Blockchain: Research and Applications 5 (2024) 100168

Contents lists available at ScienceDirect

Blockchain: Research and Applications

journal homepage: www.journals.elsevier.com/blockchain-research-and-applications

Research Article

Secure smart contract-based digital twins for the Internet of Things

Iakovos Pittaras [a,*], Nikos Fotiou [a], Christos Karapapas [a], Vasilios A. Siris [a], George C. Polyzos [a,b]

[a] Mobile Multimedia Laboratory, Department of Informatics, School of Information Sciences and Technology, Athens University of Economics and Business, Athens, 10434, Greece

[b] School of Data Science, The Chinese University of Hong Kong, Shenzhen, 518172, China

ARTICLE INFO

Keywords:
Digital twins
Internet of Things
Web of Things
Distributed ledger technologies
Hyperledger Fabric
Ethereum

ABSTRACT

The proliferation of Internet of Things (IoT) devices that operate unattended providing a multitude of important and often sensitive services highlights the need for seamless interoperability and increased security. We argue that digital twins of IoT devices, with the right design, can enhance the security, reliability, auditability, and interoperability of IoT systems. The salient features of digital twins have made them key elements for the IoT and Industry 4.0. In this paper, we leverage advances in W3C's Web of Things (WoT) standards and distributed ledger technologies (DLTs) to present a novel design of the smart contract-based digital twins with enhanced security, transparency, interoperability, and reliability. We provide two different variations of that general design using two different blockchains (one public and one private, permissioned blockchain), and we present design trade-offs. Furthermore, we introduce an architecture for accessing and controlling IoT devices securely and reliably, providing full auditability, while at the same time using the proposed digital twins as an indirection mechanism (proxy). The proposed architecture leverages the blockchain to offer notable properties, namely,decentralization, immutability, auditability, non-repudiation, availability, and reliability. Moreover, it introduces mass actuation, easier management of IoT devices, and enhanced security to the IoT gateways, enables new business models, and makes consumer devices (vendor-)agnostic.

1. Introduction

The Internet of Things (IoT) is envisioned to be an ecosystem of interconnected devices that merge the cyber with the real world, meaning to provide a multitude of services that improve the quality of our lives. However, there are some IoT challenges that need to be addressed. First of all, IoT systems are fragmented. There is a plethora of IoT devices from different manufacturers that use different protocols and standards. In order to deal with that, the Web of Things (WoT) W3C working group [1] leverages Web technologies and standards to deal with interoperability issues among different types of IoT devices and platforms by developing an interoperable IoT architecture. The WoT builds on well-known Web protocols and enables IoT device discovery and access using REST-based APIs, over popular application layer protocols, such as HTTP(s). The WoT comes with a lot of benefits and addresses the problems of fragmentation and lack of interoperability in the IoT.

In addition, securing IoT services, i.e., sensing and, in particular, actuation, requires complex security solutions using advanced cryptographic techniques and algorithms. However, these solutions have not been designed for the IoT, in which many IoT devices are usually less powerful than typical computers, hence they cannot perform complex security and cryptographic operations. Therefore, more lightweight solutions that take into consideration the limitations of IoT devices are required. Furthermore, since the real world can be directly impacted by the IoT, and IoT devices are even physically exposed to many, if not to all users, security, safety, and privacy are serious concerns. One solution in the direction of securing IoT systems and IoT devices is the use of digital twins. A digital twin is a virtual replica of a physical (IoT) device, system, or asset [2]. In most IoT systems, digital twins are typically used for testing, monitoring, and simulating IoT devices. In this work, we propose the use of digital twins as an isolation, protection, and indirection mechanism, instead of interacting directly with the actual IoT devices. In particular, instead of communicating with the actual IoT device, users communicate only with its digital twin. Then, all valid state modifications of the (virtual) digital twin will be securely transmitted to the actual IoT devices, which will consequently perform the requested actions.

* Corresponding author.
E-mail addresses: pittaras@aueb.gr (I. Pittaras), fotiou@aueb.gr (N. Fotiou), karapapas@aueb.gr (C. Karapapas), vsiris@aueb.gr (V.A. Siris), polyzos@acm.org (G.C. Polyzos).

https://doi.org/10.1016/j.bcra.2023.100168
Received 9 January 2023; Received in revised form 6 November 2023; Accepted 6 November 2023
Available online 22 November 2023

Digital twins usually operate in a more powerful and secure network location than the actual device, such as a Web server or a Cloud. Many companies, such as Amazon[1] and Microsoft[2], are already providing such services. Nonetheless, these solutions are vendor-specific and often result in "vendor lock-in" situations. Furthermore, these services lack transparency; given the pervasiveness of IoT systems and their access to sensitive and private data, security and privacy concerns arise. Finally, these systems may occasionally suffer from outages due to their centralized nature, making IoT devices inaccessible [3]. To address all these, we propose using distributed ledger technologies (DLTs) to create and "host" secure and reliable digital twins.

In this paper, we summarize and extend our efforts presented in Refs. [4,5], for designing smart contract-based digital twins for WoT-enabled IoT devices and using them to perform actions (actuation and sensing) on physical IoT devices. In particular, the contributions of this paper are as follows.

- We design and present secure, decentralized, reliable, auditable, and flexible smart contract-based digital twins of WoT "virtual entities", i.e., entities that integrate one or more IoT devices.
- We implement our solution in two different blockchain types: Ethereum and Hyperledger Fabric. Each of them constitutes a better fit for different use cases and purposes.
- We design, implement, and evaluate an IoT system that uses smart contract-based digital twins to offer secure sensing and actuation.

Our solution achieves the following. It improves interoperability, by adopting the WoT architecture, and auditability, since every interaction is recorded immutably on the DLT, enhances security by removing the need for a trusted entity that hosts the digital twin, and increases system availability by implementing its core functionality in a smart contract. In addition, we make consumers oblivious to IoT devices and IoT device (vendor-)agnostic, since they communicate with the digital twin and not the actual IoT devices, allowing abstractions and virtualization.

The rest of the paper is organized as follows. In Section 2, we present background information about technologies used in our work, as well as, the related work in the area. In Section 3, we introduce the design of smart contract-based digital twins, while in Section 4, we present the architecture of the proposed IoT system. In Section 5, we present the implementation and evaluation of the system. Finally, Section 6 concludes the paper and discusses future work.

2. Background and related work

2.1. Distributed ledger technologies and smart contracts

A blockchain is an append-only ledger of transactions distributed throughout a network of trustless nodes. Hence, they are also referred to as DLTs. A blockchain may be public or private, even though finer distinctions can be made in some cases. A popular public, permissionless blockchain is Ethereum [6]. The Ethereum virtual machine (EVM) is capable of executing immutable computer programs, called smart contracts, of any complexity, in a Turing complete language, such as Solidity.[3] A smart contract is a distributed application that "lives" and operates in the blockchain and runs deterministically in the context of the EVM. Immutability refers to the fact that once the smart contract is deployed, its code cannot change. Every interaction with a smart contract is recorded in the blockchain, giving us the opportunity to audit all the actions.

