Decentralized Beacons: Attesting the Ground Truth of Blockchain State for Constrained IoT Devices

Santeri Paavolainen*† and Pekka Nikander*

***Abstract*—Internet of Thing devices (IoT devices) are often constrained in terms of computing, memory, storage, power, and network resources. This makes them ill-suited to operate as first-class citizens on a blockchain, such as Ethereum, preventing the IoT devices from attaining the security guarantees that are available to better resourced nodes that are able to operate as full, validating nodes on the blockchain. IoT devices may use so-called light protocols to interact with the blockchain with minimal resource requirements, but these protocols provide only probabilistic security guarantees. In this position paper, we propose a new mechanism where an operator of IoT devices is able to send a "ground truth state" to the devices via a new mechanism, which we call "decentralized beacons", enabling them to gain full security guarantees of the blockchain state.**

***Index Terms*—Internet of Things; Blockchain; Ethereum**

I. Introduction

While the initial use of blockchains was primarily to transfer cryptocurrency between accounts on the blockchain network [1], the potential benefit of blockchains for Internet-of-Things (IoT) devices has since became apparent [2]. Given the large variability of computing, memory, storage, power and network capabilities of IoT devices, several different integration models have been identified [3]. These each offer differing security guarantees for the devices. From the security point of view, the most desirable situation for an IoT device would be to operate as a *full node*, as this gives the best security properties; however, this is often out of the reach of constrained IoT devices. For example, to participate in the Ethereum blockchain [4] as a full node requires several gigabytes of storage to maintain the blockchain state [5]. In practice, the two realistic blockchain integration models for constrained IoT devices are to either *rely on a gateway* and/or trusted server, or to operate as a *thin client*.

A thin blockchain client — also known as a light client — utilizes a blockchain-specific *light protocol*. For Ethereum, this is the Light Ethereum Subprotocol (LES) [6]. The core idea of a light protocol is to both minimize the bandwidth requirements and to provide cryptographic proofs of the consistency of the data, which the light protocol client can independently verify. However, it is well-known that light protocols are able to provide only probabilistic security guarantees, as the cryptographic proofs are able to only show that the provided data is internally consistent; it is unable to provide a global consistency guarantee. For example, LES cannot prove that the state a light client is receiving is really the "true" blockchain state as viewed by the majority of the network nodes.

* Department of Communications and Networking, School of Electrical Engineering, Aalto University, Finland (email: first.last@aalto.fi)
† LMF Ericsson, Finland (email: first.last@ericsson.com)

The recommended behavior for light clients is to contact many light protocol servers over the Internet and try to determine the prevailing consensus with the assumption that with a sufficiently large number of connections, the majority of the connected nodes are honest. However, IoT devices are often constrained also by their network connectivity, meaning that they have to operate under a threat model where *all network traffic* may be intercepted and controlled by a single adversary. Under this model, such an adversary has full control of the visibility of the light protocol servers on the Internet. Therefore, an adversary can masquerade as any number of such servers. Another solution to the issue of attaining confirmation on the global blockchain state is to rely on human intervention, e.g. with the human independently cross-checking the root block hash from independent sources such as Etherscan.[1] For IoT devices deployed at scale such human intervention is unfortunately not economically nor operationally feasible.

There is a strong trend to add networking capabilities even to the lowest-cost devices, where the low cost of the device sets limits on computational and storage resources such devices can afford. Consequently, while technological development can increase the capacity of a device *at a fixed price*, this is often counteracted by the desire to instead use manufacturig advancements to *decrease unit costs* without increasing the computing and storage resources of the device. Thus, in the future, we believe that the proportion of constrained IoT devices unable to operate as full nodes on a blockchain will increase even as the absolute number of IoT devices capable of being full nodes will increase.

In this paper, we propose a mechanism where an IoT device can operate as a light client while simultaneously attaining the full security properties available to a full node, albeit under limited circumstances, and at a cost of depending on a trusted party. Despite these limitations, we believe that this would make new use cases for IoT-blockchain systems feasible in terms of security, latency and costs.

The rest of this paper is structured as follows. First, in Section II we review existing research on the field, after which we provide an overview of Ethereum in Section III. Then, in Section IV, we describe *decentralized beacons,* a new method of transmitting and using trusted attestations of the blockchain state at IoT devices. This is followed, in Section V, by a discussion and a description of future work, with conclusions presented in Section VI.

