IoT Resource Access utilizing Blockchains and Trusted Execution Environments

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 consider an IoT resource with a Trusted Execution Environment (TEE) and propose a model to provide trusted resource access that is linked to blockchain payments, ensuring the integrity and confidentiality of the IoT data. The model is built on the widely used OAuth 2.0 open authorization framework, which provides delegated authorization for IoT resources. We utilize hash-lock and time-lock mechanisms to cryptographically link trusted resource access, provided by the IoT resource's TEE, to authorization grants and blockchain payments. The model is implemented in the OP-TEE open source port for the Raspberry Pi that uses ARM's TrustZone and is evaluated on the Rinkeby public Ethereum testnet.**

***Index Terms*—delegated authorization, distributed ledgers, Trusted Execution Environment (TEE), hash-locks and time-locks**

I. Introduction

Blockchains provide a decentralized trust system for executing transactions and code. However, trust is only ensured for data and code confined within the limits of a blockchain and does not extend to the blockchain's interaction with the real world, e.g., calling outside APIs or obtaining external data. Moreover, a blockchain does not provide confidentiality for the data it records, since the data is replicated on all blockchain nodes; however, such replication provides high availability and transparency, which are important advantages of blockchain systems. Finally, public blockchains based on Proof-of-Work (PoW) consensus have a high transaction delay, high energy consumption, and limited scalability.

Trusted Execution Environments (TEEs) provide a secure environment for executing code and storing data. A TEE runs in isolation and in parallel to the normal (or "rich") operating system, ensuring the confidentiality and integrity of code and data. However, because a TEE runs on a single device, it cannot provide high availability nor decentralized trust. Combining blockchains and TEEs can enable systems with the complementary gains of both: decentralized trust and high availability from blockchains on one hand, and privacy and trust when interacting with the real world through TEEs on the other hand. Moreover, combining TEEs from different manufacturers with blockchain technology can reduce the reliance on manufacturers building trustworthy TEE hardware, which has been a key criticism of TEEs. Interestingly, combining TEEs with blockchains can potentially provide an environment for transparently detecting and creating immutable proofs of manufacturers that build untrustworthy hardware. In any case, care must be taken when combining TEEs and blockchains such that their combination does not cancel each other's advantages [1], [2], [3].

Blockchains and TEEs can be combined following two directions. In the first, TEEs can be used to execute blockchain functionality, such as transaction verification and decentralized consensus; using TEEs can improve the privacy and trust of a single node, which in turn can help increase the overall efficiency and scalability of the blockchain system. This is the direction investigated in [2], [3], [4], [5], [6], [7], [8], and by Intel's Microsoft's Coco framework [9]. The second direction utilizes TEEs to provide trust and privacy when blockchains interact with the real world. For example, [10], [11], AnyLedger [12], and Weeve [13] investigate TEEs for developing secure wallets. The works in [14], [15] and Weeve [13] use TEEs for ensuring the integrity and privacy of IoT measurements. Another line of work proposes an *overlay* of oracles that utilizes TEEs for enabling blockchains to interact with the real world in a trusted manner [16] or for generating random numbers [17]. The works in [18], [19] use TEEs for improving off-chain transactions. We further discuss related work and identify how it differs from the work in this paper in Section V.

Our goal is to investigate how TEEs in IoT devices can be utilized together with a blockchain to provide delegated authorization and trustful access to IoT resources. The proposed model utilizes hash-lock and time-lock mechanisms for cryptographically linking blockchain payments and authorization grants, while the IoT device's TEE ensures access to the resource if the linked payment is performed. Hash-lock and time-lock mechanisms are interledger or *cross-chain* mechanisms [20], [21]; hence, a TEE can be viewed as a *trusted local ledger*, and interledger mechanisms are used to bind transactions and capabilities across blockchains, distributed ledgers, and TEEs; see Figure 1.

