Decentralized Authorization in Constrained IoT Environments exploiting Interledger Mechanisms*

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 models that utilize smart contracts and interledger mechanisms to provide decentralized authorization for constrained IoT devices. The models involve different trade-offs in terms of cost, delay, complexity, and privacy, while exploiting key advantages of smart contracts and multiple blockchains that communicate with interledger mechanisms. These include immutably recording hashes of authorization information and policies in smart contracts, resilience through the execution of smart contract code on all blockchain nodes, and cryptographically linking transactions and IoT events recorded on different blockchains using hash-lock and time-lock mechanisms. In the case of two ledgers, an authorization and a payment ledger, the authorization ledger can be a private Ethereum network or a permissioned ledger such as Hyperledger Fabric. For decentralized authorization where a subset of m-out-of-n authorization servers are required, we present two policies for selecting the m servers. The first policy can utilize statistics of the authorization servers such as transaction cost and response time. The second policy selects the first m servers that respond. The proposed models are evaluated on the public Ethereum testnets Rinkeby and Ropsten, and for different implementations on the Hyperledger Fabric permissioned ledger, in terms of execution cost (gas), delay, and reduction of data that needs to be sent to the constrained IoT devices.

Keywords: decentralized authorization, interledger, hashed time-lock contracts, constrained IoT environments

1 Introduction

Blockchains have different properties, e.g., Ethereum is a public and permissionless blockchain with Turing-complete smart contracts and an intrinsic cryptocurrency. Being public, Ethereum,

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

as Bitcoin, provides large-scale decentralized trust at the expense of high computation costs, hence high transaction fees, and high transaction delays. On the other hand, permissioned blockchains or distributed ledgers such as Hyperledger Fabric and R3 Corda operate with a restricted set of peers and incur a smaller cost and delay, while supporting different levels of write and read access. Combining multiple blockchains to implement different functionality, such as authorization and payment, allows different cost, delay, complexity, and privacy tradeoffs. However, multiple blockchains or distributed ledgers must be interconnected in a way that securely binds the transactions on different ledgers.

Authorization for constrained IoT devices (Things), with limited connectivity and computation power, requires support from an *authorization server* (AS) [1]; relevant use cases range from authorization for home door locks and health monitoring devices to smart container tracking and industrial actuator control [2]. Offloading authorization functionality from IoT devices to an AS also facilitates the collective management of authorization policies. However, providing authorization through a single AS can have a low reliability, since the single AS would represent a single point of failure in the case of server faults and misbehavior. The goal of this paper is to propose and investigate models for providing decentralized authorization with multiple ASes that utilize two blockchains, one for authorization and the other for payments, in order to reduce the transaction cost and delay compared to a single (public) blockchain.

The proposed models are general and can exploit blockchain and smart contract technology in the context of authorization in constrained IoT environments. Our realization of the models considers the OAuth 2.0 delegated authorization framework, which is based on *access tokens* and is a widely used IETF standard, currently being investigated for authorization in IoT environments by IETF's Authentication and Authorization for Constrained Environments (ACE) Working Group [3]. We also consider the CBOR (Concise Binary Object Representation) Web Token (CWT) format [3, 4], a recently proposed standard for compactly encoding *claims* in access tokens which is more efficient than the JSON Web Token (JWT) format.

The contributions of the paper are the following:

- We investigate interledger mechanisms for securely linking transactions on two blockchains, a private or permissioned chain and a public chain, to reduce the execution cost and delay compared to a single (public) chain.

- We propose an approach for decentralized authorization in constrained IoT environments that utilizes multiple authorization servers (ASes) and includes two mechanisms for reducing the amount of data that needs to be sent to the constrained IoT devices (Things).

- For decentralized authorization we present and investigate two policies for selecting the m-out-of-n servers: The first policy utilizes statistics of the authorization servers such as transaction cost and response time. The second policy selects the first m servers that respond to an authorization request.

- We evaluate the proposed models on two public Ethereum testnets, Rinkeby and Ropsten, in terms of execution cost (gas), delay, and reduction of the amount of data that needs to be sent to IoT devices. In the case of two blockchains, we evaluate Hyperledger Fabric (a permissioned ledger) interconnected with Rinkeby.

Previous work on decentralized authorization based on smart contracts assumes that IoT devices communicate with the blockchain or are capable of executing public/private key cryptographic functions. To the best of our knowledge, this is the first work to investigate the application of smart contracts and multiple blockchains with interledger mechanisms for decentralized authorization to constrained IoT devices. The current paper extends our previous conference paper [5] by describing and evaluating two policies for selecting a subset m-out-of-n servers in the case of decentralized authorization and presenting experimental results for different implementations in the Hyperledger Fabric permissioned ledger.

The remainder of the paper is structured as follows: In Section 2 we present related work, identifying where it differs from the work presented in this paper. In Section 3 we present some background on authorization in constrained IoT environments. In Section 4 we present four models for decentralized authorization. The first two models correspond to baseline scenarios where only hashes are recorded on a public blockchain (first model) and where a smart contract runs on a public blockchain (second model). The third model focuses on our first contribution that combines two blockchains with interledger mechanisms, while the fourth model focuses on both key contributions of the paper: decentralized authorization for constrained IoT devices utilizing two blockchains with interledger mechanisms. In Section 5 we evaluate the models in terms of transaction cost, delay, and reduction of data that needs to be sent to constrained devices. Finally, in Section 6 we conclude the paper and present future research directions.

2 Related work

The work in [6] presents a blockchain-based decentralized authorization system where authorization proofs can be efficiently verified. The work in [7] presents a decentralized access control system where IoT devices are required to interact directly with the blockchain and are assumed to be always connected, while [8, 9] present solutions where policies and access control decisions are directly recorded on Bitcoin's blockchain. The work in [10] present a system based on OAuth 2.0 where a smart contract generates authorization tokens, which a key server verifies in order to provide private keys that allow clients to access a protected resource. The work in [11] contains a high level description of using smart contracts with OAuth 2.0 to provide an architecture where a user can freely select the server to provide authorization for the user's protected resource. Finally, threshold signatures can be used to achieve authorization from a subset of parties that possess a share of a private signing key [12].

All the above works assume that the IoT devices interact directly with the blockchain or are capable devices, i.e. they are always connected to the Internet and are capable of implementing public/private key cryptographic functions. We do not make these assumptions, and propose a scheme where the authorization function is distributed across multiple servers. Our previous work [13] considered the case of a single AS and a single chain, and proposed an approach to verify that the IoT device and AS share a common secret.

3 Authorization in constrained environments

OAuth 2.0 is a framework for delegated authorization to access a protected resource [14]. It enables a third party application (client) to obtain access with specific permissions to a 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. Authorization is provided for different levels of access, which are termed *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 do 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 using TLS (Transport Layer Security). OAuth 2.0 also assumes that the protected resource is always connected to the Internet, hence can communicate with the AS to check the validity and scope of the access tokens presented 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 [15]. JWTs can use the JSON Web Signature (JWS) structure to digitally sign or integrity protect claims with a Message Authentication Code (MAC) [16]. Hence, unlike bearer tokens, JWT/JWS tokens are self-contained, i.e., they include all the information for the resource to verify their integrity without communicating with the AS. The JWT format is considered by the W3C Credentials Community Group for *Verifiable Credentials*, which can be combined with *Decentralized Identifiers* or *DIDs* [17]. A more efficient encoding derived from JWTs but based on CBOR (Concise Binary Object Representation) is the CBOR Web Token (CWT) [3, 4], which can be extended to create and process signatures, MACs, and encrypted data [18].