[1] https://etherscan.io/

978-1-7281-2171-0/19/$31.00 ©2019 IEEE

II. Related work

Over the last few years there has been an increasing interest in IoT device and blockchain integration, as summarized e.g. by Reyna et al. [2]. Similarly, different integration patterns, such as an IoT device acting as a full node, blockchain access via a trusted gateway, or blockchain access via a light protocol have been identified [3]. Since it is common knowledge that many common public blockchains, such as Ethereum, have resource requirements that rule out most IoT devices from operating as a full node [5], an approach taken for example by IOTA[2] is to try to create a blockchain protocol and network that is suitable for direct use by IoT devices. Other distributed ledger systems, such as Hyperledger Fabric,[3] offer more lightweight integration options due to their inherent permissioned model that contains explicit trust anchors.

Another approach is to use gateways, but in ways where the protocol between the gateway and the IoT device (or across an IoT device mesh) provide security guarantees needed by the system, such as in Beekeeper [7]. While not always, these systems are more suitable for authenticating information transferred from the IoT device, but less frequently they address authentication and validating blockchain state as transferred *to* the device. While not explicitly using a trusted gateway, IoT devices are often used as full nodes on a (small) private network, either implying the use of an IoT-friendly blockchain, or the use of gateways in a more realistic, full-scale deployment [8]. Similarly, the proposed overlay network by Dorri et al. [9] establishes a layer of trusted nodes, pushing the security boundary outside of the most constrained nodes on the network.

III. Background

A. Ethereum

Ethereum [4] is a public blockchain technology and a network[4] of thousands of decentralized nodes. The network consists of three types of nodes: miners, full nodes (validating nodes) and light nodes. In the network all parties can submit transactions, but only miners can seal transactions into a new block. This requires solving a cryptographic puzzle (proof of work) which is a compute-intensive task. When a new block is announced on the network, both miners and full nodes validate the validity of the block (rejecting invalid blocks) and update their own internal view of the blockchain state. In contrast, light nodes are unable to maintain an internal view of the blockchain state, and have to rely on other nodes to provide the blockchain state upon request.

Transactions in the Ethereum network can interact with *smart contracts*, pieces of code stored in the Ethereum blockchain state and evaluated under a well-defined and deterministic virtual machine model. Since miners and full nodes keep their internal view of the blockchain state up-to-date by evaluating all transactions in blocks, all parties on the network can also rely on a smart contract's state being same across the whole network. The heteregoneus and decentralized blockchain network provides a high level of finality for all sealed transactions — after a short delay transactions can be considered to have become permanently recorded in the blockchain.[5]

While miners and full nodes form a peer-to-peer network, light nodes in contrast operate in a client-and-server model using *Light Ethereum Subprotocol* (LES) to communicate with light protocol servers. The light protocol are conventionally assumed to be full nodes. This protocol offers significantly different security properties than what is available to full nodes using the normal peer-to-peer protocol, and is described in Section III-C..

B. Ethereum and IoT devices

The permanence and decentralized nature of blockchains make them interesting from IoT devices' point of view. For example, Boudguiga et al. describe a mechanism for distributing information on IoT device firmware updates via a blockchain [10]. Similarly, smart contracts can be used as as integration points, providing a highly secure and robust source of information for the device. Since operations on a smart contract that do not require modification of the contract state can be evaluated locally, the device can use the smart contract to query values or make decisions based on input values from the device — for example, a sensor could check that a conventional HTTPS-based request is signed by a public key from the list of keys maintained by such a smart contract. Since smart contracts can encode arbitrary program logic they are highly flexible and adaptable to different requirements.

However, IoT devices are often constrained in their computing, storage and networking capabilities. As noted before, being a full node on the Ethereum network requires maintaining a local copy of the blockchain state. This requires computing resources (evaluating transactions' effects on the state), persistent storage for storing the state, and network capacity to receive information on new blocks and transactions. In the general case we have to assume that IoT devices are unable to operate as full nodes on the network, and have to operate as *light nodes*.

