicts the latency of the system for different block sizes (BS) and block times (BT). This process is also done to assess the throughput, which is shown in Table 8. As shown in Fig. 12 (a-e), systems latency is almost stable for $C \geq 200$ (i.e., when concurrent requests are more than 200). Therefore, based on the definition of scalability, we can claim that the system is scalable and can maintain low latency, based on system configuration in a large-scale request environment.

To analyze the system throughput Table 7 is provided. As shown in the table, by increasing the BS and decreasing the BT to a threshold of the system requirement, the throughput will increase (see Table 7 for $BS = 30$ and $BT = 5$). For example in $BS = 30, BT = 5$ (low complexity of consensus puzzle and a large number of transactions fit in each block) the throughput has the highest amount, while $BS = 15, BT = 15$ (high complexity of consensus puzzle and the small number of transactions fit in each block) has the lowest throughput. But an important issue in this configuration is the system's security; it means decreasing the block time of the Blockchain results in an easier consensus puzzle, that can increase the risk of integrity violation in the PoW model. Based on the requirements of the system, the admin must specify a trade-off between consensus complexity and throughput.

TABLE 7. Parameters of analysis.

Parameter	values	description
Concurrent requests (C)	50, 100, 200, 500, 700	The number of the virtual clients in the system. all requests are sent concurrently and all the virtual clients are full nodes that participate in consensus
Block Size (BS)*	15, 30	The number of transactions fitting in one block.
Block time (BT)	5, 10, 15	The difficulty of consensus puzzle which leads to extraction of blocks in predefined time. The average block time for Ethereum is 13.12 seconds on 08/04/2021.

*in real-world public Blockchains, this parameter is higher, but due to the limitation of the libraries, the mentioned values are tested.

It is important to mention that in $BS = 15, BT = 15$, due to several exceptions of web3j library, we could not reach a precise result for parameters in concurrent request 700. Based on our observation, because the difficulty of consensus puzzle is high and Block size is in minimum, web3j library throws time-out exception. Therefore, in Fig. 12 (a-e), we exceptionally finished the assessment of this configuration in 500.

To assess the latency and throughput of the system for the execution of different processes, Fig. 13 (a-b) depicts these two parameters in the different concurrent requests for $BT = 10, BS = 30$. As shown in Fig. 13, user registration, ABAC procedure, and payment are mostly need more time for execution, while inserting the valid token for registration and service provider registration are the fastest procedures.

3) DISCUSSION ON EXPERIMENTS

In the previous section, we analyzed the system performance and scalability for at most 700 concurrent requests. In the

TABLE 8. System throughput with different parameters.

P	SP registration				Add registration token				User registration				ABAC procedure				Payment			
BT	5		15		5		15		5		15		5		15		5		15	
BSC	15	30	15	30	15	30	15	30	15	30	15	30	15	30	15	30	15	30	15	30
50	1.64	3.20	0.82	1.63	1.63	3.20	0.82	1.62	1.62	3.20	0.77	1.44	1.63	3.19	0.77	1.43	1.64	3.19	0.78	1.47
100	1.62	2.05	0.89	1.30	2.17	3.24	0.93	1.63	2.13	3.21	0.87	1.45	2.16	3.19	0.87	1.45	2.17	3.22	0.88	1.47
200	1.26	1.69	0.87	1.43	2.57	4.24	0.93	1.85	2.14	4.05	0.88	1.62	2.14	3.22	0.88	1.62	2.17	4.16	0.89	1.66
500	0.78	4.81	-	0.90	2.64	4.58	0.95	1.88	2.26	3.82	0.90	1.67	2.4	3.84	0.90	1.66	2.29	3.81	0.91	1.70
700	1.08	2.68	-	1.00	2.58	4.64	-	1.95	2.35	3.91	-	1.87	2.31	3.60	-	1.87	2.35	3.77	-	1.87

real-world implementation of our use-case, it is important to assess system performance when an enormous number of requests and smart contracts exist in the system. Regarding this assessment, two features can be taken into account: 1) latency and throughput of the system, and 2) storage scalability.

- *Throughput and latency:* As shown in Fig. 12 and Table 8, the system is highly scalable in terms of the number of requests. Also, an increasing number of user's in the system results in increasing the number of validators. Due to these two reasons, we can state that the system's latency and throughput are not highly dependent on the number of requests.
- *Storage:* Increasing number of transactions, smart contracts, and Blockchain logs, need high storage space for full-nodes in the system. For the time being, in the current version, we store the logs, transactions, and smart contracts directly into the Blockchain, which can be a challenging issue in the real implementation of the system. This issue is discussed in the Section VII-E.