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 [3]. PoP tokens include a normal access token, such as a JWT/CWT, and a PoP key: access to the protected resource requires both the access token and the PoP key, which is used to secure the link between the client and the IoT device. The implementation of the decentralized authorization models presented in this paper adopts the CWT format, proposing two schemes to further reduce the amount of data that needs to be transmitted to the constrained device.

4 Blockchain-based authorization

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 proofs in the case of disagreements.

- Smart contracts can encode authorization policies in an immutable and transparent manner. Policies can depend on payments as well as other IoT events that are recorded on the same or on different blockchains.

- Leveraging smart contracts to make resource access requests provides a higher protection against DoS attacks. Although individual blockchain nodes could be subject to targeted DoS attacks, hampering a smart contract's execution would entail attacking *all* blockchain nodes at once, a rather impractical venture.

There are tradeoffs that involve the above features, namely between transparency and privacy and between the additional functionality of smart contract and the execution cost when smart contracts are executed on public ledgers. Specifically, recording data, such as hashes, and executing smart contracts on a public ledger provide wide-scale decentralized trust, since anyone can participate in the consensus mechanism of a public ledger, in addition to wide-scale transparency. However, these features come at a high transaction cost and delay due to the PoW (Proof-of-Work) consensus mechanism used in public blockchains. On the other hand, storing data and executing smart contracts on a private or permissioned ledger provides decentralized trust and transparency among the nodes participating in the ledger. Moreover, a private or permissioned ledger provides higher security and has a significant lower transaction cost and delay. The experimental results presented in Section 5 will quantify the transaction cost and delay gains in the case of decentralized authorization in constrained IoT environments.

We present four models that allow different tradeoffs in terms of cost, delay, complexity, and privacy:

- Linking authorization grants to blockchain payments

- Smart contract handling authorization requests

- Smart contract and two blockchains for authorization and payment with interledger mechanisms

- Decentralized authorization with multiple ASes

The first two models correspond to our baseline scenarios: in the first, only hashes of authorization information are immutably recorded on the blockchain and smart contracts are not used, whereas the second model utilizes a smart contract but on a single (public) blockchain. The third model, which focuses on our first contribution, exploits two blockchains whose transactions are securely linked using interledger mechanisms and quantifies the significant cost reduction that can be achieved by moving smart contract authorization functionality to a permissioned or private blockchain. The fourth model focuses on both key contributions of the paper: decentralized authorization for constrained IoT devices utilizing two blockchains with interledger mechanisms.

A hash-lock is a cryptographic lock that can be unlocked by revealing a secret whose hash is equal to the lock's value h. Unlocking a hash-lock can be one of the conditions for performing a

transaction or for executing a smart contract function. On a single blockchain, a hash-lock can be linked to an off-chain capability, e.g., message decryption, if the hash-lock secret is the secret key that can decrypt the message.

Hash-locks can be used on two or more blockchains that support the same hash function, to link a transaction on one chain to a transaction on the other chain: if the two transactions have hash-locks with the same value, then unlocking one hash-lock would reveal the secret that unlocks the other; hence, the two transactions are cryptographically linked through a dependence relation. More generally, hash-locks combined with AND/OR logic operators can implement elaborate dependencies involving transactions on multiple chains.

Time-locks are blockchain locks that can be unlocked only after an interval has elapsed. This interval can be measured in absolute time or in the number of blocks mined after a specific block. One usage of time-locks are refunds: a user (payer) can make a deposit to a smart contract address. The smart contract can have a function, which typically also includes a hash-lock, for a second user to transfer the deposit to another account (the payee's account). However, if the second user never calls this function, then the first user's deposit could be locked indefinitely in the smart contract's account. To avoid this, the smart contract can also include a refund function that allows the first user to transfer the amount he/she deposited back to the his/her account; however, this function can be called only after some time interval, which is the interval in which the second user must transfer the deposit from the smart contract account to the payee's account.

Contracts that include both hash and time-locks are referred to as hashed time-lock contracts (HTLCs) [19]. HTLCs have been used for atomic cross-chain trading (atomic swaps) [20, 21] and for off-chain transactions between trustless parties [22]. HTLCs can be implemented in blockchains with simple scripting capabilities, such as the Bitcoin blockchain, without requiring the advanced functionality of smart contracts. Smart contracts do not increase the capabilities of *interledger* mechanisms based on hash and time-locks, but increase the *intra-ledger* functionality. We investigate these features for decentralized authorization to constrained IoT devices.

In all the models presented below, the client sends a resource access request to the URL of the AS (model 1) or to the address of the smart contract responsible for handling access to the IoT device (models 2, 3, and 4). The URL or smart contract address can be obtained by the client sending a query to the IoT device or reading a QR code on it. However, this approach cannot ensure that the legitimate URL or smart contract address is provided by the IoT device. This can be ensured if the client uses a registry service that resides on the blockchain and contains a binding between the IoT device's URI and the URL of the AS or the smart contract address handling authorization, or by including this information in Decentralized Identifier (DID) documents [23].

Finally, in all models we assume that the client, the resource owner, and the ASes have an account (public/private key pair) on the blockchain (both the authorization and the payment blockchains for models 3 and 4).

4.1 Model #1: Linking authorization grants to payments and recording authorization information on the blockchain

With this model the initial communication between the client and the Authorization Server (AS) follows the normal authorization message exchange, such as OAuth 2.0, Figure 1. Specifically, in step 1 the client requests resource access from the AS. The AS generates a random PoP key which

