Fotiou, Nikos; Siris, Vasilios A.; Voulgaris, Spyros; Polyzos, George C.; Lagutin, Dmitrij

Bridging the cyber and physical worlds using blockchains and smart contracts

Published in:
Workshop on Decentralized IoT Systems and Security

DOI:
10.14722/diss.2019.23002

Accepted/In press: 01/01/2019

Document Version
Peer reviewed version

Please cite the original version:
Fotiou, N., Siris, V. A., Voulgaris, S., Polyzos, G. C., & Lagutin, D. (Accepted/In press). Bridging the cyber and physical worlds using blockchains and smart contracts. In Workshop on Decentralized IoT Systems and Security Internet Society. https://doi.org/10.14722/diss.2019.23002

Bridging the cyber and physical worlds using blockchains and smart contracts

Nikos Fotiou, Vasilios A. Siris, Spyros Voulgaris, George C. Polyzos
Mobile Multimedia Laboratory, Dept. of Informatics
School of Information Sciences and Technology
Athens University of Economics and Business
{fotiou,vsiris,voulgaris,polyzos}@aueb.gr

Dmitrij Lagutin
Dept. of Communications and Networking
School of Electrical Engineering
Aalto University
dmitrij.lagutin@aalto.fi

***Abstract*—We address the limitations of existing information security solutions when applied to the cyber-physical world. In particular, we consider the case of Internet of Things (IoT) actuation and we argue that it is hard to secure such a process. To this end, we propose a "damage control" approach, where service time is divided into slots and users perform micro-service transactions, paying essentially in advance for each one, corresponding to one service slot. Under these circumstances, in the case of service disruption, a user, in the worst case, may lose the amount of money that corresponds to a single micro-service transaction in a single time slot. We implement our solution by leveraging blockchain-based smart contracts, off-chain payments, and one-time Hash-based Message Authentication Code (HMAC) passwords. Our solution supports IoT devices with limited processing capabilities and which are not necessarily connected to the Internet. Moreover, with our solution, IoT devices do not interact directly with the blockchain. In fact, they are oblivious to the use of blockchain technology. They do not store any user-sensitive information, neither are payments made to or is value stored on the devices.**

I. Introduction

An interesting aspect of the Internet of Things (IoT) is that it intertwines the cyber with the physical world, i.e., through the IoT and by using actuation devices users can alter the physical world (e.g., turn a light on, charge a device, unlock a door, make a human heart beat at a specific pace, etc.). On the other hand, it is hard to verify an actuation process from the standpoint of the cyber world in a reliable and undeniable way. For example, consider the case of a "pay to charge" service for charging electrical vehicles; it is difficult for the entity that handles the payments to verify that the charging station indeed worked as anticipated by the users.[1]

[1]If a cyber process attempts to determine whether the service has been provided at the expected level, it will have to rely on information provided by the involved entities, which however cannot be fully trusted to provide truthful or reliable data, as they have incentives to misreport. Or, the process could rely on a trusted third party to report truthfully, however, this approach negates the decentralised character of the solution we promote in this work.

Workshop on Decentralized IoT Systems and Security (DISS) 2019
24 February 2019, San Diego, CA, USA.
ISBN
https://dx.doi.org/10.14722/diss.2019.23002
www.ndss-symposium.org

In this paper, we are concerned with actuation processes that have a reasonably long duration, which can be divided into periods with the following properties: the intended process is the union of the processes during all the periods and the cost of the service is additive, so that the total cost (or price) is the sum of the costs (or prices) for the periods. Thus, if needed, the process can be stopped at the end of each period and the correct and acceptable payment from both sides would be the sum for the completed periods. I.e., we consider each period as a service quantum, indivisible, and we assume that both sides can determine whether the service quantum was provided, without further qualifications. This allows the service periods to be treated separately. An example of a service that can be reasonably modelled in this way is that of a charging station charging an electric car (as opposed to unlocking a door). Users could be charged based on the duration of the actuation process (e.g., for the duration of the charging) or in instalments based on the units of energy provided in the various periods of time (e.g., for each kWh).