C. THREATS SCENARIO

In this section we provide an assessment of the system based on several threat scenarios:

1) SINGLE POINT OF FAILURE (DOS/DDOS)

Scenario 1: there is a centralized point in the access control process, which can be a single point of failure.

Analysis: As shown in Fig. 4, registration, the access control, and the payment process (i.e., steps 1, 3, 4, 5) in the system are performed on-chain. This means there are several Blockchain nodes in the system to receive a user's request, validate the request based on the rules in the smart contracts, reach consensus on the validation result, update the ledger and send the result to the user or other contracts. In this process, the failure of a single node does not have a significant effect on the functionality of the whole system.

Finally, we can claim that it is feasible to outsource the access management of service providers and network providers to our system, without being concerned about having a single point of failure or the need for a trusted third party. The system is resistant to DoS/DDoS attacks, as well as node failures.

2) UNAUTHORIZED USE OF SERVICE (MITM)

Scenario 2: Adversary $\mathcal{A}$ aims to directly connect to the service provider without authorization, and use the service subscribed by another user. This thread can be assumed as a Man-in-the-Middle scenario.

Analysis: To use the service, the user should send a one-time valid token as well as a random number (i.e., *nonce*) to the *sp* (Fig. 7, step 11). The token is generated in the *AC_Manager* contract (Fig. 7, Step 7) and is stored in the Blockchain. So, it is feasible to get the token. The token is the concatenation of $Addr_u$ and $Addr_{sp}$ with $hash(nonce)$ which is sent by the user. It means $\mathcal{A}$ can restore hash(nonce) from the Blockchain.

nonce is a strong random number that is generated off-chain for each connection. Besides, the hash function is one-way, and if its input has enough entropy, finding the clear message by having its hash is not feasible in acceptable time. Since *u* sends the $hash(nonce)$ in the first request, $\mathcal{A}$ cannot find the *nonce*. Therefore, the attempt of $\mathcal{A}$ to use the service registered by the other user is failed.

Scenario 3: Adversary $\mathcal{A}$ aims to connect to the system, but use the service subscribed by another user.

Analysis: Because the access tokens are generated based on the $Addr_u$, $\mathcal{A}$ must be able to use the identity of the legitimate user to call the *AccessToService*(). In the token generation, we use *msg.sender*, which will remove the possibility of changing the address of the sender. So, the attempt of $\mathcal{A}$ is failed.

Scenario 4: Adversary $\mathcal{A}$ uses the old tokens for connection.

Analysis: Adversary $\mathcal{A}$ can regenerate the old token, because it has all of the parameters. But in checking the validity of the token by *sp*, this attempt will fail. Because after checking the validity of the token by the *sp*, *AC_Manager* removes the token from the list of valid tokens (Fig. 7, step 12). Moreover, $\mathcal{A}$ cannot insert the regenerated token to the Blockchain, because this capability is limited to *AC_Manager*.

3) TAMPERING THE INTERNET PRICE(IMMUTABILITY)

Scenario 5: Adversary $\mathcal{A}$ creates a *SP_NP_Contract* with modified $price_{internet}$. This scenario has a motivation of using the internet without paying the network provider.

