Pervasive Smart Contracts for Blockchains in IoT Systems

Amir Taherkordi[†‡] and Peter Herrmann[‡]
[†]University of Oslo, Oslo, Norway
amirhost@ifi.uio.no
[‡]Norwegian University of Science and Technology (NTNU), Trondheim, Norway
herrmann@ntnu.no

ABSTRACT

Thanks to its decentralized structure and immutability, blockchain technology has the potential to address relevant security and privacy challenges in the Internet of Things (IoT). In particular, by hosting and executing *smart contracts*, blockchain allows secure, flexible, and traceable message communication between IoT devices. The unique characteristics of IoT systems, such as heterogeneity and pervasiveness, however, pose challenges in designing smart contracts for such systems. In this paper, we study these challenges and propose a design approach for smart contracts used in IoT systems. The main goal of our design model is to enhance the development of IoT smart contracts based on the inherent *pervasive attributes* of IoT systems. In particular, the design model allows the smart contracts to encapsulate functionalities such as contract-level communication between IoT devices, access to data-sources within contracts, and interoperability of heterogeneous IoT smart contracts. The essence of our approach is structuring the design of IoT smart contracts as self-contained software services, inspired by the microservice architecture model. The flexibility, scalability and modularity of this model make it an efficient approach for developing pervasive IoT smart contracts.

CCS CONCEPTS

• **Applied computing → Service-oriented architectures**;

KEYWORDS

Blockchains, Internet of Things, Smart Contracts, Microservices.

ACM Reference Format:
Amir Taherkordi[†‡] and Peter Herrmann[‡]. 2018. Pervasive Smart Contracts for Blockchains in IoT Systems. In *2018 International Conference on Blockchain Technology and Application (ICBTA 2018), December 10–12, 2018, Xi'an, China.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3301403.3301405

ICBTA 2018, December 10–12, 2018, Xi'an, China

ACM ISBN 978-1-4503-6646-5/18/12.
https://doi.org/10.1145/3301403.3301405

1 INTRODUCTION

Smart devices have facilitated the pervasive presence of a variety of things, interacting and cooperating with each other through unique addressing schemes—Internet of Things (IoT). These devices often exchange huge amounts of security, safety-critical and privacy-sensitive data. This makes them tempting targets for various cyber attacks [25]. Equipping such devices with appropriate security and privacy support mechanisms is a challenging task due to their resource limitations, in particular, their often limited computing power and energy supply. In addition, many state-of-the-art security solutions are highly centralized and not well-suited for IoT systems due to the lack of scalability, many-to-one nature of the traffic, and single points of failure [20]. With respect to privacy, existing privacy preserving methods rely on revealing noisy, incomplete or summarized data to the data requester, while many IoT applications require users to reveal precise data in order to receive personalized services [5]. For all these reasons, many existing security and privacy technologies do not fit well to IoT systems which demand more lightweight, scalable, and distributed security and privacy solutions. Blockchain technology has the potential to meet these demands thanks to its decentralized, secure, and private nature [4].

Blockchain treats message exchanges between IoT devices similarly to financial transactions in Bitcoin, a popular application of this technology. To enable message exchanges, the IoT devices leverage *smart contracts* defining the rules according to which the communication has to take place [3]. IoT systems can also benefit from smart contracts for other purposes such as tracing consumer-to-machine and machine-to-machine transactions. The unique characteristics of IoT systems, such as autonomy of IoT devices and their interactions, device heterogeneity, and pervasive communication, however, make the design and development of IoT smart contracts often challenging.

There exists a number of smart contract programming models such as Solidity [21] which are suitable for conventional blockchain systems. Moreover, their design is based on simple function-based programming, without high-level abstractions for better engineering of smart contract code which is required for pervasive and heterogeneous systems like IoT. In the context of smart contracts design for IoT, most existing approaches mainly focus on addressing the scalability issue for blockchains integrated with large-scale IoT systems in terms of limited bandwidth connections and processing capabilities in IoT deployments [5].