On the other hand, a popular private blockchain is Hyperledger Fabric [7] (for simplicity, we will refer to it as Fabric). Fabric is a private, permissioned, and open-source blockchain, where membership in the network is controlled, by a special node called membership service provider (MSP). The MSP implementation in Fabric follows the public key infrastructure (PKI) model using X.509 certificates issued by certificate authorities (CAs). Fabric, like Ethereum and other popular blockchains, supports the execution of smart contracts (in Fabric, they are called chaincodes, however, for simplicity and uniformity, we will refer to them as smart contracts), written in a general-purpose programming language. A smart contract in Fabric is executed simultaneously and in parallel by several special nodes, called endorsing peers. Fabric introduces a new model for transactions, called execute-order-validate, in contrast to other blockchains that follow the order-execute model. Thus, in Fabric, the transaction flow consists of three steps. Initially, the transaction is executed, then it is ordered by the consensus protocol, and finally, it is validated against an endorsement policy (e.g., n out of m endorsing peers should execute the smart contract and produce the same output) before committing it to the ledger. Finally, the results are gathered by another special node called the orderer, which creates the block and broadcasts it to the network. The flow of a transaction in Fabric is shown in Fig. 1.

2.2. Web of Things

The WoT architecture [8] structures well-known Web protocols and tools for connecting IoT devices to the Web. In the WoT architecture communication model, IoT devices are made available through REST-based APIs, which are used by consumers to access device properties, trigger device actions, and receive device-generated events. In order to improve the interoperability and usability of IoT platforms, the WoT model uses a common format for describing IoT devices, referred to as the thing description (TD) [9]. The TD is machine-readable and includes metadata about the IoT device, such as its identifier, title, and security definitions among others, as well as IoT device properties, actions, and events that can be accessed or invoked through Web links and forms.

Listing 1 provides an example of a WoT TD for a smart lamp, which is encoded using JSON-LD. This TD includes information about how this IoT device can be accessed, i.e., via an HTTP request to the specified URI. In particular, to switch on/off the smart lamp, a POST HTTP request should be sent to https://example.com/things/lamp1/toggle. Similarly, in order for someone to read the current status of the smart lamp, a GET HTTP request has to be sent to https://example.com/things/lamp1/status.

```
{''@context'': ''https://www.w3.org/2019/wot/td/v1'',
 ''id'': ''lamp1'',
 ''title'': ''My lamp'',
 ''securityDefinitions'': {...},
 ''security'': [...],
 ''properties'': {
    ''status'': {
       ''type'': ''string'',
       ''forms'': [{''href'':
                   ''.../things/lamp1/status''}]
  }
 },
 ''actions'': {
    ''toggle'': {
       ''type'':''boolean'',
       ''forms'': [{''href'':
                   ''.../things/lamp1/toggle''}]
  }
 },
 ''events'': {...}
}
```

Listing 1: WoT thing description for a smart lamp.

[1] https://aws.amazon.com/iot-core/?c=i&sec=srv.
[2] https://azure.microsoft.com/en-us/services/digital-twins.
[3] https://solidity.readthedocs.io/en/v0.6.10/.

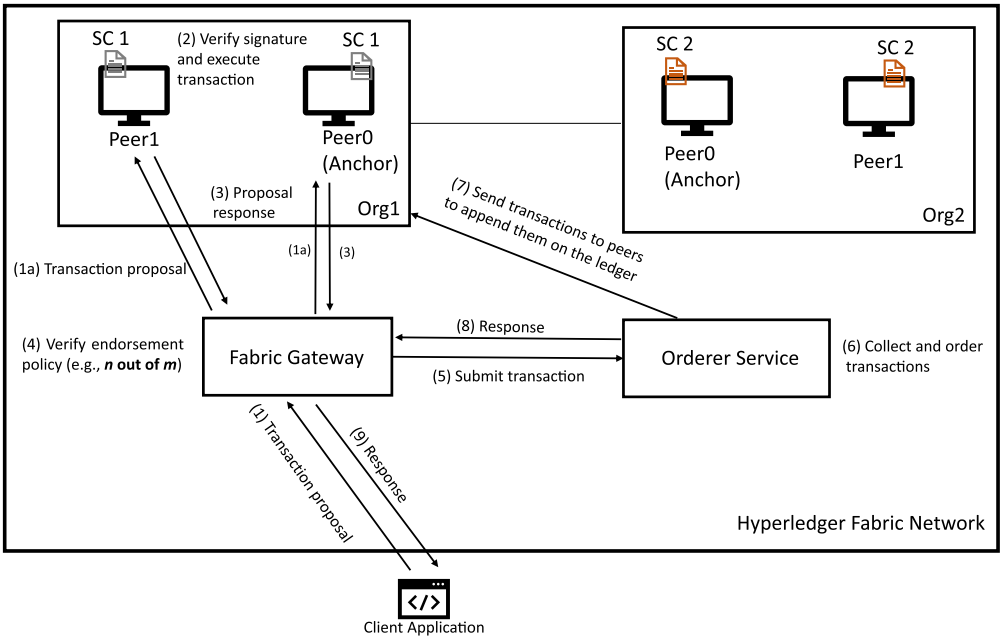

Fig. 1. Transaction flow in a Hyperledger Fabric blockchain network.

2.3. Related work

Over the past few years, there has been significant consideration of incorporating blockchain technology into IoT systems and applications. Many works are trying to integrate blockchain infrastructure into various IoT systems and architectures, e.g., for the energy sector [10], food supply chains [11], smart agriculture [12], and gaming [13], among others. Other research efforts, like the one presented by some of the authors [14], discuss how blockchains can be used to provide novel security mechanisms for the IoT. In our work, we use blockchains to secure the digital twins of WoT-enabled IoT devices.

In recent times, digital twins have witnessed increasing attention. There are several research efforts that have tried to integrate digital twins into IoT applications. Chevallier et al. [15] presented an architecture for creating and managing digital twins for smart buildings. Liu et al. [16] proposed a framework based on digital twins for an indoor safety management system. Moreover, Mohammadi and Taylor [17] proposed a smart city digital twin paradigm. Similarly, White et al. [18] presented a design for digital twins of smart cities. These works use digital twins mainly as monitoring tools and for performing simulations, which cannot be performed in the actual IoT systems. However, in our work, we use the digital twin as an indirect mechanism for the actual IoT devices. We use them to perform real actions on IoT devices instead of using them to extract information about the physical devices.

On the other hand, many efforts have investigated the intersection of blockchains and digital twins, i.e., blockchain-based digital twins. Yaqoob et al. [19] presented some potential use cases, architectures, and technologies that can enhance digital twins to be more effective in industrial problems. They used the blockchain as storage for digital twin data. Similarly, Khan et al. [20] proposed a framework that uses the blockchain to securely store digital twin data. More recent efforts [21–23] have used the blockchain to address the problem of sharing the data generated by the digital twins. In particular, Putz et al. [21] introduced a decentralized application (DApp) that facilitates digital twin data sharing among multiple parties without the need for trusted third parties. Dietz et al. [22] tried to address the same problem by examining how DLTs can be used to provide secure sharing of data generated by the digital twins. All these efforts highlight the advantages of integrating DLTs and digital twins. However, these works use the blockchain as a means for digital twin data sharing, while we are using the blockchain to implement the actual digital twins as smart contracts to isolate and secure the physical IoT devices.

3. Smart contract-based digital twins design

In this section, we present the design of smart contract-based digital twins of IoT devices. As IoT devices, we consider sensors and actuators that follow the WoT standards and specifications, namely, they expose a TD. The use of the WoT addresses the problems of fragmentation and interoperability that the IoT faces by providing a set of standardized technologies, following the well-known Web paradigm. Furthermore, we follow the most typical architectural pattern in legacy IoT systems, where the IoT devices are paired with a more powerful device than them, a gateway. In this pattern, instead of interacting directly with the IoT devices, users interact with the gateway. Therefore, in our design, we consider IoT devices that are paired with WoT gateways. Each of the WoT gateways in the system represents a WoT "virtual entity". A virtual entity is the composition of one or more IoT devices, e.g., a building consisting of several IoT devices. A virtual entity provides a single WoT TD that includes the actions, the properties, and the events of all its IoT devices. Therefore, if in our system there are two WoT gateways, including many IoT devices, then we have two virtual entities and two TDs accordingly. This is depicted in Fig. 2. IoT devices are identified by URIs (see Listing 1) that can be used for performing sensing and actuation processes. In our design, the URIs are composed of the URL of the WoT gateway, the name of the IoT device, and the action or data that are provided from the corresponding IoT device, e.g., http://gw1.iot/lamp1/toggle.