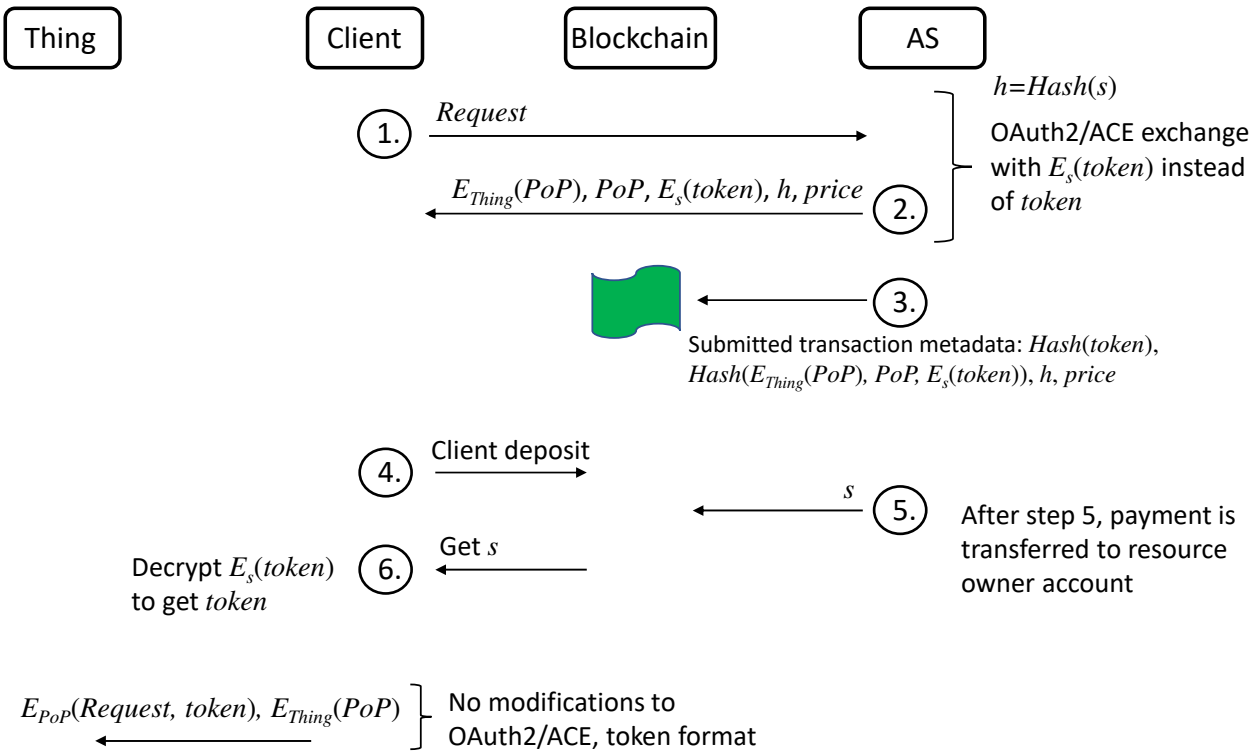

Figure 1: Model 1: Authorization grants are linked to blockchain payments and the hashes of the authorization information exchanged are recorded on the blockchain for verification in the case of disputes.

it sends to the client[1] together with its encryption with the secret key[2] K_{Thing} shared by the Thing (IoT device) and the AS; the client will later use the PoP key to establish a secure communication link with the Thing. Also, the AS sends to the client the access token encrypted with a secret s, i.e., $E_s(token)$, the hash $h = Hash(s)$ of the secret s, and the price for the requested level of resource access. The secret s is a secret randomly generated by the AS and is required for the client 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 client allows different levels of resource access, encoded in the access token's scopes, to correspond to 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 client once payment has been confirmed. The second is the hash of three items: $E_{K_{Thing}}(PoP)$, the PoP key, and $E_s(token)$; the second hash serves as proof of the information that is communicated using OAuth between the AS and the client. Note that the above authorization exchange does not ensure that the access token the client obtains from the AS indeed allows access to the Thing.

Also in step 3 a hashed time-lock payment is initiated on the blockchain, which allows the

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

[2]The secret key that the Thing and AS share is established during the provisioning (or commissioning) phase, when the Thing is bound to the AS.

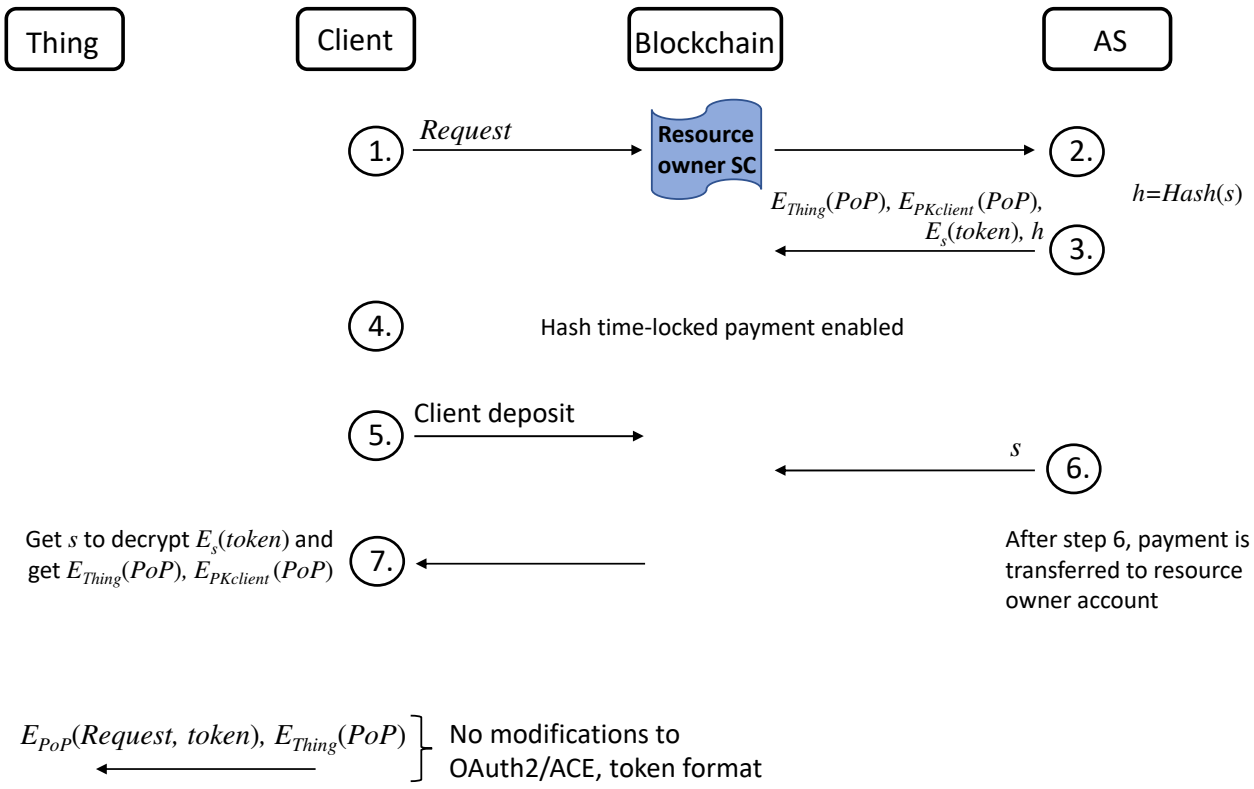

Figure 2: Model 2: Smart contract handling authorization requests and encoding authorization policies.

client to deposit an amount equal to 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 client can request a refund of the amount it deposited. Once the secret s is revealed, the client can get s from the blockchain (step 6) and decrypt $E_s(token)$, and thus obtain the access token. At this point, the client has all the necessary information to request access from the Thing, using normal OAuth 2.0 with the modifications from the ACE framework.

4.2 Model #2: Smart contract handling authorization requests

In the second model a smart contract is used to transparently record prices and other authorization policies defined by the resource owner, which is also the owner of the smart contract. Examples of such policies include permitting resource access to specific clients, determined by their public/private key pairs on the blockchain, and adding dependence of access authorization on IoT events that are recorded on the blockchain.

Whereas in the previous model the client and the AS interacted directly, in this model the interaction is through the smart contract, Figure 2. The smart contract code is executed by all blockchain nodes, providing a secure and reliable execution environment; this provides higher protection against DoS attacks, compared to the model in Section 4.1 where resource access requests are sent directly to the AS.