Under these conditions we propose a solution that enables users to pay as they go, i.e., as long as they receive service. In particular, we divide service time in (small) slots (consider them equal in time, or price, for simplicity) and the parties agree on a price per slot (not necessarily monetary). Thus, a user (pre-)pays for a time slot and if he receives satisfactory service during the slot, he proceeds with (pre-)paying for another slot, and so forth. With this approach and in case of service disruption, a user may in the worst case lose the amount of money that corresponds to the last single slot.

Furthermore, we consider a realistic setup where a device has limited computation power, it is not connected to the Internet, but it has local connectivity, e.g., using Wi-Fi Direct, Bluetooth, or NFC to communicate with its users, and it cannot (or will not) handle payments. Our solution is based on a semi-trusted third party, referred to as the *Authorization Server* (inspired by OAuth 2.0 [1]) and leverages blockchain-based smart contracts, off-chain micropayments, and one-time Hash based Message Authentication Code (HMAC) passwords [2].

A. Background

A blockchain is an append-only ledger of transactions replicated throughout a network. Transactions are validated by a number of network nodes and are added to the ledger

upon consensus, assuring this way that no single entity has control over the ledger. A smart contract is a replicated application that "lives" in the blockchain. Users can interact with a smart contract by sending transactions to its "address" in the blockchain. For any interaction with a smart contract, all operations are executed by the blockchain, in a deterministic and reliable way. Smart contracts can verify blockchain user identities and digital signatures and they can perform a number of operations. The code of a smart contract is immutable and it cannot be modified even by its owner or creator. Moreover, all transactions sent to a contract are recorded in the blockchain. Blockchains and smart contracts are considered a "democratic" way for maintaining transactions [3] and are envisioned to provide novel security mechanisms [4] for the IoT. Our solution has been built using Ethereum [5] smart contracts.

Various solutions related to our work have been recently proposed. Lundqvist et al. [6] built a Bitcoin-based system for micropayments between IoT devices. Their system assumes that IoT devices "understand" the Bitcoin protocol and they can interact with the Bitcoin blockchain. This may be a strong assumption for many IoT devices. In our work IoT devices are completely oblivious to the blockchain. Furthermore, the scripting capabilities of Bitcoin are limited, compared to an Ethereum-based smart contract. For this reason, their solution requires more interactions with the blockchain. Similarly, Huang et al. [7] propose a Bitcoin-based system for outsourcing computations to Fog devices, Zhang et al. [8] propose a similar system for outsourcing computations to a Cloud system, and Król and Psaras [9] implement a micropayment system that enables IoT devices to offload resource-heavy computations to Edge devices. Various research efforts propose off-chain payment channels using trusted hardware. For example, Lind et al. [10] propose a payments channel based on trusted hardware. Similarly, Bentov et al. [11] leverage trusted hardware to implement a real-time decentralized "currency exchange". Our solution does not require any special execution environment. In our previous work [12] we used the concept of the authorization server to enable a user to pay for an IoT service using blockchain technology, but we did not consider off-chain transactions nor micropayments. In this paper we focus on actuation processes and off-chain micropayments.

B. Contributions

With this paper we are making the following contributions:

- We limit the cost that the disruption of an actuation service has to a user to a predetermined amount. This amount can be pre-agreed between the two parties.
- We provide fast, blockchain-based micro-payments to constrained IoT devices, incapable of performing public-key encryption, or (directly) participating in the blockchain, or storing blockchain-related secrets.
- We enable "payment delegation" allowing users not owning blockchain credentials to perform payments, up to a specific, pre-configured amount, for a specific service.
- We support many-to-one payments, enabling multiple users, authorized by a single entity that owns blockchain credentials, to pay for a service.
- We build a presently feasible solution that relies on existing, already deployed technologies.

II. System overview

Our system considers the following entities: an actuation device, referred to as the *IoT device* or simply the *device*, a *client* interacting with the device, and an *Authorization Server* (AS), which is in charge of authenticating clients and authorizing them to access devices. In order to better understand our system entities and their interactions, throughout the paper we consider the use case of a car charging service. In this use case, charging stations hold the role of the device and cars hold the role of the client. A car can initiate the charging process only if it gets authorized by an AS managed by the charging station's owner. Cars (or car owners as we will see) pay the AS through a blockchain. The entity that performs the payments is the owner of a blockchain *wallet*. Charging stations are oblivious about the payment process, they are disconnected from the Internet, and they can interact directly only with a car using Wi-Fi Direct, Bluetooth, or NFC.