We design and implement digital twins on the two most popular representatives of public/permissionless and private/permissioned blockchains, Ethereum and Fabric, taking into consideration the limitations and challenges introduced by each type.

3.1. Ethereum-based digital twins

First, we introduce the design based on the public, permissionless Ethereum blockchain. Ethereum and Ethereum's smart contracts introduce some limitations and challenges that need to be addressed in our proposed design. In particular, Ethereum smart contracts cannot communicate directly with the "outside" world (off-chain); instead, they

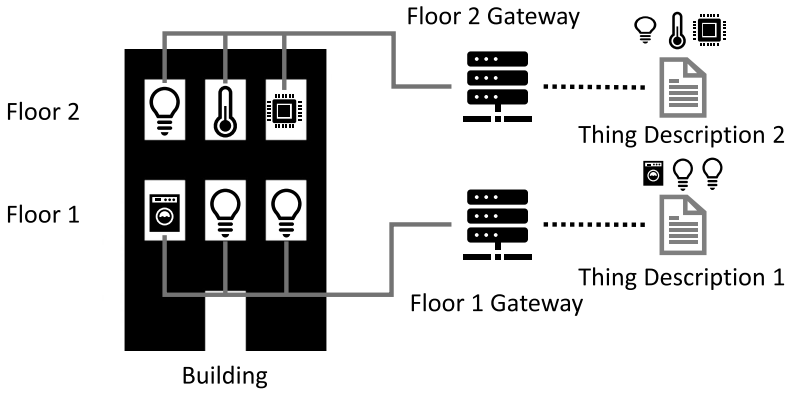

Fig. 2. The Internet of Things/Web of Things architecture considered in our design.

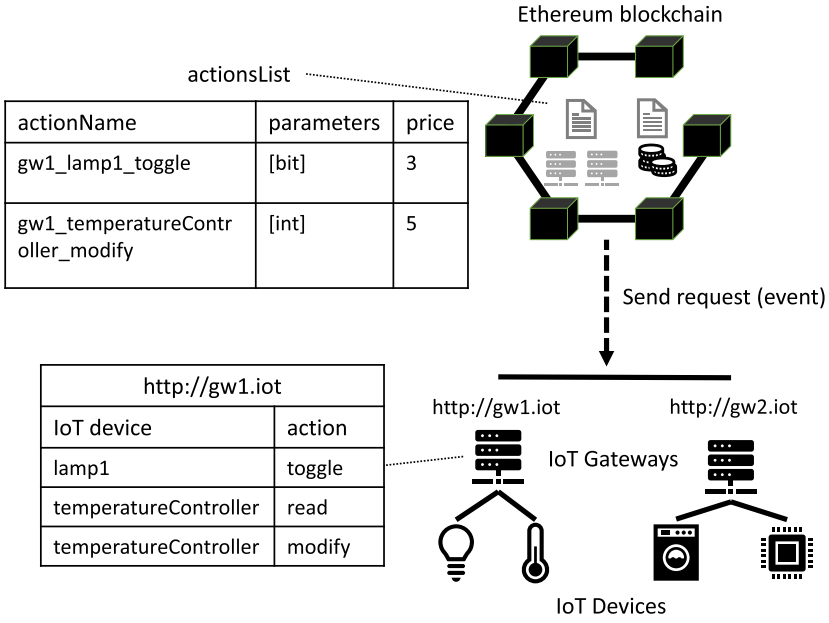

Fig. 3. Digital twin's structure on the Ethereum.

can only access data that are on-chain. Furthermore, all actions that involve the invocation of a function in a smart contract incur a transaction cost, which in Ethereum is expressed as gas. Gas is Ethereum's unit for measuring the computational and storage demands of a transaction. In addition to the monetary cost, Ethereum also introduces transaction delays, which depend on the block mining time and cannot be considered negligible. In particular, the average time required for an operation to be executed on the Ethereum blockchain is ≈ 15 s. Finally, since Ethereum is a public blockchain, everyone can send transactions to the blockchain and read the blockchain and the deployed smart contracts. Therefore, there are considerations, when designing digital twins as Ethereum smart contracts.

Since Ethereum smart contracts cannot communicate (or send requests, call a REST API, etc.) with an application, or anything that is off-chain, we cannot consider a design, in which the smart contract communicates with the WoT gateway directly to learn the available actions and properties of the virtual entity. Thus, we decide to embed the TD of the virtual entities into the smart contract. However, in Ethereum, it is too costly to store the actual TD in the smart contract. To avoid having enormous monetary costs, we store a stripped-down version of the virtual entities' TDs in the smart contract. Each action of the TD is stored in a data structure in the smart contract, called the *actionsList*. This structure contains (a) an action name, (b) the input parameters, including the type and the number of parameters, and (c) the defined price for each action, expressed in ERC 20 tokens. This is shown in Figs. 3 and 4, where the design and the source code of the smart contract are illustrated. The action name is composed of the name of the gateway underscore the name of the IoT device underscore the name of the action. The parameter field is essentially the data schema that describes the data format of the actions, properties, and events of a TD.[4]

Finally, since Ethereum is public and everyone can have access to the smart contracts deployed on it, to restrict access to the IoT devices, we propose a form of access control. Consumers can gain access to the smart contract-based digital twin, by obtaining some owner-specific tokens (the owner is the person who owns the virtual entity), implemented using the ERC 20 token standard [24]. Therefore, only the consumers who have obtained these tokens can perform operations on the virtual entity. To obtain some tokens, consumers have to communicate with the owner directly, offline, and off-chain. This feature can also enable various business models. For example, depending on the use case, the owner can define a price for these tokens and sell them to consumers, or consumers may sell their tokens to other consumers, or even consumers can form "alliances" and purchase a large number of tokens, potentially achieving a discount and forming a secondary market.

[4] https://www.w3.org/TR/wot-thing-description/#dataschema.

```
pragma solidity >=0.4.16 <0.9.0;
import "./ERC20DT.sol";

contract DigitalTwin is ERC20DT {
    //The structure that represents an action/property of an IoT device
    struct Actions { string actionName; string[] parameters; uint price; }
    //stripped-down version of the TD
    Actions[] public actionsList;
    //event that ends up on the GW
    event PerformAction(string actionName, string[] parameters, address client);

    function performAction(uint actionIndex, string[] parameters) public {
        //check that the request is valid
        require(actionIndex < actionsList.length);
        Actions action = actionsList[actionIndex];
        require(balances[msg.sender] > action.price);
        require(parameters.length == action.parameters.length);
        //transfer the tokens from the consumer to the smart contract address
        balances[msg.sender] = balances[msg.sender] – action.price;
        balances[address(this)] = balances[address(this)] + action.price;
        //emit the event
        emit PerfromAction(action.actionName, action.parameters, msg.sender); }

    function endOfAction(uint actionIndex, address client, bool success) public {
        require(msg.sender == owner);
        Actions action = actionsList[actionIndex];
        //transfer the token to the owner or to the
        if (success) {
            balances[owner] = balances[owner] + action.price;
            balances[address(this)] = balances[address(this)] – action.price; }
        else {
            balances[client] = balances[client] + action.price;
            balances[address(this)] = balances[address(this)] – action.price; } }

    function getAction() { … }
    function addAction(string actionName, string[] parameters, uint price) public {
        Actions action = Actions(actionName, parameters, price);
        actionsList.push(action); }
    function deleteAction() { … }
    function modifyAction() { … } }
```

Fig. 4. Source code of the smart contract-based digital twin on the Ethereum.