In this paper, we study the challenges in designing smart contracts for blockchains integrated with IoT systems, including autonomous operations of smart devices, heterogeneity of smart contract terms, and intermittent communication between IoT devices involved in a transaction. We term the smart contract that addresses these design challenges as *pervasive smart contract*. The main design principle behind a pervasive smart contract is encapsulating functionalities specific to IoT smart contracts, including contract-level communication between IoT devices, access to external data-sources within contracts, and supporting interoperability of heterogeneous IoT smart contracts. To this end, we adopt a self-contained lightweight development model for IoT smart contracts, inspired by the microservice architecture model. The flexibility, scalability, and technology diversity of the proposed model make it an efficient solution for developing pervasive smart contracts. Our initial implementation of the proposed development model for pervasive smart contracts focuses on tackling issues caused by the secured and isolated sandboxed runtime environment within which the contracts are hosted.

The rest of this paper is organized as follows. In Section 2, an overview of integration of IoT and blockchains is provided. The design aspects of smart contracts in IoT systems are discussed in Section 3, followed by a microservice-based design model proposed for developing pervasive IoT smart contracts. In Section 4, the implementation highlights and challenges are discussed. Related work and concluding remarks are presented in Sections 5 and 6, respectively.

2 BLOCKCHAINS AND IOT: AN OVERVIEW

Currently, digital trade mostly relies on trusted third-party authorities for handling financial or operational transactions, *e.g.* a bank which confirms the delivery of money to a person. These trusted third parties control and manage all data and information and typically use a centralized costly system for transaction processing. Moreover, they can be hacked, compromised or administered by malicious agents. This is where *blockchain technology* is introduced. It solves the mentioned issues by creating a decentralized system without the need for such third parties. A blockchain is basically a distributed data structure, or a public ledger of all transactions or digital events executed and shared among participating parties [10]. Each transaction in the public ledger is immutable and verified by consensus of a majority of the participants in the system. Blockchain makes trustless, peer-to-peer messaging possible without the need for third-party brokers. Thus, it can be used for financial services, *e.g.* cryptocurrencies such as Bitcoin [13].

IoT systems often consist of a large number of heterogeneous smart devices that interact with each other or with other networks and platforms such as clouds. IoT devices usually generate a huge amount of data that has to be coordinated. Due to the varying nature, context, and location of the nodes, this coordination can be quite difficult. Moreover, the transmitted data is often sensitive since it may contain personal information about device owners and users. In addition, the users' behaviors and preferences may be revealed. To preserve the security and privacy of IoT data and coordinate the flow of IoT data among different devices and systems, several security frameworks have been proposed which are usually highly centralized. As discussed above, that makes these frameworks inept for IoT systems due to the difficulty to scale the systems, the many-to-one nature of the IoT data traffic, and the existence of single points of failure [20, 27].

The decentralized nature of blockchain without the need to involve trusted third parties makes it possible to eliminate single points of failure and centralized management of sensitive data by a third party. This makes blockchain an ideal solution to provide a secure tamper-proof IoT network. The blockchain treats the processing of transactions and communication and coordination between IoT devices similarly to financial transactions in Bitcoin. In this way, a more resilient and unified ecosystem is created for smart devices in an IoT network to interact and exchange data securely. When two IoT devices want to exchange messages, they negotiate so-called *smart contracts* that state the terms to be met by the two parties in their interaction [3]. The concept of smart contract was first introduced by Nick Szabo as "*a computerized transaction protocol that executes the terms of a contract*"[23].

3 DESIGNING SMART CONTRACTS FOR IOT SYSTEMS

A smart contract is realized as a script that is stored on the blockchain with a unique address. The script contains references to transactions that are automatically executed for data exchanged between two devices following the smart contract. In this way, the cooperating parties can directly deal with each other without having to rely on a central system. The smart contracts are stored in blocks that are electronically linked to one another in a blockchain. Since all users own a copy of the stored contracts, all kinds of exploits and contract tampering are prevented.