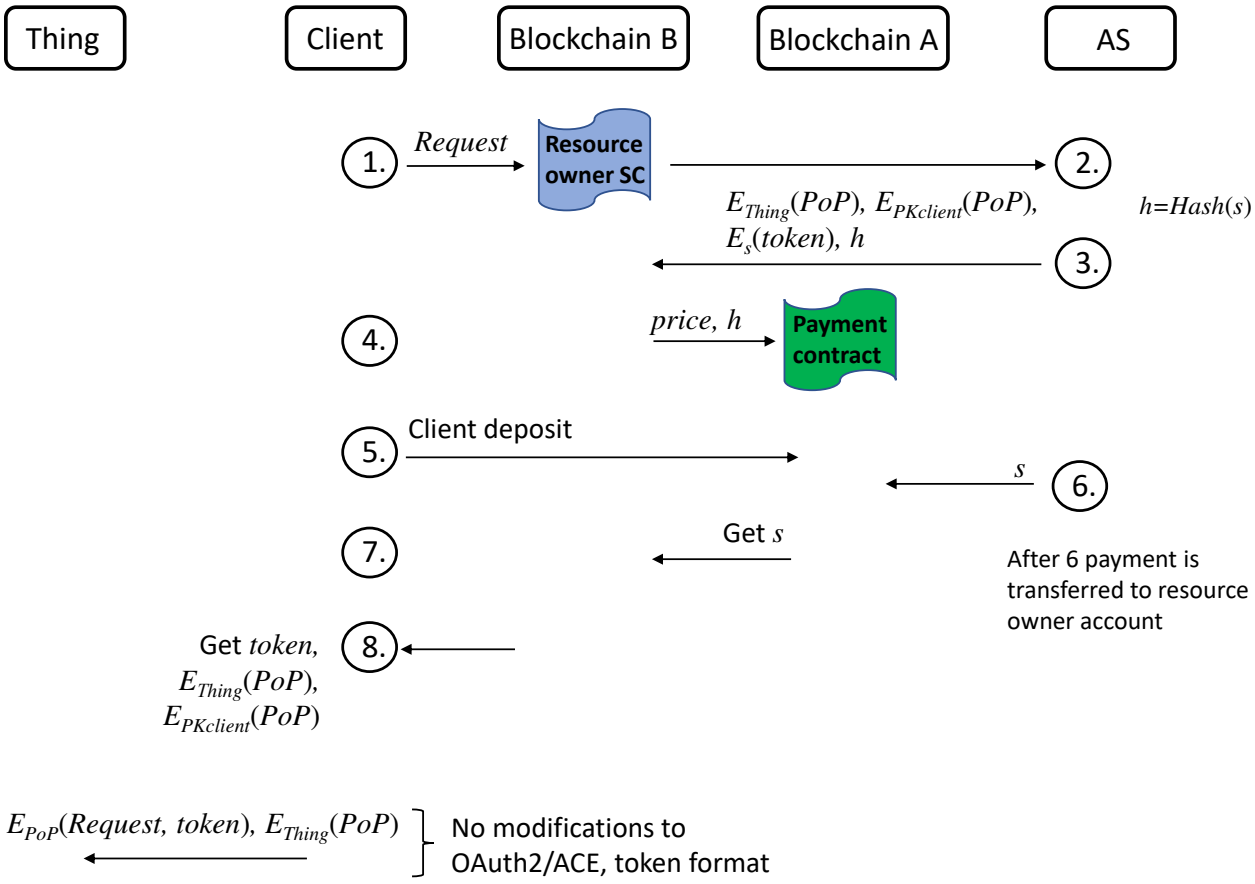

Figure 3: Model 3: Smart contract handling authorization requests and two chains interconnected with interledger mechanisms. The authorization blockchain B runs the authorization contract and the payment blockchain A handles payments.

4.3 Model #3: Smart contract and two blockchains with interledger mechanisms

In the model of this subsection the smart contract handling authorization requests and encoding policies is located on an *authorization blockchain*, while payments for resource access are performed on a *payment blockchain*, see Figure 3. Depending on whether the authorization chain is a public or a permissioned blockchain, different tradeoffs between transaction cost, delay, and privacy can be realized.

A hashed time-lock payment is initiated on the payment blockchain, where the client can deposit an amount corresponding to the resource access price. The amount will be transferred to the resource owner's account if the secret s is revealed. Once revealed, the secret s can be submitted to the smart contract on the authorization blockchain, which serves as a record on this blockchain that the payment was successfully performed. The client can obtain the secret s from the authorization blockchain together with the other necessary authorization information to access the protected resource.

One issue with the above model is how the payment contract on the payment chain is triggered

by the resource owner smart contract residing on the authorization chain. One alternative is to have an *interledger gateway* read the price and hash h from the authorization chain and submit it to the payment chain to initiate a payment (Step 4 in Figure 3), and later read the secret s submitted by the AS on the payment blockchain and record it on the authorization chain (Step 7); the interledger gateway can receive a fee for performing this function. Another alternative is to have the interledger functionality be performed by the AS or the client.

4.4 Model #4: Decentralized authorization with multiple ASes

Note that all of the operations of the authorization process cannot be moved onto the blockchain, since some operations involve processing secret information: keys to produce token signatures and keys shared with the Thing. Performing the authorization functions redundantly in the nodes of a private or permissioned blockchain would provide a higher level of resilience to node failures compared to a single AS, but results in reduced security since compromising a single blockchain node would lead to secret keys being disclosed.

Rather than moving all the authorization functionality to the blockchain, we propose an alternative approach for decentralized authorization that ensures security and provides fault tolerance if some number of ASes are faulty or misbehave. Let n be the number of ASes that are collectively responsible for providing authorization. Each AS i shares a different secret key, K_{Thing_i}, with the protected resource (Thing). Authorized access to the Thing requires tokens from m out of n servers. The policy specifying the required number of ASes is defined in the smart contract and is also known to the Thing. Fault tolerance is provided by having n ASes which can respond to requests, but requiring only $m < n$ for authorization to proceed. Compared to having a single AS, the proposed scheme provides higher security since m ASes need to agree in order for the client to access the protected resource.

There are two alternatives for how m servers are selected to provide authorization. With the first, the smart contract selects the specific m servers, based on their previous behavior that can involve response time or transaction cost. The second policy selects the first m servers that respond to an authorization request.

4.4.1 Selection of m-out-of-n ASes based on previous performance

This policy requires that the smart contract handling the requests maintains the list of ASes along with their performance for previous authorization requests. The performance can be in terms of the time they responded to requests or their transaction cost. Such information allows the smart contract to prioritize ASes in order to select those that respond quickly, hence avoid ASes that have a high delay, are faulty, or have a high transaction cost. This intelligent selection of ASes requires maintaining historic information, which would require higher memory and processing by the smart contract at the authorization ledger. This policy can be implemented by including in the notification event in step 2 of Figure 4 the information for the m ASes that were selected by the smart contract. Moreover, in step 3 the hashes corresponding to the secrets of the selected m ASes are included in the payment contract. Hence, in step 6 only the selected m ASes can respond. Note that in case of an error or fault of one of the selected m ASes, the client can have a timeout to restart the authorization procedure from the beginning. Moreover, in the case of an

error or fault the smart contract can keep information for the faulty AS, in order to avoid it in subsequent authorization requests.

4.4.2 Selection of m ASes that respond first

With the second policy the smart contract allows all ASes to respond to the authorization request in step 3 of Figure 4, and selects the first m ASes that respond. With this approach the smart contract does not need to maintain the list of ASes, nor information of their performance for previous authorization requests. However, there is a possibility that the smart contract receives more than m responses. This depends on the duration for mining a block on the blockchain (in the case of public blockchains with Proof-of-Work consensus) or for obtaining consensus to add it to the blockchain (in the case of permissioned blockchains). In public blockchains, these responses can incur a gas cost independent of whether the ASes that gave the response were among the m ASes to provide decentralized authorization. Once the first m AS responses are received, then in step 4 of Figure 4 the hashes corresponding to the secrets of the first m ASes that responded are included in the payment contract. Hence, in step 6 only the first m ASes that responded in step 3 can submit the secret corresponding to their hash. As in the previous policy, to handle the case where an AS exhibits an error or is faulty in step 6 of Figure 4, the client has a timeout after which it restarts the authorization procedure from the beginning.

