2019 IEEE International Congress on Internet of Things (ICIOT)

Confidential Discovery of IoT devices through Blockchain

Christian Rondanini
DiSTA
University of Insubria, Italy
Email: c.rondanini@uninsubria.it

Barbara Carminati
DiSTA
University of Insubria, Italy
Email: barbara.carminati@uninsubria.it

Elena Ferrari
DiSTA
University of Insubria, Italy
Email: elena.ferrari@uninsubria.it

***Abstract*—Selection criteria regulating IoT device discovery involve confidentiality issue on the information the constraints convey. A promising approach to cope with this issue is leveraging on blockchain technology and smart contracts to implement the overall discovery process deployment. However, due to the blockchain design, data within the blockchain is public and smart contracts cannot access data outside the blockchain, unless through the exploitation of Oracles. On the one hand, this brings benefits of trust decentralization, transparency, and accountability of the discovery process. On the other hand, it carries serious consequences on confidentiality and privacy as well as on Oracles trustworthiness. For these reasons, in this paper, we investigate how to ensure data confidentiality during the discovery process of IoT devices on blockchain even in the presence of an untrusted Oracle.**

***Index Terms*—Internet of Things; IoT Device Discovery; Blockchain; Confidentiality.**

I. Introduction

Today, Internet of Things (IoT) represents one of the most significant field of interest from the industrial, as well as the academic and research community, with near $874M raised in the Q2/2018 and a trillion dollar impact.[1] Past studies have predicted that by 2020 more than 50 billion IoT devices will be connected to the internet. At the time of writing, this number is close to 24 billion, with an extremely high variety of involved applications and services, that are increasing every day.

In the IoT environment, "things" are connected devices (e.g., sensors) with remote sensing and/or actuating capabilities, that can exchange collected data with other connected devices and applications. In general, they can collect and process the data either locally, or remotely, through cloud-based application back-ends [1]. The scale of IoT solutions can range from a simple application that monitors and reacts to the temperature of a sensor, to a full-blown enterprise application that manages and controls a set of buildings.

As a single IoT device has often a limited computational power, many of the most interesting IoT services are offered through a collaboration of a series of IoT devices. The real added value is achieved when the connected IoT devices are able to communicate with each other and integrate with vendor-managed inventory systems, customer support systems, business intelligence applications, and business analytics. For example, smart things, such as sensors and actuators can be composed together to keep the air condition optimal in an house while optimizing the electricity usage. In this example the air conditioner communicates with air circulators, windows actuators, humidity sensors, and power grids.

To make the deployment of services relying on the collaboration of IoT devices possible, the first step is to make available discovery mechanisms to search the required devices distributed in the smart environment. This discovery process has to be driven by a set of *search requirements*. For instance, constraints referring to the type of service a given device has to provide (e.g., temperature sensing), or to some expected features of the devices (e.g., the sensor should be placed into a given location, its latency should be below a given threshold). A key point is therefore to have the assurance that all requirements have been correctly considered during the discovery process. Indeed, an untrusted discovery process might fail in evaluating the specified requirements in several ways. It might return devices that do not satisfy the requirements (e.g., not placed in the required location) as well omit those that indeed satisfy them, with the final result of selecting IoT devices not implementing the required service. As such, it is mandatory that the discovery process provides a proof of the fact that it has retrieved all and only those devices satisfying the specified search requirements.

At this purpose, we propose to exploit blockchain to ensure the correctness of the IoT discovery process execution. Blockchain technology can cope with the above mentioned requirements by leveraging on its tamper-proof design and on the key feature of enabling the computation of pre-defined programs, called "smart contracts" [2]. More precisely, we propose to have a IoT discovery process driven by the specified search requirements, implemented via a smart contract. Blockchain network validation of the smart contract execution provides assurance that all search requirements have been correctly evaluated. However, it brings also some issues regarding the privacy and confidentiality of the involved data. Indeed, sensitive data can be exposed during the evaluation process (e.g., information about the device latency). Therefore, in this paper, we design an approach that allows smart contract execution by preserving, at the same time, the confidentiality of the contained sensitive data. A further concern arises from the need of an *Oracle* to handle interactions of blockchain with external entities [3], which in turn requires to carefully

[1] https://iot-analytics.com/iot-investments-m-and-a-market-update-2018/

978-1-7281-2714-9/19/$31.00 ©2019 IEEE
DOI 10.1109/ICIOT.2019.00014

consider possible data confidentiality breaches. For this reason, in this paper, we investigate how to ensure data confidentiality during the discovery process even in the presence of an untrusted Oracle.

Literature presents several approaches (e.g.,[4], [5], [6]) aiming at achieving a distributed discovery service over IoT devices capabilities (e.g., multi-attribute, range queries). Although, these works propose scalable, self-configuring, and reliable architectures, they do not provide any assurance on requirements evaluation as well as they do not consider the confidentiality issue.

Moreover, the integration of IoT within the blockchain technology is currently under research on several fields (see surveys in [7], [8] for a comprehensive overview of the challenges and issues). An application scenario is the one of achieving secure and trustworthy access control mechanisms for IoT relying on smart contracts (see [9], [10]). Furthermore, blockchain platforms have been exploited to address the collaboration among IoT components, exploiting the smart contracts to overcome the lack of trust among them (e.g., [11], [12], [13]). The idea of a smart contract acting as mediator to control the collaborative process has been investigated in other research field, not closely related to IoT, such as in web services (e.g., [14], [15]). The confidentiality issues that arise with a smart contract as a mediator of service composition have been investigated also in [16], where encryption schemes have been proposed to achieve a privacy-preserving computation of the smart contract in the presence of an untrusted Oracle. However, to the best of our knowledge, we are the first proposing an approach leveraging on blockchain for the confidential discovery of IoT devices driven by specified search requirements.