IoT systems can benefit from smart contracts for different purposes such as consumer-to-machine and machine-to-machine transactions, developing traceability applications, etc. For instance, in cloud-based manufacturing platforms, smart contracts act as agreements between the service consumers and the manufacturing resources to provide on-demand manufacturing services [2]. As another example, in supply chain systems, smart contracts can maintain a registry of products and track their position through different points in a supply chain through cryptographically verifiable receipts for product delivery [3].

In the following, we discuss the main issues in designing IoT smart contracts that result from the nature of IoT data, the IoT network architecture, and the unique properties of IoT applications:

Autonomous execution: This feature enables the autonomous operation of smart devices without the need for a centralized authority. In blockchains, the autonomy of smart contracts is typically limited to the automatic execution of contract terms, while triggering the execution of a smart contract function is basically performed through a user transaction in the blockchain. In the case of IoT, a higher level of autonomy is required since the functions in a smart contract are often triggered and executed based on the particular contextual situation of the devices in the environment. For instance, two

devices managing the handing over of a physical good between two transport vehicles only need to coordinate when the vehicles are close by.

Heterogeneous contracts: Key players in smart contracts for IoT systems are the IoT devices themselves. They may directly interact with each other to fulfill a requirement or take part in executing a workflow, *e.g.* in logistics management systems. In a manufacturing scenario, for instance, plenty of smart devices may need to communicate, where each device possesses its own settings in terms of semantics for describing blockchain transactions and programming smart contracts and their dependencies (*e.g.* database access or network communication). This implies that we cannot rely on a single pre-defined smart contract description model that can serve as a general model for designing and developing smart contracts for IoT systems, but we need to deal with a plethora of different contracts. Therefore, a high-level heterogeneity support is required to enable developing and deploying IoT smart contracts that may contain terms and transactions with different semantics.

Intermittent information flow: IoT devices communicate intermittently. Thus, when a device is granted and verified to perform a transaction, we should not expect that the device is necessarily online. This may lead to a temporary halt in the information flow that involves smart contracts. For example, in the case of a supply chain, the lack of network connectivity during the delivery of a shipment may result in the lack of delivery confirmation, and consequently no payment to the supplier will be made. Smart contracts for IoT systems should be designed in such a way to minimize the effect of intermittent IoT communication on their execution and the information flow.

In the following, we propose a software design model for smart contracts in IoT systems which is aimed to address the above mentioned concerns. We term such types of contracts as *pervasive smart contracts*.

3.1 Pervasive IoT Smart Contracts

The state-of-the art smart contract design and development models are mainly focused on supporting primitive actions such as transferring of digital currencies or assets between parties under certain conditions. From a development perspective, the scope of a smart contract implementation code is often limited to manipulating variables (*e.g.* currencies) that are globally defined in the blockchain hosting the smart contracts. We aim to enhance this traditional development model with features that primarily not only support intra-blockchain interactions within smart contracts, but also enable inter-blockchain communication to other nodes and resources in the IoT network. Beyond the challenges in enabling interaction between a contract with the external world, the enhanced development model has to adhere to the core implementation model of a smart contract—a self-contained software module with all required dependencies embedded in the implementation of the contract.

The basic idea behind our approach is structuring the design of IoT smart contracts as self-contained software services inspired by the microservice architecture. This is a novel software development method focused on building single-function modules with well-defined interfaces and operations. In a monolithic application, built as a single software unit, a change made to a small section of code might require building and deploying an entirely new software version which leads to low flexibility and scalability in engineering applications. The microservice architecture is defined as a design approach in which a monolithic application is built as a suite of small services, each running in its own process and communicating using lightweight mechanisms. Such a service is small, independently deployable, highly decoupled and usually carries out only small tasks [15]. Microservices may be developed in different languages and use different data storage techniques, while they promise scalable and flexible development of systems.

