Black-Box IoT: Authentication and Distributed Storage of IoT Data from Constrained Sensors

Panagiotis Chatzigiannis
George Mason University
pchatzig@gmu.edu

Foteini Baldimtsi
George Mason University
foteini@gmu.edu

Constantinos Kolias
University of Idaho
kolias@uidaho.edu

Angelos Stavrou
Virginia Tech
angelos@vt.edu

ABSTRACT

We propose Black-Box IoT (BBox-IoT), a new ultra-lightweight black-box system for authenticating and storing IoT data. BBox-IoT is tailored for deployment on IoT devices (including low-Size Weight and Power sensors) which are *extremely constrained* in terms of computation, storage, and power. By utilizing core Blockchain principles, we ensure that the collected data is immutable and tamper-proof while preserving data provenance and non-repudiation. To realize BBox-IoT, we designed and implemented a novel chain-based hash signature scheme which only requires hashing operations and removes all synchronicity dependencies between signer and verifier. Our approach enables low-SWaP devices to authenticate removing reliance on clock synchronization. Our evaluation results show that BBox-IoT is practical in Industrial Internet of Things (IIoT) environments: even devices equipped with 16MHz micro-controllers and 2KB memory can broadcast their collected data without requiring heavy cryptographic operations or synchronicity assumptions. Finally, when compared to industry standard ECDSA, our approach is two and three orders of magnitude faster for signing and verification operations respectively. Thus, we are able to increase the total number of signing operations by more than 5000% for the same amount of power.

1 INTRODUCTION

The commercial success of low Size Weight and Power (SWaP) sensors and IoT devices has given rise to new sensor-centric applications transcending traditional industrial and closed-loop systems [31, 93]. In their most recent Annual Internet Report [2], CISCO estimates that there will be 30 billion networked devices by 2023, which is more than three times the global population. While very different in terms of their hardware and software implementations, Industrial IoT (IIoT) systems share common functional requirements: they are designed to collect data from a large number of low-SWaP sensor nodes deployed at the edge. These nodes, which we refer to as edge *sensors*, are resource-constrained devices used in volume to achieve a broader sensing coverage while maintaining low cost. Thus, while capable of performing simple operations, low-SWaP sensors usually depend on battery power, are equipped with limited storage, and have low processing speed [23].

In practice, edge sensors are usually controlled by and report to more powerful gateway devices (which we refer to as *aggregators*) that process and aggregate the raw sensory data. For instance, in an Industrial (IIoT) environment, sensors are devices such as temperature sensors are broadcasting their measurements to the network router, which in turn submits it to the cloud through the Internet. Until recently, due to processing and storage constraints, many IoT designs were geared towards direct to cloud aggregation and data processing. However, latency, bandwidth, autonomy and data privacy requirements for IoT applications keep pushing the aggregation and processing of data towards the edge [58]. In addition, in most use cases, IoT devices need to be mutually *authenticated* to maintain system integrity and the data origin has to be verified to prevent data pollution attacks [61, 78] and in “model poisoning” where an attacker has compromised a number of nodes acting co-operatively, aiming to reduce the accuracy or even inject backdoors to the resulting analysis models [17, 40].

The use of distributed, immutable ledgers has been proposed as a prominent solution in the IoT setting allowing rapid detection of inconsistencies in sensory data and network communications, providing a conflict resolution mechanism without relying on a trusted authority [13]. A number of relevant schemes has been proposed in the literature [72, 76], which as we discuss in Section 6, propose various ways to integrate distributed ledgers (commonly referred to as *Blockchain*) with IoT.

The Challenge: One of the main roadblocks for using Blockchain-based systems as “decentralized” databases for sharing and storing collected data is their dependency on asymmetric authentication techniques. Typically, to produce authenticated data packets, sensors have to digitally sign the data by performing public key cryptographic operations, which are associated with expensive sign and verification computations and large bandwidth requirements. Although some high-end consumer sensor gateways and integrated sensors might be capable of performing cryptographic operations, a large number of edge sensors have limited computational power, storage and energy [20, 49]. To make matters worse, sensors try to optimize their power consumption by entering a “sleep” state to save power resulting in intermittent network connectivity and lack of synchronicity. Given such tight constraints, an important challenge is allowing low-SWaP devices being extremely constrained both in terms of computational power and memory (categorized as Class 0 in RFC 7228 [18] ref. Section 5.1), to authenticate and utilize a blockchain-based data sharing infrastructure.