The rest of the paper is organized as follows. Section II presents some background information on blockchain. Section III introduces the search requirements we have considered and the main design choices underlying our proposal. Section IV presents the proposed approach, whilst Section V presents algorithms behind it. Section VI shows preliminary experimental results. Finally, Section VII concludes the paper.

II. Blockchain Basics

Blockchain [17] is a distributed append-only public ledger technology, initially intended for cryptocurrency, e.g., Bitcoin. It is replicated and shared among members of a network, to keep track of every exchange of resources or assets between participants, called transactions. As shown in Figure 1, the blockchain structure is composed of a sequence of time-stamped blocks, linked together by the hash values. Each block includes several transactions and it is identified by its hash value (i.e., the value returned by a cryptographic hash function applied on the block content).

Transactions are inserted into blocks only if they are considered valid by the network participants through a distributed consensus protocol that, in general, is deemed to be secure if the majority of network participants are honest. At the time of writing, several distributed consensus protocols have been proposed, starting from the well-known Byzantine Fault Tolerance algorithm (BFT) [18], passing through Proof of work (PoW), Proof of Stake (PoS), up to newer ones, tailored for specific blockchain frameworks. They differentiate on the selection method exploited to insert a new block into the chain. As an example, in PoW, nodes have to solve a computationally intense puzzle, while in PoS, the block creator is selected based on the amount of stake held by network nodes.

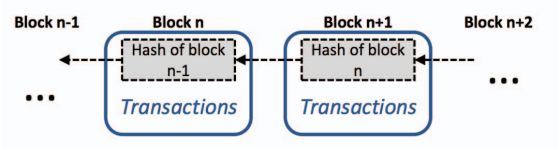

Figure 1. Blocks forming a blockchain

The computation involved in transaction validation can be encoded into pre-defined programs, called *smart contracts*. These can be seen as "a computerized transaction protocol that executes the terms of a contract". In this view, blockchain plays the role of a distributed ledger storing results (i.e., transactions) of smart contracts whose correct evaluation have been validated by network participants. Smart contract enabled blockchains are emerging as general purpose application platforms, among which the most popular one is Ethereum,[2] whereas Hyperledger[3] is the most significant standardization initiative. It is relevant to note that a smart contract cannot directly retrieve data outside the blockchain, as interactions with external entities are not directly possible. To overcome this limitation, blockchain platforms rely on the presence of a trusted mediator, called *Oracle*, that delivers data from an external source to smart contracts. This provides flexibility to the framework but, at the same time, might bring some security issues, as it will be discussed in Section III-B.

III. Requirement-driven IoT device discovery

An IoT device collaboration can be modeled as a network of IoT devices that have to interact in order to achieve a common goal. From the point of view of the discovery process, an IoT device collaboration can be represented as the set N of nodes of the underlying IoT network. The IoT discovery process plays a key role in the deployment of an IoT collaboration, as it has to find which is the most appropriate device to be assigned with each node. Before going into the details of our proposal, we will introduce an example used throughout the paper.

Example 1: Let us assume a smart city scenario where an application has to search for devices to create an IoT network for traffic control. The purpose of the application is to limit the vehicle's circulation in a particular area (say, $BakerStreet$), depending on the pollution rate and the current weather. At this purpose, the application is looking for devices collecting information about weather and pollution as well as

[2]https://www.ethereum.org/
[3]https://www.hyperledger.org/

controlling the traffic status (e.g., traffic control camera) near $BakerStreet$.

As mentioned in the introduction, the discovery process is usually driven by a set of *search requirements*. To give the requesting entity the assurance that all the search requirements have been correctly considered in the discovery process, we leverage on blockchain. This is achieved by having a smart contract, called $Assigner$, implementing the IoT discovery process driven by the specified search requirements. Here, we assume the $Assigner$ smart contract is able to retrieve relevant information about available IoT devices, aka *device profiles*, and evaluate on them the specified search requirements. The blockchain network validation of the $Assigner$ execution provides to the requiring application assurance that all search requirements have been correctly evaluated.

Before presenting the proposed blockchain based IoT discovery process, in this section, we illustrate in more details the type of search requirements we support, and the design principles we have considered in defining the process.

A. Search Requirements

Application search criteria. The first type of search requirements refers to those specified directly by the application requiring the IoT collaboration. As an example, an application requiring the composition illustrated in Example 1 might impose that the traffic status device adopts high-security standards and that its latency is less than 10ms. Hereafter, we refer to a search criteria applied on a single node of the IoT collaboration N as a *local application search criteria* (e.g., the weather device latency has to be lesser than 10ms) and to a search criteria that has to be satisfied by each device in N as *global application search criteria* (e.g., each device has to be located near Baker Street). In addition, we also assume that the requesting user can state, for a specific node, a *blacklist* of devices he/she does not want to interact with. For example, a user might prefer not having ACME company as a supplier of the weather device.

Provider constraints. We also take into consideration that the providers of the devices participating in the IoT collaboration might have additional requirements that need to be considered in the discovery process. A relevant type of constraint is the one modelling *conflict of interests* (COI) among organizations (e.g., due to partnerships, being competitors, etc.). Hereafter, we model a COI constraint specified by a device (aka, the providing organization) as a list of conflicting device providers. Additionally, organizations might impose constraints on all the devices participating the to IoT collaboration. We refer to this type of constraints as *Global provider constraints*. Referring to Example 1, for instance, the traffic device might require that each device in N adopts strong security mechanisms.