The OAuth 2.0 delegated authorization framework is a widely used IETF standard that is currently being investigated for authorization in IoT environments by IETF's Authentication and Authorization for Constrained Environments (ACE) Working Group [22], [23]. The OAuth 2.0 framework

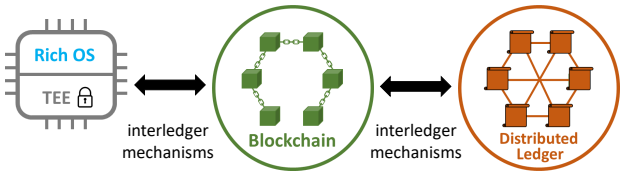

Fig. 1. Hash-lock and time-lock interledger mechanisms can be used for linking transactions across blockchains, distributed ledgers, and TEEs, which can be viewed as *trusted local ledgers*.

can also be used in cases where centralized management of authorization policies is advantageous. 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 approach presented in this paper for exploiting TEEs with blockchains can be applied to the general context of IoT resource access authorization.

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

- We present a model that combines TEEs and blockchains using hash-locks and time-locks to cryptographically link authorization grants to payments, and ensure access to IoT resources.
- We implement the model using the OP-TEE port for the Raspeberry Pi, which uses ARM's TrustZone, and evaluate it using the Rinkeby public Ethereum testnet.

The remainder of the paper is structured as follows: In Section II we present some background on delegated authorization using OAuth 2.0, hash and time-locks, and ARM's TrustZone TEE. In Section III we present a model that combines TEEs and blockchains using hash and time-locks, and in Section IV we present its implementation and evaluation. Finally, in Section V we present related work and in Section VI we present conclusions and our ongoing work.

II. Background

In this section we present some background on the OAuth 2.0 delegated authorization framework, hash-lock and time-lock mechanisms, and ARM's TrustZone TEE.

A. OAuth 2.0 delegated authorization

OAuth 2.0 is a framework for delegated authorization to access a protected resource [24]. It enables a third party application (client) to obtain access with specific permissions to a resource, with the consent of the resource owner. Access to the resource is achieved through *access tokens*, created by an authorization server. The specific format of the access tokens, which will be discussed in more detail later, is opaque to the clients and to OAuth 2.0. The authorization consent by the resource owner is provided after the owner is authenticated; however, the authentication procedure is not part of OAuth 2.0. Authorization is provided for different levels of access, such as read and write/modify, 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 authorization server. 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 authorization server's response containing the access token.

One type of access tokens are *bearer tokens*. Bearer tokens allow the holder (bearer) of the token, independently of its identity, to access the protected resource. OAuth 2.0 assumes secure communication between the different entities. Moreover, it assumes that the protected resource is always connected to the Internet, hence can communicate with the authorization server to check the validity and scope of the access tokens presented by clients requesting resource access. Both of the above two requirements are not always possible in constrained environments [22].

JSON Web Token (JWT) is an open standard that defines a compact format to transmit claims between parties as a JSON object [25]. JWTs can use the JSON Web Signature (JWS) structure to digitally sign or integrity protect claim with a Message Authentication Code (MAC) [26]. 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 authorization server. Of course, this requires that during its initialization phase the protected resource is cryptographically bound with the authorization server.

In constrained environments, in addition to intermittent or no connectivity, the communication between the client and the protected resource is not secure, hence transmitting bearer tokens or even self-contained JWTs over such insecure links can allow other parties to obtain them through eavesdropping. For this reason in constrained environments Proof-of-Possession (PoP) tokens are used [23]. PoP tokens include a normal access token, such as a JWT/JWS, and a PoP key [27]: access to the protected resource is not possible solely with the access token; the PoP key is necessary. Hence, the PoP key must be kept secret and not transmitted in cleartext over insecure links. Finally, more efficient encoding of JWTs based on CBOR (Concise Binary Object Representation) is proposed to reduce the amount of data transferred [23].

B. Hash-locks and time-locks

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 key that decrypts 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 one on the other chain: if the two transactions have hash-locks with the same value, then unlocking one would reveal the secret that unlocks the other; hence, the two transactions are cryptographically linked through a dependence relation.

Time-locks are locks on a blockchain that can be unlocked only after an interval has elapsed. The time 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 transfer an amount of currency (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 such indefinite locking of funds, 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. The above example shows how time-locks can be used to allow some functionality only after some time interval has passed.

Contracts that include both hash and time-locks are referred to as hashed time-lock contracts (HTLCs) [20]. HTLCs can be implemented in blockchains with simple scripting capabilities, such as the Bitcoin blockchain, without requiring the advanced functionality of smart contracts.