By exploiting the simplicity of the procedure for selecting the subset of ASes, a variant of the second policy is to avoid having the client send the initial request to the smart contract on the authorization chain in step 1 of Figure 4. Instead, the client can send the request directly to all ASes; this requires that the client knows the ASes. The advantage of this variant of the second policy, as we will see in Section 5, is the lower total delay, since the number of transactions on the authorization chain is reduced by one.

In response to the client's authorization request, each AS sends a different PoP key PoP_i, encrypted with the Thing's secret key and the client's public key, and an access token with a MAC tag to ensure its integrity (Figure 4). The client thus obtains m different PoP keys, which it XORs to obtain the secret PoP key that will be used to establish a secure communication link with the Thing. These m PoP keys, encrypted with the Thing's key K_{Thing_i} that it shares with each of the m ASes, are also sent to the Thing. Hence, if the Thing performs the same XOR function on the m PoP keys it will obtain the same PoP key as the client.

Recall from the discussion in Section 3 that a requirement is to reduce the amount of data transmitted to constrained devices. We propose two schemes for reducing the authorization information the client sends to the Thing: aggregate MAC tags and transmission of common token fields once. With aggregate MAC tags [24], the client sends to the Thing the token payloads received from the m ASes, but only one aggregate MAC tag that is computed by taking the XOR of the m MAC tags the client receives from the m ASes. With the second optimization, the client sends the token fields that are common to all ASes only once (these correspond to $token'_1, \ldots token'_m$ in Figure 3). The common token fields include the subject (Thing) the token refers to, the scope of access, the token creation time, the token validity time, and the token type. The fields which are different include the AS and token id fields. The reduction of the amount of data the client sends to the Thing will be evaluated in the next section.

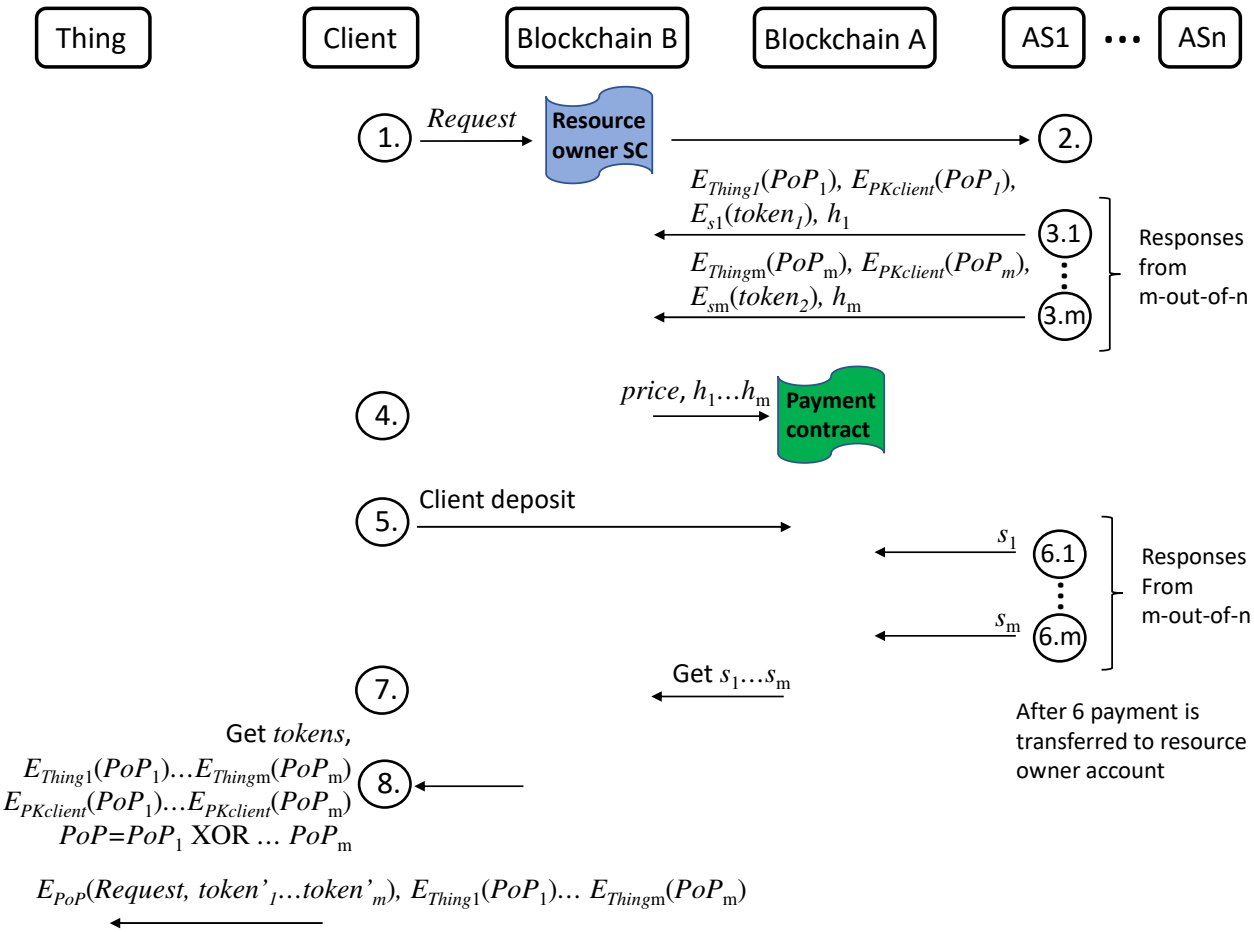

Figure 4: Model 4: Decentralized authorization: Each authorization grant requires m out of n AS responses.

5 Evaluation

For the evaluation we utilized the Rinkeby public Ethereum testnet[3] and the Ropsten testnet[4]. We used the Infura Ethereum node cluster[5] for submitting transactions to the two testnets. The ASes were based on a PHP implementation of the OAuth 2.0 framework[6], extended to support CWT's CBOR encoding[7]. The client used Web3.js to interact with the blockchain.

For the scenarios involving two blockchains that are shown in Figures 3 and 4, for the authorization chain (blockchain B in the figures), we considered both a private Ethereum network and Hyperledger Fabric. For the private Ethereum case we deployed a node running the Go-Ethereum software. Hyperledger Fabric [25] is a permissioned blockchain system that is part of Linux Foundation's open-source Hyperledger project. A key feature of Fabric is that smart con-

[3] https://www.rinkeby.io/
[4] https://ropsten.etherscan.io/
[5] https://infura.io
[6] https://github.com/bshaffer/oauth2-server-php
[7] https://github.com/2tvenom/CBOREncode

tracts, which are called chaincode in Fabric, are executed in a distributed manner following an execute-order-commit flow pattern instead of the common order-execute-commit pattern. The chain-code is submitted by clients to special nodes, called endorsing nodes, that execute (or simulate) the chaincode. This execution identifies the read-write dependencies and can involve checking that a transaction conforms to business rules identified in the chaincode. After a transaction has received a sufficient number of endorsements, it is sent to ordering nodes that order and group the transactions in blocks. Finally, the transactions are validated to ensure the consistency of the read-write dependencies. After the transactions are validated, they are committed, i.e., the ledger is updated.