Both the above types of search criteria can be modeled as conditions to be evaluated on a device profile. For this purpose, we assume that each device exposes a *device profile*, modelled as a list of pairs $(Attribute, Value)$ encoding all the device properties and their corresponding values (e.g., Latency,10ms; Encryption, AES256). To formally represent these search requirements, we introduce introduce the following formal definitions:

Definition 1: **Search requirement.** Let Att be the set of attribute names used in the device profiles. A search requirement Req on Att is defined as a boolean expression on a set of predicates $\mathcal{R} = r_1, \ldots, r_n$, where, for each $r_i \in \mathcal{R}$, $r_i = a \oplus Value$, having $a \in Att$, $\oplus$ a conditional operation (i.e., = , $>$, $<$, $\geq$, $\leq$), and $Value$ a threshold value in the domain of a.

Definition 2: **Application search criteria.** Let DC be a device collaboration required by an application. Let N be the set of nodes involved in DC. Application search criteria specified by u on DC are defined as $C_a = \{Local, Global, Blacklist\}$ where:

- $Local = \{(n_1, Req_1), \ldots, (n_n, Req_n)\}$, where Req_j is a search requirement defined according to Definition 1 applied only to node $n_j \in N$, $j = \{1, \ldots, |N|\}$.
- $Global = \{Req_1, \ldots, Req_n\}$, where Req_j is a search requirement defined according to Definition 1 applied to any node $n_j \in N$, $j = \{1, \ldots, |N|\}$.
- $BlackList = \{(n_1, [pid_1, \ldots, pid_n]), \ldots, (n_n, [pid_1, \ldots, pid_n])\}$, where $j = \{1, \ldots, |N|\}$, and pid_k, $k \in \{1, \ldots, m\}$ is the identifier of a device provider which should not be assigned to node n_j.

Example 2: Let us consider again Example 1, and suppose that the application has the following criteria: the location of the weather device has to be Baker Street; the encryption used by each device has to be AES 256bit; ACME devices must not be involved in the weather sensing activity. These requirements are modeled through the following criteria: $Local = \{weather, location = BakerStreet\}$, $Global = \{Encryption = AES256\}$, $Blacklist = \{weather, [ACME]\}$.

Definition 3: **Provider constraints.** Let DC be a device collaboration required by an application. Let N be the set of nodes involved in DC. Let d be a device involved in the collaboration. Provider constraints specified by d on DC are defined as $C_d = \{Global, COI\}$:

- $Global = \{Req_1, \ldots, Req_n\}$, where Req_j is a requirement defined according to Definition 1 applied to any node $n_j \in N$, $j = \{1, \ldots, |N|\}$.
- $COI = \{pid_1, \ldots, pid_n\}$ defines the list of providers having a COI with d, where pid_j is a provider identifier.

Example 3: Consider a scenario where a device poses constraints requiring that all devices in the composition have to adopt a specific encryption method (e.g., AES 256bit) and it does not want to collaborate with ACME. These constraints are modeled, respectively as : $Global = \{Encryption = AES256\}$ and $COI = \{ACME\}$.

B. Design principles

In the design of the proposed requirement-driven IoT discovery process on Blockchain we have taken into account the two following main principles.

Best selection. The purpose of IoT discovery is to assign to every node in the IoT network the proper device, that is, a device that satisfies all the requirements posed by the requesting application as well as by the providers of the devices associated with the nodes of the composition. However, we have to take into account that, given a node n, the discovery process might find several devices satisfying the associated requirements. In order to select just one device, we have to rank them according to some criteria, thus to select the best one. Among possible criteria (e.g., reputation, QoS, etc.), in this paper, we rank selected devices based on how much their profiles are close in satisfying search criteria. To compute this rank, we exploit metrics to compare the distance between the attribute value of the device profile and the threshold value contained in the constraint. As an example, given the search criteria "latency<30ms" and two devices, say D1 and D2, having two latency values say 20ms and 11ms, that both satisfy it, $Assigner$ selects D2, since its value is closer to the threshold value.

Confidentiality. Thanks to distributed consensus, the evaluation of a requirement-driven discovery process via a smart contract ensures the correctness of requirements evaluation. However, it poses new challenges. Indeed, by design, a smart contract is defined as public and, according the adopted distributed consensus protocol, it has to be evaluated by blockchain network participants, called validators, to validate its output. This might expose sensitive data contained in the $Assigner$ smart contract. As an example, each validator has to be able to access the device profile as well as the search criteria and provider constraints, that may convey very sensitive data. For instance, information about COI among device providers may be confidential for business purposes. For this reason, in designing the $Assigner$, we have to adopt a mechanism to *ensure a confidential execution of the requirement-driven discovery process*, that is, we have to execute the smart contract by preserving, at the same time, the confidentiality of the contained sensitive data.

A further concern arises from the need of an *Oracle* to handle interactions of blockchain with external entities [3]. Although Oracles allow the blockchain to be more flexible, they introduce the presence of a mediator between the smart contract and external services, which require to carefully consider possible data confidentiality breaches. As an example, since the $Assigner$ has to inquiry the Oracle to retrieve device profiles, a malicious Oracle is able to read all contained information, which could be highly sensitive (e.g., device latency). As such, a further relevant requirement is to ensure the *Oracle data confidentiality*, by preventing the Oracle to access the sensitive information contained into the device profile.