The authentication problem in an environment that includes constrained devices is a challenging and multifaceted one, and any scheme will have multiple trade-offs out of necessity. For example, the client, device, and the AS need to be in contact with each other at some point of time. The important question is: which communication should be minimized? Inclusion of blockchains introduces yet another optimization angle, since a scheme should consider the cost of blockchain transactions and smart contracts. Finally, a scheme should consider various options when it comes to payment flexibility (e.g., one-to-one payments, many-to-many) and to security, such as what is the window of tolerable losses from both the device and the client? Our solution minimizes the communication between the device and the AS, making it suitable for cases where the device and the AS do not have constant network connectivity. Furthermore, with our constructions we support one-to-one and many-to-one payments. Finally, our solution tries to minimize client losses and supports zero losses for the devices that offer the actuation service.

From a high-level perspective our system operates as follows. A client (or a client owner) makes a "deposit" to a smart contract in order to use an actuation service. Then, it requests from an AS an "one-time password" that can be used for invoking the actuation process for a time slot. This password in provided in exchange to a "payment receipt". The payment receipt can be used by the AS to claim from the smart contract (part of) the deposit. If a client needs more passwords, it produces more receipts, and the process goes on. Our solution has the following properties

- A deposit is claimed using only a single payment receipt, even in the case of many-to-one payments. This minimizes the interactions with the smart contract and makes the smart contract implementation simpler (avoiding this way possible implementation or application logic mistakes).
- Payment receipts are provided off-chain. Furthermore, the generation and validation of a receipt involves only the calculation of a digital signature, whereas the generation and evaluation of an one-time password

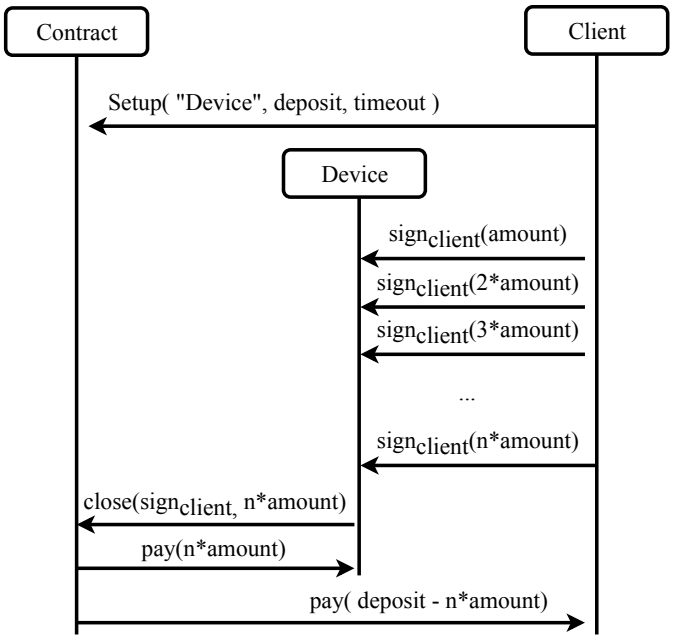

Fig. 1. A payment channel that can be used for off-chain payments.

involves the calculation of a keyed-hash message authentication code. Hence, this process is fast and therefore small time slots can be used (minimizing the losses in case of service disruption).

- A device and an AS have to be pre-configured with a shared secret key. No further interaction is required between these two entities.
- The communication channel between a client and a device does not have to be secured (as opposed to the communication channel between a client and an AS).
- Except from the validation of an one-time password, a device does not have to perform any other operation with respect to our solution.

III. System design

A. Notation

In our system an entity may have a public-private key pair. We denote the public key of an entity A with P_A. Furthermore, the digital signature of a message M using the private key of an entity A is denoted by $sign_A(M)$. By definition, given a public key P_X and a digital signature $sign_X(M)$, any entity (including a smart contract) can verify that the digital signature has been generated by the owner of P_X. Our system also relies on symmetric encryption and keyed-Hash Message Authentication Code (HMAC): the encryption of a message M using a (symmetric) key key is denoted by $E_{key}(M)$, whereas the HMAC of M using key is denoted by $H_{key}(M)$.