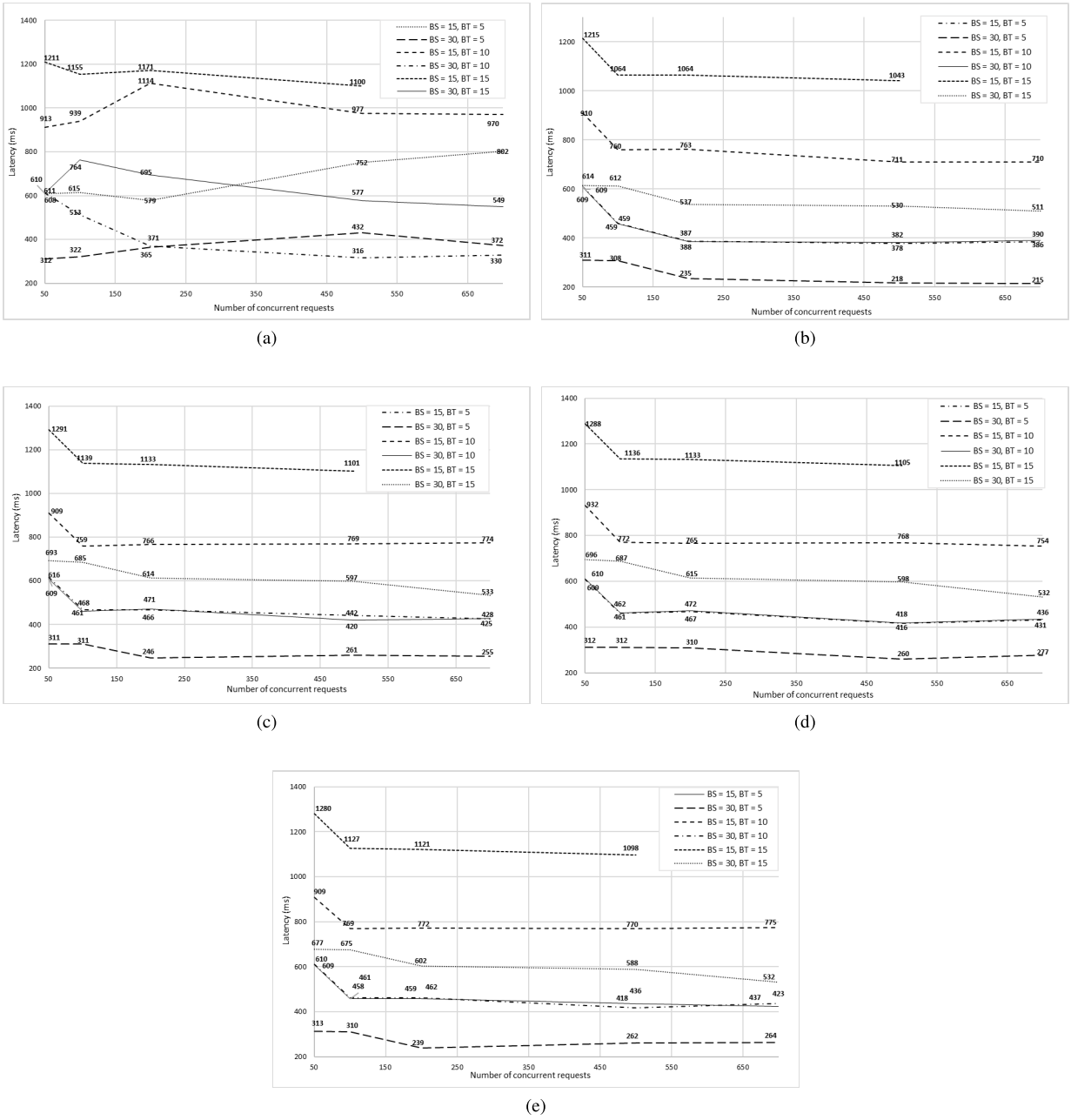

FIGURE 12. System latency with different values for Block Time (BT) and Block Sizes (BS) in several concurrent requests for service provider registration (a), inserting valid registration token (b), user registration (c), ABAC procedure (d), and payment to the network provider (e).

Analysis: As shown in Fig. 5, when *sp* reaches an agreement with the *np*, a one-time registration token is generated by the *np* (Fig. 5, step 2). The attempt of $\mathcal{A}$ will be failed because:

1) The token can be stored in the network, only when it is sent by the *np*. So, $\mathcal{A}$ cannot insert any valid token into the Blockchain.
2) The $price_{internet}$ exists in the token; therefore, any change in the price will lead to $T \neq T'$.
3) As T' is generated using *msg.sender*, if $\mathcal{A}$ can find a token with low $price_{internet}$ and send it to the Blockchain, instead of agreement with the network provider, $Addr_{\mathcal{A}} \neq Addr_{sp}$, therefore, $T \neq T'$ and the scenario is failed.

4) TAMPERING ACCESS ATTRIBUTES (IMMUTABILITY)

Scenario 6: Adversary $\mathcal{A}$ aims to request registration with modified access attributes (not based on agreement).

Analysis: This analysis is the same for both registration and update scenarios, but to make it short, we only explain the scenario for registration. As shown in step 2 of Fig. 6, when the user pays for the subscription, *sp* generates a token

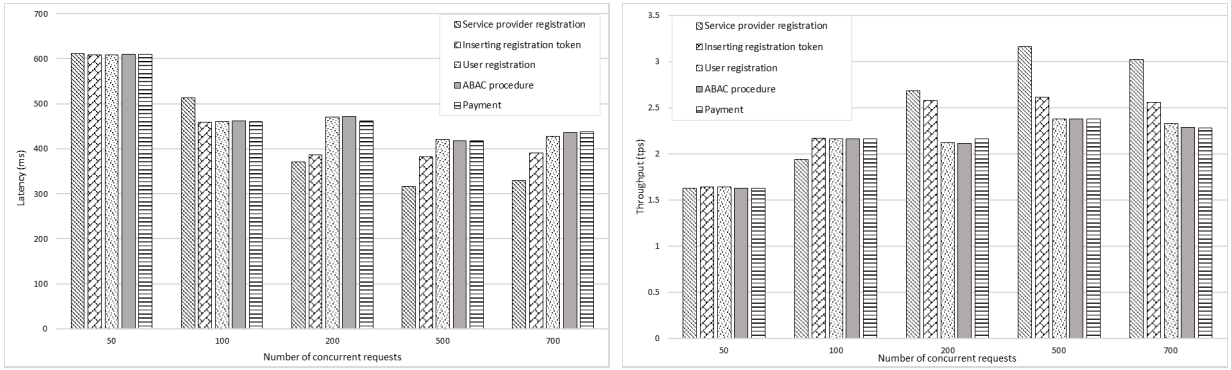

FIGURE 13. System latency and throughput in BT= 10s and BS = 30 for all processes in the system with several concurrent requests. (a) is the latency of the system and (b) is the throughput.