Our Contributions: We design and implement BBox-IoT, a complete blockchain-based system for Industrial IoT devices aimed to create a decentralized, immutable ledger of sensing data and operations while addressing the sensor and data authentication challenge for extremely constrained devices. We aim to use our system as a "black-box" that empowers operators of an IIoT enclave to audit sensing data and operational information such as IIoT communications across all IIoT devices.

To perform sensor and data authentication operations *without* relying on heavy cryptographic primitives, we introduce a novel hash-based digital signature that uses an onetime hash chain of signing keys. While our design is inspired by TESLA broadcast authentication protocol [69, 70], our approach *does not* require any timing and synchronicity assumptions between signer and verifier. Overcoming the synchronicity requirement is critical for low-SWaP devices since their internal clocks often drift out of synchronization (especially those using low cost computing parts) [34, 79]. Our signature construction is proven secure assuming a pre-image resistant hash function. Also, we achieved logarithmic storage and computational (signature/verification) costs using optimizations [47, 92]. Our proposed scheme further benefits by the broadcast nature of the wireless communication. Indeed, in combination with the immutable blockchain ledger, we are able to ferret out man-in-the-middle attacks in all scenarios where we have more than one aggregators in the vicinity of the sensors. To bootstrap the authentication of sensor keys, we assume an operator-initiated device bootstrap protocol that can include either physical contact or wireless pairing using an operator-verified ephemeral code between sensors and their receiving aggregators. Our bootstrap assumptions are natural in the IoT setting, where sensors often “report" to specific aggregators and allows us to overcome the requirement for a centralized PKI. Note that our signature scheme is of independent interest, in-line with recent efforts by NIST for lightweight cryptography [80].

For the blockchain implementation where a *consensus* protocol is needed, we consider a *permissioned* setting, where a trusted party authorizes system participation at the aggregator level. Our system supports two main types of IoT devices: low-SWaP sensors who just broadcast data and self-reliant aggregators who collect the data and serve as gateways between sensors and the blockchain. While our system is initialized by a trusted operator, the operator is not always assumed present for data sharing and is only required for high-level administrative operations including adding or removing sensors from the enclave. We build the consensus algorithms for BBox-IoT using a modified version of Hyperledger Fabric [7], a well known permissioned blockchain framework, and leverage blockchain properties for constructing our protocols tailored for constrained-device authentication. However, BBox-IoT operations are designed to be lightweight and do not use public key cryptography based on the RSA or discrete logarithm assumptions, which are common, basic building blocks of popular blockchain implementations. We describe our system in details considering interactions between all participants and argue about its security.

We implemented and tested a BBox-IoT prototype in an IIoT setting comprising of extremely constrained sensors (Class 0 per RFC 7228). We employed 8-bit sensor nodes with 16MHz micro controllers and 2KB RAM, broadcast data every 10 seconds to a subset of aggregators (e.g. IIoT gateways) which in turn submit aggregated data to a cloud infrastructure. The evaluation shows that the IIoT sensors can compute our 64-byte signature in 50ms, making our signature scheme practical even for the least capable of IIoT platforms. Our evaluation section shows results by considering a sensor/gateway ratio of 10:1. When compared with ECDSA signing operations, our scheme is significantly more efficient offering two (2) and three (3) orders of magnitude speedup for signing and verification respectively. Our theoretical analysis and implementation shows that we can achieve strong chained signatures with half signature size, which permits accommodating more operations in the same blockchain environment. BBox-IoT is also over 50 times more energy-efficient, which makes our system ideal for edge cost-efficient but energy-constrained IIoT devices and applications.

Finally, we adopt the same evaluation for Hyperledger Fabric considered in previous work [7] and estimate the end-to-end costs of BBox-IoT when running on top of our Hyperledger modification, showing it is deployable in our considered use-cases.