In the following section, we show how the above principles have been considered in the design of the proposed $Assigner$.

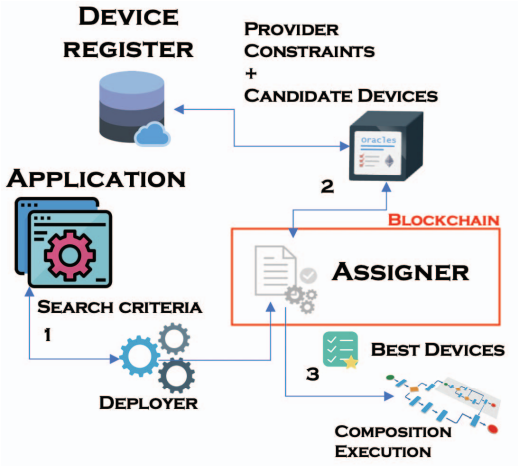

Figure 2. The general picture

IV. Confidential requirement-driven IoT discovery process

In this section, we present the reference architecture in support of the proposed IoT device discovery process (see Figure 2). The process starts when an application submits the criteria for the discovery of IoT devices. Since we assume an application might require to setup an IoT network, we model the input as a list N_{app} of N search criteria, one for each node required in the IoT network. This list is submitted to the $Deployer$, which is an off-chain component in charge of the creation of the $Assigner$ smart contract, tailored for the required discovery process. In particular, given N, the process to be encoded in the smart contract has to perform the following tasks.

(I) For each node $n_j \in N$, it has to retrieve the set of available IoT devices, their profiles, and their provider constraints, if any. This is performed by exploiting the *Device Register*, an entity that keeps a record of the available devices and their profiles.[4] In particular, the $Assigner$ inquiries the *Device Register* via an Oracle (see Figure 2). The retrieved devices are referred as *Candidate Devices List* for node n_j, CDL_{n_j}.

(II) For each node n_j, the $Assigner$ has to prune from the corresponding CDL_{n_j} those devices that do not satisfy the application search criteria or provider constraints. This implies that, given a candidate device d in CDL_{n_j}, the $Assigner$ has to evaluate whether d is compatible with devices assigned to previous nodes, that is, if d's profile satisfies all the constrains posed by providers of devices assigned to previous nodes (i.e., $n_1, \ldots n_{j-1}$). If d's profile does not satisfy even one of these provider constraints, the $Assigner$ has to remove d from CDL_{n_j}. The devices obtained by this evaluation are referred as *Selected Devices List* for node n_j, SDL_{n_j}, which

[4]The $Assigner$ has to retrieve all those devices that perform a given type of service. This holds under the assumption that, for each node, the application has specified the required service type as a mandatory search criteria.

could also be empty. In this respect, there can be two cases. The first one is when SDL_{n_j} is empty because none of the devices in CDL_{n_j} satisfy the application search criteria. In this case, the $Assigner$ has to terminate the discovery process, since for node n_j is not possible to find a proper device. The second case is when the SDL_{n_j} is empty because none of the devices in CDL_{n_j} is compatible with previous assigned devices. Here, rather than terminating the discovery process, the $Assigner$ has to reconsider the previous assignments. In particular, it starts by selecting an alternative device for n_{j-1} and verifies if, with this new assignment, the SDL_{n_j} is not empty (e.g., at least a node in CDL_{n_j} is compatible with the new device assigned to n_{j-1}). If this is not the case, the $Assigner$ proceeds to re-assign node n_{j-2} and so on till SDL_{n_j} is not empty, or it reaches the first node n_1. Even if after the re-assigment of all the previous nodes, the list is still empty, the $Assigner$ has to terminate the discovery process.

(III) In case SDL_{n_j} is not empty, as a final step, the $Assigner$ has to select, among the devices in the list, the best one to be assigned to n_j. The idea is to rank devices in SDL_{n_j} based on how much their attribute values are close in satisfying search criteria and provider constraints (cfr. Section III-B). To compute this rank, we exploit metrics to compare the distance between the attribute value of the device profile and the threshold value contained in the search criteria/provider constraint.[5] In case search criteria/provider constraints contain different conditions posed on different profile attributes, we assume them ordered according to a priority specified by the application on profile attribute. The rank is thus computed by ranking services according to highest-priority conditions.

These three tasks represent the main operations that have to be executed by the $Assigner$ smart contract. We note that the best selection principle is taken into account by task (III). In contrast, the confidentiality principle requires to encode the process into the $Assigner$ smart contract such that sensitive data are not accessible neither to validators executing the smart contract, nor to the Oracle mediator. The proposed solution implies to encrypt sensitive data, contained and used in the $Assigner$ smart contracts, in such a way that they can be consumed (aka decrypted) only by authorized actors by, at the same time, allowing simple computations over it needed for the execution of the discovery process (e.g., criteria/constraint evaluation on device profiles).

V. $Assigner$ ALGORITHM

In this section, we present the algorithm implemented by the $Assigner$ smart contract. Before this, we briefly introduce the adopted encryption schemes.