B. Preliminaries

Our system is based on two constructions, namely *payment channels* and distributed authorization using *HMAC-based one-time passwords*. In the following we introduce these two concepts.

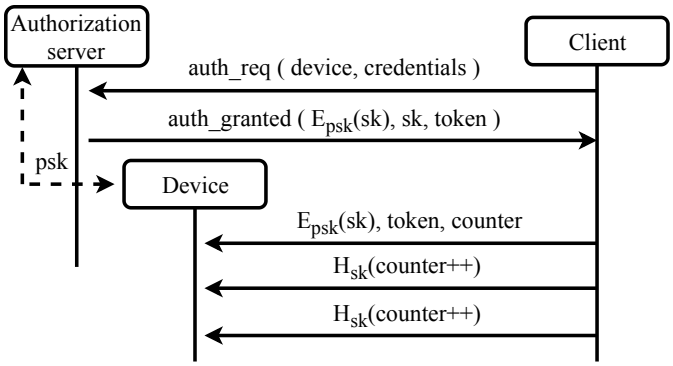

Fig. 2. Distributed authorization and hmac-based one-time passwords.

1) Payment channel: A payment channel is a construction that allows a client to make a "deposit" for a device to a smart contract. That device can claim (a portion of) this deposit by presenting an appropriate "payment receipt." Payment receipts are generated by the client and provided to the device off-chain. Furthermore, a payment channel includes a timeout after which a client can request his deposit back. In the following we present a "typical" smart contract-based payment-channel construction. This process is illustrated in Fig. 1. This construction assumes "powerful" devices, i.e., devices capable of interacting with a blockchain and trusted to maintain a blockchain wallet.

A payment channel is set up by a client who provides to the smart contract the public key of the device that can claim the deposit, the amount of the deposit, a nonce that will be included in the payment receipt, and a timeout. When the channel is set up, the corresponding amount of funds is transferred from the client to the smart contract. Afterwards, the client may send payment receipts to the device. A payment receipt is signed by the client and includes an amount and the channel-specific nonce. The device always keeps the latest valid payment receipt and accepts a new one only if (i) the amount included is bigger than the one in the latest receipt (but less than or equal to the deposit), (ii) the nonce and the signature are valid, and (iii) the payment channel is still "open".

A device can claim the amount included in the last receipt simply by sending it to the contract before the channel closes. Upon receiving a receipt, the contract transfers the corresponding amount to the device and the remainder of the deposit is transferred back to the client. At this point the payment channel closes. A smart contract has to verify that (i) the nonce included in the receipt is correct, (ii) the signature is valid, (iii) the signature has been generated by the client that created the channel, (iv) the amount included in the receipt is less than or equal to the deposit, and (v) the device that sent the receipt is owner of the public key specified by the client during the channel set up. Since the channel is closed after a payment and provided that a client uses a unique nonce for each channel, it is not possible to double spend a receipt.

2) Distributed authorization and HMAC-based one-time passwords: It is not uncommon for devices to not be capable or trusted to perform client authentication and/or authorization. In these cases, a distributed authorization protocol (such as OAuth 2.0 [1]) can be used. By using such a protocol, an