As we observe from the simplified smart contract shown in Fig. 4, the owner, in order to "create" a digital twin of an IoT device, should call the `addAction` function and insert all the actions with the corresponding parameters and the price for these actions that the IoT device supports. Similarly, to modify an action, e.g., change the price of an action, or remove completely an action, he/she calls the `modifyAction` and the `deleteAction` functions, which just modify accordingly the *actionsList*. Then, consumers can call the `performAction` function of the smart contract-based digital twin to request and perform actions on the provided IoT devices (a more detailed flow on how consumers request to perform actuation or sensing processes is presented in Section 4).

3.2. Hyperledger Fabric-based digital twins

In the previous subsection, we presented the design of smart contract-based digital twins in a public blockchain. However, using public blockchains may be inappropriate in some use cases where privacy is needed since anyone can inspect and read a public blockchain. The latter, in our case, means that anyone can find out the provided actions as well as who has invoked which actions. Therefore, to avoid that, we have also designed and implemented smart contract-based digital twins in a private, permissioned blockchain, particularly in Fabric.

Fabric does not introduce monetary costs, as opposed to Ethereum, so it is not critical to have a minimal design for the digital twin. Thus, in this design, we can store the whole TD of the virtual entities in the smart contract, making the implementation of the smart contract much simpler and straightforward. The design of Fabric is shown in Fig. 5, and a simplified version of its source code is shown in Fig. 6. Additionally, in Fabric, smart contracts can communicate directly with the "outside" world. Thus, in the Fabric-based design, the smart contract can communicate directly with the WoT gateways instead of communicating indirectly through events. However, in a design like that, every peer participating in the network that has deployed the smart contract would send a request to the gateways. Therefore, if in a system there are 100 peers and everyone has to execute the smart contract, then all will send a request on the gateway. The gateway will eventually end up receiving 100 requests for the same actuation/sensing request. This can lead to a denial of service (DoS) attack on the WoT gateways, if there are too many peers in the network. In order to address this challenge, the smart contract does not send directly any requests to the WoT gateway, but as in the Ethereum-based design, it generates an event that is caught by the WoT gateways. Furthermore, in Fabric, there is no need for creating an external access control mechanism, as we did in Ethereum using the ERC 20 tokens, since Fabric is a permissioned blockchain and the access is already checked by the MSP, which is a trusted authority. Finally, in this design, we do not have an end actuation function, since this function is useful to return the tokens back to the consumer in case the action is not completed successfully. In Fabric, we do not introduce any tokens, thus this function is useless.

4. IoT system overview

In this section, we present an overview of the proposed IoT system as a whole and its architecture, which integrates the proposed smart contract-based digital twins to offer secure and interoperable sensing and actuation services to consumers. The architecture of the system is illustrated in Fig. 7. The system is composed of the following entities and components:

- WoT gateways and IoT devices.
- The blockchain network.
- The smart contract-based digital twins.
- Owner(s) who administer WoT gateways and IoT devices.
- Consumer(s) who interact with IoT devices.

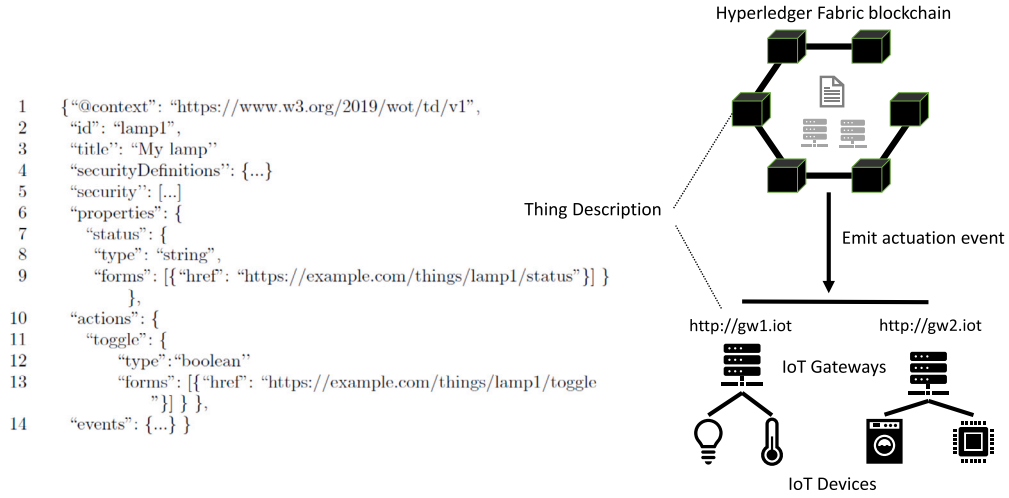

Fig. 5. Digital twin's structure on Hyperledger Fabric.

```
Const { Contract } = require('fabric-contract-api');

Class DigitalTwin extends Contract {

    //initialize ledger with the TDs
    async initLedger (ctx) {
    const TDs = [{…}, {…}];
    for (let i=0; i<TDs.length; i++) {
        //write to the ledger
        await ctx.stub.putState(TDs[i]);
    } }

    async performActuation (ctx, tdNumber, action, parameters) {
        //read the appropriate TD from the ledger
        const tdAsBytes = await ctx.stub.getState(tdNumber);
        const tdAsJSON = JSON.parse(tdAsBytes.toString());

        //check that action exists in the TD
        //check that the number of the parameters are correct

        tdAsJSON.Actions.action = parameters.action;
        tdAsBytes = Buffer.from(JSON.stringify(tsAsJSON));
        //send event
        ctx.stub.setEvent('performActuation, tdAsBytes);
        //write the result of actuation on the ledger
        ctx.stub.putState(tdAsBytes);}

    async getThingDescription (ctx) { … }
    async getAction (ctx, tdNumber, action) { … }
    async addAction (ctx, tdNumber, actionName, action) { … }
    async getProperty (ctx, tdNumber, property) { … }
    async addProperty (ctx, tdNumber, propertyName, property) { … }
    async addTD (ctx, tdNumber, td) {
        const temp = JSON.parse(td);
       await ctx.stub.putState(tdNumber, Buffer.from(JSON.stringify(temp))); }
}
```

Fig. 6. Source code of the smart contract-based digital twin on the Hyperledger Fabric.

- An ERC 20 token smart contract, applied only in the Ethereum-based design.

As we observe from Fig. 7, consumers do not access IoT devices directly; instead, they interact with their smart contract-based digital twins, which are deployed on the blockchain (either on Ethereum or Fabric). Consumers, to interact with the blockchain, should own a blockchain wallet, which includes a public/private key pair used for signing transactions sent to the blockchain network.

As we mentioned above, each WoT gateway constitutes a WoT virtual entity, and each virtual entity provides a single WoT TD, which contains the capabilities of all IoT devices. The actions of the virtual entity can be implemented either by a single IoT device, or by orchestrating multiple IoT devices. Similarly, an action may correspond to multiple interactions with an IoT device, enabling mass actuation or sensing. For example, the action "turn on all the lights on the floor" results in instructing multiple light bulbs to be switched on. The WoT gateways are responsible for mapping an action of the virtual entity to the corresponding action(s) of the real IoT device(s).