2 BACKGROUND & PRELIMINARIES

2.1 Blockchain System Consensus

In distributed ledgers including Blockchains, we can categorize the participants to: a) Blockchain maintainers also called *miners*, who are collectively responsible for continuously appending valid data to the ledger, and b) clients, who are reading the blockchain and posting proposals for new data. While clients are only utilizing the blockchain in a read-only mode, the blockchain maintainers who are responsible for “book-keeping" must always act according to a majority's agreement to prevent faulty (offline) or Byzantine (malicious) behavior from affecting its normal functionality. Reaching agreement requires the existence of a *consensus* protocol among these maintainers.

Moreover, the type of consensus protocol can be permissioned or permissionless. In permissionless blockchains, like Bitcoin [65] and Ethereum [86], anyone can participate in the consensus protocol and sybil attacks are prevented by new consensus mechanisms such as Proof-of-Work or Proof-of-Stake [51]. Although the open nature of permissionless blockchains seems attractive, it does not really fit the membership and access control requirements for IoT deployments. Instead, operators would like to exert control over IoT sensors and aggregators by authenticating all system participants.

Focusing on permissioned blockchains, consensus is computationally easier to achieve even if nodes are of limited capabilities. Distributed consensus mechanisms such as Practical Byzantine Fault Tolerance (PBFT) [26] are relatively efficient, as long as the number of maintainers is relatively small: Byzantine Fault Tolerant consensus is considered scalable up to 20 nodes [28, 81]. The distributed consesus mechanisms use inexpensive cryptographic operations such as message authentication codes (MACs). Alternatively, a trusted party TP decides who should be given participating access by issuing a private credential to each node.

In the following paragraphs we define consensus in the permissioned setting and establish the properties needed for our system. In Appendix C we present an overview of consensus algorithms in the permissioned setting inspired by [12, 24, 37], and discuss how each one would fit to BBox-IoT.

Fundamental consensus properties: Informally, the consensus problem considers a set of state machines, we also refer to them as *replicas* or *parties*, starting with the same initial state. Then given a common sequence of messages, each correct replica, by performing its private computation, should reach a state consistent with other correct replicas in the system, despite any possible network outages or other replica failures [12].

Since our system uses a blockchain, we focus on the notion of *ledger* consensus [37], where a number of parties receive a common sequence of messages and append their outputs on a public ledger. As a special case, an *Authenticated* ledger consensus protocol [55][59] only permits participation through credentials issued by a TP. The two fundamental properties of a ledger consensus protocol are [37]: (a) *Consistency*: An honest node's view of the ledger on some round j is a prefix of an honest node's view of the ledger on some round $j + \ell, \ell > 0$. (b) *Liveness*: An honest party on input of a value x, after a certain number of rounds will output a view of the ledger that includes x. We provide formal definitions in Appendix C.

BBox-IoT Permissioned Consensus: The aforementioned fundamental properties are not sufficient for our system consensus. For instance, most "classical" consensus algorithms such as PBFT [26] have not been widely deployed due to various practical issues including lack of scalability. Taking the BBox-IoT requirements into account, the system's consensus algorithm needs to satisfy the following additional properties:

i. Dynamic membership: In BBox-IoT, there is no a priori knowledge of system participants. New members might want to join (or leave) after bootstrapping the system. We highlight that the vast majority of permissioned consensus protocols assume a static membership [28]. Decoupling "transaction signing participants" from "consensus participants" is a paradigm that circumvents this limitation [74].
ii. Scalable: BBox-IoT might be deployed in wide-area scenarios (e.g. IIoT), so the whole system must support in practice many thousands of participants, and process many operations per second (more than 1000 op/s) [81].
iii. DoS resistant: For the same reason above, participants involved in consensus should be resilient to denial-of-service attacks [12].

In Appendix C.8 we also mention some desirable (but not required) consensus properties, and show how some well-known consensus protocols satisfy our required properties above in Table 7.