C. Trusted Execution Environment and ARM TrustZone

TEEs provide a secure environment for executing code and storing data. A TEE runs in isolation and in parallel to the normal (or "rich") operating system (OS), ensuring the confidentiality and integrity of code and data. The TEE system architecture is defined in [28] by GlobalPlatform, a non-profit industrial association. An important feature of TEEs is that they can provide privileged and secure access to peripherals. Device peripherals can be hardware-isolated from the rich OS and controlled only by the TEE, thus malware running in the rich OS cannot access those peripherals.

ARM TrustZone is one example of a TEE following a System on a Chip (SoC) approach. ARM's TrustZone introduces a special *secure* CPU mode alongside the regular (or normal) mode, thereby establishing the notions of a "secure world" and a "normal world". When secure mode is active, the software running on the CPU has a different view of the whole system than software running in non-secure mode. In this way, system functions, in particular security functions and cryptographic credentials, can be hidden from the normal world. ARM's TrustZone functionality is defined by the software running in secure mode, which can range from a slave process to a full operating system, hence TrustZone is significantly more flexible than TPM chips whose functionality is hard-wired. TrustZone differs from Intel's Software Guard eXtension (SGX), which is a hardware extension on Intel CPUs, in that the latter supports multiple secure enclaves in the same system, while TrustZone supports only one secure world. AMD and Nvidia also have products supporting TEE. Other related work includes the Keystone open-source project[1] for building TEEs with secure hardware enclaves based on the RISC-V architecture. ARM's TrustZone is widely deployed on mobile devices such as smartphones and micro-controller devices.

[1] https://keystone-enclave.org/

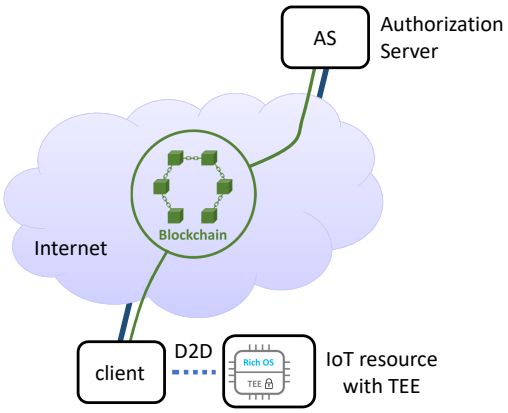

Fig. 2. High-level architecture for delegated authorization exploiting an IoT resource's Trusted Execution Environment (TEE).

III. Delegated Authorization Utilizing Blockchains and TEEs

The high-level architecture of the proposed model is shown in Figure 2. Authorization for IoT resources is outsourced to an authorization server (AS), which can provide authorization for multiple IoT resources. Resource access is provided by a device with a TEE, which supports the integrity and confidentiality. Depending on the specific type of TEE technology, different restrictions can exist. For example, ARM's TrustZone supports a single secure enclave (secure world), whereas Intel's SGX can support multiple enclaves.

Figure 2 shows that the client device and the AS interact with the blockchain, whereas the IoT resource does not have continuous network connectivity. The client accesses the IoT resource directly using device-to-device communication. Moreover, because the device-to-device link is insecure, the client and IoT resource need to establish a shared secret key to secure their direct link; this is achieved using PoP tokens that were discussed in Section II-A. Note that remote attestation of the IoT resource can still be performed in periods where the IoT resource has network connectivity. Alternatively, remote attestation can be performed *on-demand*, using the client as an intermediate node, similar to how the client is the intermediate node between the IoT resource and the AS for the authorization procedure described below.

We assume that the client, the AS, and the resource owner, have an account (public/private key pair) on the blockchain. The client will use his account to pay for accessing the IoT resource. A client's deposit, assuming the authorization procedure is smoothly completed, will be transferred to the resource owner's account. Finally, the AS has an account to send transactions in order to set up a Hashed Time-Lock Contract (HTLC) as we discuss below.