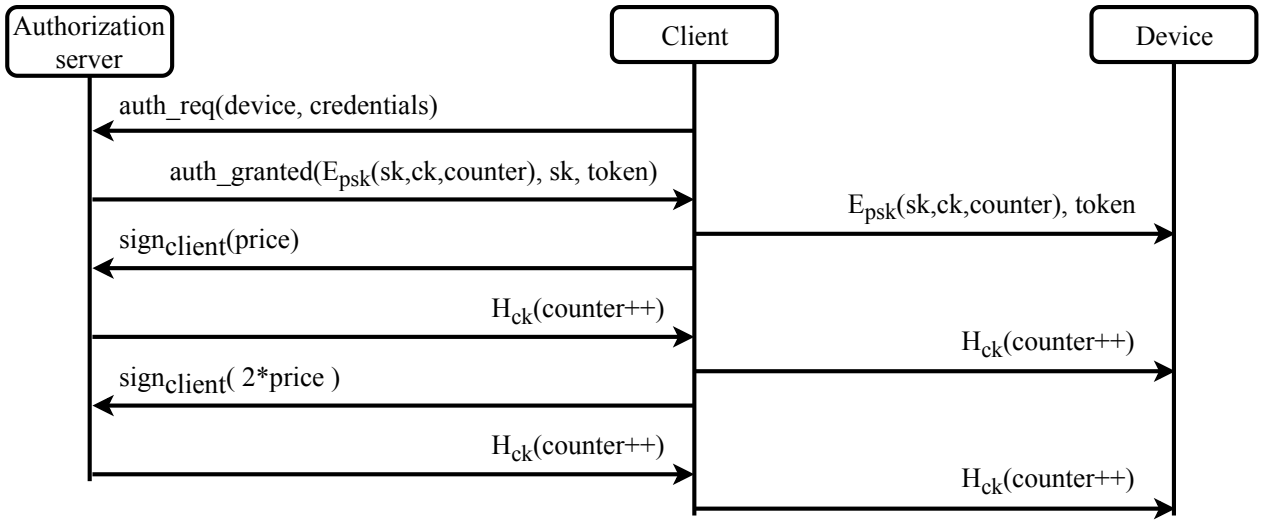

Fig. 3. Our simple construction.

authorization server (AS) authenticates and authorizes a client on behalf of a device and generates a token that can be used by the client for accessing the device. Furthermore, the AS can provide clients with keying material for generating "one-time passwords" (OTP) that can be used for subsequent interactions with the device. In the following, we give a high-level overview of a system that uses OAuth 2.0 based authorization and HMAC-based one-time passwords [2] (illustrated in Fig. 2).

In this setup, the communication channel between the device and the client is unprotected, whereas the communication channel between the AS and the client is secured (e.g., using TLS). Furthermore, the device and the AS have a pre-shared secret key, denoted by psk. A client requests authorization from an AS by providing its credentials and the identifier of the device it wants to access (e.g., a URL). If the client can be authorized, the AS generates a *token* (whose format is application specific) and a *session key* (sk). Then it sends the token, sk, and $E_{psk}(sk)$ back to the client. The client sends to the device $E_{psk}(sk)$, the token, and a *counter*. SK can be used for encrypting all subsequent messages. The OTP used for all subsequent messages is calculated simply by increasing the value of the counter by one and by calculating $H_{sk}(counter)$; note that the device can also calculate this value and validate the password.

IV. Constructions

We now describe our constructions. Remember that the device (i.e., the charging station in our use case) is not connected to the Internet and it is unaware of the existence of the blockchain.

A. Simple construction

In this section we consider the case of a single car (client) that uses the charging station. The car is also the blockchain wallet owner. The car establishes a payment channel indicating that the authorization server (AS) can claim the deposit (i.e., it includes the public key of the authorization server during channel set up). Then, the car proceeds with the authorization protocol described in the previous section. However, in our construction, we make the following modifications: in addition to the session key sk, the AS generates an additional key, i.e., the *counter key* (ck). Moreover, the AS encrypts ck and the $counter$ with the pre-shared key psk (note that only the device, i.e., the charging station, and the AS know this key). Finally, an OTP is generated by calculating $H_{ck}(counter)$. Hence, the car cannot generate the required OTPs by itself, instead it relies on the AS: in order for the AS to generate the next OTP the car must provide a valid payment receipt.

This construction is illustrated in Fig. 3. As it can be seen in this figure, the client exchanges two payment receipts for two OTPs, hence it uses the device for two time slots. The last payment receipt can be used by the AS to claim its money.

B. Payment delegation to a single entity

We now extend the previous construction and we consider a car (client) which is not trusted to have access to a blockchain wallet. Instead, its *owner* is responsible for setting up the payment channel and for making the initial deposit. However, the car should be able to generate valid payment receipts. For this reason, we consider that the car has a public-private key pair used for signing payment receipts. Furthermore, the public key of the car P_{car} is included in the payment channel setup. When the payment channel is closed, the contract, instead of verifying that the payment receipt has been signed by the entity that created the channel, it verifies the the receipt has been signed by the owner of P_{car}. All other operations are not modified.

