Trusted D2D-based IoT Resource Access using Smart Contracts

Vasilios A. Siris, Dimitrios Dimopoulos, Nikos Fotiou, Spyros Voulgaris, George C. Polyzos

Mobile Multimedia Laboratory, Department of Informatics
School of Information Sciences & Technology
Athens University of Economics and Business, Greece
{vsiris, dimopoulosd, fotiou, voulgaris, polyzos}@aueb.gr

***Abstract*—We present and evaluate models that allow clients to access IoT resources using secure and trusted device-to-device (D2D) communication, while utilizing smart contracts to obtain the benefits of blockcain technology. These benefits include immutability, high availability, and distributed trust. The models consider different network connection capabilities of the clients and the IoT resources, namely continuous network connectivity and D2D-only connectivity. We describe two approaches for utilizing blockchains and smart contracts in the authorization process: in the first approach, only hashes of the authorization information are recorded on the blockchain. In the second approach, a smart contract handles authorization requests. We implement the approaches using the OAuth 2.0 delegated authorization framework and evaluate the implementations on the public Ethereum testnet Rinkeby, in terms of execution cost, contract creation cost, and delay. Our evaluation quantifies the tradeoffs of blockchain cost and smart contract functionality, such as blocking and non-blocking operation, and the reduction of the transaction cost that can be achieved when multiple authorization requests are concatenated in a single transaction.**

***Index Terms*—delegated authorization, blockchains, distributed ledgers, constrained IoT devices**

I. Introduction

The Internet of Things will involve a huge number of devices, acting both as consumers of services (or clients) and providers of services (or resources), which can be sensors or actuators. Some of these devices will be constrained in terms of memory, processing, energy, and network connectivity. Due to their constrained resources, performing authentication and authorization by the IoT devices themselves will not be possible in the majority of cases. Additionally, due to the huge number of devices, there can be significant advantages in terms of scalability, total cost reduction, and improved security if the authentication and authorization functionality is offloaded from the IoT devices to nodes with sufficient processing and storage resources to handle authentication and authorization for a large number of devices.

Distributed ledger technologies such as blockchains provide an immutable ledger offering high availability and distributed trust. These features can provide advantages when applied to the IoT [1]. However, not all devices would be capable of interacting directly with the blockchain and even if they could, due to security and overall management, direct interaction with blockchains might not be desirable. The interaction with a blockchain, similar to the authentication and authorization functionality, can be offloaded to capable nodes.

The goal of this paper is to present and evaluate different architectures for providing delegated authorization using smart contracts that enable clients to use secure and trusted device-to-device (D2D) communication to access IoT resources. Clients and IoT resources can have different capabilities in terms of network connectivity, processing power, and memory. Secure refers to keeping the authorization information that needs to be exchanged confidential, whereas trusted refers to all entities performing the actions intended by their owners. D2D communication can be provided by technologies such as bluetooth, WiFi direct, and cellular D2D.

Our realization of the proposed models considers the widely used IETF standard OAuth 2.0 for delegated authorization, which is based on *access tokens*. OAuth 2.0 is being investigated for authorization in IoT environments by IETF's Authentication and Authorization for Constrained Environments (ACE) Working Group [2], [3]. The OAuth 2.0 framework can also be used in cases where centralized management of authorization policies is beneficial. An important feature of OAuth 2.0 is that it provides authorization for different levels of access, termed scopes. OAuth 2.0 mainly defines the format of the authorization message exchange, and the models presented in this paper can be applied to the general context of IoT resource access authorization.

In summary, the contributions of the paper are the following:

- We present and discuss models for providing secure and trusted D2D-based IoT resource access based on smart contracts that consider different network connection capabilities of the clients and the IoT resources.
- We implement the proposed models in smart contracts with blocking and non-blocking operation, and with support for the concatenation of authorization requests.
- We evaluate the implementations on the public Ethereum testnet Rinkeby, in terms of the execution cost, contract creation cost, and delay, and quantify the tradeoffs between blockchain cost and smart contract functionality.

Our previous work [4], [5], [6], [7] focused on the case where the IoT device does not have continuous network connectivity, whereas the client both has continuous network connectivity and interacts directly with the blockchain. In the current paper

we consider scenarios where the client doesn't have continuous network connectivity, and assume that the client does not directly interact with the blockchain.

The remainder of the paper is structured as follows: In Section II we discuss some motivating use cases and in Section III we present some background on delegated authorization using OAuth 2.0 and the advantages from utilizing blockchains and smart contracts for authorization in IoT environments. In Section IV we present different architectures for D2D-based IoT resource access that utilize blockchains, considering different network connection capabilities of the client and the IoT resource. In Section V we present the message exchange between the entities involved, and in Section VI we present the implementation and the evaluation of the proposed models. Finally, in Section VII we present related work and in Section VIII we present conclusions and our ongoing work.

II. Use cases

Next we discuss some motivating use cases which involve clients and IoT resources with different network connection capabilities, and different processing and storage resources.

The first use case that we consider involves electronic door locks (IoT resources) in the rooms of a hotel or an apartment that is rented on a short-term basis. The electronic door lock can be opened using a digital key (access token), which a client sends directly to the lock through D2D communication, e.g., bluetooth. A person seeks to reserve a hotel room or rent the apartment for some number of nights. The person sends a requests from his/her smartphone (client) to the authorization server (AS) that handles authorization requests for the door lock. The same AS can handle authorization requests for many door locks. The AS sets up a blockchain contract to receive the payment for the requested number of nights. The client can make the payment through some proxy, or client authorization server (CAS). Once the AS verifies that the price for resource access has been paid, the AS provides the client with the necessary credentials to unlock the door for the duration of the requested stay. In this use case the electronic door lock is a constrained IoT resource: it has only bluetooth connectivity for receiving the access token from the client. The client can be a smartphone, which is a device with continuous network connectivity that can also communicate with the door lock using D2D communication to provide the access token to unlock the door.