based on all parameters of the agreement. Then it registers the valid token in the Blockchain and sends it to the user. If Adversary $\mathcal{A}$ changes any parameter in Fig. 6, step 4, will cause to $T \neq T'$ in step 5.

Scenario 7: Adversary $\mathcal{A}$ aims to change its (or other one's) access attributed in the system by updating the existed $User\_Contract_{u,sp}$.

Analysis: To update the access attributes, 2 scenarios can be assumed:

1) $\mathcal{A}$ calls the *Update*() function of *Registration* (Algorithm 6). As mentioned in the update process, using the update tokens will result in the failure of the adversary attempt.
2) $\mathcal{A}$ calls the *update*() function of $User\_Contract_{u,sp}$. This function checks the sender of the message, and if $msg.sender \neq Addr_{Reg}$, the transaction will fail. As mentioned before, $Addr_{KAddr}$ is stored in this contract from deployment step. *update*() fetches the $Addr_{Reg}$ from *Known_Addresses*, and this scenario fails because $Addr_{\mathcal{A}} \neq Addr_{Reg}$.

5) THREAT OF MAINTAINABILITY

Scenario 8: The controller contracts (i.e., the contracts which are not used as a distributed database), needed to be replaced to be adapted to the new needs of new business models.

Analysis: As shown in Fig. 10, two contracts *AC_Manager* and *Registration*, as two controller contracts in the system, are replaceable. These two contracts do not store any hardcoded parameter. Based on [37], the maintainability problem can be resolved by assigning the variables dynamically.

6) IMMUTABILITY OF STORED ADDRESSES

Scenario 9: Adversary $\mathcal{A}$ aims to change the addresses stored in *Known_Addresses*.

Analysis: If this scenario can be executed, all other threads in the system would be feasible. To protect the system against this threat, we defined that:

1) The addresses of the database contracts (see Fig. 10), cannot be changed in any circumstances, and
2) Controller contracts can only be replaced by a new contract if $msg.sender == Addr_{admin}$. Note that $Addr_{admin}$ is stored in *Known_Addresses* in the setup phase.

D. COMPARISON

Table 9 shows the comparison among the proposed method and other related works. As mentioned before, to the best of our knowledge, we could not find a paper that shares the same concerns as us. Thus, we compare the more related state of the arts. So, several works implemented the ABAC method (i.e., [23], [24], [26], [28]–[30], [33], [34]), while other works proposed different access control methods. Important to mention that, in the related works we only focused on the papers that are more related to our use-case and select among them; because of this some papers are not related to the ABAC model.

Focusing on access control solution, our proposed method can be compared with [23], [24], [26]–[30], [33], and [34]. As shown in Table 9, [23], [24], [26]–[30] use Blockchain as a distributed database for rules and policies. They have a central point for access decisions, that can be a single point of failure. While, in our solution, we use Blockchain for both access management and distributed database for rules. This approach can remove any single point of failure in the access control procedure. So, we can claim that the proposed system is more fault-tolerant. References [33], [34] are similar regarding the performance, scalability, and fault-tolerant. But, their use-cases are different from our paper and they also do not support payment capabilities.

VII. CONCLUSION AND FUTURE DIRECTIONS

Blockchain and smart contracts are disruptive technologies that can change different aspects of businesses. In this paper, we proposed a smart contract-based access control mechanism. A high-level abstract of the proposed method is depicted in Fig. 14. The main purpose of the proposed method is to provide a flexible and scalable access control solution

TABLE 9. Comparison of proposed method with existing systems.