C. Light Ethereum Subprotocol

The Light Ethereum Subprotocol (LES) is a protocol [6] where a client receives block announcements from a server and can request block headers, transaction details, and state proofs from the server. The LES protocol itself is a peer-to-peer protocol, but in the context of IoT devices it is operated purely as a client-server protocol. In this context, an example of a highly simplified exchange of LES messages between a client (IoT device) and a server (full node) is shown in

[2]https://www.iota.org/

[3]https://www.hyperledger.org/projects/fabric

[4]The ethereum protocol and implementation can also be used in private blockchains. This paper focuses on the use of the public Ethereum network.

[5]The mechanics and economics of mining, permanence of past blocks, and risks concerning the whole network's behavior are beoynd the scope of this paper, and an interested reader is instructed to check the literature.

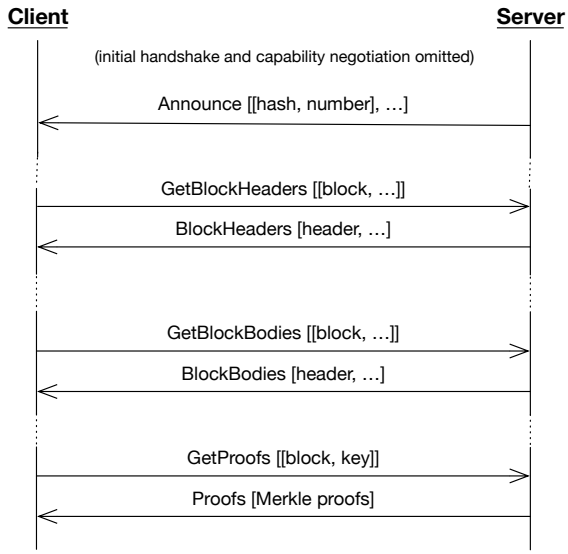

Figure 1. Simplified exchange between a LES client and server. The server announces a new block, which is then subsequently queried in detail by the client, ending with the client requesting state and its associated Merkle proofs for a specific account or smart contract.

Figure 1, where the client fetches block header, block body and state of an address on the blockchain as a result of receiving announcement of a new block.

As a result of the message exchange, the client can verify the consistency of all responses up to the block hash, through the use of hashing, Merkle-Patricia trees, and Merkle proofs. We denote this as the *internal consistency* of a block: there exists a chain of proofs from the block hash to all the retrievable block state, and the chain of proofs can be independently verified by the client. That is, the *block hash* validates the block header itself, whose proof-of-work can be checked by the client, and any state retrieved from the server can then be linked to the block header's Merkle-Patricia tree root hashes via state proofs. Therefore, once the client has hold of a block hash (or a block header), all of the state for the block is essentially securely frozen — any attempt to return inconsistent state data would result in an invalid proof. This internal consistency does not, however, provide any *global consistency* guarantees: the client cannot validate (in any easy way) that the state transition rules have been correctly followed by the miner(s) across blocks.[6]

Normally, any a block that does not follow the Ethereum blockchain transition rules will be rejected by other nodes. However, if an adversary has forked the blockchain, the adversary can provide an *invalid view* of the blockchain state to the client as it has no need to pass these invalid blocks to the general Ethereum network. If such an adversary can prevent the client from contacting any trustworthy nodes, the adversary can arbitrarily manipulate the state of any address in the invalid blocks it mines and presents to the client; for example, it may substitute a smart contract's code, or arbitrarily modify the balance of an account. A trivial client implementation would be unable to detect such a change,[7] as the block that had been mined by the adversary would contain a valid proof-of-work and the root hashes of Merkle-Patricia trees would correspond to the *modified* state, for which the server would be able to provide valid proofs. The block provided to the IoT device would be internally consistent, but not globally consistent.

[6]Validating that the state transition rules have been followed requires evaluating the block and its transactions and maintaining the full blockchain state, which, by definition, would equal for the client being a full node.

IV. Decentralized Beacons

There is an underlying conflict between constrained IoT devices, which want to minimize the bandwidth, storage and processing requirements, and the security of the light protocol, where increased security properties come at the price of increased use of bandwidth, storage and processing resources. To address this fundamental conflict between minimizing resource use and maximizing security guarantees, it is possible to rely on a trusted third party. The usual way is to use a gateway or a separate trusted server that the IoT device communicates directly with, pushing the boundary of blockchain interaction away from the device.