From a high-level perspective, the entities in our system interact with each other as follows. Initially, the owner physically deploys the IoT devices and pairs them with a WoT gateway. Then, he/she creates the digital twin of the virtual entity, composed of all IoT devices paired with the corresponding gateway, and deploys them on the blockchain network. Depending on the use case, the design of the digital twin slightly changes (see Section 3). When the setup has been completed, a

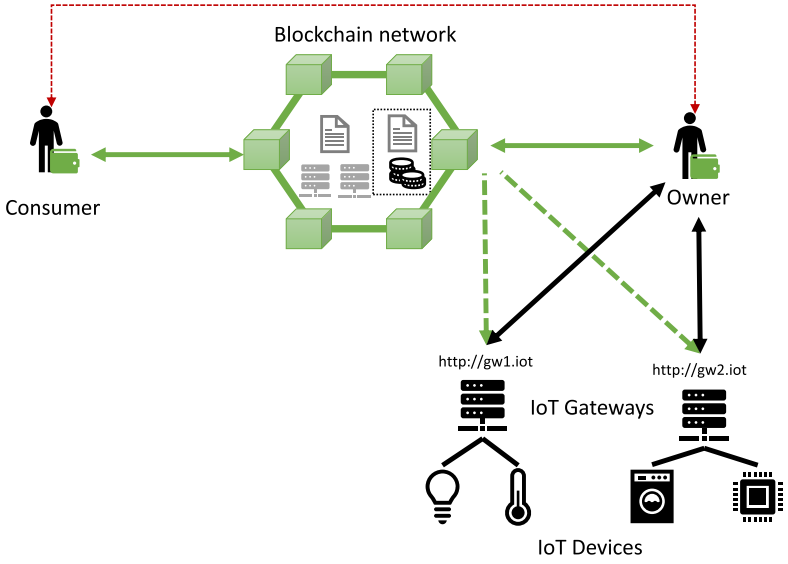

Fig. 7. An overview of the system's architecture.

consumer can gain access to the provided actions, hence to the IoT devices. To that end, he has to obtain permission from the owner. Then, the consumer learns from the smart contract all the available actions and the required parameters. From this point on, a consumer can perform an IoT device access request by sending a transaction to the smart contract-based digital twin. The smart contract verifies the transaction and forwards the request to the appropriate WoT gateway. Finally, the WoT gateway forwards the request to the appropriate IoT device(s), which eventually performs the requested action. The communication between the WoT gateway and the IoT devices is specified by the device vendor and it is out of the scope of this paper. The requested action may lead IoT devices to produce (sensitive) data. We should note here that these data are not stored in the smart contract or in the ledger. As in legacy IoT systems, the data can be stored in a cloud, in the (WoT) gateways, in the endpoint devices, or even in a decentralized storage system, such as the InterPlanetary File System (IPFS) [25]. This choice does not affect our design (in both digital twin implementations) or the system as a whole. However, depending on the selected option, small changes might be needed in the source code. The life cycle of the system involves the following phases.

4.1. Setup phase

Initially, the owner physically deploys the IoT devices and the corresponding WoT gateways. A WoT gateway includes the TD of the virtual entity and exposes all the actions, properties, and events of that virtual entity. Furthermore, the owner creates a smart contract (depending on the use case, he/she chooses the appropriate design, Ethereum-based or Fabric-based), which represents the digital twin of the virtual entity, and he/she deploys it on the blockchain network. The smart contract's address on the blockchain is considered well-known.

In both cases, the owner also has to configure the WoT gateways to "watch" the blockchain for events generated by the corresponding smart contract. In the case of the Ethereum blockchain, he/she should also create and deploy another smart contract that creates and manages the owner-specific tokens, implementing the ERC 20 token standard. A consumer, in order to acquire tokens, contacts the owner to come to an agreement on the price of a token. Then, he/she has to pay the agreed amount of money to the owner in fiat currency. The payment can also be done with cryptocurrencies, e.g., Ether or BTC, with small changes to the source code of the ERC 20 smart contract. In the current design, this process takes place offline and off-chain using fiat currency in order to keep consumer–owner business relationships private. When the payment is completed, the owner transfers the agreed amount of the blockchain-based tokens to the consumer's blockchain wallet. This process does not happen in the Fabric-based design since Fabric is a private blockchain. Thus, in Fabric, a consumer, in order to be able to interact with the smart contract-based digital twin, has to obtain an identity, namely an X.509 certificate. To do so, he/she communicates with the administrator of the network and the corresponding CA to obtain the certificates.

Finally, consumers can learn all the available operations provided by the IoT devices, the appropriate parameters, and the cost of each action (this only applies to the Ethereum-based design) by just calling a function of the smart contract called `getAction`, which returns them.

4.2. IoT device access

From this point on, a consumer can perform an IoT device actuation request or access data request. To do so, he/she sends a transaction to the smart contract-based digital twin that includes the requested action and the corresponding parameters. The smart contract-based digital twin verifies that the transaction is valid. Namely, it checks that the action exists in the actionsList or in the TD and that the parameters are correct. From now on, depending on the case, the flow slightly changes.

(a) Ethereum-based design

In the case of the Ethereum-based design, the consumer has to purchase the required number of ERC 20 tokens. To do so, the consumer pays the owner and the owner calls the `transfer` function of the ERC 20 smart contract to transfer the agreed tokens to consumer. Therefore, the smart contract-based digital twin has to verify that the blockchain address of the consumer has the required number of tokens. In order to perform this verification, it interacts with the smart contract that manages the ERC 20 tokens, and calls the `balanceOf` function, which returns how many tokens an address has. Then, these tokens are deposited at the smart contract's address. If all requirements are met, then an event, named `PerformAction` is triggered, as we can see from Figs. 3 and 4. The event includes the action name, the parameters, and the blockchain address of the consumer. The address of the consumer is needed in order to send the tokens back to him/her, if the action is not completed successfully.

The event is eventually "caught" by the appropriate WoT gateway, which finds out which device should serve the request. Subsequently, the WoT gateway forwards the request to the appropriate IoT device(s), which performs the requested action. When the action has been completed, the WoT gateway configured with the owner's

blockchain wallet, sends a transaction to the blockchain. In particular, it calls the `endOfAction` function to transfer the tokens from the smart contract-based digital twin address to the owner address, if the action is completed successfully, otherwise, it transfers the tokens back to the consumer's blockchain wallet.

In the case of a consumer not spending all of his/her ERC 20 tokens and does not want to use the provided services anymore, he/she has two options. The first option is to give back the remaining tokens to the owner and receive back fiat currency. The second option is to sell his/her tokens to other consumers. The latter allows the creation of secondary markets and enables various new business structures.

(b) Hyperledger Fabric-based design

In Fabric, the flow is the same with small differences. After the validation of the transaction, the smart contract-based digital twin generates an event, which is "caught" by the WoT gateway. Then, the WoT gateway forwards the request directly to the appropriate IoT devices, which eventually perform the requested action. In this case, we do not have a token smart contract, since Fabric is a private, permissioned blockchain, meaning that we do not need to introduce additional access control mechanisms.

4.3. IoT device management

The smart contract-based digital twins include either the whole TD or a stripped-down version of it, the *actionsList*. Both of them contain all the properties, actions, and events of the virtual entity. These structures in smart contracts can only be modified by the owner. In particular, the owner can add a new entry, as well as modify, or even delete an existing one. These operations do not involve any interaction with the consumers. Additionally, an owner can replace a physical IoT device since the consumer interacts with the virtual entity, which is device-independent. This replacement affects only the configuration of the corresponding WoT gateway and its smart contract-based digital twin.