The first goal of the evaluation is to quantify the gains, in terms of reduced execution cost and delay, that can be achieved by combining a public and a private or permissioned ledger. A second goal is to compare the two policies for selecting the subset of the ASes that are required to send authorizations. A higher number of ASes that need to respond to perform authorization incurs a higher cost, which can be reduced when a public and permissioned chain are combined instead of using a single public chain. Finally, a third goal is to highlight the impact that different implementation choices can have on the delay incurred on the Hyperledger Fabric permissioned ledger.

In Table 1, the smart contract & one blockchain is similar to model in Figure 3 (smart contract & two blockchains), but with one blockchain for both authorization and payment instead of two blockchains, where one handles authorization and the other payment. Similarly, the decentralized & one blockchain models are similar to the model in Figure 4 (decentralized authorization & two blockchains) with one blockchain for both authorization and payment instead of two blockchains, where one handles authorization and the other payment. For the results with two blockchains, we use the public blockchain (Rinkeby or Ropsten) as the payment chain and a private Ethereum network as the authorization chain. The results shown include the gas and delay due to transactions on the public blockchain only. Also, the results in Table 1 consider the second policy discussed in Section 4.4.2.

5.1 Execution cost (gas)

The second column in Table 1 shows that the execution cost of a smart contract on the Ethereum Virtual Machine (gas) on the public Rinkeby testnet is significantly higher compared to simply recording hashes (first line in Table 1). However, when the smart contract authorization functionality is moved to a private blockchain (models with 2 BCs in Table 1), then the execution cost is significantly reduced: For 1, 2, and 3 ASes the execution cost when two blockchains are used is 33.2%, 23.1%, and 21.1% of the execution cost when a single blockchain is used. In the case of two blockchains, the execution cost involves the transactions on the payment blockchain, which for Table 1 is the Rinkeby public Ethereum testnet.

5.2 Delay on the Ethereum blockchain

The delay on the Ethereum blockchain is due mainly to the block mining time. The smart contract model with one blockchain has four transactions, while the model that records only hashes has three; hence, the delay for the smart contract model is expected to be 33% higher; this agrees with

Table 1: Execution cost (Gas) and delay - Rinkeby

Model	Gas	Delay in secs (s) (95% conf. int.)
Hashes of auth. inform.-Fig. 1	102 489	43,2 (42,3, 44.1)
SC & 1 BC	258 166	59.3 (57.6, 61.1)
SC & 2 BCs-Fig. 3	85 682	43.0 (39.8, 46.2)
Dec-Auth 2-of-4 & 1 BC	1 440 540	60.5 (54.4, 67.3)
Dec-Auth 2-of-4 & 2 BCs-Fig. 4	332 569	42.1 (39.4, 44.8)
Dec-Auth 3-of-4 & 1 BC	2 124 249	63.7 (57.2, 70.2)
Dec-Auth 3-of-4 & 2 BCs-Fig. 4	447 940	44.7 (39.6, 49.9)

Table 2: Delay - Ropsten

Model	Delay in secs (s) (95% conf. int.)
Hashes of auth. inform.-Fig. 1	53.2 (40.3, 66.1)
SC & 1 BC	64.4 (52.3, 77.1)
SC & 2 BCs-Fig. 3	57.8 (46.0, 69.7)
Dec-Auth 2-of-4 & 1 BC	76.7 (61.5, 92.0)
Dec-Auth 2-of-4 & 2 BCs-Fig. 4	49.1 (37.7, 60.5)
Dec-Auth 3-of-4 & 1 BC	77.5 (60.5, 94.6)
Dec-Auth 3-of-4 & 2 BCs-Fig. 4	52.2 (42.6, 61.7)

the results in the third column of Table 1, according to which the smart contract model with one blockchain has average delay 59.3s, which is 37.3% higher than the delay when only hashes are recorded, 43.2s (also shown is the confidence interval from 20 runs). Table 1 quantifies the reduced delay when a public chain is combined with a private chain (note that in this case the average delay shown is the delay on the public chain only): e.g., the 2 out of 4 decentralized model with two chains has average delay 42.1s, which is 30.4% smaller than the delay with one chain, 60.5s. Table 1 also shows that for both one and two blockchains, the average delay is not significantly influenced by the number of ASes. Also, for two blockchains the average delay is close to the delay when only hashes are recorded. The above results show the advantages, in terms of lower delay, that can be achieved by combining public and private/permissioned ledgers.

Table 2 shows that the delays and confidence intervals for the Ropsten testnet are higher than the Rinkeby testnet. We attribute this difference to the fact that Rinkeby uses the Proof-of-Authority (PoA) for distributed consensus, while Ropsten uses Proof-of-Work, as the Ethereum mainnet. For both the Rinkeby and Ropsten testnets, the delay depends on the gas price that is given when a transaction is submitted.

5.3 Comparison of two policies for selecting m-out-of-n ASes

Next we compare the two policies discussed in Section 4.4 for selecting m-out-of-n ASes, in terms of the execution cost. The first policy, which is described in Section 4.4.1, requires that the past performance of all ASes is maintained and the smart contract selects the subset of ASes based on

this performance. Unlike the first policy, the second policy, which is described in Section 4.4.2, allows all ASes to respond to authorization requests and selects the subset of ASes that responded first. The two policies are implemented in the smart contract that handles authorization requests in step 1 of Figure 4. The results we present below consider the case where the authorization contract and the payment contract run both in the Ethereum public testnet and the case where the authorization contract runs on a private blockchain and only the payment contract runs on the Ethereum public testnet.

Figure 5 shows that in the case of one blockchain, i.e., both the authorization smart contract and the payment contract run on the public Ethereum testnet, the first policy has a higher execution cost, measured in terms of the execution cost of the smart contract running on the Ethereum network, compared to the second policy. This is due to the past history that this policy maintains in order to select the ASes for authorization; maintaining the past history incurs a high execution and storage cost when it is maintained on a public blockchain. On the other hand, the second policy does not maintain historic information but rather selects the first m ASes that respond. Figure 5 also shows that, as expected, when more ASes need to send authorizations the execution cost is higher. Moreover, the incremental cost is the same for both policies. The higher execution cost is exchanged with the higher level of security and trust when more ASes are required to send authorizations.

Figure 5 also shows that both policies have the same execution cost in the case of two blockchains, a public and a private/permissioned blockchain, since in this case only the payment transactions on the Rinkeby testnet, which are the same for both policies, incur an execution cost. The results for two blockchains illustrate the gains from utilizing a private/permissioned and a public ledger: The private/permissioned ledger can implement more elaborate authorization policies and store information that would incur a high cost on a public ledger. Of course, in a private/permissioned ledger trust is maintained among the participating nodes, rather than on a wide-scale as in the case of public ledgers. Depending on the application requirements, trust among a limited set of nodes may be sufficient, making the combination of private and public ledgers an attractive and practical approach.