2.2 Hyperledger

One of the most promising examples of permissioned blockchains is the Hyperledger project, established by the Linux Foundation and actively supported by companies such as IBM and Intel [45]. The Hyperledger project aims to satisfy a wide range of business blockchain requirements, and has developed several frameworks with different implementation approaches, such as Hyperledger Fabric, Indy, Iroha and Sawtooth. Each framework uses a different consensus algorithm, or even supports "pluggable" (rather than hardcoded) consensus like Hyperledger Fabric [7]. To make pluggable consensus possible, Hyperledger Fabric introduces the "execute-order-validate" paradigm in its architecture, instead of the traditional "order-execute" [7]. In this paradigm, the "maintainer" participants are decoupled from the "consensus" participants (called *Peers* and *Orderers* respectively as we will see below), which eventually leads to satisfying dynamic membership and scalability.

In the following we focus on Hyperledger Fabric which seems to fit best in our system and provide a high-level description of its architecture (shown in Figure 1a). Its main components are categorized as follows:

(1) **Clients** are responsible for creating a transaction and submitting it to the peers for signing. After collecting a sufficient number of signatures (as defined by the system policy), they submit their transaction to the orderers for including it in a block. Client authentication is delegated to the application.
(2) **Peers** are the blockchain maintainers, and are also responsible for endorsing clients' transactions. Notice that in the context of Hyperledger, "Endorsing" corresponds to the process of applying message authentication.
(3) **Orderers** collectively form the "ordering service" for the system. After receiving signed transactions from the clients, the service establishes *consensus* on total order of a collected transaction set. Then the ordering service by delivering blocks to the peers, and ensures the consistency and liveness properties of the system.
(4) The **Membership Service Provider** (MSP) is responsible for granting participation privileges in the system.

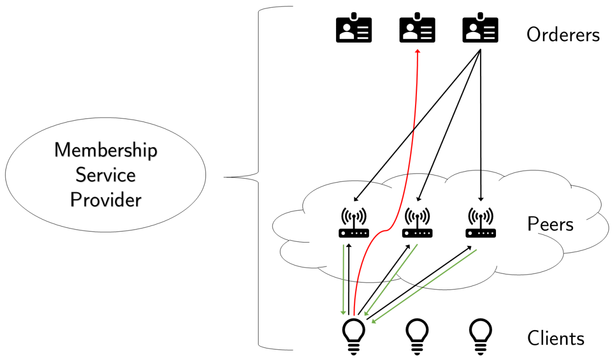
In its initial version v0.6, Hyperledger Fabric used the Byzantine fault-tolerant PBFT consensus algorithm [46], which only supports static membership for the ordering service participants. It's current version (v2.2) offers the *Raft* [67] consensus algorithm, which provide crash fault tolerance but not Byzantine fault tolerance, thus preventing the system from reaching consensus in the case of malicious nodes. Hyperledger Fabric could potentially use BFT-SMART in the future [16, 77].

Regarding scalability, although the original version of Hyperledger Fabric has several potential bottlenecks in its architecture, proposals exist to improve its overall performance in terms of operations per second [39]. These proposals suggest storing blocks in a distributed peer cluster for further scalability improvements. Also while operations (or transactions) per second is one scalability aspect in our setting, the other one is the actual number of peers the system can practically support without heavily impacting its performance. To the best of our knowledge, this has only been experimentally shown for up to 100 peers [7]. This experiment indicates however a low impact of the number of peers, especially if the network latency is low. Our setting allows such an assumption, since our use-case scenarios are all instantiated in a specific geographical location, as later discussed in section 5.

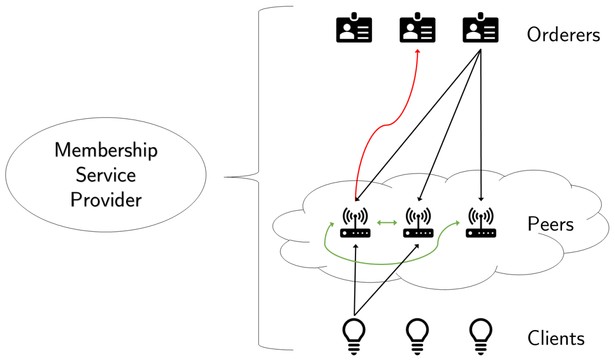
2.3 Modifying Hyperledger Fabric

While Hyperledger Fabric's *execute-order-validate* architecture offers several advantages discussed previously, we cannot directly use it in our BBox-IoT system, since we assume that lightweight devices (which for Hyperledger Fabric would have the role of "clients") are limited to only broadcasting data without being capable of receiving and processing. In Hyperledger Fabric, clients need to collect signed transactions and send them to the ordering service, which is an operation that lightweight devices are typically not capable of performing.