A. Adopted encryption schemes

We recall that in our scenario we have to protect (aka encrypt) the data contained into the smart contract (e.g., attributes storing the device latency values), by, at the same time, making possible to execute some computation on it (e.g., evaluating whether the latency is less than a given threshold). At this purpose, we cannot exploit standard encryption schemes, rather the idea is to exploit alternative encryption schemes supporting simple operations over encrypted data, such as homomorphic encryption. These encryption schemes are defined such that, given two ciphertexts `E(a)` and `E(b)`, the computation of an operator $\oplus$ on them, i.e., `E(a)`$\oplus$`E(b)`, generates an encrypted value `E(c)`, whose decryption corresponds to the result of $\oplus$ applied on the plaintexts `a` and `b`, i.e., `E(a)`$\oplus$ `E(b)=E(c)`, where `a`$\oplus$`b=c`. The scheme can be fully (FHE) homomorphic, supporting arbitrary computation on ciphertexts, or partially (PHE) homomorphic, supporting only a subset of mathematical functions on encrypted data. Among the available homomorphic schemes, we have selected the Order Preserving Encryption (OPE) scheme [20]. This scheme allows us to perform simple condition operations ($<, >, =$) on encrypted values, making thus possible the evaluation of search requirements (see Definition 1) directly over encrypted devices attributes' values and encrypted thresholds.

In order to adopt this scheme, we assume that a pair of OPE public/private keys is generated for every instance of $Assigner$ execution, that is, for every new request of an IoT discovery process. These keys are generated directly by the *Deployer*, which has to use them to encrypt the sensitive data contained in the $Assigner$ (e.g., threshold values specified in the search requirements). Also the device profile attributes have to be encrypted with these keys. As such, we assume the *Device Register* is aware and complaint with the proposed solution, as it has to encrypt the device profile's attributes with the OPE public/private key associated with the discovery process.

B. Algorithms

The underlying logic implemented by the $Assigner$ smart contract is represented in Algorithm 1. This takes as input, from the $Deployer$, the search criteria specified by the application for each node $n \in N$, encoded into a list called N_{app}, and returns the list of best devices to be assigned to each node (BRL_N in the algorithm). It is relevant to note that the $Deployer$ builds N_{app} such that the threshold's values and the criteria specified in search criteria are encrypted with OPE key associated with the $Assigner$.

As described in Section IV, at runtime the $Assigner$ performs three main tasks. As a first task, it has to inquiry the *Device register* through an Oracle ($Query_{DeviceRegister}$ in the algorithm) in order to retrieve, for each $n \in N$, the list of devices able to carry out n, called *Candidate Device List* (CDL_n in the algorithm) (lines 8-35). Once this information is retrieved, for each node n in N, $Assigner$ has to evaluate each device in CDL_n to select only those that satisfy the application search criteria (i.e., local, global and blacklist) specified in N_{app} for n, say C_a (lines 16-25). As a first step, it checks whether the considered device D, aka its providers, is not included into the blacklist specified in C_a (line 16). If this is the case, it evaluates the local and global constraints.

[5] An example of these distance metrics can be found in [19], where it is analyzed how much an access request is close to satisfy an access control policy.

It is relevant to recall that to guarantee *privacy-preserving computation*, and the *Oracle data confidentiality* requirements while enabling $Assigner$ to perform the requirement evaluation (lines 8-23), the device's profiles/providers constraints in CDL are encrypted with OPE scheme. This means that, for the evaluation of the criteria $\mathtt{C}_1 \oplus \mathtt{V}_1$, where $\mathtt{C}_1$ is a criteria and $\mathtt{V}_1$ is a value, the OPE scheme is $\mathtt{C}_{1\mathtt{OPE}} \oplus \mathtt{V}_{1\mathtt{OPE}}$. As an example, for the condition latency<10, the criteria latency and the value 10 are encrypted with OPE. Exploiting OPE, the $Assigner$ first evaluates the local criteria in C_a (line 17). The evaluation is done by $Evaluation_{constraint}$ (see Procedure 2), which takes as input the device D and the constraint $Cons$ that D has to satisfy. This procedure considers the disjunctive normal form (DNF) of $Cons$ and evaluates separately each clause. For each clause, it checks if all included predicates are satisfied. This is done by matching the threshold value specified in the predicate with the corresponding attribute value in the device profile.[6]

Algorithm 1 $Assigner$

1: **Input**: N_{app}, the list of search criteria.
2: **Output**: BRL_N, the list of best selected devices for each $N_{app}[i]$.
3: Let $Nodelist$ be the list of nodes in N.
4: Let SDL_N be a matrix initialized empty
5: Let CDL_n, BRL_N be initialized empty
6:
7: $matrixIndex = 0$
8: **for all** $n \in NodeList$ **do**
9: Let C_a be the search criteria associated with $N_{app}[i]$
10: Let req_n be the criteria associated with node n in $C_a.local$
11: CDL_n = $Query_{DeviceRegister}(n)$
12: Let SDL_n be a vector initialized empty
13: $ListIndex = 0$
14: **for all** $D \in CDL_n$ **do**
15: Let pid be the identifier of D's provider
16: **if** $(pid \notin C_a.blacklist)$ **then**
17: **if** $Evaluation_{constraint}(D, req_n)$ **then**
18: satisfy = true
19: **for all** $Req \in C_a.global$ **do**
20: **if** satisfy **then**
21: **if** $Evaluation_{constraint}(D, Req)$ == false **then**
22: satisfy = false
23: **end if**
24: **end if**
25: **end for**
26: **if** satisfy **then**
27: $SDL_n[ListIndex] = D$
28: $ListIndex++$
29: **end if**
30: **end if**
31: **end if**
32: **end for**
33: $SDL_N[matrixIndex] = SDL_n$
34: $matrixIndex++$
35: **end for**
36:
37:
38: **for** $Index = 0$ to $|Nodelist|$ **do**
39: $Sort(SDL_N[Index])$
40: $BRL_N[Index] = SDL_N[Index].[0]$
41: **end for**
42:
43: $Index_{D_i} = 0$
44: $Index_{D_j} = 0$
45: Solution = $Provider_{constraints}$ $(BRL_N, SDL_N, Index_{D_i}, Index_{D_j})$
46: **if** Solution == 1 **then**
47: **return** "no solution"
48: **else**
49: **return** BRL_N
50: **end if**

[6]For sake of clarity, the procedure shows only the pseudocode of a generic operator $\oplus$.