Features \ refs	[19]	[23]	[24]	[25]	[26]	[27]	[28]	[29] [30]	[31]	[32]	[33]	[34]	[35] [36]	Proposed
Access control automation	No	No	No	No	No	No	No	No	Yes	Yes	Yes	Yes	Yes	***Yes***
Purpose of blockchain usage*	DB	DB	DB	DB	DB	DB	DB	DB	ACM	ACM	ACM	ACM	ACM	***ACM***
Scalability+	H	H	N/A	N/A	N/A	N/A	N/A	H	N/A	N/A	M	N/A	N/A	***H***
Access decision making**	C	O	C	C	C	C	C	N/A	D	D	D	D	D	***D***
Fault tolerance in AC process	No	No	No	No	No	No	No	Yes	Yes	Yes	Yes	Yes	Yes	***Yes***
Remove single point of Failure	No	No	No	No	No	No	No	No	Yes	Yes	Yes	Yes	Yes	***Yes***
Payment capability	No	No	No	No	No	No	No	Yes	No	No	No	No	No	***Yes***
Access control model++	CB	AB	AB	CB	AB	AB	AB	AB	G	RB	AB	AB	G	***AB***

* DB (Using blockchain as a database for rules or terms of the agreement) or ACM (Using blockchain for both database and access control process)
\+ Tolerance of latency is low for more than 500 concurrent requests (High(H)), up to 500 concurrent requests (Moderate(M)) in different parameters
**C (Centralized), O (the Owner), and D (Distributed)
++AB (ABAC), CB (CapBAC), RB (RBAC), and G(Generic)

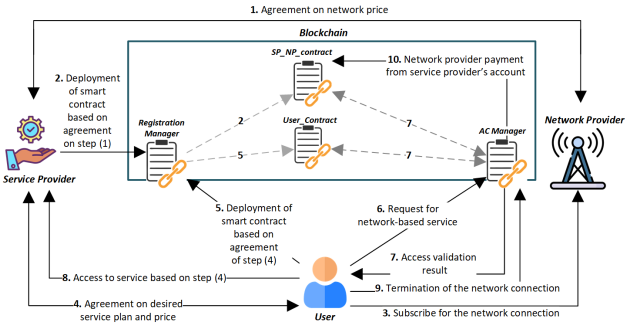

FIGURE 14. A high-level abstract of the proposed access control method.

without the need for a trusted third party to enable service providers to outsource their access control needs. Our method also supports a new business model to decrease the user's payment, improves the access management automation, removes the single point of failure (increasing the fault tolerance) in access management, increases the security, accountability, and reliability of connection, and addresses the user's, network provider's and service provider's needs. This method also guarantees payment security based on blockchain properties in a trustless environment.

Measuring the scalability of the system using two parameters (i.e., latency and throughput) indicates that increasing the number of nodes in the system can result in decreasing the latency. This change happens, because the validators in the system, participate in the consensus procedure, increase. The variation in latency and throughput in the case of having several different concurrent requests (50 to 700), several block sizes (15 and 30 transactions per block), and different block times (5s, 10s, 15s), show that this system also brings high scalability.

Due to the many challenges in the implementation of a new model, we have postponed some improvements for future work. Detailed suggestions for future work are given next.

A. AUTHENTICATION

As mentioned before, our proposed method uses the network provider's AKA-based authentication method to send the user's request to the Blockchain. As future directions, we may have two suggestions:

1) In addition to AKA-based authentication (i.e., a centralized solution), we need to propose another distributed authentication solution. This solution must consider the requirements of all parties (i.e., AKA is only done in network provider side).
2) From another perspective, even though the proposed access control method is done after the AKA-based authentication on the network provider side, in the future it can be introduced as the first step to eliminate the AKA-based access control mechanisms. Also, it can provide an idea and vision about how to implement more software-based and loosely-coupled networks, by outsourcing the authentication and access control process and decoupling it from the network. This solution can remove any single point of failure in whole process; Privacy of all parties, latency, scalability and trust are main challenges to solve in this solution.

B. PRIVACY PRESERVING

In this version of the proposed method, we did not focused on privacy issues because, we do not store any identifier data about users (e.g., email, phone number, etc.). The only available data of the users in the system is their Blockchain address, a random parameter that cannot be the user's identifier. Despite this reason, when implementing the method in a real-world scenario, the users' and the service provider's privacy would be of vital challenge to focus on.

C. DECREASING THE LATENCY

One of the main concerns about using Blockchain technology for access management is its delay in getting a response. To overcome this problem, we propose to use a private/consortium Blockchain with a small number of users. In a real-world scenario, the latency of the system must be decreased as much as possible. To do so, one key research challenge will be to reduce the number of messages exchanged in the system while maintaining the constraints and requirements to protect the system's safety and security.

D. DELEGATION

Using the current system is limited to one user. It means no one rather than the user herself, can use the system. To increase the flexibility of the system, the user can delegate her access permissions to another user to use the services. To do so, several challenges in security and trust will rise. Resolving these challenges can be a future direction to improve the functionality of the method.