The above use case can be extended if the person staying in the hotel is accompanied by a second guest who also wants to be able to open the room's door. However, instead of using a smartphone, the accompanying guest wants to use his/her smart watch. To achieve this, the first person's smartphone can send a copy of the access token, and possibly other authorization credentials, to the accompanying guest's smart watch. In the extended use case, both the client (smart watch) and the door lock are constrained devices without continuous network connectivity; despite not having network connectivity, the smart watch and door lock can communicate in a trusted manner using D2D communication to transfer the access token that unlocks the door. Furthermore, we assume that neither the smart watch nor the smartphone, despite being capable, interacts directly with the blockchain. Instead, the interaction with the blockchain is performed through a proxy, the CAS, which operates on the behalf of the clients.

The second use case we consider involves purchasing items from a smart vending machine. We assume that the smart vending machine has continuous network connectivity. However, it does not interact directly with the blockchain. A person can purchase items from the vending machine through his/her smart wristband. In this case, the client (wristband) is a constrained device while the IoT resource (vending machine) is not constrained in terms of network connectivity. As in the first use case, we again assume that neither the client nor the IoT resource interact directly with the blockchain. The client with constrained network connectivity must interact with a proxy, the CAS, operating on its behalf, to perform the payment (on the blockchain) for the item purchased and obtain the necessary access credentials that will allow the client to obtain the purchased item. It is interesting to note that in this use case the client (smart wristband) has only D2D communication capability, but can communicate with its proxy (CAS) through the connected IoT resource. Of course, all the above communication and transactions must be performed in a secure and trusted manner.

III. Background

In this section we provide some background information on delegated authorization using OAuth 2.0 and the benefits from utilizing blockchains and smart contracts for authorization in IoT environments.

A. Delegated authorization using OAuth 2.0

OAuth 2.0 is a framework for delegated authorization to access a protected resource [8]. It enables a third party application (client) to obtain access with specific permissions to the protected resource, with the consent of the resource owner. Access to the resource is achieved through access tokens, which are created by an authorization server (AS). The specific format of the access tokens, which are discussed below, is opaque to the clients and to OAuth 2.0. The consent for authorization by the resource owner is provided after the owner is authenticated. Alternatively, authorization requests can be accepted automatically by the AS, based on policies defined a priori by the resource owner [3]. Authorization can allow different access levels, called *scopes*, and for a specific time interval. The OAuth 2.0 authorization flows can involve intermediate messages exchanged before the access token is provided by the AS. However, the details of the authorization flow does not impact the general approach of the proposed models, hence in our discussion we only consider the initial client request and the AS's response with the access token.

One type of access tokens are *bearer tokens*. Bearer tokens allow the holder (bearer), independently of its identity, to access the protected resource. Bearer tokens are the default for OAuth 2.0, which assumes secure communication between

the different entities based on TLS (Transport Layer Security). OAuth 2.0 also assumes that the protected resource has continuous network connectivity, hence can communicate with the AS to check the validity and scope of the access tokens provided by the clients. Meeting the above two requirements is not always possible in constrained environments [2].

JSON Web Token (JWT) is an open standard that defines a compact format for transmitting claims as JSON objects [9]. JWTs can use the JSON Web Signature (JWS) structure to digitally sign or integrity protect claims with a Message Authentication Code (MAC) [10]. Hence, unlike simple bearer tokens, JWT/JWS tokens are self-contained, i.e., they include all the necessary information for the protected resource to verify their integrity without communicating with the AS. The JWT format is also considered by the W3C Credentials Community Group for *Verifiable Credentials* which can be combined with *Decentralized Identifiers* or *DIDs* [11]. A more efficient encoding of claims, which is derived from JWTs but is based on CBOR (Concise Binary Object Representation), is the CBOR Web Token (CWT) [12], [3]. CWTs reduce the amount of data that needs to be sent to constrained IoT devices and can be extended to create and process signatures, MACs, and encrypted data [13]. The implementation of the models presented in this paper adopts the CWT format.

In constrained environments, in addition to limited connectivity, the communication between the client and the protected resource is not secure, hence transmitting bearer tokens or self-contained JWTs/CWTs over such insecure links make them a target for eavesdropping. To avoid this Proof-of-Possession (PoP) tokens are used in constrained environments [14]. PoP tokens include a normal access token, such as a JWT/CWT, and a PoP key [15]: access to the protected resource requires both the access token and the PoP key, which can be used to secure the link between the client and the protected resource.

If the client is a constrained device, some of the authorization functionality must be performed on behalf of the client by a proxy, the client authorization server (CAS) [16]. In the models we present in the next section, we will consider that the CAS, in addition to implementing the authorization functions, also interacts with the blockchain on behalf of the client.

B. Delegated authorization utilizing blockchains and smart contracts

The advantages from combining authorization based on frameworks such as OAuth 2.0 with blockchain and smart contracts are the following:

- Blockchains can immutably record hashes of the information exchanged during authorization and cryptographically link authorization grants to payments and other IoT events recorded on the blockchain. These records serve as indisputable receipts in the case of disagreements.
- Smart contracts can encode authorization policies in an immutable and transparent manner. Policies can depend on payments or other IoT events.
- Smart contracts run on all nodes of a blockchain, hence sending resource access requests to smart contracts can protect against DoS attacks that involve a very high resource request rate.

In the models presented below we assume that the CAS, the AS, and the resource owner, have an account (public/private key pair) on the blockchain. The CAS will use its account to pay for the client's access to the IoT resource. Initially, the CAS will deposit the amount for resource access to a Hashed Time-Lock Contract (HTLC) [17], as we discuss below. The deposit is transferred to the resource owner's account when the AS reveals a secret that corresponds to the hash-lock. The same secret is used by the CAS to obtain the necessary authorization credentials that the client needs in order to access the resource. If the AS does not provide the secret within some time interval, the CAS can submit a refund request to return the deposit to its account.