5.4 End-to-end delay when Hyperledger Fabric is interconnected with Rinkeby

In this section we investigate the delay that is induced when the authorization contract is implemented on the Hyperledger Fabric permissioned ledger, identifying how different implementation choices influence this delay. Fabric's execute-order-commit flow pattern allows faster execution of parallel transactions. However, when concurrent transactions access the same state variable then a commit error may occur. Specifically, if prior to committing a transaction that modifies a variable, a second transaction that modifies the same variable is executed and endorsed, then the second transaction will fail validation and a commit error will occur. Concurrent transactions occur in our scenario, since multiple ASes can respond to an authorization request. One approach to address the issue of concurrent transactions is to add a FIFO (First-In-First-Out) module that serializes the transactions from ASes and sends them to the underlying Fabric system with a small delay between consecutive transactions. A disadvantage with this approach is that the serialization operation adds delay, which increases as the number of ASes that need to respond for authorization

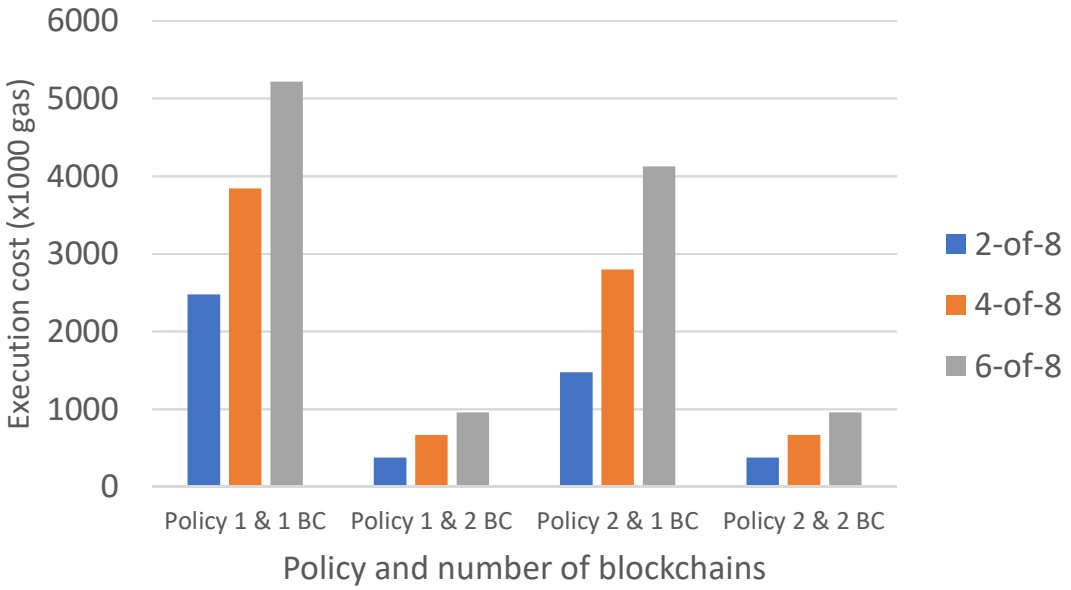

Figure 5: Execution cost (gas) for the two policies, for the case of one and two blockchains. For two blockchains, the cost involves only the cost for transactions on the payment blockchain.

increases. An alternative approach, which is the approach we consider in the evaluation results shown in Figure 6, is to maintain a separate state, in the form of a table, where the information submitted by each AS is stored. This table also maintains a boolean variable indicating whether the specific AS has responded to the authorization request. Then, a module that is external to Hyperledger Fabric periodically reads the aforementioned boolean variables to determine whether the necessary number of ASes has responded; this corresponds to steps 3.1 to 3.m in Figure 4. Once the required number of ASes has responded, the procedure can continue with step 4 in Figure 4.

Finally, a third approach is to perform the above functionality in the interledger gateway. As discussed in Section 4.3, the interledger gateway is responsible for reading the price and hash h from the authorization chain and submitting it to the payment chain to initiate a payment (Step 4 in Figure 3). After the secret is submitted by the ASes on the payment blockchain, the interledger gateway reads the m secrets s_1 to s_m from the payment chain and submits them to the authorization chain. Moreover, instead of periodically reading the number of ASes that responded as done by the external module in the second approach described above, the interledger gateway can be made aware of the number of ASes m that need to respond, hence can wait for m events that correspond to the responses 3.1 to 3.m of the m ASes in Figure 4. After the interledger gateway receives the m events, the procedure can continue with step 4 in Figure 4.

Note that all three of the approaches presented above for handling concurrent transactions from multiple ASes can be used to implement the two policies presented in Sections 4.4.1 and 4.4.2. Indeed, the performance in terms of delay is the same for both policies.

Figure 6 shows the end-to-end delay when Hyperledger Fabric (authorization blockchain) is interconnected with Rinkeby (payment blockchain), when the second approach described above is used for handling concurrent transactions on Hyperledger Fabric. The 95% confidence interval for these results are less than 2% of the average value that is shown. The figure shows that the delay on the Rinkeby network is consistent with the delay shown in Table 1 and is independent

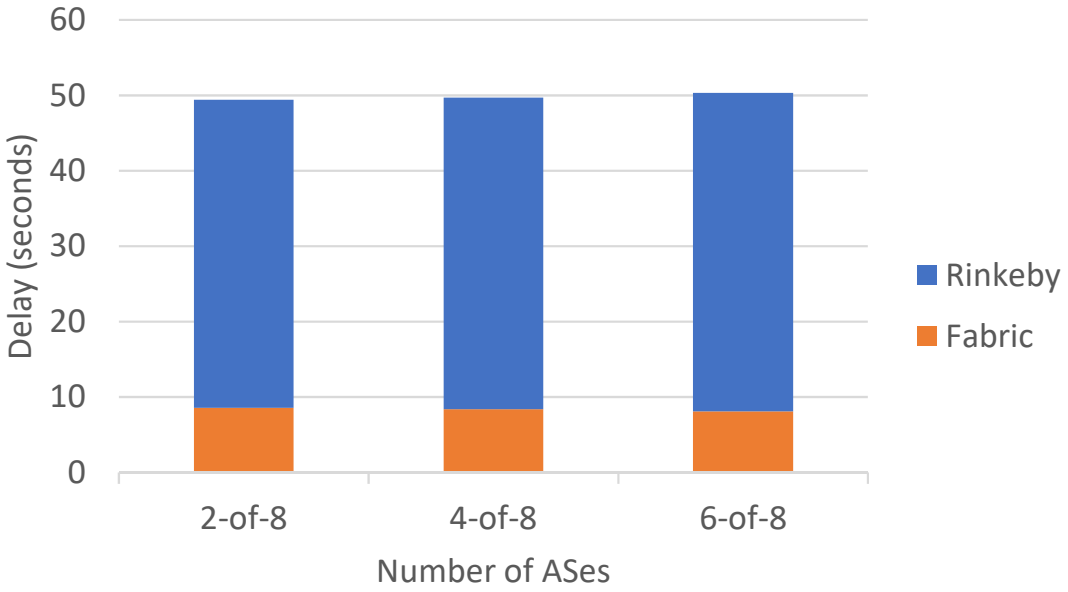

Figure 6: End-to-end delay when Hyperledger Fabric (authorization blockchain) is interconnected with the public Ethereum testnet Rinkeby (payment blockchain).

of the number of ASes. Moreover, the figure shows that approximately 16-18% of the total delay is due to Hyperledger Fabric. With the third approach presented above, which requires that the interledger gateway is aware of the number of ASes that need to respond and receives individual event notifications from the ASes that respond to authorization requests, the delay on the Hyperledger Fabric network is further reduced by approximately 28%, Figure 7, and becomes 12.5-13.5% of the total delay.