E. STORAGE OPTIMIZATION

Storage requirement is one of the challenges to implement the proposed system in a real-life scenario. Many users of this system utilize resource constraint devices such as mobile phones and laptops that are not highly capable of storing a large amount of data in their storage capacity. So, optimization of storage can be an important upcoming challenge. Two scenarios can be effective for the future direction in this part:

- It is possible to use cloud storage to store Blockchain data (i.e., smart contracts and their data), and then, we can only store the URL of those data in the Blockchain. In this scenario, another authentication and access control is needed to access to cloud and avoid data leakage;
- Separation of full-nodes and light-weight nodes can be another solution. To do so, we can designate several nodes with an enormous amount of storage and processing power for consensus, validation, and storage purpose, and all users can participate in the system as light-weight nodes. In this scenario, the challenge of trust must be taken into account.

F. ANALYSIS IMPROVEMENTS

To improve the assessment and analysis of the proposed method, in the future we aim to perform a detailed study of the cost and performance of the system:

1) With different scenarios of using public, private, or consortium Blockchains. In these experiments, using real traffic patterns and finding the best fitting for shaping the on-chain operations (to maximize the throughput) is the main challenge.
2) With different scenarios of using several Blockchain implementations such as Hyperledger Fabric, Quorum, etc., to select the most appropriate solution for our use-case.
3) With different consensus models such as PoS, PBFT, etc. to find the best solution for the system. The main challenge in this part of the analysis is to find a solution that can provide a suitable compromise between security and the performance of the system.

REFERENCES

[1] S. Nakamoto. (2008). *Bitcoin: A Peer-to-Peer Electronic Cash System*. [Online]. Available: https://bitcoin.org/bitcoin.pdf
[2] *Secure Property Titles with Owner Authority | Satoshi Nakamoto Institute?*. Accessed: Mar. 19, 2021. [Online]. Available: https://nakamotoinstitute.org/secure-property-titles/
[3] M. A. Ferrag, M. Derdour, M. Mukherjee, A. Derhab, L. Maglaras, and H. Janicke, "Blockchain technologies for the Internet of Things: Research issues and challenges," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 2188–2204, Apr. 2019, doi: 10.1109/JIOT.2018.2882794.
[4] A. A. Monrat, O. Schelen, and K. Andersson, "A survey of blockchain from the perspectives of applications, challenges, and opportunities," *IEEE Access*, vol. 7, pp. 117134–117151, 2019.
[5] M. S. Ali, M. Vecchio, M. Pincheira, K. Dolui, F. Antonelli, and M. H. Rehmani, "Applications of blockchains in the Internet of Things: A comprehensive survey," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 2, pp. 1676–1717, 2nd Quart., 2019.
[6] Z. Zheng, S. Xie, H.-N. Dai, X. Chen, and H. Wang, "Blockchain challenges and opportunities: A survey," *Int. J. Web Grid Services*, vol. 14, no. 4, pp. 352–375, 2018.
[7] Y. Xiao, N. Zhang, W. Lou, and Y. T. Hou, "A survey of distributed consensus protocols for blockchain networks," *IEEE Commun. Surveys Tuts.*, vol. 22, no. 2, pp. 1432–1465, 2nd Quart., 2020, doi: 10.1109/COMST.2020.2969706.
[8] R. Wattenhofer, *The Science of the Blockchain*. North Charleston, SC, USA: CreateSpace Independent Publishing Platform, 2016.
[9] M. Jakobsson and A. Juels, "Proofs of work and bread pudding protocols (extended abstract)," in *Proc. Secure Inf. Netw. Commun. Multimedia Secur. IFIP TC6/TC11 Joint Workshop Conf. Commun. Multimedia Secur. (CMS)*, vol 23. Boston, MA, USA: Springer, Sep. 1999, pp. 258–272.
[10] S. King and S. Nadal. (Aug. 2012). *PPCoin: Peer-to-Peer Crypto-Currency With Proof-of-Stake*. [Online]. Available: https://peercoin.net/assets/paper/peercoin-paper.pdf
[11] M. Castro and B. Liskov, "Practical Byzantine fault tolerance," in *Proc. OSDI*, vol. 99. 1999, pp. 173–186.
[12] V. Y. Kemmoe, W. Stone, J. Kim, D. Kim, and J. Son, "Recent advances in smart contracts: A technical overview and state of the art," *IEEE Access*, vol. 8, pp. 117782–117801, 2020.
[13] D. Ferraiolo, D. R. Kuhn, and R. Chandramouli, *Role-Based Access Control*. Norwood, MA, USA: Artech House, 2003.
[14] Y. Nakamura, Y. Zhang, M. Sasabe, and S. Kasahara, "Exploiting smart contracts for capability-based access control in the Internet of Things," *Sensors*, vol. 20, no. 6, p. 1793, Mar. 2020.
[15] B. Anggorojati, P. N. Mahalle, N. R. Prasad, and R. Prasad, "Capability-based access control delegation model on the federated IoT network," in *Proc. 15th Int. Symp. Wireless Pers. Multimedia Commun.*, 2012, pp. 604–608.
[16] B. W. Lampson, "Protection," *ACM SIGOPS Operating Syst. Rev.*, vol. 8, no. 1, pp. 18–24, 1974.
[17] R. S. Sandhu, "Role-based access control," in *Advances in Computers*, vol. 46. Amsterdam, The Netherlands: Elsevier, 1998, pp. 237–286.
[18] V. C. Hu, D. R. Kuhn, and D. F. Ferraiolo, "Attribute-based access control," *Computer*, vol. 48, no. 2, pp. 85–88, Feb. 2015.
[19] R. Xu, Y. Chen, E. Blasch, and G. Chen, "BlendCAC: A smart contract enabled decentralized capability-based access control mechanism for the IoT," *Computers*, vol. 7, no. 3, p. 39, Jul. 2018.
[20] G. Ali, N. Ahmad, Y. Cao, S. Khan, H. Cruickshank, E. A. Qazi, and A. Ali, "XDBAuth: Blockchain based cross domain authentication and authorization framework for Internet of Things," *IEEE Access*, vol. 8, pp. 58800–58816, 2020.
[21] F. Ghaffari, E. Bertin, J. Hatin, and N. Crespi, "Authentication and access control based on distributed ledger technology: A survey," in *Proc. 2nd Conf. Blockchain Res. Appl. Innov. Netw. Services (BRAINS)*, 2020, pp. 79–86.
[22] Q. Zhou, H. Huang, Z. Zheng, and J. Bian, "Solutions to scalability of blockchain: A survey," *IEEE Access*, vol. 8, pp. 16440–16455, 2020.
[23] S. Shafeeq, M. Alam, and A. Khan, "Privacy aware decentralized access control system," *Future Gener. Comput. Syst.*, vol. 101, pp. 420–433, Dec. 2019.
[24] X. Qin, Y. Huang, Z. Yang, and X. Li, "An access control scheme with fine-grained time constrained attributes based on smart contract and trapdoor," in *Proc. 26th Int. Conf. Telecommun. (ICT)*, Apr. 2019, pp. 249–253.
[25] L. Tan, N. Shi, C. Yang, and K. Yu, "A blockchain-based access control framework for cyber-physical-social system big data," *IEEE Access*, vol. 8, pp. 77215–77226, 2020.
[26] S. Wang, Y. Zhang, and Y. Zhang, "A blockchain-based framework for data sharing with fine-grained access control in decentralized storage systems," *IEEE Access*, vol. 6, pp. 38437–38450, 2018.