IV. D2D-BASED IOT RESOURCE ACCESS USING SMART CONTRACTS

In this section we present models for access authorization using blockchains that consider different network connection capabilities of the client and the IoT resource:

- The IoT resource does not have continuous network connectivity, but only D2D connectivity, whereas the client requesting resource access has continuous network connectivity. This combination corresponds to the electronic door lock use case discussed in the previous section, where a smartphone is the client.
- Both the client and the IoT resource do not have continuous network connectivity, i.e., they both having only D2D connectivity. This combination corresponds to the electronic door lock use case, where now the client device is a smart watch (or wristband).
- The client has only D2D connectivity, whereas the IoT resource has continuous network connectivity. This combination corresponds to the smart vending machine use case discussed in the previous section, where the IoT resource (smart vending machine) has continuous network connectivity and the client is a smart wristband (or watch) with only D2D connectivity.

In addition to whether a device (client or IoT resource) has continuous network connectivity or only D2D connectivity, a second dimension is whether the device is constrained in processing and memory. A device without processing and memory constraints can perform asymmetric key cryptographic functions, while a device that is constrained can perform only symmetric key cryptographic functions. Hence, processing and memory constraints influence the type of access tokens that can be used, and in particular the type of integrity verification that will be incorporated in the JWT/CWT token: if the device is capable, then signatures using public/private keys can be used. On the other hand, if the device is constrained, then MAC integrity verification must be used.

In all three scenarios presented below there is a client authorization server (CAS) and an authorization server (AS) that handle requests and responses on behalf of the client and

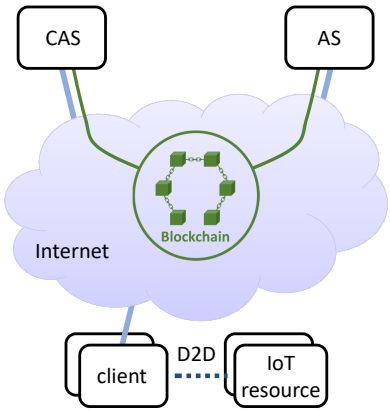


Fig. 1. The client has network connectivity while the IoT resource has only D2D connectivity. The client acts as an intermediate node that forwards messages between the IoT resource and the AS, which handles authorization requests on behalf of the IoT resource. The client AS (CAS) interacts with the blockchain and the AS on behalf of the client.

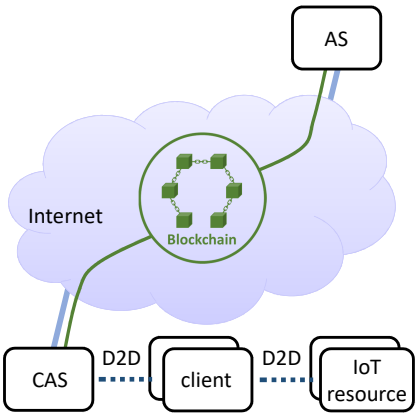


Fig. 2. Both the client and IoT resource have only D2D connectivity. Prior to requesting access, the client must obtain the authorization credentials from the CAS. Once it has the credentials, the client can request access to the resource using D2D communication, without requiring synchronous network connectivity or simultaneous D2D connectivity with the CAS.

IoT resource, respectively [16]. The CAS and AS also handle interactions with the blockchain, in order to link authorization grants to blockchain payments. To take actions on behalf of the client and resource, the CAS and AS must have the consent of the client owner and the resource owner; one way to provide such consent is through verifiable credentials [11], which is more general than considering that the client and resource owners control the CAS and AS, as assumed in [16].

During their initialization, both the client and the IoT resource establish with the CAS and AS, respectively, shared keys to be able to securely communicate over insecure D2D links and/or through intermediate nodes. If a device, either the client or the IoT resource, is constrained in terms of processing, then during its initialization it must establish a common secret key with the CAS or the AS. This shared secret key is used to add MAC integrity verification to the messages exchanged between the client and the CAS, and between the AS and the IoT device. If the device has sufficient processing to perform asymmetric key cryptographic functions, then the CAS and/or the AS can use public key cryptography to sign messages they send to the client and/or the IoT resource, respectively. Note that if the IoT resource has continuous network connectivity, then instead of using signed or MAC integrity protected access tokens, simple access tokens can be used; in this case, the IoT resource can use introspection to verify the validity and scope of the access token [8], [3].

A. Connected client and disconnected IoT resource

In the first model we discuss, the client has continuous network connectivity whereas the IoT device does not have continuous network connectivity, but only D2D connectivity. This is the case investigated in our previous work [4], [5], [6], [7]. The difference with the model considered in the current paper is that the client, despite having network connectivity, does not interact directly with the blockchain, Figure 1.

Because the client does not interact directly with the blockchain, the CAS performs blockchain transactions on behalf of the client. The client can send authorization requests to the CAS, which handles the exchange of authorization messages with the AS and interacts with the blockchain. The goal of this interaction is to link authorization grants with blockchain payments. Specifically, the CAS obtains the necessary access token and PoP key from the AS only if it performs the blockchain payment, on behalf of the client. When the CAS receives the authorization credentials, it forwards them to the client. The client can then use the credentials to request service from the IoT resource. As we will see more detail when we discuss the message exchange, when the client requests access to the IoT resource, the client acts as an intermediate node that forwards messages between the IoT resource and the AS, which handles authorization requests on behalf of the IoT resource. Specifically, the AS accepts authorization requests from the CAS and provides authorization credentials once it verifies that the appropriate blockchain payment has been performed. As shown in Figure 1, a single CAS and AS can handle multiple clients and IoT resources, respectively.