C. Payment delegation to multiple entities

Our final construction enables many-to-one payments. In particular, it considers multiple clients, also not trusted to have access to a blockchain wallet, belonging to the same *owner*. In terms of our use case, this construction enables multiple cars owned by the same entity (this can be for example a car rental company) to use multiple charging stations, administered however by the same AS. This construction faces two challenges: (i) how to configure the payment channel with the public keys of the cars that are allowed to generate payment signatures,

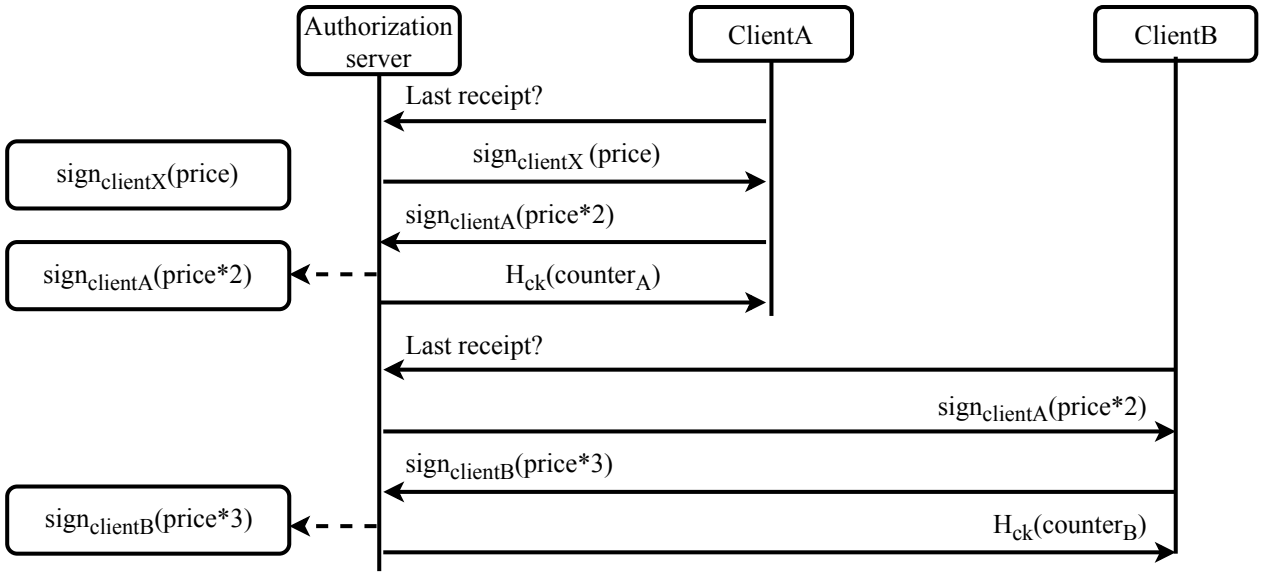

Fig. 4. Many-to-one payments.

and (ii) how to coordinate payment receipt generation among cars such that using only the last payment receipt the AS can close the payment channel.

The first challenge can be overcome using a *Merkle tree*. A Merkle tree is a binary tree in which every leaf node contains the hash of a public key (the public key of an authorized car in our use case) and every non-leaf node contains the hash of its children. Given the root of a Merkle tree and a *Merkle proof*, any entity can verify that (a hash of) a key is included in the tree in logarithmic time to the number of leaves. An entity can prove that its key is included in a tree by providing a Merkle proof, i.e., a list of the siblings of all nodes in the path from its (leaf) node to the root node (therefore, the number of nodes required in a Merkle proof is logarithmic to the number of leaves). With this construction, we extend the payment channel opening process to include the root of the Merkle tree (instead of a public key).