Figure 3 shows the messages exchanged among the client, the AS, and the IoT resource. We assume that service discovery during which the IoT resource is discovered and

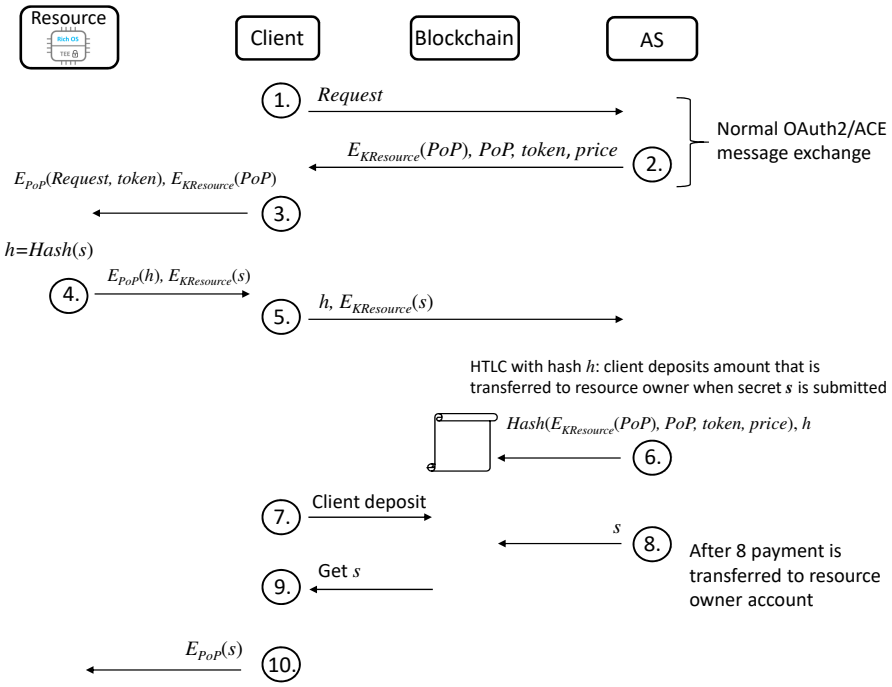

Fig. 3. Message exchange for access to the IoT resource with a TEE. The Trusted Application running in the TEE is responsible for generating the secret s, computing its hash $h = Hash(s)$, and ensure that the client provides the true secret to obtain resource access (Step 10).

discovery of the AS that handles authorization requests for the IoT resource has already occurred and is not shown in Figure 3. The AS that handles the authorization requests can be discovered by sending an initial unauthorized resource request message to the IoT resource [23] or through a QR code on the IoT device. Steps 1 and 2 include the normal OAuth 2.0 message exchange between the client and the AS. According to OAuth 2.0, the communication between the client and the AS is secured using TLS, hence the information exchanged in Steps 1 and 2 are secured. After these two steps, the client has obtained the PoP key with which it can establish a secure link with the IoT resource. This is possible since the client also receives from the AS and forwards to the IoT device over the device-to-device connection the PoP key encrypted with the secret key $K_{Resource}$ that the IoT resource shares with the AS; sharing of the secret key in the IoT resource can be performed during the resource's initialization. In Step 3, the client forwards the encrypted PoP key to the IoT resource along with the access token. The above procedure is followed because in the scenario shown in Figure 3, the IoT resource is disconnected. On the other hand, if the IoT resource had continuous network connectivity, then it could alternatively obtain the PoP key directly from the AS. As in Figure 3, interaction of the client and the IoT resource can still be performed using device-to-device communication, even if the IoT resource had continuous network connectivity.

When the IoT device receives the access token, it verifies its validity. If the access token is formatted as a signed JWT, then this verification involves checking the signature included in the JWT token. Alternatively, if the IoT resource had continuous network connectivity, then it could use introspection to communicate with the AS in order to verify the access token and the corresponding access rights that it allows [23].

After verifying the access token, the IoT resource generates a secret s and computes its hash $h = Hash(s)$, which will be used in the hash-lock of the payment contract. In Step 4, over the secure communication channel it established with the client, the IoT resource sends the hash h and the secret s encrypted with the secret key $K_{Resource}$ that the IoT resource shares with the AS. After receiving the hash h and $E_{K_{Resource}}(s)$, the client forwards both to the AS in Step 5. In Step 6, the AS creates a hashed time-lock payment on the blockchain. The hash-lock is the same h that the AS received from the resource through the client, while the price is what the AS sent to the client in Step 2. Additionally, in Step 6 the hash of the authorization information the AS sent to the client in Step 2 is also submitted to the blockchain; in the case of disputes, this hash is a non-repudiated receipt of the information that the AS sent to the client.