B. Disconnected client and disconnected IoT resource

Next we discuss the case where both the client and the IoT resource are constrained devices. As in the previous scenario, the authorization requests for the IoT resource are handled by the AS and the authorization requests on behalf of the client are handled by the CAS, see Figure 2. Moreover, both the CAS and the AS directly interact with the blockchain. The client prior to communicating using D2D with the IoT resource, must obtain the necessary authorization credentials (access tokens and PoP keys) from the CAS. This may be achieved at any point prior to the time the client requests resource access, when the client has intermittent connectivity to the CAS using D2D communication. Once it has obtained the authorization credentials, the client can request access to the IoT resource through its D2D communication link, without requiring synchronous network connectivity or simultaneous D2D connectivity with the CAS. The communication between

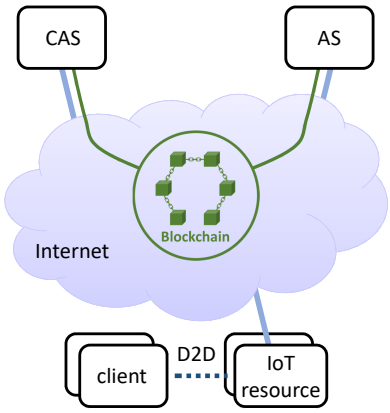


Fig. 3. The client has only D2D connectivity while the IoT resource has continuous network connectivity. The IoT resource acts as an intermediate node that forwards messages between the client and the CAS, which handles authorization requests on its behalf.

the CAS and the AS to request resource access on behalf of the client and to obtain the authorization credential after the corresponding blockchain payment is the same as the message exchange in the previous scenario.

C. Disconnected client and connected IoT resource

In the third model the client is disconnected while the IoT resource has continuous network connectivity. As in the model of the previous subsection, the CAS submits authorization requests to the AS and interacts with the blockchain on behalf of the client. The client communicates with the CAS using the connected IoT device as the intermediate node. The AS is responsible for handling authorization requests on behalf of the IoT resource, see Figure 3.

The CAS and AS interact in the same way as in the first two models. Once the CAS obtains the authorization credentials, which include the access token and the PoP key, it must transfer these to the client before the client requests service from the IoT resource; this transfer is performed through the connected IoT resource.

Because the IoT resource has continuous network connectivity, it can use introspection to verify the validity and scope of the access token [8], [3]. Hence, unlike the first two models, the access token does not need to contain a signature or a MAC for verifying its authenticity.

V. Message Exchange

In this section we present the message exchange between the various entities, namely the client, IoT resource, CAS, AS, and blockchain.

A. CAS-AS message exchange

We present two approaches for the message exchange between the CAS, which operates on behalf of the client, and the AS, which operates on behalf of the IoT resource. In the first approach, the authorization requests and responses are communicated directly between the CAS and AS. In this approach the blockchain is used only to record hashes of the authorization information exchanged between the CAS and the AS and to link blockchain payments to authorization grants. The motivation for recording hashes of the authorization information exchanged between the CAS and AS is that they serve as indisputable receipts in the case of disputes.

In the second approach, authorization requests and responses go through a smart contract, which is owned by the resource owner. Because smart contracts are executed by all blockchain nodes, a blockchain provides a secure execution environments with high availability. This offers higher protection against DoS attacks, compared to the first approach where access requests are sent directly to the AS. Moreover, in the second approach a smart contract can be used to transparently record prices and other authorization policies defined by the resource owner. Examples of such policies include permitting resource access to specific CASes/clients, determined by their public keys on the blockchain, and dependence of access authorization on IoT events that are recorded on the blockchain. An additional advantage achieved by allowing a smart contract to handle authorization requests is that the smart contract can securely bind the IoT resource with the AS responsible for providing authorization grants for that resource.

The two approaches are similar to the message exchange in our previous work [6], [7], where however the authorization message exchange occurred between the client and the AS; the client was assumed to have continuous network connectivity and could interact directly with the blockchain. An additional contribution of this paper is that, in Section VI, we evaluate different smart contract implementations that can be realized for the scenarios presented in Section IV: a non-blocking implementation that allows multiple authorization exchanges to be conducted in parallel and an implementation that allows the CAS and the AS to request and respond to multiple concatenated authorization requests that are included in a single message and transaction.

1) Linking authorization grants to blockchain payments and recording hashes of authorization information: With this approach the initial communication between the CAS and the AS occurs as in the normal OAuth 2.0 framework, Figure 4. However, instead of the AS providing the CAS with authorization credentials after consent is given by the resource owner, the authorization credentials are disclosed only after the payment for resource access is recorded on the blockchain.

Specifically, in Step 1 the CAS sends on the behalf of the client to the AS a request for accessing the IoT resource. The AS generates a random PoP key which it sends to the CAS[1] together with the PoP key encrypted with the secret key $K_{AS-resource}$ shared by the AS and the IoT resource, which is set during the IoT resource's initialization[2]; the client will later use the PoP key to establish a secure D2D link with the IoT resource. Also, the AS sends to the CAS the access token encrypted with a secret s, i.e., $E_s(token)$, the hash

[1]The communication link between the CAS and the AS is secured, hence the PoP key cannot be leaked through eavesdropping.

[2]If the resource has sufficient processing, then the AS can use asymmetric cryptography and encrypt the PoP key with the resource's public key.

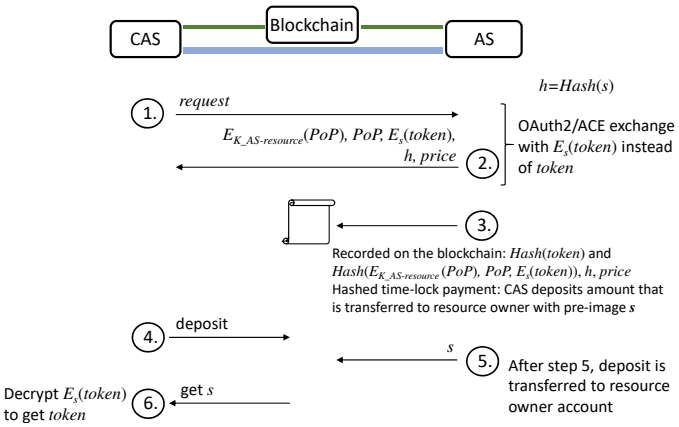


Fig. 4. CAS-AS message exchange when authorization requests are sent directly to the AS. Hashes of the authorization information are recorded on the blockchain, which provide indisputable receipts in case of disagreements. Disclosure of authorization credentials is linked to blockchain payments.