4.4. Ephemeral interactions and revocation

There might be cases where the owner should grant ephemeral permissions to guests. These cases concern mainly the Ethereum-based design, which is a public blockchain, and this design is used in use cases where many untrusted entities interact with the IoT devices. In such cases, the guest should come to an agreement with the owner for the price of tokens and the period during which they need them. Then, the owner sends to the guest the agreed number of tokens and adds the guest to a list, called *guestList*, along with the corresponding period. This list exists in the WoT gateways. From now on, the guest can interact with the IoT devices as any other consumer. The difference is that when the agreed period is up, the owner calls the `withdraw` function of the smart contract that manages the ERC 20 tokens to take back the remaining tokens from the guest. This function can only be invoked by the owner and, in essence, resets the token balance of a consumer, returning all the tokens back to the owner's blockchain wallet. Thus, this function can also be used in cases of security breaches. Therefore, the `withdraw` function can also act as a revocation mechanism. The consumers whose tokens are revoked can no longer have access to IoT devices and perform any action; thus, the revocation is instantaneous.

The Fabric-based design also supports revocation. In the Fabric blockchain, the admin of the network (who can be the owner of the IoT devices/gateways) can revoke the certificate of a consumer. This process creates a certificate revocation list (CRL), which includes the revoked consumers. Then, this CRL is included in the Fabric's network configuration, and the consumer can no longer have access to the blockchain network or to the deployed smart contracts.

Table 1
EVM execution cost of our construction building blocks.

Smart contract	Operation	Cost measured in gas
Digital twin	contract deployment	2341723
	performAction	52329
	endOfAction	43628
	addAction	119934
	modifyAction	41064
	removeAction	46632
	getActions	-
	getAction	-
ERC-20	contract deployment	805618
	balanceOf	-
	transfer	33664
	mintTokens	34786
	withdraw	29450

5. Implementation and evaluation

5.1. Implementation

We developed a proof-of-concept implementation of the presented IoT system. Our WoT gateway is based on Eclipse's Thingweb,[5] which is a Node.js implementation of the WoT model. As an IoT device, we emulated a smart lamp. The smart contract-based digital twin in Ethereum was implemented using Solidity, while in Fabric, it was implemented using JavaScript. Both implementations of smart contract-based digital twins are open-source and available at GitHub,[6] while a simplified version of them is shown in Figs. 4 and 6. Furthermore, our client application for Fabric, which is responsible for acquiring the certificate and interacting with the smart contract-based digital twin through the Fabric gateway, is implemented in JavaScript using the Fabric SDK, while the client application for Ethereum is also implemented in JavaScript using the Web3.js library. Similarly, the corresponding event listeners (i.e., the piece of software that "catches" events generated by the smart contract-based digital twins) are also implemented in JavaScript.

5.2. Cost and performance evaluation

Cost evaluation concerns only the Ethereum-based design, since only that introduces monetary costs. First of all, all actions performed in our system involve the invocation of a function implemented in a smart contract. As we have already mentioned in the previous sections, the invocation of a function incurs a transaction cost, which is the cost of gas for executing transactions on the EVM. In order to measure the cost required by our smart contracts, we deployed the smart contracts on the Rinkeby Ethereum test network.[7] The cost, measured in gas units, for each of the functions is shown in Table 1.

The functions that only access the blockchain for reading introduce zero cost. As we observe from Table 1, the cost introduced by some functions is not negligible. The most expensive functions are those required for deploying smart contracts on blockchain (≈ $49.46 and $17.02),[8] which however take place only once. The function that is responsible for adding a new action to the *actionsList*, i.e., the `addAction` function, costs ≈ $2.53. Similarly, the function used by consumers for invocation, i.e., the `performAction` function, costs ≈ $1.10.

However, we should note here that the price of the actions expressed in fiat currency is not stable since it depends heavily on the price of Ethereum's cryptocurrency (known as Ether), which fluctuates highly. Thus, right now, given the price of ether, the cost may not be too high,

[5] https://www.thingweb.io/.
[6] https://github.com/mmlab-aueb/DLT-DigitalTwins.
[7] https://www.rinkeby.io/#stats.
[8] Prices for December 2022.

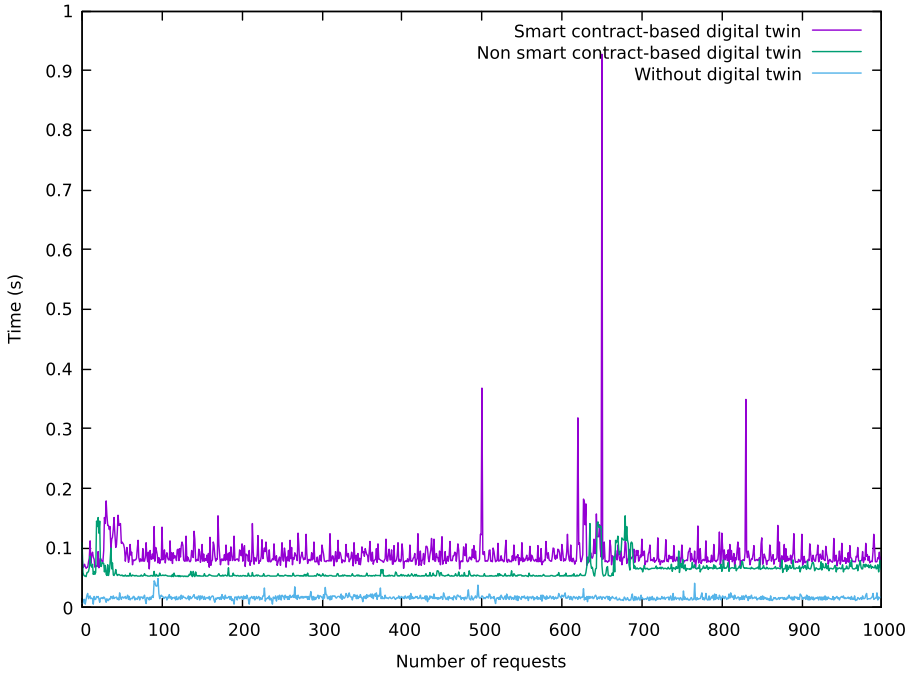

Fig. 8. Time required for requesting an actuation/sensing process.

but it may be prohibitive for some use cases, such as the use case of a smart home.

In addition to gas, the use of the public Ethereum blockchain incurs a transaction delay. The average time required for an operation to be executed on Ethereum is ≈ 15 s. On the other hand, read transactions do not depend on the block mining time; thus, they do not incur any transaction delay.

To evaluate the performance of our Fabric-based design, we conducted some experiments to find out whether the overhead of the blockchain is bearable or not. To evaluate that, we measured the time required for requesting an actuation/sensing process. In particular, we measured the time from the moment a user sends a request to the digital twin until that request ends up at the WoT gateway. For comparison, we implemented an instance of our system without blockchain, i.e., the digital twin is just a Web service, as well as an instance of the solution without digital twin, i.e., the user sends the request directly to the WoT gateway. We conducted this experiment 1000 times. The results are presented in Fig. 8. The average time required for the smart contract-based design is 88 ms; for the non smart contract-based design is 61 ms; and for non digital twin-based is 16 ms. As we observe from the results, the overhead added from the blockchain is ≈ 20 ms, which is tolerable in almost any case.

In Fig. 8, we observe that there are several high spikes that often happen in the smart contract-based digital twin scenario. This is happening due to the ordering service of Fabric. As we mentioned in Section 2.1 (and shown in Fig. 1), transactions are sent to the orderer node in order to be ordered into blocks and then, blocks are sent back to peers to be appended in the ledger. A block is created either after an adequate number of transactions has been collected or after the expiration of a configurable timeout period. Because our experiment does not always generate many transactions, in some cases, the orderer has to wait for the whole timeout period: this is a spike.