Microservices has been recently introduced for IoT systems because of the continuous evolvement of IoT applications and growth in the scale of monolithic applications with more complexity in their structure [22]. Considering their unique characteristics and the challenges discussed above, microservices promise to be an efficient design choice for building pervasive IoT smart contracts. In particular, microservices can encapsulate all design concerns of a pervasive contract in a single software unit. Therefore, we use them as a fundamental model for our approach.

Figure 1 shows the main elements of the model used to integrate IoT and the blockchain based on the design concerns of pervasive smart contracts. On the right side, there is a peer-to-peer network for hosting and maintaining the blockchain in which each node has a copy of the blockchain. It should be noted that the nodes hosting the blockchain are assumed to be quite powerful. Thus, they are different from the IoT devices which serve only as the *clients of a blockchain*. Each block in the blockchain can contain normal transactions and/or the bytecode of smart contracts. Having the address of the smart contract code available, other transactions can execute a smart contract function and create new transactions. Furthermore, IoT devices should be equipped with client code to interact with the blockchain. The interaction can be either a *normal transaction* or a *smart contract transaction*. The former refers to creating typical transactions which should be stored in a block, *e.g.* transferring digital assets like Bitcoin from an account to another one when a delivery is performed in the supply chain application. In contrast, the smart contract transaction can be either creating and posting a new smart contract to the blockchain, or invoking a function of a given smart contract deployed on the blockchain.

As mentioned above, smart contracts are basically computer code stored in blocks, containing a set of functions implementing the terms of a contract, so-called Contract Functions. The top right part of Figure 1 includes three sample smart contracts for the logistic and supply chain applications. In their traditional design, each one includes only Contract Functions that can, for instance, be implemented using the language Solidity in the Ethereum blockchain platform [21]. The top left part of the figure shows our proposed model for implementing pervasive smart contracts as microservices. In this model, a microservice contains not only the Contract Functions, but also functionalities that are essentially specific to a pervasive smart contract and address the aforementioned IoT smart

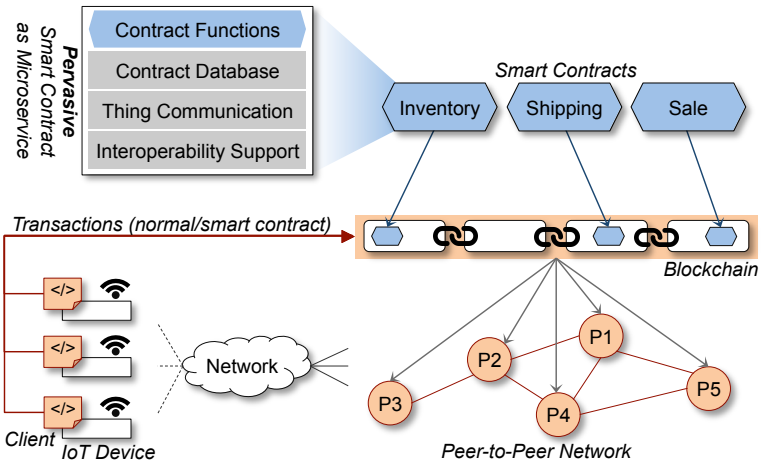

Figure 1: Overall integration model for IoT and blockchain with pervasive smart contracts as microservices

contract design concerns. In the rest of this section, we discuss these functionalities in details.

Thing Communication is a key functional requirement, arising from the fact that smart contracts cannot directly access and fetch the data they require, *e.g.* traffic-related information for estimating cost in the workflow of goods delivery. For that, the smart contract requires to communicate with a third-party system or another IoT device to complete the execution of a contract function. For example, to fetch the required information, the device can establish a RESTful communication with another one having IP-access. This functionality also handles failures in communication with other devices to overcome the issue of intermittent information flow. For that, the communication between devices and the contract can exploit message-oriented middleware technologies such as the Advanced Message Queuing Protocol (AMQP) [24] or the Message Queuing Telemetry Transport (MQTT) [11].