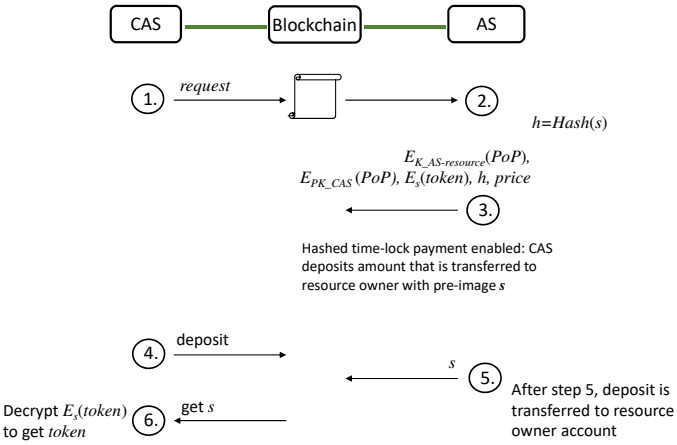


Fig. 5. CAS-AS message exchange when a smart contract handles authorization requests. Authorization information is exchanged through the blockchain. As in the approach of Figure 4, disclosure of authorization credentials is linked to blockchain payments.

$h = Hash(s)$ of the secret s, and the price for the requested resource access scope. The secret s is a secret randomly generated by the AS and is required for the CAS to decrypt $E_s(token)$ and obtain the access token; the AS will reveal the secret s once it confirms that the payment for resource access has been committed on the blockchain. Communicating the price from the AS to the CAS allow different levels of resource access to be offered for different prices.

In Step 3, two hashes are submitted to the blockchain: the first is the hash of the token that the AS will reveal to the CAS once payment has been confirmed. The second is the hash of three items: $E_{K_{AS-resource}}(PoP)$, the PoP key, and $E_s(token)$; the second hash serves as proof of the authorization information that is exchanged using OAuth between the AS and the CAS. Note that the above authorization exchange does not ensure that the access token the client obtains from the AS indeed allows access to the IoT resource.

Also, in Step 3 a hashed time-lock payment is initiated on the blockchain, which allows the CAS to deposit the requested price (Step 4). This amount will be transferred to the resource owner's account if the secret s (hash-lock) is submitted to the contract by the AS (Step 5) within some time interval. If the time interval is exceeded, then the CAS can request a refund of the amount it deposited. Once the secret s is revealed, the CAS can get s from the blockchain (Step 6) and decrypt $E_s(token)$, thus obtaining the access token. After Step 6, the CAS has all the credentials that are necessary for the client to request access from the IoT resource.

2) Smart contract for handling authorization requests: Unlike in the previous approach where the CAS and the AS communicated directly, in the approach discussed next the interaction is through the smart contract, corresponding to Steps 1 and 2 in Figure 5.

In response to the authorization request it received from the CAS, in Step 3 of Figure 5 the AS sends to the smart contract the PoP key encrypted both with the secret key shared by the AS and the IoT resource, $E_{K_{AS-resource}}(PoP)$, and with the CAS's public key, $E_{PK_{CAS}}(PoP)$. On the other hand, in the previous approach the PoP key was sent from the AS to the CAS over a secure communication link, hence encrypting the PoP key was not necessary.

As in the first approach, a hash time-locked payment is enabled, allowing the CAS to deposit the amount corresponding to the resource access price (Step 4). The amount is transferred to the resource owner's account if the secret s that unlocks the hash-lock is revealed (Step 5). Once revealed, the CAS can obtain the secret s (Step 6), together with the other necessary authorization credentials that will allow the client to access the protected resource. If the blockchain is public, then s can be read by anyone, hence everyone can obtain the access token. However, the access token cannot be used alone, since the PoP key is also required for accessing the resource. Nevertheless, if privacy of the access token is important, then the secret s can be encrypted using the CAS's public key PK_{CAS} and the hash-lock set to $h = Hash(E_{PK_{CAS}}(s))$.

B. Client-CAS and client-IoT resource message exchange

The message exchange between the client, the CAS, and the IoT resource when the IoT resource does not have continuous network connectivity is shown in Figure 6. Note that this message exchanges applies to both the case where the client has continuous network connectivity and the case where the client has only D2D connectivity. Initially the client communicates with the CAS by sending a message with its intent to access the IoT resource (Step 1). After receiving the request from the client, the CAS performs either of the two message exchanges presented in Section V-A. Next, in Step 2 the client receives the authorization credentials from the CAS and in Step 3 it sends its access request to the IoT resource.

The message exchange when the client has only D2D connectivity and the IoT resource has continuous network connectivity is shown in Figure 7. Now, the client communicates with the CAS that handles authorization requests on its behalf using the connected IoT resource as an intermediate node.

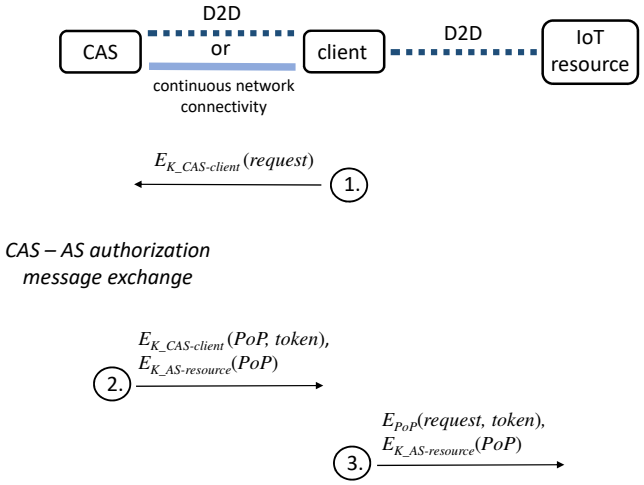


Fig. 6. Message exchange between CAS, client, and resource when the IoT resource has only D2D connectivity (Figures 1 and 2). The CAS-AS message exchange can follow the sequence in Figure 4 or 5.