Our modification. To address this issue, we propose a modification in Hyperledger architecture. In our modified version, as shown



(a) Original architecture. Clients collect signatures from peers for a transaction, then submit the signed transaction to the ordering service which then returns a block containing packaged transactions to peers.



(b) Modified architecture. Clients only broadcast the transaction to the peers, who are then responsible themselves for signing it before submitting it to the ordering service.

Figure 1: Modified Hyperledger Fabric architecture.

in Figure 1b, a client broadcasts its “transaction" message to all nearby peer nodes. However, the transaction is handled by a *specific* peer (which are equivalent to an aggregator as we discuss in the next section), while peers not “responsible" for that transaction disregard it. That specific peer then assumes the role of the “client" in the original Hyperledger architecture simultaneously, while also continuing functioning as a peer node. As a client, it would be responsible for forwarding this transaction to other peers, and collecting the respective signatures, as dictated by the specified system policy, in a similar fashion to original Hyperledger Fabric. It would then forward the signed transaction to the ordering service, and wait for it to be included in a block. The ordering service would send the newly constructed block to all peers, which would then append it to the blockchain.

Security of our modifications. The proposed modifications of Hyperledger do not affect the established security properties (i.e. Consistency and Livenessas we define them in Appendix C and interpreted in [7]), since a peer node simultaneously acting as a client, can only affect the signing process by including a self-signature in addition to other peers' signatures. However, because the signing requirements are dynamically dictated by the system policy, these could be easily changed to require additional signatures or even disallow self-signatures to prevent any degradation in security. We also note that while this modified version of Hyperledger effectively becomes agnostic to the original client, which otherwise has no guarantees that its broadcasted transaction will be processed honestly, our threat model discussed in the next section captures the above trust model.

3 BBOX-IOT SYSTEM PROPERTIES

In BBox-IoT there are five main types of participants, most of them inherited by Hyperledger Fabric: the MSP, orderers, local administrators, aggregators and sensors. Aggregators are equivalent to *peers* and sensors to *clients* in our modified Hyperledger Fabric architecture discussed in the previous section. We provide a high level description of each participant's role in the system and include detailed definitions in Appendix D .

- The MSP is a trusted entity who grants or revokes authorization for orderers, local administrators and aggregators to participate in the system, based on their credentials. It also initializes the blockchain and the system parameters and manages the system configuration and policy.
- Orderers (denoted by O) receive signed transactions from aggregators. After verifying the transactions as dictated by the system policy they package them into blocks. An orderer who has formed a block invokes the consensus algorithm which runs among the set of orderers $\mathcal{O}$. On successful completion, it is transmitted back to the aggregators with the appropriate signatures.
- Local administrators (denoted by LAdm, are lower-level system managers with delegated authority from the MSP. Each LAdm is responsible for creating and managing a local device group G, which includes one or more aggregators and sensors. He grants authorization for aggregators to participate in the system with the permission of the MSP. He is also solely responsible for granting or revoking authorization for sensors in his group, using aggregators to store their credentials.
- Aggregators (denoted by Ag) are the blockchain maintainers. They receive blocks from orderers and each of them keeps a copy of the blockchain. They store the credentials of sensors belonging in their group and they pick up data broadcasted by sensors. Then they create blockchain “transactions" based on their data (after possible aggregation), and periodically collect signatures for these transactions from other aggregators in the system, as dictated by the system policy. Finally, they send signed transactions to the ordering service, and listen for new blocks to be added to the blockchain from the orderers.
- Sensors (denoted by S) are resource-constrained devices. They periodically broadcast signed data blindly without waiting for any acknowledgment. They interact with local administrators during their initialization, while their broadcasted data can potentially be received and authenticated by multiple aggregators.

We then define the security and operational properties of BBox-IoT, in accordance with evaluation principles adopted in [19, 32, 38, 76].