[27] B. Waters, "Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization," in *Proc. Int. Workshop Public Key Cryptogr.*, 2011, pp. 53–70.
[28] L. Guo, X. Yang, and W.-C. Yau, "TABE-DAC: Efficient traceable attribute-based encryption scheme with dynamic access control based on blockchain," *IEEE Access*, vol. 9, pp. 8479–8490, 2021, doi: 10.1109/ACCESS.2021.3049549.
[29] X. Ling, J. Wang, T. Bouchoucha, B. C. Levy, and Z. Ding, "Blockchain radio access network (B-RAN): Towards decentralized secure radio access paradigm," *IEEE Access*, vol. 7, pp. 9714–9723, 2019.
[30] X. Ling, Y. Le, J. Wang, Z. Ding, and X. Gao, "Practical modeling and analysis of blockchain radio access network," *IEEE Trans. Commun.*, vol. 69, no. 2, pp. 1021–1037, Feb. 2021.
[31] C. Yang, L. Tan, N. Shi, B. Xu, Y. Cao, and K. Yu, "AuthPrivacyChain: A blockchain-based access control framework with privacy protection in cloud," *IEEE Access*, vol. 8, pp. 70604–70615, 2020.
[32] J. Paul Cruz, Y. Kaji, and N. Yanai, "RBAC-SC: Role-based access control using smart contract," *IEEE Access*, vol. 6, pp. 12240–12251, 2018.
[33] H. Liu, D. Han, and D. Li, "Fabric-iot: A blockchain-based access control system in IoT," *IEEE Access*, vol. 8, pp. 18207–18218, 2020.
[34] Y. Zhang, S. Kasahara, Y. Shen, X. Jiang, and J. Wan, "Smart contract-based access control for the Internet of Things," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 1594–1605, Jun. 2018.
[35] T. Sultana, A. Almogren, M. Akbar, M. Zuair, I. Ullah, and N. Javaid, "Data sharing system integrating access control mechanism using blockchain-based smart contracts for IoT devices," *Appl. Sci.*, vol. 10, no. 2, p. 488, Jan. 2020.
[36] T. Sultana, A. Ghaffar, M. Azeem, Z. Abubaker, M. U. Gurmani, and N. Javaid, "Data sharing system integrating access control based on smart contracts for IoT," in *Proc. Int. Conf. P2P, Parallel, Grid, Cloud Internet Comput.*, 2019, pp. 863–874.
[37] J. Chen, X. Xia, D. Lo, J. Grundy, X. Luo, and T. Chen, "Defining smart contract defects on Ethereum," *IEEE Trans. Softw. Eng.*, Apr. 20, 2020, doi: 10.1109/TSE.2020.2989002.
[38] G. Bertoni, J. Daemen, M. Peeters, and G. V. Assche. (Jan. 2011). *The Keccak SHA-3 Submission*. [Online]. Available: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.361.9781&rep=rep1&type=pdf
[39] D. C. Nguyen, P. N. Pathirana, M. Ding, and A. Seneviratne, "Blockchain for 5G and beyond networks: A state of the art survey," *J. Netw. Comput. Appl.*, vol. 166, Sep. 2020, Art. no. 102693.
[40] S. Behrad, E. Bertin, and N. Crespi, "Securing authentication for mobile networks, a survey on 4G issues and 5G answers," in *Proc. 21st Conf. Innovation Clouds, Internet Netw. Workshops (ICIN)*, Feb. 2018, pp. 1–8, doi: 10.1109/ICIN.2018.8401619.
[41] C. Dannen, *Introducing Ethereum and Solidity*. Berkeley, CA, USA: Apress, 2017. [Online]. Available: https://link.springer.com/book/10.1007/978-1-4842-2535-6
[42] G. Wood. (Mar. 2018). *Ethereum: A Secure Decentralised Generalised Transaction Ledger*. [Online]. Available: https://ethereum.github.io/yellowpaper/paper.pdf
[43] I.-C. Lin and T.-C. Liao, "A survey of blockchain security issues and challenges," *IJ Netw. Secur.*, vol. 19, no. 5, pp. 653–659, 2017.
[44] P. W. Eklund and R. Beck, "Factors that impact blockchain scalability," in *Proc. 11th Int. Conf. Manage. Digit. Ecosyst.*, 2019, pp. 126–133.
[45] J. Xie, F. R. Yu, T. Huang, R. Xie, J. Liu, and Y. Liu, "A survey on the scalability of blockchain systems," *IEEE Netw.*, vol. 33, no. 5, pp. 166–173, Sep. 2019.
[46] M. Schäffer, M. Di Angelo, and G. Salzer, "Performance and scalability of private Ethereum blockchains," in *Proc. Int. Conf. Bus. Process Manage.*, 2019, pp. 103–118.