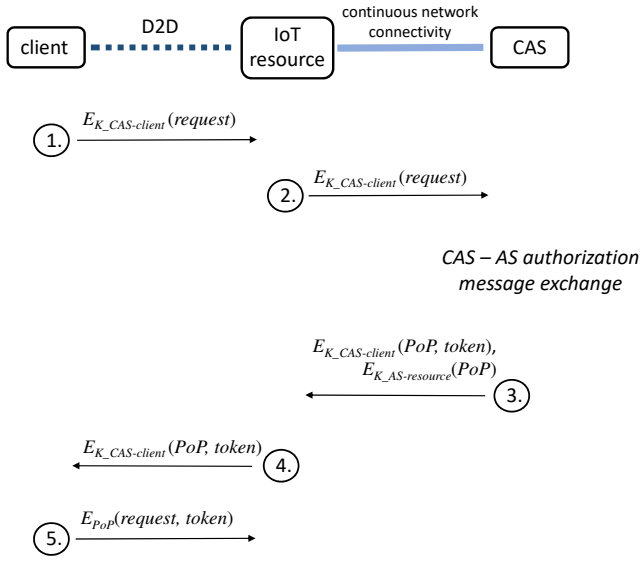


Fig. 7. Message exchange between client, resource, and CAS message when the client has only D2D connectivity while the IoT resource has continuous network connectivity (Figure 3). The CAS-AS message exchange can follow the sequence in Figure 4 or 5.

Note that the communication of the client and the CAS is secured, since they share a secret key $K_{CAS-client}$ that was configured during the client's initialization.

VI. Evaluation

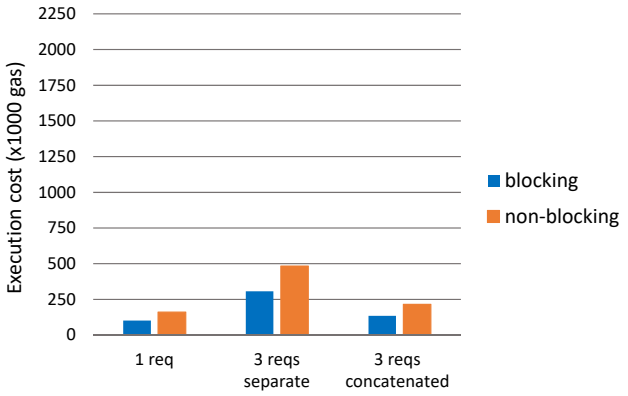
For the evaluation we used a local Ethereum node running Go-Ethereum[3] that was connected to the public Ethereum testnet Rinkeby[4]. Smart contracts were written in Solidity with the Remix[5] web-based editor. The AS was based on a PHP implementation of the OAuth 2.0 framework[6]. The CAS and AS used Web3.js to interact with the Rinkeby blockchain.

[3]https://geth.ethereum.org/
[4]https://www.rinkeby.io/
[5]https://remix.ethereum.org/
[6]https://github.com/bshaffer/oauth2-server-php

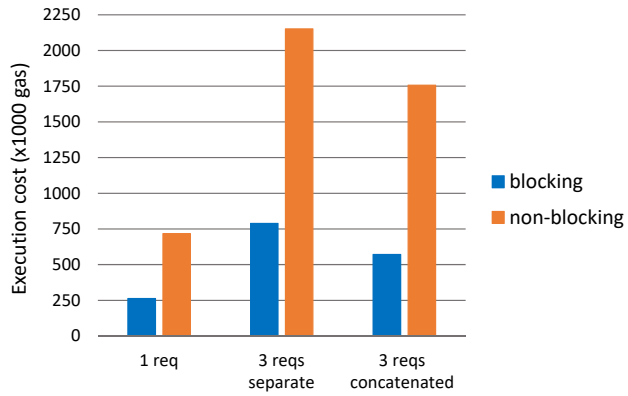
We compare the two approaches presented in Section V-A: the first records hashes of the authorization information on the blockchain (Figure 4) and the second involves a smart contract handling authorization requests (Figure 5). For each of the two approaches we compare four implementations: The first is the baseline implementation where the smart contract operates in a blocking mode where only one authorization request can be handled at a time ("1 req" in Figure 8(a)). The second implementation also operates in a blocking mode, but each message includes three authorization requests ("3 reqs concatenated" in Figure 8(a)), which are sent by the same CAS; similar to the requests, we assume that the responses are also concatenated, which means that the authorizations are handled by the same AS. The third implementation operates in a non-blocking mode, allowing more than one authorization requests, each in a separate message, to be ongoing at the same time ("1 req" in Figure 8(b)). Finally, the fourth implementation operates in a non-blocking mode, as the previous - third - implementation, but each message includes three authorization requests ("3 reqs concatenated" in Figure 8(b)). The "3 reqs separate" columns in Figures 8(a) and 8(b) correspond to the case where three authorization requests and their responses are sent and received separately with blocking and non-blocking operation, respectively.

Figure 8 shows the execution cost (gas), which quantifies the amount of EVM (Ethereum Virtual Machine) resources (computation and storage), for each of the above implementations. Comparison of the corresponding columns in Figures 8(a) and 8(b) shows that, for a blocking implementation, a smart contract that handles authorization requests requires approximately 2.5 times more gas than the approach that records only hashes of the authorization information on the blockchain. For the non-blocking implementation, the ratio is larger and close to 4 times. Figure 8(a) shows that the gas is 88% higher for the non-blocking implementation compared to blocking when only hashes are recorded on the blockchain, while Figure 8(b) shows that it is approximately 190% higher in the case of a smart contract handling authorization requests. The above results quantify the higher execution cost for smart contracts with more functionality.