We propose a new mechanism, called *decentralized beacons*. Instead of an IoT device trusting an intermediary gateway, or a directly connected server, the trust is placed on a separate, trusted entity that provides *attestations* of the blockchain state, but is not in direct communication with the IoT device. These attestations are encoded as transactions on the very same blockchain network the IoT device is connected to (using a light protocol). The trusted entity is assumed to have sufficient resources to operate as a secure and well-connected full node, providing sufficient guarantees on the persistence and commitment properties of the attested blockchain state.

The main benefit of this method is that the IoT device and the attestation entities do not need to have any direct communication whatsoever, other than being members of the same blockchain. Hence, it is possible to deploy any number of IoT devices relying on the same attesting node(s) without adding any new load. An IoT device may also check the attestation state from multiple independent attesters without adding any communications load.

A. Overview

The overall structure of an IoT system using decentralized beacons is shown in Figure 2 and discussed in detail below.

While the idea of establishing a trusted entity in order to operate on a decentralized blockchain may appear to fly in the face of the goal of attaining decentralized security guarantees, one must realize that most IoT devices already implicitly or

[7]More complex implementation could recurse and fetch the state of the smart contract from earlier blocks for cross-correlation. With sufficient recursion depth the client could have a probabilistic assurance that the chain of blocks could not have been mined by an adversarial miner. This does increase the complexity of the light client as well as the bandwidth requirements, and in the end, can only provide probabilistic security guarantees.

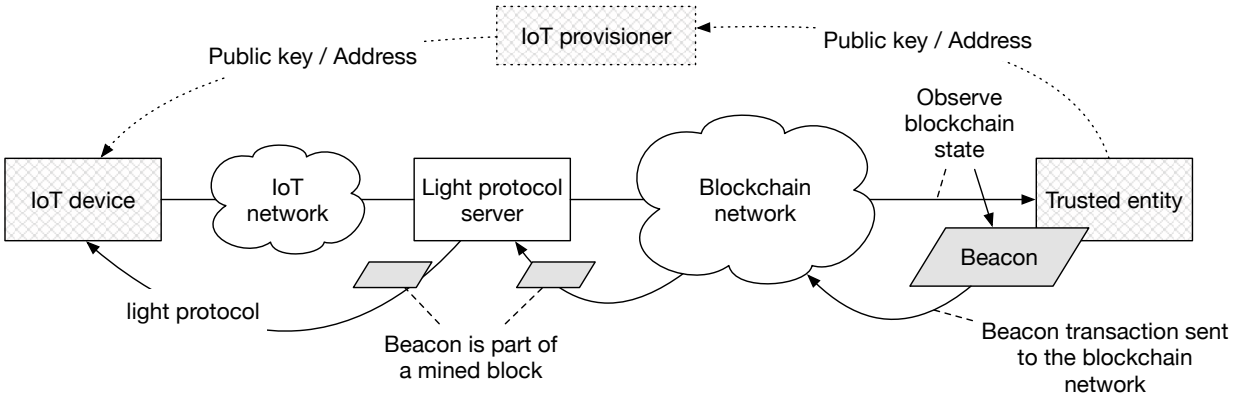

Figure 2. The IoT device is limited to use the light protocol server to interact with the blockchain. The light protocol server is a full node, and communicates directly with the blockchain network. The trusted entity observes the blockchain state, and generates beacon transactions and sends them to the network for processing. The IoT device is assumed to have configured with the trusted entity's public key or address during provisioning as well as any other parameters, such as the light protocol server address, as needed. If and when the beacon is part of a block the device receives from the light protocol server, the device is able to securely confirm that the transaction is sent by the trusted entity.

explicitly trust their manufacturer, owner, installer, or other such entity. Hence, in this context, it is not a significant step to assume the existence of a entity that provides trusted blockchain state attestations, especially if that entity can be used to provide an operational solution solving the problems outlined above. Also, since there is no need for global trust, decentralized beacons do not require any global coordination among the blockchain nodes or between the trusted entities and the IoT devices relying on them. The deployment and use of the beacons are *decentralized decisions*, made by a each IoT device owner, operator, or manufacturer, separately.