3.1 Threat model & Assumptions

Physical layer attacks and assumptions. While our system cannot prevent physical tampering with sensors that might affect data correctness, any data discrepancies can be quickly detected through comparisons with adjacent sensors given the blockchain immutability guarantees [89]. Similarly, any malicious or erroneous data manipulation by an aggregator will result in detectable discrepancies even when one of the aggregators is not compromised simultaneously. Of course, if all aggregators become compromised instantaneously, which is hard in a practical setting, our system will not detect any discrepancies. This raises the bar significantly for an adversary who might not be aware or even gain access to all aggregator nodes at the same time. Finally, attacks such as flooding/jamming and broadcast interception attacks are out of scope in this paper.

Trust Assumptions. We assume that MSP is honest during system bootstrapping only, and that device group participants (Local administrators, aggregators and sensors) may behave unreliably and deviate from protocols. For instance, they might attempt to statically or dynamically interfere with operations of honest system participants (e.g. intercept/inject own messages in the respective protocols), even colluding with each other to do so. This behavior is expected which our system is designed to detect and thwart.

Consensus Assumptions. As in Hyperledger, we decouple the security properties of our system from the consensus ones. For reference, this implies tolerance for up to 1/3 Byzantine orderer nodes, with a consensus algorithm satisfying at least the fundamental and additionally required properties discussed in Section 2.

Given the above adversarial setting, we define the following security properties[1]:

- S-1 Only authenticated participants can participate in the system. Specifically:
  - a. An orderer non-authenticated by the MSP is not able to construct blocks (i.e., successfully participate in the consensus protocol). The ordering service can tolerate up to f malicious (byzantine) orderers.
  - b. An LAdm non-authenticated by the MSP is not able to form a device group G.
  - c. If an aggregator is not authenticated by the MSP, then its signatures on transactions cannot be accepted or signed by other aggregators.
- S-2 *Sensor health:* Sensors are resilient in the following types of attacks:
  - a. Cloning attacks: A non-authenticated sensor cannot impersonate an existing sensor and perform operations that will be accepted by aggregators.
  - b. Message injection - MITM attack: A malicious adversary cannot inject or modify data broadcasted by sensors.
- S-3 *Device group safety:* Authenticated participants in one group cannot tamper with other groups in any way, i.e.:
  - a. An LAdm cannot manage another group, i.e. add or revoke participation of an aggregator or sensor in another device group, or interfere with the functionalities of existing aggregators or sensors at any time.
  - b. An aggregator (or a coalition of aggregators) cannot add or remove any sensor in device group outside of their scope, or interfere with the functionalities of existing aggregators or sensors at any time.
  - c. A sensor (or a coalition of sensors) cannot interfere with the functionalities of existing aggregators or other sensors at any time.
- S-4 *Non-repudiation and data provenance:* Any BBox-IoT node cannot deny sent data they signed. For all data stored in BBox-IoT, the source must be identifiable.
- S-5 *DoS resilient:* BBox-IoT continues to function even if MSP is offline and not available, or an adversary prevents communication up to a number of orderers (as dictated by the consensus algorithm), a number of aggregators (as dictated by the system policy) and up to all but one sensor. Also an adversary is not able to deny service to any system node (except through physical layer attacks discussed before).
- S-6 *System policy and configuration security:* BBox-IoT policy and configuration can only be changed by MSP.
- S-7 *Revocation:* The system is able to revoke authentication for any system participant, and a system participant can have its credentials revoked only by designated system participants.

[1]We do not consider data confidentiality in our system, however as discussed later our model could be extended to satisfy confidentiality as well.

4 CONSTRUCTIONS

We first set the notation we will be using throughout the rest of the paper. By λ we denote the security parameter. By $b \leftarrow B(a)$ we denote a probabilistic polynomial-time (PPT) algorithm B with input a and output b. By := we denote deterministic computation and by $a \rightarrow b$ we denote assignment of value a to value b. We denote a protocol between two parties A and B with inputs x and y respectively as $\{\mathsf{A}(x) \leftrightarrow \mathsf{B}(y)\}$. By $(\mathsf{pk}, \mathsf{sk})$ we denote a public-private key pair. A list of elements is denoted by [ ]. We denote a block B being appended on a blockchain BC as $\mathsf{BC}||\mathsf{B}$.