For the second challenge we define a simple query-response protocol that can be used by a client to retrieve from the AS the last valid payment receipt. We make the assumption that every new payment receipt increases the amount by a certain pre-defined value. In more detail, an AS always maintains the last valid payment receipt; prior to generating a receipt, a client queries the AS for the last valid receipt, the AS responds, the client validates the receipt, extracts the amount, and generates a new one adding to the amount the pre-defined value. The AS will accept this new receipt–and mark it as the last valid receipt–if (i) the payment channel is still open, (ii) the receipt is signed by a client whose public key is included in the Merkle tree, and (iii) it includes the amount of the last recorded receipt increased by the pre-defined value. In order for the AS to perform all these validations, the payment receipt must also include the Merkle proof. This protocol is illustrated in Fig. 4. In this example, initially the AS has stored a payment receipt signed by $Client_X$. For simplicity (i) we assume that all payment receipts are valid, and (ii) we omit the first step of the authorization process during which clients obtain $E_{psk}(sk, ck, counter)$. Then, $Client_A$ wishes to access the actuation device: it queries AS for the last receipt and the AS responds with the one signed by $Client_X$. $Client_A$ validates the receipt, creates a new one that includes a higher amount and sends it back to the AS. The AS accepts it, marks it as the last valid receipt and generates the OTP. The same process is repeated with $Client_B$. It should be noted that there can be cases where two (or more) clients request almost simultaneously the last valid payment receipt. In that case only one client will generate a valid payment receipt: the other receipts will be refused and the clients that generated them will have to re-execute the protocol.

With this construction, an AS can claim its money and close the payment channel by providing to the smart contact only the last valid payment receipt. Now the contract, instead of verifying that the payment receipt has been signed by the entity that created the channel, it verifies that it has been signed by a client whose public key is included in the Merkle tree. All other operations are not modified.

V. Implementation and evaluation

We have implemented the presented solution using Ethereum smart contracts.[2] We are using public-private key pairs that are constructed using the secp256k1 elliptic curve, i.e., the same elliptic curve used by Ethereum. As an HMAC function we have selected SHA256 which is well supported by various libraries and operating systems for constrained devices. Finally, for constructing the Merkle tree and since a Merkle proof has to be verified by a smart contract, we are using the keccak256 hash function, which is currently the recommended hash function for using inside Ethereum smart contracts. We have implemented a smart contract, which is deployed in a local testbed, that realizes the payment channels used in our three constructions.

The invocation of a smart contract operation creates some computational overhead to the blockchain network. In

[2]Source code of our implementation can be found at https://gitlab.com/mmlab-group/projects/ndss-diss-2019

Ethereum, this overhead is measured in "gas" units: the amount of gas "consumed" by an operation depends on the operation's complexity. Furthermore, a user that invokes a smart contract operation pays a transaction fee which is calculated based on the consumed gas and the price per gas unit. Currently, the average price of a gas unit is[3] $\$0.012 \times 10^{-4}$. Table 1 below illustrates the cost, measured in Ethereum gas, for invoking each channel operation. The calculation of the cost of the "close" operation of our second construction considers a two-level Merkle tree (i.e., four cars); that cost is proportional to the depth d of the used Merkle tree. In particular the cost of that function can be calculated using the following formula $36258+d\times36$, where 36258 is the cost for verifying a payment receipt and 36 is the cost of calculating a hash function.

First construction	
Operation	**Cost measured in gas**
open channel	43700
close channel	36258
Second construction	
Operation	**Cost measured in gas**
open channel	50388
close channel	36258
Third construction	
Operation	**Cost measured in gas**
open channel	50388
close channel	36330

TABLE I. COST FOR INVOKING SMART CONTRACT OPENING AND CLOSING PAYMENTS CHANNELS

In order for client to start producing payment receipts its deposit must be "recorded" in the blockchain. Currently, this operation in the public Ethereum blockchain requires in average 14.51 sec.[4]

Endpoints are implemented using JavaScript. Interactions with the Ethereum blockchain, digital signatures using the secp256k1 elliptic curve, and hash calculations are implemented using the Ethereum JavaScript API.[5]

VI. CONCLUSION AND FUTURE WORK

This paper presented a solution that realizes micro-payments for actuation services minimizing user loses in case of service disruption. This solution is based on smart contracts and blockchains. Smart contracts are used because they support features required to solve this particular problem, that cannot be provided (at least easily) by traditional means of payment. In particular, using blockchains and smart contracts it is possible (i) to deposit an amount of money that can be claimed by a third party, and (ii) to authorize a user who has no relationship with that particular blockchain technology, to spend this amount for any purpose. Our system leverages existing technologies and makes realistic assumptions about the capabilities of the IoT devices. Furthermore, by hiding all blockchain-specific operations from the IoT devices, we believe that existing authorization systems (e.g., OAuth 2.0-based systems) can be relatively easily extended to support our solution.