Comparison of columns "3 reqs separate" and "3 reqs concatenated" in Figure 8(a) show that, for the blocking implementations, the gas when three requests and their responses are concatenated is smaller than the gas when the requests are sent separately by 56% when only hashes are recorded, and smaller by 28% when a smart contract handles requests. The gains for non-blocking, Figure 8(a), are 55% and 18% when only hashes are recorded and when a smart contract handles authorization requests, respectively. These results show that concatenation of requests can provide gains in terms of reduced execution cost; indeed, the gains are significantly higher for simple contracts that record only hashes. Additional experiments not shown due to space indicate that, as expected, the gains are higher when more requests are concatenated. Specifically, for non-blocking, when 9 requests are concatenated the gains are 67% (higher than the 55% gain when 3 request are concatenated)

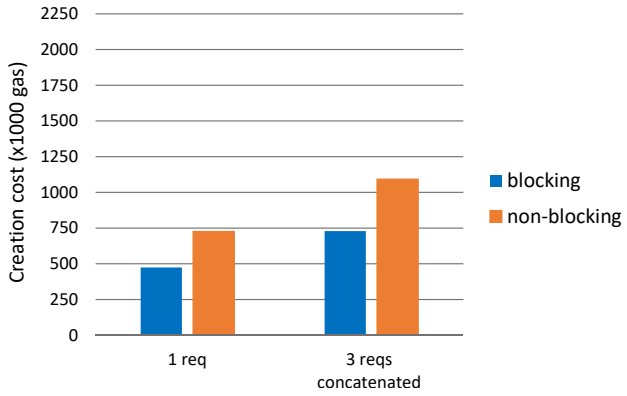


(a) Recording only hashes on the blockchain

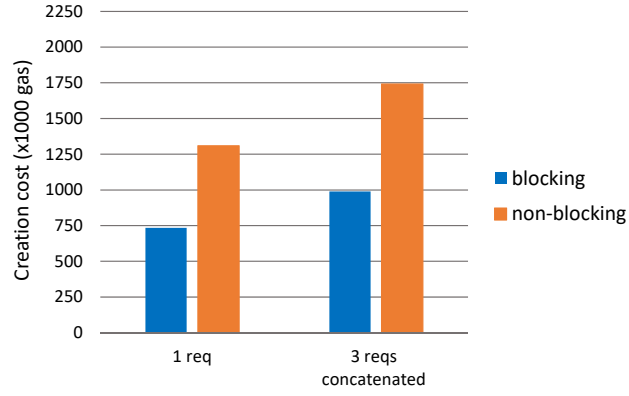


(b) Smart contract handling authorization requests

Fig. 8. Contract execution cost. The top graph corresponds to the approach in Figure 4 where only hashes are recorded on the blockchain. The bottom graph corresponds to the approach in Figure 5 where a smart contract handles authorization requests.



(a) Recording only hashes on the blockchain



(b) Smart contract handling authorization requests

Fig. 9. Contract creation cost. The top graph corresponds to Figure 4 where only hashes are recorded on the blockchain. The bottom graph corresponds to Figure 5 where a smart contract handles requests. The contract creation cost for "3 reqs separate" in Figure 8 is the same as the cost for "1 req".

when only hashes are recorded and 25% (higher than the 18% gain when 3 request are concatenated) when a smart contract handles requests.

Concatenation of authorization requests can be performed in the space domain, when CASes and ASes handle multiple clients and IoT resources. Alternatively, concatenation can be performed in the time domain by aggregating requests received by a CAS in a time interval, before sending them to the AS. Such time domain aggregation of requests adds a delay to the authorization process, which needs to be considered together with the blockchain transaction time.

The contract creation cost is shown in Figure 9. Note that this figure does not contain the contact creation cost for "3 reqs separate", since it uses the same contract as "1 req". The figure shows that the increase of the contract creation cost for the second approach, where authorization requests are handled by the smart contract, compared with the simpler scheme, where only hashes of authorization information are recorded on the blockchain, is smaller for the non-blocking compared to the blocking implementation: Comparison of the corresponding columns in Figures 9(a) and 9(b) shows that the contract creation cost for smart contracts handles authorization requests is 36-80% higher than the creation cost for contracts that record only hashes. An additional conclusion from the comparison of Figure 9 and Figure 8 is that for simple contracts that record only hashes and are blocking, the contract creation cost dominates the execution cost, while for more complex smart contracts such as the ones handling authorization requests and are non-blocking, the execution cost becomes comparable to the creation cost.

Finally, 20 executions of each of the non-blocking implementations have shown that the average transaction delay when only hashes of authorization information are recorded on the blockchain is 44 seconds, with a 95% confidence interval $\pm$ 5 seconds; the delay for the blocking implementation with three separate requests is higher by approximately 29 seconds, due to the serialization that blocking imposes. For a smart contract handling authorization requests the delay is 58 seconds, with a 95% confidence interval $\pm$ 6 seconds. The above results show that the delay is approximately 32% higher for the smart contract approach compared to the approach that records only hashes. This result is expected, since the recording only hashes involves three transactions on the blockchain, Figure 4, whereas a smart contract handles authorization requests involves four transactions, Figure 5.

VII. Related work

The work in [18] presents a blockchain-based authorization system where authorization proofs can be efficiently verified. The work in [19] presents a decentralized access control system where IoT devices have continuous network connectivity and interact directly with the blockchain, while [20] presents a system where policies and access control decisions are directly recorded on Bitcoin's blockchain. The work in [21] presents a smart contract-based system for providing access control to IoT devices while satisfying access policies in terms of the minimum time interval between consecutive accesses. The above works all assume that the IoT device can directly access the blockchain, which is not possible in constrained IoT environments. Unlike the above, our work considers different network connection capabilities of the clients and the IoT resources, which includes the case where either the client, or the IoT resource, or both, have only D2D connectivity.