4.1 Our Hash-based Signature Scheme

Our construction is inspired by Lamport passwords [54] and TESLA [69, 70] but *avoids the need for any synchronization* between senders and receivers which is a strong assumption for the IoT setting. Instead, we assume the existence of a constant-sized state for both the sender and receiver between signing operations. Our scheme allows for a fixed number of messages to be signed, and has constant communication and logarithmic computation and storage costs under the following requirements and assumptions:

- There's *no* requirement for time synchronization, and a verifier should only need to know the original signer's pk.
- The verifier should immediately be able to verify the authenticity of the signature (i.e. without a "key disclosure delay" that is required in the TESLA family protocols, described in more detail in Section 6.2).
- Network outages, interruptions or "sleep" periods can be resolved by requiring computational work from the verifier, proportional to the length of the outage.
- We do not protect against Man-in-the-Middle attacks in the signature level, instead, we use the underlying blockchain to detect and mitigate such attacks as we discuss later in Section 4.3.

Let $h : \{0,1\}^* \to \{0,1\}^\lambda$ be a preimage resistant hash function.

$(\mathsf{pk}, \mathsf{sk_n}, s_0) \leftarrow \mathsf{OTKeyGen}(1^\lambda, n)$

– sample a random "private seed" $k_n \leftarrow \{0,1\}^*$

– generate hash chain $\mathsf{pk} = k_0 = h(k_1) = h(h(k_2)) = \ldots = h^i(k_i) = h^{i+1}(k_{i+1}) = \ldots = h^{n-1}(k_{n-1}) = h^n(k_n)$

– hash chain creates n pairs of $(\mathsf{pk_i}, \mathsf{sk_i})$ where:

$(\mathsf{pk_1}, \mathsf{sk_1}) = (k_0, k_1) = (h(k_1), k_1)$,

$(\mathsf{pk_2}, \mathsf{sk_2}) = (k_1, k_2) = (h(k_2), k_2)$,

... ,

$(\mathsf{pk_i}, \mathsf{sk_i}) = (k_{i-1}, k_i) = (h(k_i), k_i)$,

...,

$(\mathsf{pk_n}, \mathsf{sk_n}) = (k_{n-1}, k_n) = (h(k_n), k_n)$

– initialize a counter $\mathsf{ctr} = 0$, store ctr and pairs as $[(\mathsf{pk_i}, \mathsf{sk_i})]_1^n$ to initial state s_0

– output $(\mathsf{pk} = \mathsf{pk_1}, \mathsf{sk_n}, s_0)$.

Note: Choosing to store only $(\mathsf{pk}, \mathsf{sk_n})$ instead of the full key lists introduces a storage-computation trade-off, which can be amortized by the "pebbling" technique we discuss in this section.

$(\sigma, \mathsf{sk_i}, s_i) \leftarrow \mathsf{OTSign}(\mathsf{sk_{i-1}}, m, s_{i-1})$

– parse s_{i-1} and read $\mathsf{ctr} \to i-1$

– compute one-time private key $\mathsf{sk_i} = k_i$ from $n-i$ successive applications of the hash function h on the private seed k_n (or read k_i from $[\mathsf{sk}]_1^n$ if storing the whole list)

– compute $\sigma = h(m||\mathsf{pk_i})||\mathsf{sk_i} = h(m||k_{i-1})||k_i = h(m||h(k_i))||k_i$

– increment $\mathsf{ctr} \to \mathsf{ctr}+1$, store it to updated state s_i

$\mathsf{OTVerify}(\mathsf{pk}, n, m, \sigma) := b$

– parse $\sigma = \sigma_1||\sigma_2$ to recover $\sigma_2 = k_i$

– Output $b = (\exists j < n : h^j(k_i) = \mathsf{pk}) \wedge (h(m||h(k_i)) = \sigma_1)$

Note: The verifier might choose to only store the most recent k_i which verified correctly, and replace pk with k_i above resulting in fewer hash iterations.

Construction 1: n-length Chain-based Signature Scheme

$$\begin{tikzcd} k_0 & k_1 \arrow[l, "h"'] & k_2 \arrow[l, "h"'] & k_3 \arrow[l, "h"'] & k_4 \arrow[l, "h"'] & k_5 \arrow[l, "h"'] \end{tikzcd}$$

Figure 2: Key generation for $n = 5$ and seed k_5. First signature uses as $\mathsf{pk} = k_0$ and $\mathsf{sk} = k_1$.