Note that after Step 6, due to the transparency of transactions stored on the blockchain, the client can verify that the payment contract hash is the same as the value h that it had received from the IoT resource. Additionally, the payment contract also has a time-lock, i.e. the actions we discuss next such as the client deposit and AS revealing the secret s occur within a maximum time interval; the time-lock allows either parties (client or AS) to abort the procedure, if the other party has delayed taking the action on its part. hence, there is no need for trust between the client and the AS.

The client deposits the price for resource access to the blockchain contract in Step 7. The deposit is not transferred directly to the IoT resource owner's account, but is transferred

to the payment contract's account which acts as an escrow service. The deposit is transferred to the resource owner's account only when the AS reveals to the contract the secret s that unlocks the hash-lock; note that it is necessary that the hash function algorithm used at the IoT resource for computing the hash is the same as the hash function in the blockchain contract. If the AS does not reveal the secret s within the time defined by the contract's time-lock, then the client can submit a request for the deposit to be returned to the client's account; hence, by jointly using hash-locks and time-locks, the payment contract is a Hashed Time-Lock Contract (HTLC) [20] and the two actions, client deposit and AS revealing the secret s, either both happen or neither of the two happens, i.e., they are *atomic*. If the above events occur smoothly, then the secret s would be revealed on the blockchain. Hence, the client can obtain the secret s in Step 9 and send it to the IoT resource in order to obtain access (Step 10). Note that once the secret s is revealed on the blockchain anyone can obtain it; however, the secret alone is not enough to gain access to the IoT resource since both the access token and the PoP key are also required.

Since the secret and hash were produced in the IoT resource's TEE and access to the resource is also provided through the TEE, knowledge of the secret s, together with the access token and PoP key, ensure that the client can access the IoT resource according to the scope defined in the access token. From a high-level perspective, the TEE can be viewed as a *trusted local ledger*. The proposed model uses hash-locks and time-locks, which are *interledger* mechanisms that enable *atomic cross-chain trading* or *atomic swaps* [21], to cryptographically bind authorization grants and access through a TEE with blockchain payments. Using the same interledger mechanisms for cryptographically linking transactions on different blockchains, distributed ledgers, and TEEs, which are viewed as a local trusted ledger, allow for simplicity that can provide higher security and efficiency.

IV. Evaluation

Our implementation of the IoT resource is based on the OP-TEE (Open Portable Trusted Execution Environment) open source port for the Raspberry Pi[2], which uses ARM's TrustZone. OP-TEE follows the GlobalPlatform TEE system architecture [28]. The secure world TEE runs the Linux OP-TEE operating system. The module providing the IoT resource access runs as a Trusted Application in the OP-TEE OS and performs the security critical operations that include generating the secret s, computing its hash $h = Hash(s)$, and verifying that the client provides the true secret s to obtain resource access. Note that the module for providing the IoT resource's service is also executed in the TEE; this ensures[3] that the IoT resource will provide the intended service to the client, ensuring the IoT data's integrity and confidentiality, as long as the client provides the true secret s. A Client Application running in the normal world uses GlobalPlatform's TEE client API to communicate with the Trusted Application.

[2]https://www.op-tee.org/docs/rpi3/

[3]The degree to which this is guaranteed depends on the frequency/time that attestation is performed.

TABLE I
Gas and delay
Gas price=2.5 Gwei. 1 Gwei=$\$1.1 \cdot 10^{-7}$ on Feb 2, 2019.

Transaction	Gas (cost in $)	Delay in seconds
Contract creation	473 508 (0.130)	-
Set payment information (Step 6)	32 231 (0.009)	13.6
Client deposit (Step 7)	28 287 (0.008)	14.9
Send secret s (Step 8)	41 949 (0.012)	15.0

For the evaluation we have deployed a local Ethereum node running Go-Ethereum[4] connected to the Rinkeby public Ethereum testnet[5]. The smart contract implementing the transaction logic were written in Solidity with the Remix server[6]. The authorization server was based on the OAuth 2.0 server software[7], which is written in Php. The client used Web3.js, which is a part of Node.js.