Additionally, Thing Communication encompasses logic for the autonomous execution of contract functions based on different contextual situations of the IoT application. For instance, in the asset tracking use case, the `sendMoney` function will be executed if a container and a retailer share the same location [3]. To this end, every stakeholder carries a BLE, GSM or LTE radio to maintain the current location of smart devices. Then, the IoT application triggers the blockchain Client on the devices that are co-located.

The Contract Database allows a contract to communicate with a trusted data provider. For example, the data source can be a secure application running on an hardware-enforced Trusted Execution Environment (TEE) such as the Industrial IoT TEE for Edge Devices (IIoTEED) [19]. The Contract Database serves as a data access point for an individual smart contract. A relevant example of a data source is IPFS (Interplanetary File System). IPFS files are content-addressed and identified by their hashes. In order to fetch a data file, the entire network is searched for a file corresponding to a particular hash. Thus, it is an ideal file storage and sharing technology for developing decentralized IoT access control models [1].

It should be noted that, according to the general specification of smart contracts, their communication with the off-chain world is either limited (*i.e.* to other smart contracts) or not allowed at all. For instance, the Ethereum Virtual Machine (EVM) [7] is completely isolated in a sandbox, *i.e.* the code running inside the EVM has no access to the network, file system or other processes. To communicate with other parties like a data source, we need to make sure that the data fetched from the original data-source is genuine and untampered. One solution, developed by Oraclize [17], is to accompany the returned data together with a document called authenticity proof, which can be built using technologies such as auditable virtual machines and TEE. In the next section, we discuss this issue in detail.

Interoperability Support is proposed to support heterogeneity among devices interacting with each other based on a smart contract. The most common type of heterogeneity appears in the semantics for describing the transactions added to the blockchain by executing different smart contracts. For example, the high diversity of IoT devices in logistic applications can lead to workflow transactions that are semantically heterogeneous. Interoperability Support encompasses mechanisms for interpreting transactions produced by the corresponding smart contract to a general form interpretable and traceable by the blockchain.

4 IMPLEMENTATION HIGHLIGHTS

A number of well-known blockchain platforms exist featuring smart contract functionality, such as Ethereum [6], Hyperledger Fabric [8], and NEO [14]. In this paper, we adopt Ethereum as the smart contract development framework. Its main advantage is the high degree of standardization and support it offers. In particular, the providers of Ethereum have made extensive effort to improve the development and operation of smart contracts. Additionally, it comes with a set of well-defined rules to develop smart contracts which make the development process easier and less risky. Ethereum features its own high-level programming language *Solidity* for smart contracts which facilitates the development and setting up of smart

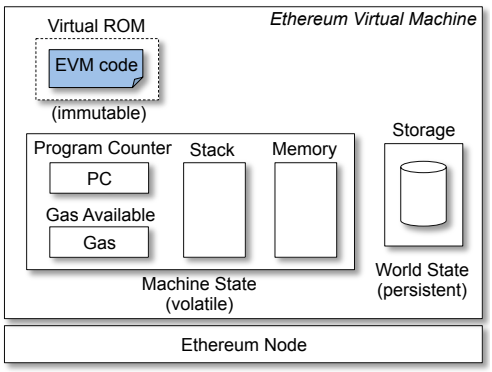

Figure 2: The main architectural elements of the EVM

contracts [21]. Solidity is influenced by C++, Python and JavaScript and is designed to let smart contracts run on the Ethereum Virtual Machine (EVM). The language is statically typed, and supports inheritance, libraries and complex user-defined types.

Our main implementation goal is to provide a programming framework that enables the developer to encapsulate the contract functions and functionalities in a microservice. This requires to understand the runtime environment of smart contracts in Ethereum. In particular, we need to investigate the features and limitations of the *secured* runtime environment for smart contracts. As mentioned before, Ethereum smart contracts run on the EVM which is a sandboxed, completely isolated runtime environment. Thus, no smart contract hosted by the EVM has access to the network, file system, data sources, or other processes running on the computer hosting the EVM. This makes the implementation of the proposed microservice model challenging, in particular, with respect to the functionalities that need interaction with one or more of the aforementioned sources. To tackle this issue, we first need to look into the architectural design of the EVM.