5.3. Security evaluation

Our design provides increased protection to WoT gateways. In particular, since digital twins act as an indirect mechanism between the consumers and the WoT gateways, the location of WoT gateways does not have to be known to the consumers; the gateway can even be unreachable through the Internet and disconnected from the outside world. In the Ethereum-based design, where the smart contract-based digital twins and the WoT gateways communicate through events generated by the digital twins, the URLs of the WoT gateways are not included in the smart contract. On the other hand, in Fabric-based design, the URLs are included in the smart contract since the whole TD is included. However, since Fabric is a permissioned blockchain, we assume that consumers who have access to the smart contract-based digital twins are considered trusted by the owner. Furthermore, since the communication with the WoT gateways is achieved through events, we can completely remove the URL of the WoT gateways from the TD. Therefore, our design secures the WoT gateways from a variety of risks and attacks that are common to Web services, such as DoS attacks.

Moreover, in both our designs, we provide a form of access control. In Fabric-based design, we rely on the permission system of Fabric. In the case of the Ethereum-based design, we leverage the ERC 20 token standard. In particular, only users who have acquired the owner-specific tokens can request and invoke the available actions. Thus, these tokens, except for enabling new business models, also act as access control tokens (ACTs). This is quite interesting, since with this design, we offer effective revocation, which has an immediate effect. However, once consumers acquire ERC 20 tokens, they can transfer them to anyone they want. In order to avoid this, the corresponding ERC 20 smart can be configured to allow consumers to transfer their tokens only to the owner's address, e.g., to get reimbursed. Our proposed access control mechanism has some intriguing properties. Mainly, it allows for the swift and immediate resolution of compromised blockchain keys, all thanks to the effective revocation mechanism. Importantly, these actions can be taken without requiring any interaction with the consumer or his/her token.

In addition, our system inherits all the properties of blockchains. DLTs offer reliability, robustness, and increased availability by design since transactions and blockchain are replicated in all miners participating in the blockchain network. Therefore, there is no single point of failure, and the data, the actions, and the smart contracts are always available to the consumers. Thus, by implementing the digital twins as smart contracts, we guarantee that they will always be live and not depend on trusted third parties. Blockchains also offer auditability. Every transaction, hence, every function invocation on the blockchain, is

recorded in a block appended immutably to the ledger. This property enhances our system with high auditability, in the sense that every interaction with an IoT device is recorded and can be revisited at any time. Therefore, in the case of security incidents, blockchains can provide undeniable auditing information, offering non-repudiation.

Our system also follows the same threat model with blockchains, since blockchain is involved in most interactions in the system. In the case of the first design (Ethereum-based), the most important security threats are the 51% attacks, long-range attacks, eclipse and routing attacks, and private key attacks. In the 51% attacks, an attacker controls 51% of the blockchain, and he/she can rewrite the history of the blockchain or refuse to add transactions to it, performing a DoS attack against its users. With these attacks, the attacker gains full control of the blockchain. In addition, by compromising a user's private key, an attacker can generate transactions or perform actions on his/her behalf. On the other hand, Fabric's security threats differ from the popular public, permissionless blockchains. For example, the 51% attacks are not significant threats, since in Fabric users are known and access is managed by access control lists (ACLs). The most significant security threats on Fabric are DoS attacks, MSP compromises, private key attacks, consensus manipulation, and smart contract exploitation [26].

The only part of the system that does not involve the blockchain is the communication between the WoT gateways and the IoT devices. The security of this communication is managed by the protocols used by each vendor, which is, however, out of the scope of this paper.

Finally, combining the WoT with blockchains, that is, storing the WoT TD within a smart contract, does not create new security issues or risks. The WoT architecture and standards are supposed to be implemented in a public Web server, so no new security vulnerabilities arise by implementing them in a (public) smart contract.

5.4. Discussion

Our proposed system has many desirable usability properties, particularly compared to legacy IoT systems. First, new IoT devices can be seamlessly added to our system since the owner has only to update the WoT gateways and the TDs or the *actionsList* in smart contract-based digital twins without having to redeploy the smart contract or the client application. Similarly, the owner can add new WoT gateways that expose new TDs without having to change anything in the underlay system. Furthermore, consumers are IoT device (vendor-)agnostic since they communicate with the IoT devices through the blockchain infrastructure. Consumers do not need to know anything specific about the IoT devices except the actions or data they provide, which they can easily learn from their digital twins. In addition, consumers do not have to deploy and use different client applications for the IoT devices, which is otherwise a very common case. Therefore, to interact with any IoT device of any vendor, they just have to send a transaction to the smart contract-based digital twin.

Our system does not inherit only the benefits of blockchain technology but also its drawbacks. First, as we showed in the performance evaluation, blockchains incur a transaction cost and delay, which in some use cases can be intolerable. Another drawback, especially in the Ethereum-based design, is the need for storage. Blockchain nodes need to store the whole blockchain, which is over 300 GB in Ethereum. However, this does not directly affect our system. Furthermore, there are many workarounds, e.g., light nodes, RPC nodes, etc. In addition, the scalability of our system is also affected. Blockchain scalability is the ability of a blockchain network to process many transactions. In this direction, many solutions have been proposed for scaling blockchains. Regarding the Fabric blockchain, some findings show that Fabric can scale up to 20k transactions per second [27]. On the other hand, many scaling solutions have also been proposed for the Ethereum blockchain, such as sharding, proof of stake variations, or Layer-2 (off-chain) solutions. In our design, the issue regarding scalability lies in the fact that in the smart contract-based digital twin, we use a mapping (*actionsList*) to store the provided actions. Therefore, the question arises as to whether the system is affected by the size of this mapping. Nevertheless, as we insert more actions in the *actionsList*, the gas cost and the transaction delay will not be affected or increased, as Solidity reserves the maximum storage when creating and initializing the mapping (which is 2^{256} slots of 32 bytes).

Next, we discuss the differences between the two implementations of smart contract-based digital twins. Fabric smart contracts include the whole TD of the "virtual entities" as opposed to the Ethereum smart contracts, which should not store the whole TD since that would be too costly. Furthermore, Fabric presents better performance than Ethereum, as we have shown in our previous work [13]. Moreover, Fabric can serve many different use cases since it introduces the concept of organizations. In a Fabric network, there can be two, or more, organizations. Each organization can have its own digital twins without the others knowing anything about them. In Fabric, we can even have smart contracts within the same organization that cannot be accessed by some peers of the same organization through the use of private channels, a technology introduced by Fabric. This is something that cannot be achieved with the use of the Ethereum blockchain, in which everything is public and everything recorded on the ledger can be accessed and inspected by anyone. Therefore, Fabric is a much better fit, both performance and cost-wise. However, as a permissioned blockchain, it is not appropriate for use cases, where openness, decentralization, and full transparency are desired.

6. Conclusions and future work

In this paper, we present a novel design of secure, available, and reliable smart contract-based digital twins of IoT devices that combines WoT standards and blockchain technology. By using the WoT standards, we successfully address the problems of interoperability and fragmentation in the IoT. Moreover, by implementing digital twins as smart contracts, we achieve decentralization, high availability, robustness, flexibility, and auditability. To cover a wide range of IoT use cases and applications, we offer designs in two different blockchains, a public and a private blockchain, each introducing different properties. Furthermore, to verify the feasibility of these digital twins, we designed and developed an IoT system prototype that offers secure sensing and actuation. Our solution secures the real IoT devices and the corresponding WoT gateways by allowing consumers to interact with them only through their smart contract-based digital twins. It also makes the users oblivious to the actual IoT device details and IoT device vendor-agnostic.