The proposed solution utilizes the Ethereum blockchain, where minimizing the complexity of smart contracts is very important. A "permissioned" blockchain (e.g., Hyperledger Fabric [13]) would significantly decrease the cost of blockchain transactions and smart contracts executions. We are currently considering this alternative. Another topic for future work is the analysis of how more complex smart contracts could enhance the system, i.e., in terms of security, flexibility, or reducing the need for network communication. Finally, many-to-many payments can be realized, i.e., multiple ASes accepting payments from multiple clients. In that case, ASes may maintain a ledger, for keeping up with the payments, and use an inter-ledger protocol (e.g., ILP [14]) for claiming their money.

[3]As measured by https://ethgasstation.info on 21 Dec. 2018

[4]As measured by https://etherscan.io/chart/blocktime on 21 Dec. 2018

[5]https://github.com/ethereum/web3.js/

ACKNOWLEDGMENT

The research reported here 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] D. Hardt (ed.), "The OAuth 2.0 authorization framework," IETF, RFC 6749, 2012.

[2] D. M'Raihi (ed.), "HOTP: An hmac-based one-time password algorithm," IETF, RFC 4226, 2005.

[3] J. Cohn, P. Finn, S. Nair, and P. Sanjai, "Device democracy: Saving the future of the Internet of Things," IBM Institute for Business Value, 2014, (last accessed 30 Aug. 2018). [Online]. Available: http://www-01.ibm.com/common/ssi/cgi-bin/ssialias?htmlfid=GBE03620USEN

[4] G. C. Polyzos and N. Fotiou, "Blockchain-assisted information distribution for the Internet of Things," in *Proceedings of the 2017 IEEE International Conference on Information Reuse and Integration*, 2017, pp. 75–78.

[5] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum Project Yellow Paper*, vol. 151, 2014.

[6] T. Lundqvist, A. de Blanche, and H. R. H. Andersson, "Thing-to-thing electricity micro payments using blockchain technology," in *2017 Global Internet of Things Summit (GIoTS)*, June 2017, pp. 1–6.

[7] H. Huang, X. Chen, Q. Wu, X. Huang, and J. Shen, "Bitcoin-based fair payments for outsourcing computations of fog devices," *Future Generation Computer Systems*, vol. 78, pp. 850 – 858, 2018.

[8] Y. Zhang, R. H. Deng, X. Liu, and D. Zheng, "Blockchain based efficient and robust fair payment for outsourcing services in cloud computing," *Information Sciences*, vol. 462, pp. 262 – 277, 2018.

[9] M. Król and I. Psaras, "SPOC: secure payments for outsourced computations," in *NDSS 2018 Workshop on Decentralised IoT Security and Standards(DISS)*, 2018.

[10] J. Lind, I. Eyal, P. R. Pietzuch, and E. G. Sirer, "Teechan: Payment channels using trusted execution environments," *CoRR*, vol. abs/1612.07766, 2016. [Online]. Available: http://arxiv.org/abs/1612.07766

[11] I. Bentov, Y. Ji, F. Zhang, Y. Li, X. Zhao, L. Breidenbach, P. Daian, and A. Juels, "Tesseract: Real-time cryptocurrency exchange using trusted hardware," Cryptology ePrint Archive, Report 2017/1153, 2017, https://eprint.iacr.org/2017/1153.

[12] N. Fotiou, V. A. Siris, and G. C. Polyzos, "Interacting with the Internet of Things using smart contracts and blockchain technologies," in *Proc. of the 7th International Symposium on Security and Privacy on Internet of Things (SPIoT 2018)*, 2018.

[13] "Hyperledger fabric home page," The Linux Foundation, 2018, (last accessed 20 Dec. 2018). [Online]. Available: https://www.hyperledger.org/projects/fabric

[14] "Interledger protocol," W3C, 2018, (last accessed 20 Dec. 2018). [Online]. Available: https://interledger.org/