Figure 2 depicts the main architectural element of the EVM. The byte code of the smart contract is hosted in an immutable virtual ROM within the EVM which manages three different kinds of data: memory, stack, and storage. Memory and stack are volatile spaces used to store data during execution and small local variables, *e.g.* passing arguments to internal functions. Storage is a persistent read-write word-addressable space in which the contract stores its persistent information. Compared to other operations of the EVM, the transaction costs required to save data into the storage (called *gas* in the blockchain context) are considerably high.

The inside architecture of the EVM shows that smart contracts have the capability to communicate with the storage for storing and retrieving data, even though this space is limited to some extent because it is structured as a key-value mapping of 2^{256} slots of 32 bytes each. Moreover, there is no possibility to communicate with entities out of the EVM. Considering the proposed microservice design, the storage of the EVM can serve as Contract Database. In this way, for each smart contract C we can create a new *database contract* C_d which is used only to store and retrieve data of C. By separating the contract database from the contract itself, the deployment and management of new versions of the contract in the blockchain will become less costly. In order to get access to C_d from C, cross-contract message calls can be leveraged. Contracts can call other contracts through *message calls*. To call a function of another contract in Solidity, so-called *external functions* should be defined. The following code snippet exemplifies the structure of such a database contract C_d:

```
pragma solidity ^0.4.0;
contract MyContractDB {
    uint public dataA = valueA;
    uint public dataB = valueB;
    ....
    function setDataA(uint value)
               external returns (uint) {
        dataA = value;
        return dataA;
    }
    function  getDataA(uint value)
                external returns (uint) {
        ...
    }
}
```

Using the contract Application Binary Interface (ABI), other contracts in Ethereum can interact with C_d.

As mentioned above, the size of storage is somewhat limited and, beyond this, the gas cost of interacting with the storage is quite high. A complementary solution to the above data access model is using Oraclize [17]. It supports various types of data sources, such as URL (to access to any webpage or HTTP API endpoint) and IPFS (to access to any content stored on an IPFS file). Data from these sources can be retrieved using queries. A query is an array of parameters that are evaluated to complete a specific data source type request. For instance, in the case of sample smart contracts in Fig. 1, the Shipping contract requires to communicate with a device-specific locally-deployed service that calculates the shipment cost of a specific IoT device category. Using Oraclize, the corresponding query is described as:

```
oraclize_query("URL", "http://127.0.0.1/
                       ShipmentCost?device=VerticalPump")
```

The result of executing the query will be the execution of a transaction carrying the result. In the default configuration, the transaction will execute a `_callback` function which is implemented by the developer in the smart contract.

To conclude, the Contract Database can be either realized as a new contract co-located with the main contract in the EVM or deployed as a separate data-source service external to the contract deployed in the EVM. In the latter case, the service will be accessible through a REST API or IPFS, using Oraclize libraries. Likewise, for functionalities related to Thing Communication, Oraclize can be leveraged to communicate with IoT services that are external to Ethereum nodes. For implementing Interoperability Support, similar to C_d, we propose developing new smart contracts with a set of well-defined external functions that merely perform semantic analysis and mapping. In this way, such functionality will serve as a reference for semantic interoperability between heterogeneous IoT smart contracts. Shared by all three functionalities introduced above, our implementation approach meets the essential programming requirements for developing pervasive IoT smart contracts as microservices.

5 RELATED WORK

Although smart contracts have recently received considerable attention by the research community and industry, most existing work on IoT smart contracts has so far focused on the issues in integrating blockchains with IoT, such as designing lightweight blockchains for IoT.