FARIBA GHAFFARI received the M.Sc. degree in information security. She is currently pursuing the Ph.D. degree with the Institut Polytechnique de Paris (Telecom SudParis). She has been involving in the field of authentication, and access control mechanisms based on blockchain and smart contracts Orange Labs, France, since 2019.

EMMANUEL BERTIN (Senior Member, IEEE) received the Ph.D. and Habilitation degrees in computer science from Sorbonne University. He is currently an Expert Community Leader with Orange Labs and an Adjunct Professor with the Institut Mines-Telecom, France. His activities are focused on 5G and 6G, blockchain, and service engineering, with more than 100 published researched articles.

NOEL CRESPI (Senior Member, IEEE) received the master's degree from the Universities of Orsay and Canterbury, the Diplome d'Ingénieur from Telecom ParisTech, and the Ph.D. and Habilitation degrees from Paris VI University. He joined the Institute Mines-Telecom, in 2002. He is currently a Professor and the M.Sc. Program Director, leading the Service Architecture Laboratory. He coordinates the standardization activities for the Institute Telecom, ETSI, 3GPP, and ITU-T. He is also an Adjunct Professor with KAIST, South Korea, an Affiliate Professor with Concordia University, Canada, and a Guest Researcher with the University of Goettingen, Germany. He is the Scientific Director of the French-Korean laboratory ILLUMINE. He is the author/co-author of 400 articles and contributions in standardization. His current research interests include data analytics, the Internet of Things, and Softwarisation.

SHANAY BEHRAD received the Ph.D. degree in computer science from the Institut Polytechnique de Paris. She worked on designing authentication and access control mechanisms for 5G networks during her Ph.D. degree at Orange Labs. She is currently working with the Institute of Research and Technology b-com, as a Research Engineer. Her research interests include 5G networks, microservices, and security.

JULIEN HATIN received the Ph.D. degree in computer science from the University of Caen and the Diplome d'Ingénieur from Ecole National Superieur Ingénieur de Caen. He is currently working with Orange Labs, as a Research Engineer. His research interests include smart contract, blockchain, identity management, and authentication.