Table I shows the gas, which quantifies the amount of EVM (Ethereum Virtual Machine) resources used, for the contract creation and the three transactions in Steps 6, 7, and 8 of Figure 3. The transactions are submitted with gas price 2.5 Gwei. The creation of the contract has the largest gas cost, which is one order of magnitude larger than the cost of the three transactions. Step 8 has higher gas than Steps 6 and 7 because it includes checking that the secret s submitted by the AS satisfies $h = Hash(s)$. Also, Step 6 has higher gas than Step 7 because the former submits two hashes to the blockchain, while Step 7 submits only the deposit, Figure 3.

The transaction delay shown in Table I are the average values from 20 executions. The 95% confidence interval was smaller than ± 0.7 seconds of the averages shown. Both the average transaction delay and its standard deviation depend on the gas price: using a gas price smaller than 2.5 Gwei would result in both a higher average delay and a higher standard deviation. As expected, the blockchain transaction delay is significantly higher than the delay when the client interacts with the AS, Steps 1 and 2 in Figure 3, and when the client interacts with the IoT resource, Steps 3 and 4, which are both less than 0.08 seconds.

V. Related work

Teechain [19] is an off-chain payment network that uses TEEs to secure funds in payment channels, without requiring synchronous access to the blockchain as in the case of payment channels between devices without TEEs. The work in [18] utilizes TEEs to link off-chain payments with outsourced computations. Specifically, a client requesting outsourced computations verifies the attestation from the TEE of the executing node before providing an off-chain payment channel update. With this approach the attestation functionality is bundled with the payment channel functionality. On the other hand, the approach in this paper keeps the two functions distinct, which yields a simpler design, while also allowing

[4]https://geth.ethereum.org/

[5]https://www.rinkeby.io/

[6]https://remix.ethereum.org/

[7]https://github.com/bshaffer/oauth2-server-php

each to be modified independently without affecting the other. Furthermore, the functions can be performed by different entities, which offers greater flexibility.

Prior work has investigated TEEs for data privacy and integrity. In particular, [10] and [11] investigates TrustZone for protecting private keys and blockchain headers that are necessary for verifying transactions in a lightweight blockchain wallet. AnyLedger [12] and Weeve [13] have a similar target. The works in [14], [15], [13] consider TEEs for ensuring the integrity and privacy of IoT measurements. In [14], a verifier performs remote attestation to ensure the trustworthiness of IoT measurements utilizing ARM's TrustZone. Another line of work proposes an overlay of oracles that utilizes TEEs for allowing blockchains to interact with the real world [16] or for generating random number [17]. The work in [2], [5], [6] considers an overlay of hardware enclave to execute and ensure the confidentiality of smart contracts, while ensuring integrity and availability based on existing public blockchains like Ethereum. Related work that utilizes TEEs and blockchain functionality, such as transaction verification and decentralized consensus, essentially fusing TEE and blockchain technology, is investigated in [4], [7], [8] and by Microsoft's Coco framework [9]. Unlike the above, this paper uses hash and timelock mechanisms for cryptographically linking transactions and functionality in TEEs and blockchains.

Finally, other works such as [29], [30] investigate blockchain-based authorization without however considering TEEs. This is also the case for our previous paper [31] which investigates authorization based on smart contracts and [32] which investigates micropayments for IoT resource access.

VI. Conclusions and future work

We have presented a model for utilizing a TEE combined with blockchains to provide IoT resource access with a higher level of trust that the IoT resource will provide the intended access that is linked to the corresponding payment, ensuring the integrity and confidentiality of the IoT data. The implementation of the model uses the OAuth 2.0 authorization framework and the OP-TEE port to Raspberry Pi. Our ongoing work is investigating using different ledgers for authorizations and payments, and utilizing off-chain transactions for resource access in constrained IoT environments.

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] M. Brandenburger and C. Cachin, "Challenges for Combining Smart Contracts with Trusted Computing," in *Proc. of 3rd Workshop on System Software for Trusted Execution (SysTEX)*, 2018.

[2] R. Cheng et al., "Ekiden: A Platform for Confidentiality - Preserving, Trustworthy, and Performant Smart Contracts," arXiv:1804.05141v5, September 2018.