In [26], a smart contract-based framework is proposed to implement distributed and trustworthy access control for IoT systems. The authors use the Ethereum smart contract platform to provide an access control method for static and dynamic access rights validation. In [16], a blockchain-based solution is proposed to address scalability in managing access in large-scale IoT systems. From a different view to scalability, Dorri et al. propose a lightweight scalable blockchain model to overcome the concerns of limited scalability, significant bandwidth overheads and delays for blockchains integrated with IoT [5]. EdgeChain [18] uses a credit-based resource management system to control the resources of IoT devices that are obtained from the edge server. The authors propose using smart contracts to regulate the behavior of the IoT devices by enforcing certain policies. The above approaches are mainly focused on addressing the scalability issue in IoT smart contracts.

In [3], Christidis and Devetsikiotis discuss how smart contracts allow for automated complex multi-step IoT processes. The authors indicate that smart contracts enable cryptographic verifiability of IoT workflow and significant cost and time savings in IoT workflow execution. A decentralized, peer-to-peer blockchain platform for industrial IoT is proposed in [2] to enable cloud-based manufacturing and on-demand access to manufacturing resources . Both above approaches are mainly about the usefulness of smart contracts in executing IoT workflows. In [12], a microservice architecture is introduced for developing scalable and secure smart surveillance systems. For data protection and synchronization, the framework uses blockchain and smart contracts. However, in that framework, microservices are proposed for the components of the surveillance systems, but not for the design of smart contracts.

Among IoT-specific blockchain platforms, IOTA is a distributed ledger designed to process and execute transactions between machines in the IoT ecosystem [9]. The main focus of IOTA is building a high speed and fast transaction processing environment with better transaction validating. For that, the so-called Distributed Ledger Technology follows a Directed Acyclic Graph (DAG) approach to implement distributed consensus. For that, it features a new consensus algorithm known as Tangle. However, IOTA is not designed as a blockchain platform focussing on smart contracts. In other words, smart contracts are not yet a core feature of IOTA. The IOTA Foundation, however, is actively working on a new smart contract layer on top of the current IOTA platform.

6 CONCLUSIONS AND FUTURE WORK

Blockchain technology and smart contracts pose great potential for automating, securing and scaling message communication in IoT systems. In this paper, we studied the design concerns in using smart contracts for IoT systems such as autonomous operations of smart devices, heterogeneity of contract terms, and intermittent communication between devices involved in a transaction. To address these concerns, we proposed a microservice-based approach to develop such types of contracts, named *pervasive smart contracts*. The adoption of the microservice design model tackles challenges such as heterogeneity and pervasiveness in designing IoT smart contracts. However, implementing IoT smart contracts as microservices with the proposed functionalities comes with some programming challenges that we explored further, *e.g.* access to external data sources within a contract. In the future, we will investigate on the interoperability of pervasive contracts and requirements for *container* platforms that can host pervasive smart contracts.

REFERENCES

[1] Muhammad Salek Ali, Koustabh Dolui, and Fabio Antonelli. 2017. IoT Data Privacy via Blockchains and IPFS. In *Proceedings of the Seventh International Conference on the Internet of Things (IoT '17)*. ACM, 14:1–14:7.

[2] Arshdeep Bahga and Vijay K Madisetti. 2016. Blockchain Platform for Industrial Internet of Things. *Journal of Software Engineering and Applications* 9, 10 (2016), 533–546.

[3] K. Christidis and M. Devetsikiotis. 2016. Blockchains and Smart Contracts for the Internet of Things. *IEEE Access* 4 (2016), 2292–2303.

[4] A. Dorri, S. S. Kanhere, R. Jurdak, and P. Gauravaram. 2017. Blockchain for IoT Security and Privacy: The Case Study of a Smart Home. In *IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*. IEEE Computer, Kona, HI, USA, 618–623.