As a next step, we are in the process of implementing a smart home testbed to test the solution, as well as looking for a larger testbed, e.g., a smart city or district, to experiment with it in a real setting and assess its performance, efficiency, usability, and scalability.

CRediT authorship contribution statement

Iakovos Pittaras: Conceptualization, Investigation, Methodology, Software, Validation, Writing – original draft. **Nikos Fotiou:** Conceptualization, Writing – review & editing. **Christos Karapapas:** Writing – review & editing. **Vasilios A. Siris:** Project administration, Supervision, Writing – review & editing. **George C. Polyzos:** Conceptualization, Funding acquisition, Project administration, Supervision, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Funding

This work originated with the H2020 project SOFIE (Secure Open Federation for Internet Everywhere, Grant #779984) and has since been supported in part by the Research Center of the Athens University of Economics and Business.

References

[1] W3C, Web of Things, https://www.w3.org/WoT/. (Accessed 22 December 2017).
[2] B.R. Barricelli, E. Casiraghi, D. Fogli, A survey on digital twin: definitions, characteristics, applications, and design implications, IEEE Access. 7 (2019) 167653–167671, https://doi.org/10.1109/ACCESS.2019.2953499.
[3] The Verge, An Amazon server outage caused problems for Alexa, Ring, Disney Plus, and deliveries, https://www.theverge.com/2021/12/7/22822332/amazon-server-aws-down-disney-plus-ring-outage. (Accessed 22 December 2021).
[4] I. Pittaras, N. Fotiou, C. Karapapas, et al., Secure, mass Web of Things actuation using smart contracts-based digital twins, in: Proceedings of the 2022 IEEE Symposium on Computers and Communications (ISCC), IEEE, 2022, pp. 1–6, https://doi.org/10.1109/ISCC55528.2022.9912991.
[5] I. Pittaras, G.C. Polyzos, Secure and efficient Web of Things digital twins using permissioned blockchains, in: Proceedings of the 2022 7th International Conference on Smart and Sustainable Technologies (SpliTech), IEEE, 2022, pp. 1–5, https://doi.org/10.23919/SpliTech55088.2022.9854219.
[6] G. Wood, Ethereum: a secure decentralised generalised transaction ledger, Ethereum Project Yellow Paper 151 2014, https://ethereum.github.io/yellowpaper/paper.pdf.
[7] E. Androulaki, A. Barger, V. Bortnikov, et al., Hyperledger fabric: a distributed operating system for permissioned blockchains, in: Proceedings of the Thirteenth EuroSys Conference, EuroSys '18, ACM, 2018, https://doi.org/10.1145/3190508.3190538.
[8] M. Kovatsch, R. Matsukura, M. Lagally, et al., Web of Things architecture, https://www.w3.org/TR/wot-architecture/. (Accessed 22 December 2020).
[9] S. Kaebish, T. Kamiya, M. McCool, et al., Web of Things thing description, https://www.w3.org/TR/wot-thing-description/. (Accessed 22 December 2020).
[10] M. Andoni, V. Robu, D. Flynn, et al., Blockchain technology in the energy sector: a systematic review of challenges and opportunities, Renew. Sustain. Energy Rev. 100 (2019), https://doi.org/10.1016/j.rser.2018.10.014.
[11] S. Voulgaris, N. Fotiou, V.A. Siris, et al., Hierarchical blockchain topologies for quality control in food supply chains, in: Proceedings of the 2020 European Conference on Networks and Communications (EuCNC), IEEE, 2020, https://doi.org/10.1109/EuCNC48522.2020.9200913.
[12] M.S. Devi, R. Suguna, A.S. Joshi, et al., Design of IoT blockchain based smart agriculture for enlightening safety and security, in: A. Somani, S. Ramakrishna, A. Chaudhary, et al. (Eds.). Emerging Technologies in Computer Engineering: Microservices in Big Data Analytics, Springer, Singapore, 2019, https://doi.org/10.1007/978-981-13-8300-7_2.
[13] I. Pittaras, N. Fotiou, V.A. Siris, et al., Beacons and blockchains in the mobile gaming ecosystem: a feasibility analysis, Sensors. 21 (3) (2021), https://doi.org/10.3390/s21030862.
[14] N. Fotiou, G.C. Polyzos, Smart contracts for the Internet of Things: opportunities and challenges, in: Proceedings of the 2018 European Conference on Networks and Communications (EuCNC), IEEE, 2018, https://doi.org/10.1109/EuCNC.2018.8443212.
[15] Z. Chevallier, B. Finance, B.C. Boulakia, A reference architecture for smart building digital twin, in: 2020 International Workshop on Semantic Digital Twins, SeDiT 2020, France, vol. 2615, 2020.
[16] Z. Liu, A. Zhang, W. Wang, A framework for an indoor safety management system based on digital twin, Sensors. 20 (20) (2020), https://doi.org/10.3390/s20205771.
[17] N. Mohammadi, J.E. Taylor, Smart city digital twins, in: Proceedings of the 2017 IEEE Symposium Series on Computational Intelligence (SSCI), IEEE, 2017, https://doi.org/10.1109/SSCI.2017.8285439.
[18] G. White, A. Zink, L. Codecá, et al., A digital twin smart city for citizen feedback, Cities. 110 (2021), https://doi.org/10.1016/j.cities.2020.103064.
[19] I. Yaqoob, K. Salah, M. Uddin, et al., Blockchain for digital twins: recent advances and future research challenges, IEEE Netw. 34 (5) (2020), https://doi.org/10.1109/MNET.001.1900661.
[20] A. Khan, F. Shahid, C. Maple, et al., Toward smart manufacturing using spiral digital twin framework and twinchain, IEEE Trans. Ind. Inform. 18 (2) (2022) 1359–1366, https://doi.org/10.1109/TII.2020.3047840.
[21] B. Putz, M. Dietz, P. Empl, et al., EtherTwin: blockchain-based secure digital twin information management, Inf. Process. Manag. 58 (1) (2021) 102425, https://doi.org/10.1016/j.ipm.2020.102425.
[22] M. Dietz, B. Putz, G. Pernul, A distributed ledger approach to digital twin secure data sharing, in: 33rd S. Foley. (Eds.). IFIP Annual Conference on Data and Applications Security and Privacy (DBSec), in: LNCS, vol. 11559, 2019, pp. 281–300, https://doi.org/10.1007/978-3-030-22479-0_15.
[23] W. Shen, T. Hu, C. Zhang, et al., Secure sharing of big digital twin data for smart manufacturing based on blockchain, J. Manuf. Syst. 61 (2021), https://doi.org/10.1016/j.jmsy.2021.09.014.
[24] F. Vaogelsteller, V. Buterin, ERC-20: token standard, https://eips.ethereum.org/EIPS/eip-20. (Accessed 22 December 2015).
[25] J. Benet, IPFS-content addressed, versioned, P2P file system, arXiv. 2014. preprint. arXiv:1407.3561.
[26] Christopher Cordi, Hyperledger fabric security threats: what to look for, https://www.hyperledger.org/blog/2021/11/18/hyperledger-fabric-security-threats-what-to-look-for. (Accessed 22 December 2021).
[27] C. Gorenflo, S. Lee, L. Golab, et al., FastFabric: scaling hyperledger fabric to 20000 transactions per second, in: Proceedings of the 2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC), IEEE, 2019, pp. 455–463, https://doi.org/10.1109/BLOC.2019.8751452.