B. Beacon Structure

For the beacons to work, they must provide the client a convincing piece of evidence about the *global* state of the blockchain. On the Ethereum blockchain, a trivial beacon structure would be a transaction from the trusted entity's address and with the *data* field containing the block hash of the attested block. However, some further elaboration is necessary, as the sender's address is available only through recovering it from the transaction signature. Hence, the sender needs to send the transaction to itself (or some other predefined address). As the recipient address is explicit in the transaction, using it allows the IoT device to detect beacon transactions with less resources.

More formally, we define a *decentralized beacon* to be a mechanism for transporting an attestation $A = f_A(b_n)$ for block n, and f_A is a function of the block identified by its block hash b_n where its result satisfies the property that b_n can be inferred from it.[8] For Ethereum, we place the attestation A to a transaction's data portion, $T_d = A$, set the target address to the trusted party's own address a_s (or other predefined address), $T_t = a_s$, the transaction sequence number T_n to the next sequence number,[9] and transaction value to zero $T_v = 0$. The gas limit T_g is set to a sufficient value.[10] The gas price T_p is dynamically determined by the current gas price level of the network to ensure sufficiently fast inclusion of the transaction in a future block. Finally, the transaction must include the signature components T_w, T_r and T_s,. Consequently, an attestation transaction T is constructed as follows:

$$\begin{aligned} T &= (T_n, T_p, T_g, T_t, T_v, T_d, T_w, T_r, T_s) \\ &= (T_n, T_p, \geq 21000 + 68\,|f_A(b_n)|, \\ &\qquad a_s, 0, f_A(b_n), T_w, T_r, T_s) \end{aligned}$$

Note that from the security point of view, everything but the signature (T_w, T_r, T_s) and the attestation $A = f_A(b_n)$ are irrelevant and need only to be chosen so that the transaction is both valid, and will be accepted into a future block. The other fields could be used for other purposes — for example, if the recipient of the transaction is the sender itself, one could imagine piggy-backing other information on the value field, as sending cryptocurrency to oneself does not change the balance (apart from the transaction costs).

C. Security

The use of decentralized beacons assumes existence of a *centralized* trusted entity. The security of the system rests on the infeasibility of an adversary being able to impersonate the trusted party. In practice, this equates to the security of the private key of the trusted party. However, beacons themselves do not rely on the security of the underlying blockchain — in

[8] A trivial $f_A(b_n) = b_n$. We do, however, anticipate a need for more elaborate encoding for the attestation data.

[9] T_n is referred to as "nonce" in the Ethereum specification, but as a monotonically increasing sequence of integer numbers starting from zero value it is actually a sequence number.

[10] Since the target address is not a smart contract address, the gas usage of the transaction is $21000 + 68\,|T_d|$, but it would be prudent to set a higher limit to guard against future changes in gas usage calculation.

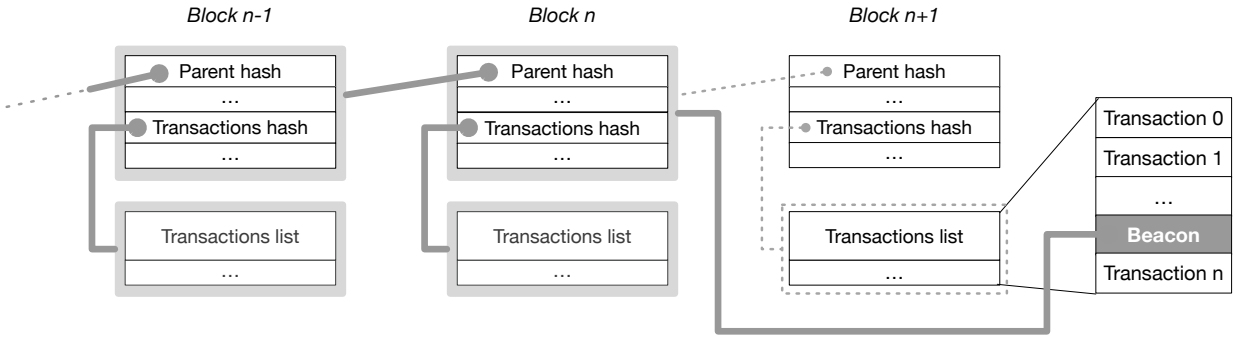

Figure 3. If the beacon referring to block n is in block $n+1$, the information the recipient can establish as "true" are highlighted by thick gray lines and borders. This includes the beacon itself, whose signature can be checked. The beacon contains block hash of the block n, e.g. hash of the header block which can be verified. The block $n-1$ can in turn be verified through the parent hash in block n. Each validated block header in turn contains hashes of other elements that are part of the block such as the list of transactions. Note that while the beacon in block $n+1$ can be verified, all of the other data in block $n+1$ has only to be locally consistent, but cannot be verified by the recipient as globally consistent.

contrast, they assume a model where an adversary is actively trying to subvert the network.