[5] Ali Dorri, Salil S. Kanhere, Raja Jurdak, and Praveen Gauravaram. 2017. LSB: A Lightweight Scalable BlockChain for IoT Security and Privacy. *CoRR* abs/1712.02969 (2017), 1–17. http://arxiv.org/abs/1712.02969

[6] Ethereum Project. Accessed: 2018. http://www.ethereum.org.

[7] Ethereum Virtual Machine. Accessed: 2018. http://ethdocs.org/en/latest/introduction.

[8] Hyperledger Fabric. Accessed: 2018. http://www.hyperledger.org/projects/fabric.

[9] IOTA. Accessed: 2018. https://www.iota.org.

[10] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou. 2016. Hawk: The Blockchain Model of Cryptography and Privacy-Preserving Smart Contracts. In *IEEE Symposium on Security and Privacy (SP)*. IEEE Computer, CA, USA, 839–858.

[11] Dave Locke. 2010. *MQ telemetry transport (MQTT) V3.1 protocol specification*. Technical Report. IBM developer Works Technical Library.

[12] Deeraj Nagothu, Ronghua Xu, Seyed Yahya Nikouei, and Yu Chen. 2018. A Microservice-enabled Architecture for Smart Surveillance using Blockchain Technology. *CoRR* abs/1807.07487 (2018), 4 pages. http://arxiv.org/abs/1807.07487

[13] Satoshi Nakamoto. 2008. Bitcoin: A Peer-to-peer Electronic Cash System,âĂİ http://bitcoin.org/bitcoin.pdf.

[14] NEO. Accessed: 2018. http://neo.org.

[15] Sam Newman. 2015. *Building Microservices*. O'Reilly Media, CA, USA.

[16] O. Novo. 2018. Blockchain Meets IoT: An Architecture for Scalable Access Management in IoT. *IEEE Internet of Things Journal* 5, 2 (2018), 1184–1195.

[17] Oraclize. Accessed: 2018. https://docs.oraclize.it.

[18] J. Pan, Jianyu Wang, Austin Hester, Ismail AlQerm, Yuanni Liu, and Ying Zhao. 2018. EdgeChain: An Edge-IoT Framework and Prototype Based on Blockchain and Smart Contracts. *CoRR* abs/1806.06185 (2018), 14 pages.

[19] S. Pinto, T. Gomes, J. Pereira, J. Cabral, and A. Tavares. 2017. IIoTEED: An Enhanced, Trusted Execution Environment for Industrial IoT Edge Devices. *IEEE Internet Computing* 21, 1 (2017), 40–47.

[20] Rodrigo Roman, Jianying Zhou, and Javier Lopez. 2013. On the Features and Challenges of Security and Privacy in Distributed Internet of Things. *Computer Networks* 57, 10 (2013), 2266–2279.

[21] Solidity Programming Language. Accessed: 2018. http://solidity.readthedocs.io.

[22] L. Sun, Y. Li, and R. A. Memon. 2017. An Open IoT Framework based on Microservices Architecture. *China Communications* 14, 2 (February 2017), 154–162.

[23] Nick Szabo. 1994 (Accessed: 2018). http://szabo.best.vwh.net/smart.contracts.html.

[24] Steve Vinoski. 2006. Advanced Message Queuing Protocol. *IEEE Internet Computing* 10, 6 (2006), 87–89.

[25] Y. Yang et al. 2017. A Survey on Security and Privacy Issues in Internet-of-Things. *IEEE Internet of Things Journal* 4, 5 (Oct 2017), 1250–1258.

[26] Yuanyu Zhang, Shoji Kasahara, Yulong Shen, Xiaohong Jiang, and Jianxiong Wan. 2018. Smart Contract-Based Access Control for the Internet of Things. *CoRR* abs/1802.04410 (2018), 1–11. http://arxiv.org/abs/1802.04410

[27] J. Zhou, Z. Cao, X. Dong, and A. V. Vasilakos. 2017. Security and Privacy for Cloud-Based IoT: Challenges. *IEEE Communications Magazine* 55, 1 (2017), 26–33.