The work in [22] presents a system based on OAuth 2.0 where a smart contract generates authorization tokens, which a key server obtains in order to provide private keys that allow clients to access a protected resource. The work of [23] contains a high level description for using smart contracts with OAuth 2.0 to allow users to freely select the server that provides authorization to their protected resource. The difference of this paper, in addition to considering different network connection capabilities of the clients and the IoT resources, is that we present two different approaches for utilizing blockchains which have different tradeoffs in terms of the contract execution and creation cost, and the smart contract functionality.

Our previous work [4], [5], [6], [7] focuses on the case where clients have continuous network connectivity and interact directly with the blockchain, while IoT resources have only D2D connectivity. Moreover, [5] investigates the reduction of the transaction cost by utilizing private channels which enable off-chain transactions, and [7] investigates the reduction of the transaction cost by moving smart contract functionality from a public blockchain to a private or permissioned blockchain.

VIII. Conclusions and future work

We have presented different architectures involving devices with different network connection capabilities that utilize smart contracts to provide secure and trusted D2D-based IoT resource access. Blockchains are used to link authorization grants to payments and record authorization information. Our evaluation quantifies the tradeoffs between blockchain cost and smart contract functionality, such as blocking and non-blocking operation, and the reduction of the transaction cost when multiple authorization requests are concatenated in a single transaction.

Ongoing work is investigating two directions: first, how smart contract functionality related not only to payments but also to IoT transactions can be moved to private or permissioned ledgers to reduce transactions costs and, second, the trusted interaction of smart contracts with the real world using decentralized oracles. This is important since as our results show, the execution cost of smart contracts increases significantly with higher functionality.

Acknowledgements

This research has been undertaken in the context of project SOFIE (Secure Open Federation for Internet Everywhere), which has received funding from EU's Horizon 2020 programme, under grant agreement No. 779984.

References

[1] K. Christidis and M. Devetsikiotis, "Blockchains and smart contracts for the Internet of Things," *IEEE Access*, vol. 4, pp. 1–1, 01 2016.

[2] L. Seitz *et al.*, "Use Cases for Authentication and Authorization in Constrained Environments," RFC 7744, IETF, January 2016.

[3] ——, "Authentication and Authorization for Constrained Environments (ACE) using the OAuth 2.0 Framework (ACE-OAuth)," IETF Draft, February 14, 2019.

[4] N. Fotiou, V. A. Siris, and G. C. Polyzos, "Interacting with the Internet of Things using Smart Contracts and Blockchain Technologies," in *Proc. of 7th Int'l Symp. on Security & Privacy on Internet of Things, in conjunction with SpaCCS*, 2018.

[5] N. Fotiou, V. A. Siris, G. C. Polyzos, and D. Lagutin, "Bridging the cyber and physical worlds using blockchains and smart contracts," in *Proc. of Workshop on Decentralized IoT Systems and Security (DISS), in conjunction with NDSS (to appear)*, 2019.

[6] V. A. Siris, D. Dimopoulos, N. Fotiou, S. Voulgaris, and G. C. Polyzos, "OAuth 2.0 meets Blockchain for Authorization in Constrained IoT Environments," in *Proc. of IEEE 5th World Forum on Internet of Things (to appear)*, 2019.

[7] ——, "Interledger Smart Contracts for Decentralized Authorization to Constrained Things," in *Proc. of 2nd Workshop on Cryptocurrencies and Blockchains for Distributed Systems (CryBlock 2019), in conjunction with IEEE INFOCOM (to appear)*, 2019.

[8] D. Hardt *et al.*, "The OAuth 2.0 Authorization Framework," RFC 6749, Standards Track, IETF, October 2012.

[9] M. Jones, J. Bradley, and N. Sakimura, "JSON Web Token (JWT)," RFC 7519, Standards Track, IETF, May 2015.

[10] ——, "JSON Web Signature (JWS)," RFC 7515, Standards Track, IETF, May 2015.

[11] M. Sporny *et al.*, "Verifiable Credentials Data Model 1.0: Expressing verifiable information on the Web," Draft Community Group Report, W3C, January 16, 2018.

[12] M. Jones, E. Wahlstroem, S. Erdtman, and H. Tschofenig, "CBOR Web Token (CWT)," RFC 8392, Standards Track, IETF, May 2018.

[13] J. Schaad, "CBOR Object Signing and Encryption (COSE)," RFC 8152, Standards Track, IETF, July 2017.

[14] L. Seitz *et al.*, "Authentication and Authorization for Constrained Environments (ACE) using the OAuth 2.0 Framework (ACE-OAuth)," IETF Draft, January 31, 2018.

[15] M. Jones *et al.*, "Proof-of-Possession Key Semantics for CBOR Web Tokens (CWTs)," IETF Draft, February 21, 2018.

[16] S. Gerdes *et al.*, "An architecture for authorization in constrained environments," IETF Draft, October 22, 2018.

[17] Bitcoin Wiki, "Hashed Timelock Contracts (HTLC)," https://en.bitcoinwiki.org/wiki/Hashed_Timelock_Contracts, last accessed 24/02/2019.

[18] M. P. Andersen *et al.*, "WAVE: A Decentralized Authorization System for IoT via Blockchain Smart Contracts," University of California at Berkeley, Tech. Rep., December 2017.

[19] R. Xu *et al.*, "BlendCAC: A BLockchain-ENabled Decentralized Capability-based Access Control for IoTs," arXiv:1804.09267v1, April 2018.

[20] D. D. F. Maesa, P. Mori, and L. Ricci, "Blockchain based access control," in *Proc. of 17th IFIP Distributed Applications and Interoperable Systems (DAIS)*, 2017.

[21] Y. Zhang *et al.*, "Smart Contract-Based Access Control for the Internet of Things," arXiv:1802.04410, February 2018.

[22] O. Alphand *et al.*, "IoTChain: A blockchain security architecture for the Internet of Things," in *Proc. of IEEE WCNC*, 2018.

[23] T. Hardjono, "Decentralized Service Architecture for OAuth2.0," March 25, 2018. [Online]. Available: IETF draft. https://tools.ietf.org/html/draft-hardjono-oauth-decentralized-02