Procedure 2 $Evaluation_{constraint}$

1: **Input**: D, a device; $Cons$, a search requirement
2: **Output**: $True$ if D satisfies the $Cons$; $False$, otherwise.
3: Let C be the set of clauses contained in the DNF form of $Cons$
4: $satisfied = false$
5: **for all** $c \in C$ **do**
6: **if** satisfied == false **then**
7: Let P be the set of predicates in c
8: state == 0
9: **for all** $p \in P$ **do**
10: **if** state=0 **then**
11: Let Att be the attribute over which p is defined
12: Let $\oplus$ be the conditional operation specified in p
13: Let val be the value associated with Att in profile of D
14: **if** $(Att \oplus val)$ **then**
15: $satisfied$ = true
16: **else**
17: $satisfied$ = false
18: state = 1
19: **end if**
20: **end if**
21: **end for**
22: **end if**
23: **end for**
24: **return** $satisfied$

Procedure 3 $Provider_{constraints}$

1: **Input** : BRL_N, the list of ranked devices, SDL_N, $Index_{D_i}$, $Index_{D_j}$
2: **Output** : 1 if there is no solution; 0 otherwise.
3:
4: $satisfied$ = 1
5: **for** i=0 to $|BRL_N|$ **do**
6: Let $cons$ be the set of global constraints associated with the device in $BRL_N[i]$
7: **for** j=0 to $|BRL_N|$ **do**
8: **if** $BRL_N[j] \neq BRL_N[i]$ **then**
9: **for all** $c \in cons$ **do**
10: Let COI be the set of COI constraints associated with the device in $BRL_N[j]$
11: **if** $Evaluation_{constraint}$ $(D_j, cons) == false$ **or** $BRL_N[i] \in COI$ **then**
12: $satisfied = Rollback_{Process}$ $(BRL_N, SDL_N, Index_{D_i}, Index_{D_j}, i, j)$
13: **end if**
14: **end for**
15: **end if**
16: $j++$
17: **end for**
18: $i++$
19: **end for**
20: **return** $satisfied$

A similar process is done to evaluate the global constraints contained in C_a (lines 19-25). If the considered device D satisfies both the global and local constraints, it is inserted into the *Selected Device List* (SDL_n in the algorithm). Once all devices for each node n have been considered, the selected devices are stored into an array called SDL_N, where the j-th element contains devices selected for node n_j.

As a second task (lines 37-41), $Assigner$ ranks devices contained into each element of SDL_N array and selects the first ranked, generating the so-called *Best Ranked List* (BRL_N in the algorithm). This sorting is performed following the best selection strategy explained in Section III, where each comparison operation among devices profile attributes/thresholds are done exploiting the OPE encryption scheme.

As a third task (lines 43-50), the $Assigner$ has to evaluate if each device in the BRL_N also satisfies provider constraints posed by other devices inserted into BRL_N. In case a constraint is not satisfied the BRL_N has to be updated accordingly.

This evaluation is done by $Provider_{constraints}$ (see Proce-

Procedure 4 $Rollback_{Process}$

Input : BRL_N, SDL_N, $Index_{D_i}$, $Index_{D_j}$, i, j
Output : 1 if there is no solution; 0 otherwise.

Let New_{D_j} be initialized empty
$Index_{D_j}++$
$New_{D_j} = SDL_N[j][Index_{D_j}]$
if $New_{D_j} = \emptyset$ **then**
  Let New_{D_i} be initialized empty
  $Index_{D_i}++$
  $New_{D_i} = SDL_N[i][Index_{D_i}]$
  if $New_{D_i} = \emptyset$ **then**
    return 1
  else
    if $New_{D_i} \notin D_j.COI$ **and** $D_j \notin New_{D_i}.COI$ **then**
      $BRL_N[i] = New_{D_i}$
      $Index_{D_i} = 0$
      $Index_{D_j} = 0$
      $Provider_{constraints}$ $(BRL_N, SDL_N, Index_{D_i}, Index_{D_j})$
    else
      $Rollback_{Process}$ $(BRL_N, SDL_N, Index_{D_i}, Index_{D_j}, i, j)$
    end if
  end if
else
  if $New_{D_j} \notin D_i.COI$ **and** $D_i \notin New_{D_j}.COI$ **then**
    $BRL_N[j] = New_{D_j}$
    $Index_{D_i} = 0$
    $Index_{D_j} = 0$
    $Provider_{constraints}$ $(BRL_N, SDL_N, Index_{D_i}, Index_{D_j})$
  else
    $Rollback_{Process}$ $(BRL_N, SDL_N, Index_{D_i}, Index_{D_j}, i, j)$
  end if
end if
return 0