A hostile entity, such as a wireless base station, can *fork* the blockchain, providing any light clients connecting to it with a state that differs from the "real" network state. The hostile entity is, however, unable to forge transactions from the trusted party. The hostile entity can arbitrarily delay or reorder beacon transactions, as well as any other transaction that is submitted to the original blockchain. It can also generate bogus transactions from addresses it has control over. These actions are, however, already available to the adversary under the assumed threat model, and the use of beacons does not introduce any new capabilities to the adversary.

If the device receives a block that contains a transaction from the trusted party, *even under an assumption that any other data or state related to the block is arbitrary and cannot be trusted*, the device can check the authenticity of the beacon transaction based on its signature, and verify that the signature matches the trusted party's public key or address. Even if the hostile party generates bogus transactions that contain a correctly formed beacon transaction, they are unable to generate a signature that the device would accept as genuine.

This method of validating blocks based on an attestation in a beacon transaction is shown in Figure 3, which shows the beacon in the list of transactions in block number $n+1$. While the block can be checked against internal consistency, e.g. it has a valid proof-of-work, its transaction hash matches the hash of transactions, etc., the device cannot assume these represent a view that the trusted party sees or will see. What it can do is to validate the beacon itself — and use the attestation in the beacon to verify the attested block and all prior blocks.

In the Figure 3, the beacon transaction in block $n+1$ contains the attestation from which the block hash b_n of an earlier block number n can be inferred. This allows the device to fetch the block header H_n and check that the hash $h(H_n) = b_n$. The chain of attestation will trace backwards to all of the earlier blocks, since the block header H_n contains also the *parent hash* e.g. block hash of the previous block b_{n-1}, allowing the client to verify recursively all past blocks.

An important consideration is that the device must check the sequence number T_n of the transaction, accepting beacons only if the sequence number is *greater* than in the previous beacon the device has received. While a valid Ethereum state transition requires a monotonic increase of the sequence number for any party sending a transaction and strictly ordered execution, a hostile party can include beacon transactions in a mined block in arbitrary order, or omit transactions causing skips in the numbering sequence. To guard against replay attacks of past beacons, a sequence number check in the client is required.

The use of decentralized beacons is not without some tradeoffs and risks to the device. It is still reliant on the light protocol server, and an adversarial server can still block any and all beacon transactions. This is a real risk for any device whose correct operation relies on access to up-to-date information. The beacon sender must also acknowledge operational risks associated from purposeful manipulation of transaction economics of the blockchain [11] or straightforward brute-force denial-of-service attacks [12].

V. Discussion and Future Work

The applicability of decentralized beacons is limited by their inherent latency: the attestation of a *past* blockchain state will take, at the minimum, a single block interval to be accepted to a future block, and potentially a lot more. Additionally, beacon transactions need to be paid in Ethereum's cryptocurrency, which incurs a real-world cost. Attempting to include a beacon transaction in every block would run into anything towards and up a thousand dollars a day. This may or may not be an obstacle — it is possible to envision an IoT service provider bearing the cost, amortized over its customer base. Alternatively, some use cases may be more tolerant to higher update latencies, in which case a beacon could be sent less frequently. Regardless, the operational costs of sending beacons may restrict its applicability.

Based on the presented outline, we are in the process of 1) implementing a proof-of-concept beacon sender and client, and will 2) evaluate the characteristics and performance of the

proof-of-concept implementation on a private test network, the Rinkeby test network and the Ethereum main network. Furthermore, we will 3) develop a version of an Ethereum client that can use beacons to establish a trusted state of the blockchain, and 4) build and evaluate the beacon-enabled client in an environment where IoT devices operate on the Ethereum blockchain, evaluating the use of smart contracts to attain, for example, configuration data or access control policies relevant to the IoT device.