[3] H. Dang, A. Dinh, E.-C. Chang, and B. C. Ooi, "Chain of Trust: Can Trusted Hardware Help Scaling Blockchains?" arXiv:1804.00399v3, August 2018.

[4] Hyperledger Sawtooth, "Proof of Elapsed Time (PoET) Specification," 2016. https://sawtooth.hyperledger.org/docs/core/releases/1.0/architecture/poet.html

[5] M. Bowman, A. Miele, M. Steiner, and B. Vavala, "Private Data Objects: an Overview," arXiv:1807.05686v2, November 2018.

[6] R. Yuan et al., "ShadowEth: Private Smart Contract on Public Blockchain," *Journal of Computer Science and Technology*, vol. 33, no. 3, pp. 542–556, May 2018.

[7] Trias, "Trustworthy and Reliable Intelligent Autonomous Systems," White Paper Version 1.0, 2018. https://www.trias.one/whitepaper

[8] M. Ahmed and K. Kostiainen, "Don't Mine, Wait in Line: Fair and Efficient Blockchain Consensus with Robust Round Robin," arXiv:1804.07391v2, January 2019.

[9] Microsoft, "The Confidential Consortium Blockchain Framework: Technical Overview," August 2017.

[10] M. Gentilal, P. Martins, and L. Sousa, "TrustZone-backed bitcoin wallet," in *Proc. of Fourth Workshop on Cryptography and Security in Computing Systems (CS2)*, 2017.

[11] W. Dai et al., "SBLWT: A Secure Blockchain Lightweight Wallet Based on Trustzone," *IEEE Access*, vol. 6, pp. 40 638–40 648, August 2018.

[12] B. Djukic and L. Pieri, "AnyLedger: Embedded wallet for decentralized IoT," 2018. www.anyledger.io/whitepaperAnyLedger.pdf

[13] M. Davidsen, S. Gajek, M. Kruse, and S. Thomsen, "Empowering the Economy of Things," 2018. https://weeve.network/weeve_whitepaper.pdf

[14] J. Park and K. Kim, "TM-Coin: Trustworthy Management of TCB Measurements in IoT," in *Proc. of IEEE PERCOM Workshop On Security Privacy And Trust In The Internet of Things*, 2017.

[15] G. Ayoade, V. Karande, L. Khan, and K. Hamlen, "Decentralized IoT Data Management Using BlockChain and Trusted Execution Environment," in *Proc. of IEEE International Conference on Information Reuse and Integration for Data Science*, 2018.

[16] F. Zhang et al., "Town Crier: An Authenticated Data Feed for Smart Contracts," in *Proc. of ACM SIGSAC Conference on Computer and Communications Security*, 2016.

[17] Oraclize, "A Scalable Architecture for On-Demand, Untrusted Delivery of Entropy," 2017. https://www.oraclize.it/papers/random_datasource-rev1.pdf

[18] M. Al-Bassam et al., "Airtnt: Fair Exchange Payment for Outsourced Secure Enclave Computations," arXiv:1805.06411, May 2018.

[19] J. Lind et al., "Teechain: Scalable Blockchain Payments using Trusted Execution Environments," arXiv:1707.05454v2, December 2018.

[20] Bitcoin Wiki, "Hashed Timelock Contracts (HTLC)," https://en.bitcoinwiki.org/wiki/Hashed_Timelock_Contracts

[21] ——, "Atomic cross-chain trading," https://en.bitcoinwiki.org/wiki/Atomic_cross-chain_trading

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

[23] ——, "Authentication and Authorization for Constrained Environments (ACE) using the OAuth 2.0 Framework (ACE-OAuth)," IETF Draft, January 31, 2019.

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

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

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

[27] M. Jones, J. Bradley, and H. Tschofenig, "Proof-of-Possession Key Semantics for JSON Web Tokens (JWTs)," RFC 7800, Standards Track, IETF, April 2016.

[28] GlobalPlatform, "TEE System Architecture v1.2," December 2018. https://globalplatform.org/specs-library/tee-system-architecture-v1-2/

[29] 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.

[30] D. D. F. Maesa, P. Mori, and L. Ricci, "Blockchain based access control," in *Proc. of 17th IFIP DAIS*, 2017.

[31] 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.

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