The above delay results apply to both policies presented in Section 4.4 for selecting the subset of ASes that participate in the authorization. However, the simplicity of the second policy allows an alternative implementation that avoids having the client send the request to the smart contract on the authorization chain in step 1 of Figure 4. Instead, the client can send the request directly to all ASes. This variant, which is only possible with the second policy, can further reduce the delay on the Hyperledger Fabric network by approximately 29%, Figure 7.

While the results of Section 5.2 show that combining public and private/permissioned ledgers can reduce the overall delay, the results of this section show that the delay on the private/permissioned ledger can be influenced by various implementation choices.

5.5 Reduction of data client sends to Thing.

Utilizing CWT encoding instead of JWT reduces the size of tokens from 310 to 122 bytes. For the smart contract with one blockchain model, the transaction cost is higher by approximately 17% compared to the cost shown in Table 1.

The proposed optimizations further reduce the amount of data that the client needs to send to the Thing. For decentralized authorization with three ASes and without the optimizations, the client sends 366 bytes (3×122 bytes) for three tokens and 96 bytes (3×32 bytes) for three PoP keys, a total of 462 bytes. With aggregate MACs, the client sends one aggregate MAC instead of three,

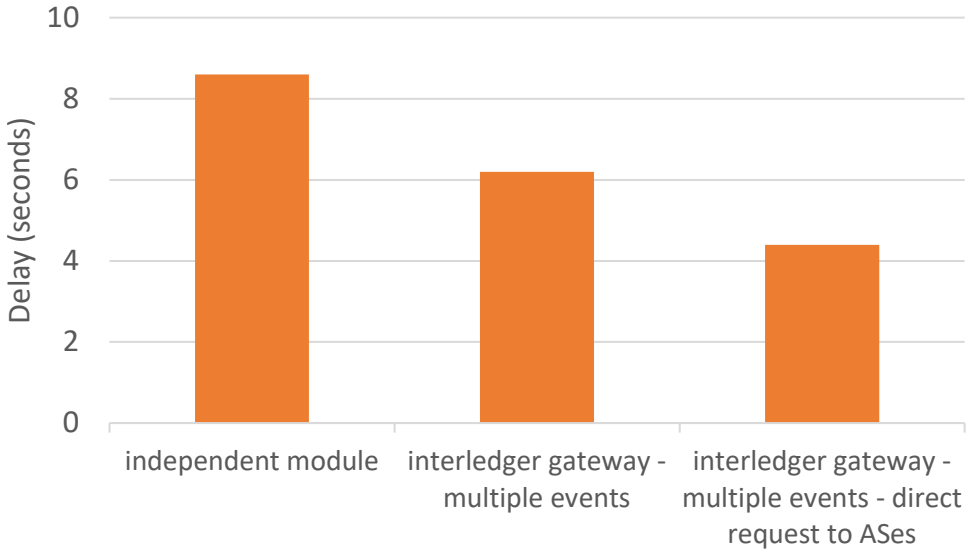

Figure 7: Transaction delay on Hyperledger Fabric (authorization blockchain) for different implementations: i) independent module, ii) interledger gateway receiving individual event notifications when ASes respond, iii) interledger gateway and client sending initial request to the ASes rather than to the smart contract.

i.e. 64 bytes less, hence a total of 398 bytes, which is a 13.9% reduction. The optimization where common token fields are sent once results in 84 bytes less, which is a 18.2% reduction, reducing the size to 314 bytes. The two optimizations together give a 32.0% reduction of the number of bytes the client needs to send to the Thing. The reduction for more ASes would be higher.

6 Conclusions and future work

Smart contracts can increase the functionality of a blockchain, and transparently encode authorization policies and logic. However, smart contracts are costly if executed on a public blockchain. Interledger mechanisms enable the interconnection of multiple blockchains, hence allow moving smart contract functionality to private or permissioned blockchains with a lower execution cost. This paper has proposed such models for decentralized authorization to constrained IoT devices and quantified their performance in terms of reduced transactions costs and delay, while also proposing mechanisms to reduce the amount of data that needs to be sent to IoT devices. Our experimental evaluation has utilized the public Ethereum testnets Rinkeby and Ropsten, and the Hyperledger Fabric permissioned blockchain. In the case of decentralized authorization where multiple ASes are involved, we have presented and evaluated two policies for selecting a subset of ASes that will participate in the authorization: the first policy selects ASes based on their past performance whereas the second policy simply selects the first ASes that respond. When executed on a public blockchain, the first policy involves a higher execution cost due to the performance history it maintains.

We are currently investigating solutions for providing more general services, such as tracking of assets in a supply chain, in a decentralized manner over multiple ledgers.

References

[1] S. Gerdes, et al., An architecture for authorization in constrained environments, Internet-Draft, IETF (April 25, 2019).

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

[3] L. Seitz, et al., Authentication and Authorization for Constrained Environments (ACE) using the OAuth 2.0 Framework (ACE-OAuth), Internet-Draft, IETF (November 21, 2019).

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

[5] V. A. Siris, D. Dimopoulos, N. Fotiou, S. Voulgaris, G. C. Polyzos, 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, 2019.

[6] M. P. Andersen, et al., WAVE: A Decentralized Authorization System for IoT via Blockchain Smart Contracts, Tech. rep., University of California at Berkeley (December 2017).

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

[8] D. D. F. Maesa, P. Mori, L. Ricci, Blockchain based access control, in: Proc. of IFIP Distr. Appl. and Interop. Sys. (DAIS), 2017.

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

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

[11] T. Hardjono, Decentralized Service Architecture for OAuth2.0 (March 25, 2018).
URL `Internet-Draft,IETF`

[12] A. Boldyreva, Efficient threshold signature, multisignature and blind signature schemes based on the Gap-Fife-Hellman-group signature scheme, IACR Cryptology ePrint Archive 2002 (2002) 118.

[13] N. Fotiou, V. A. Siris, 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.

[14] D. Hardt, et al., The OAuth 2.0 Authorization Framework, RFC 6749, Standards Track, IETF (October 2012).

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

[16] M. Jones, J. Bradley, N. Sakimura, JSON Web Signature (JWS), RFC 7515, Standards Track, IETF (May 2015).

[17] M. Sporny, et al., Verifiable Credentials Data Model 1.0: Expressing verifiable information on the Web, Draft Community Group Report, W3C (September 05, 2019).

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

[19] Bitcoin Wiki, Hashed Timelock Contracts (HTLC), https://en.bitcoinwiki.org/wiki/Hashed_Timelock_Co

[20] Bitcoin Wiki, Atomic cross-chain trading, https://en.bitcoinwiki.org/wiki/Atomic_cross-chain_trading.

[21] V. Buterin, Chain Interoperability, R3 Report (September 2016).

[22] J. Poon, T. Dryja, The Bitcoin Lightning Network: Scalable o-chain instant payments, https://lightning.network/lightning-network-paper.pdf, last accessed 30/09/2019 (January 14, 2016).

[23] D. Reed, et al., Decentralized Identifiers (DIDs) v0.13: Data Model and Syntaxes for Decentralized Identifiers, Final Community Group Report, W3C (August 13, 2019).

[24] J. Katz, A. Y. Lindell, Aggregate message authentication codes, in: Proc. of The Cryptopgraphers' Track at the RSA conference on Topics in cryptology (CT-RSA), 2008.

[25] E. Androulaki, et al., Hyperledger fabric: A distributed operating system for permissioned blockchains, in: Proc. of ACM EuroSys Conference, 2018.