dure 3), which takes as input the list of best devices BRL_N, the array of selected devices SDL_N and the indexes of the devices D in both the lists. The procedure evaluates, for each device D_i in BRL_N, if all the other devices in BRL_N satisfy D_i global constraints and do not belong to D_i's COI list. If one of these conditions is not met, $Provider_{constraints}$ has to select the device that is ranked just after D in the sorted list (SDL_n), replace D with it, and evaluate from the beginning all the constraints for all the nodes/devices. This is done by the $Rollback_{process}$ procedure (see Procedure 4), which takes as input the list of best devices BRL_N, the array of selected devices SDL_N, the indexes of the device D in both the lists and the indexes of devices already evaluated by $Rollback_{process}$. As a first step, the procedure selects the device that is ranked just after D in the SDL_n. If any (aka the list is not empty) the procedure evaluates if the new device (New_{D_j}) does not belong to the COI list of D and vice-versa. If the new device meets the conditions, the BRL_N list is updated with the new device and the constraints evaluation goes back to the beginning. Otherwise, the next device in the list is selected for evaluation. However, the procedure has to take into account the scenario where, for a given node n, there is no device in SDL_n that satisfies the application and provider-defined constraints. In such a case, $Rollback_{process}$ has to be iteratively repeated.

VI. Experimental Results

In this section, we provide results of a set of experiments we carried out to test the feasibility of our proposal. At this purpose, we have implemented $Assigner$ smart contract in the Ethereum blockchain coding the experiments in the Solidity[7] programming language by exploiting REMIX IDE and executed on the ropsten testnet.[8]

The aim of these experiments is to provide an estimation of the cost of our solution, that is, the cost of the $Assigner$ smart contract.[9] In order to estimate the $Assigner$ cost, we run several experiments varying two main dimensions to show how these impact the final cost. These dimensions arise by the consideration that the complexity of $Assigner$ smart contract computation depends on the number of search requirements evaluation it has to perform. In particular, given an IoT device collaboration modeled as a list of nodes N, the number of evaluations to be run for each node $n \in N$ depends on: the number of search requirements **NSR** associated with n and the number of available devices **CDL**, returned by the *Device Register*, able to execute n.

To test these dimensions, we have generated a set of device collaborations, modeled as a list of nodes, $\mathcal{N}$, by varying the number of available devices for each node, which represents the dimension of CDL (i.e., $CDL \in [0, 20]$). Then, we have created a smart contract modelling the algorithms for the assigner process and exploited the set $\mathcal{N}$ as input for the evaluation, with a varying number of search requirements (i.e., **NSR**$\in [1, 8]$).

Varying the number of search requirements. As a first experiment, we run $Assigner$ by varying the number of search requirements associated with each node in the IoT collaboration. Figure 3 shows the cost overhead, expressed in ethers (i.e., y-axis) by varying the number of each type of search requirement (i.e., x-axis), by also highlighting the type of search requirements. In the case of 7 constraints, the cost in USD is 0,84$.

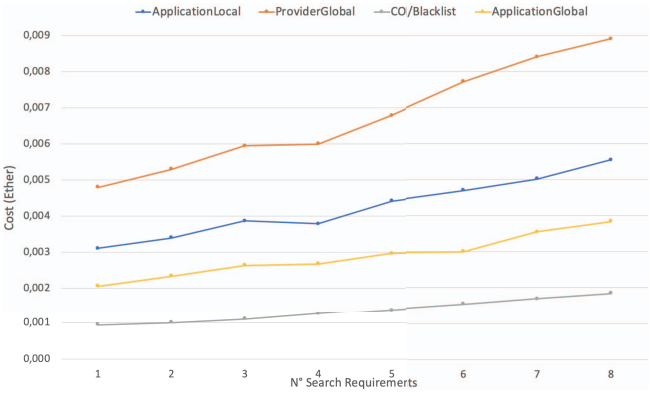

Figure 3. Cost overhead by varying the number of search requirements.

[7]https://solidity.readthedocs.io/en/v0.5.6/

[8]https://ropsten.etherscan.io/,https://remix.ethereum.org

[9]As cost, we have considered both the deploy and execution cost, expressed as gas usage in $ether(ETH)$, the currency adopted in Ethereum. At the time of writing (March 2019) 1 ETH = 163,43 USD, with a gas cost of 10 Gwei meaning a confirmation time of about $\sim$ 25 seconds.

Varying the number of candidate devices. As a second experiment, we have varied the actual dimension of the CDL list returned to the $Assigner$ by the *Device Register*. Figure 4 shows the cost overhead, expressed in ethers (i.e., y-axis) by varying the number of parameters in CDL (i.e., x-axis). In the case of 15 devices, the cost in USD is 0,91$.

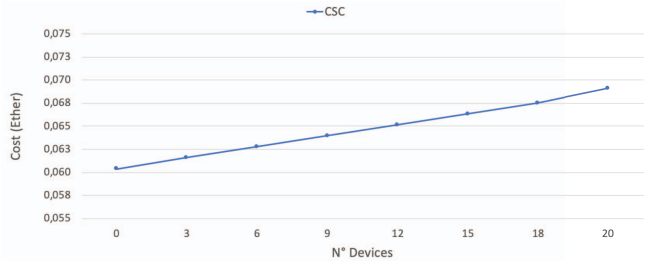

Figure 4. Cost overhead by varying the dimension of CDL.

As a summary, we can conclude that the cost of the deployment of a IoT discovery process on blockchain, under differ settings of varying complexity, varies in the range [0,001-0,1] ethers, corresponding to [0,13-13,33] USD.[10]

These experiments show how the estimated cost of the $Assigner$ smart contract linearly change w.r.t the complexity of the scenario. The additional cost introduced can be mitigated by taking into account other Blockchain platforms, such as the permissioned one.