VI. Conclusions

In this paper, we have proposed a mechanism for enabling IoT devices to establish a trusted state of a public blockchain without them being full nodes, while yet being able to gain most of the security guarantees that a fully validating node is able to gain. This requires an addition of a trusted party, but we consider this a minor inconvenience, given that most IoT devices operated are already implicitly or explicitly trusting their manufacturer, owner, installer or other such entity. The proposed method does not require any changes to the actual blockchain network or other participating nodes on the network — it can be deployed locally by any single party. Furthermore, it scales linearly w.r.t. the number of trusted parties, supporting any number of IoT devices without any additional costs occuring as more devices are added.

While the concept of using a trusted attestation is not something that can be claimed to be a giant leap of imagination or technological advancement, we do see the need for IoT devices to be able to operate as "first-class citizens" on DLTs. While the second generation blockchains, such as Ethereum, may be eventually superseded by other DLTs that are natively more IoT-friendly, the use of beacons is a solution that can potentially be deployed immediately.

Acknowledgment

This work has been written as part of the SOFIE project, and has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 779984.

References

[1] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," p. 9, 2009.

[2] A. Reyna, C. Martín, J. Chen, E. Soler, and M. Díaz, "On blockchain and its integration with IoT. Challenges and opportunities," *Future Generation Computer Systems*, vol. 88, pp. 173–190, 2018-11-01, ISSN: 0167-739X. DOI: 10.1016/j.future.2018.05.046.

[3] K. R. Özyılmaz and A. Yurdakul, "Work-in-progress: Integrating low-power IoT devices to a blockchain-based infrastructure," in *2017 International Conference on Embedded Software (EMSOFT)*, 2017-10, pp. 1–2. DOI: 10.1145/3125503.3125628.

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

[5] A. Schoedon. (2017-11-29). The Ethereum-blockchain size will not exceed 1TB anytime soon., [Online]. Available: https://dev.to/5chdn/the-ethereum-blockchain-size-will-not-exceed-1tb-anytime-soon-58a (visited on 2018-01-11).

[6] F. Zsolt. (2017-10-16). Light Ethereum Subprotocol (LES) · zsfelfoldi/go-ethereum Wiki, [Online]. Available: https://github.com/zsfelfoldi/go-ethereum/wiki/Light-Ethereum-Subprotocol-%28LES%29 (visited on 2019-02-19).

[7] L. Zhou, L. Wang, Y. Sun, and P. Lv, "BeeKeeper: A Blockchain-based IoT System with Secure Storage and Homomorphic Computation," *IEEE Access*, pp. 1–1, 2018, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2018.2847632.

[8] J. Pan, J. Wang, A. Hester, I. AlQerm, Y. Liu, and Y. Zhao, "EdgeChain: An Edge-IoT Framework and Prototype Based on Blockchain and Smart Contracts," *IEEE Internet of Things Journal*, 2018, ISSN: 2327-4662.

[9] A. Dorri, S. S. Kanhere, and R. Jurdak, "Towards an Optimized BlockChain for IoT," in *Proceedings of the Second International Conference on Internet-of-Things Design and Implementation*, ser. IoTDI '17, New York, NY, USA: ACM, 2017, pp. 173–178, ISBN: 978-1-4503-4966-6. DOI: 10.1145/3054977.3055003.

[10] A. Boudguiga, N. Bouzerna, L. Granboulan, A. Olivereau, F. Quesnel, A. Roger, and R. Sirdey, "Towards Better Availability and Accountability for IoT Updates by Means of a Blockchain," in *2017 IEEE European Symposium on Security and Privacy Workshops (EuroS PW)*, 2017-04, pp. 50–58. DOI: 10.1109/EuroSPW.2017.50.

[11] S. Paavolainen, T. Elo, and P. Nikander, "Risks from Spam Attacks on Blockchains for Intemet-of-Things Devices," in *2018 IEEE 9th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, 2018-11, pp. 314–320. DOI: 10.1109/IEMCON.2018.8614837.

[12] K. Baqer, D. Y. Huang, D. McCoy, and N. Weaver, "Stressing Out: Bitcoin "Stress Testing"", in *Financial Cryptography and Data Security*, ser. Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2016-02-26, pp. 3–18, ISBN: 978-3-662-53356-7. DOI: 10.1007/978-3-662-53357-4_1.