VII. Conclusion

In this paper, we have shown how blockchain can be used to ensure that all search requirements underlying an IoT discovery process are correctly evaluated. The key idea is to have a smart contract to implement the discovery process, where the blockchain network validation of the smart contract execution ensures the correctness of the IoT discovery process. Since sensitive data are exposed during the evaluation process (i.e., devices' profile and search requirements), in this paper, we have proposed homomorphic encryption schemes supporting the smart contract execution by preserving, at the same time, the confidentiality of the contained sensitive data. The paper shows some preliminary experimental results, highlighting the implied costs and the feasibility of the proposed approach. We plan to extend this work according to several dimensions. First, we plan to run a more extensive set of experiments, with standard IoT benchmarks. Also, we are interested in extending the proposed solution so as to deal with additional security issues related to interactions with Oracles (e.g., authenticity and integrity).

Acknowledgments

This work has received funding, in parts, from "RAIS: Real-time Analytics for the Internet of Sports", supported by the European Union's Horizon 2020 research and innovation programme under Marie Skodowska-Curie grant agreement No 813162, and from CONCORDIA, the Cybersecurity Competence Network supported by the European Union's Horizon 2020 research and innovation programme under grant agreement No 830927.

[10]We note that the cost is heavily dependent by the exchange Ether to USD, which is 1 ETH to 132,94 USD.

References

[1] A. Bahga and V. Madisetti, *Internet of Things: A hands-on approach*. Vpt, 2014.

[2] K. Christidis and M. Devetsikiotis, "Blockchains and smart contracts for the internet of things," *Ieee Access*, vol. 4, pp. 2292–2303, 2016.

[3] X. Xu, C. Pautasso, L. Zhu, V. Gramoli, A. Ponomarev, A. B. Tran, and S. Chen, "The blockchain as a software connector," in *2016 13th Working IEEE/IFIP Conference on Software Architecture (WICSA)*. IEEE, 2016, pp. 182–191.

[4] F. Paganelli and D. Parlanti, "A dht-based discovery service for the internet of things," *Journal of Computer Networks and Communications*, vol. 2012, 2012.

[5] J. Botia *et al.*, "Collaborative agents framework for the internet of things," in *Workshop Proceedings of the 8th International Conference on Intelligent Environments*, vol. 13. IOS Press, 2012, p. 191.

[6] S. Cirani, L. Davoli, G. Ferrari, R. Léone, P. Medagliani, M. Picone, and L. Veltri, "A scalable and self-configuring architecture for service discovery in the internet of things," *IEEE Internet of Things Journal*, vol. 1, no. 5, pp. 508–521, 2014.

[7] M. A. Ferrag, M. Derdour, M. Mukherjee, A. Derhab, L. Maglaras, and H. Janicke, "Blockchain technologies for the internet of things: Research issues and challenges," *IEEE Internet of Things Journal*, 2018.

[8] A. Panarello, N. Tapas, G. Merlino, F. Longo, and A. Puliafito, "Blockchain and iot integration: A systematic survey," *Sensors*, vol. 18, no. 8, p. 2575, 2018.

[9] A. Ouaddah, A. Abou Elkalam, and A. Ait Ouahman, "Fairaccess: a new blockchain-based access control framework for the internet of things," *Security and Communication Networks*, vol. 9, no. 18, pp. 5943–5964, 2016.

[10] Y. Zhang, S. Kasahara, Y. Shen, X. Jiang, and J. Wan, "Smart contract-based access control for the internet of things," *IEEE Internet of Things Journal*, 2018.

[11] A. Bahga and V. K. Madisetti, "Blockchain platform for industrial internet of things," *Journal of Software Engineering and Applications*, vol. 9, no. 10, p. 533, 2016.

[12] N. Teslya and I. Ryabchikov, "Blockchain-based platform architecture for industrial iot," in *2017 21st Conference of Open Innovations Association (FRUCT)*. IEEE, 2017, pp. 321–329.

[13] A. Dorri, S. S. Kanhere, and R. Jurdak, "Towards an optimized blockchain for iot," in *Proceedings of the second international conference on Internet-of-Things design and implementation*. ACM, 2017, pp. 173–178.

[14] I. Weber, X. Xu, R. Riveret, G. Governatori, A. Ponomarev, and J. Mendling, "Untrusted business process monitoring and execution using blockchain," in *International Conference on Business Process Management*. Springer, 2016, pp. 329–347.

[15] L. García-Bañuelos, A. Ponomarev, M. Dumas, and I. Weber, "Optimized execution of business processes on blockchain," in *International Conference on Business Process Management*. Springer, 2017, pp. 130–146.

[16] B. Carminati, C. Rondanini, and E. Ferrari, "Confidential business process execution on blockchain," in *2018 IEEE International Conference on Web Services (ICWS)*, July 2018, pp. 58–65.

[17] M. Swan, *Blockchain: Blueprint for a new economy*. " O'Reilly Media, Inc.", 2015.

[18] M. Vukolić, "The quest for scalable blockchain fabric: Proof-of-work vs. bft replication," in *International Workshop on Open Problems in Network Security*. Springer, 2015, pp. 112–125.

[19] B. Carminati, E. Ferrari, and M. Guglielmi, "Detection of unspecified emergencies for controlled information sharing," *IEEE Transactions on Dependable and Secure Computing*, vol. 13, no. 6, pp. 630–643, 2016.

[20] A. Boldyreva, N. Chenette, and A. ONeill, "Order-preserving encryption revisited: Improved security analysis and alternative solutions," in *Annual Cryptology Conference*. Springer, 2011, pp. 578–595.