- The signer has very limited computation, power and storage capabilities, but can outsource a computationally-intensive pre-computation phase to a powerful system.

Our scheme, presented in Construction 1, is a chain-based one-time signature schemesecure under an adaptive chosen-message attack as formally defined in Definition 5 in the Appendix, with each key derived from its predecessor as $k_i \leftarrow h(k_{i+1})$, $i \in \{n-1, n-2, \ldots, 0\}$ and h is a preimage resistant hash function. The keys when used in pairs (k_i, k_{i-1}) can be viewed as a public-private key pair for a one-time signature scheme, then forming a one-way hash chain with consecutive applications of h. The key k_n serves as the "private seed" for the entire key chain. In the context of integrity, a signer with a "public key" $k_{i-1} = h(k_i)$ would have to use the "private key" k_i to sign his message. Since each key can only be used once, the signer would then use $k_i = h(k_{i+1})$ as his "public key" and k_{i+1} as his "private key", and continue in this fashion until the key chain is exhausted.

For example as shown in Figure 2, we can construct a hash chain from seed k_5. For signing the 1st message m_1, the signer would use $(\mathsf{pk_1}, \mathsf{sk_1}) = (k_0, k_1)$ and output signature $\sigma = h(m_1||k_0)||k_1$. Similarly, for the 2nd message he would use $(\mathsf{pk_2}, \mathsf{sk_2}) = (k_1, k_2)$ and for the 5th message $(\mathsf{pk_5}, \mathsf{sk_5}) = (k_4, k_5)$.

Constructing the one-way hash-chain described above, given the seed k_n, would require $O(n)$ hash operations to compute $k_0 = h^n(k_n)$, which might be a significant computational cost for resource-constrained devices, as the length of the hash chain n is typically large to offset the constraint of single-use keys. While we could pre-compute all the keys, which would cost a $O(1)$ lookup operation, we would then require $O(n)$ space, which is also a limited resource in such devices. Using efficient algorithms [47, 92], we can achieve logarithmic storage and computational costs by placing "pebbles" at positions $2^j = 1 \cdots \lceil log_2(n) \rceil$, which as shown in Section 5.3 makes our construction practical for resource-constrained devices. The verifier's cost is $O(1)$ when storing the most recently-used k.

Comparison and Discussion. Our scheme is directly comparable with the TESLA Broadcast Message Authentication Protocol [69, 70], which follows a similar chain-based paradigm but requires some synchronicity between the sender and receiver, and the receiver can only verify a message after some delay. Several other chain-based schemes have been proposed [29, 41, 60], forming a "hierarchy" of chains aiming to improve their efficiency in various aspects. However, most of them do not prevent the synchronicity requirement and delayed verification, in fact some even introduce additional requirements, e.g. special "commitment distribution" messages [60], where a verifier won't be able to verify a long series of signatures if those are lost. As our scheme is hash-based, we compare with another family of hash-based signatures schemes that follow a tree structure, e.g. XMSS [14] and SPHINCS [22]. While these schemes do not have any synchronicity assumptions, their performance is not suited for the low SWaP sensors we consider (even with resource-constrained device optimizations [44]which we compare in detail in Appendix B.5). In Table 1 we compare with other hash-based schemes in terms of properties (i.e. no synchronicity or delays, denoted as NoSync and NoDelay respectively). In Table 2 we provide a concrete comparison with the rest of the schemes satisfying the above properties. In Section 6 we discuss some of the above schemes in more detail.

Table 1: Hash-based scheme comparison.

Scheme	Architecture	NoSync	NoDelay
TESLA [69, 70]	Chain	✗	✗
μTESLA 2-level chain [60]	Chain	✗	✗
Sandwich, 1-level, light chain [41]	Chain	✗	✗
Comb Skipchain [41]	Chain	✓	✗
Short Hash-Based Signatures [29]	Chain	✓	✓
XMSS [22]	Tree	✓	✓
BPQS [27]	Chain	✓	✓
SPHINCS [14]	Tree	✓	✓
Our construction	Chain	✓	✓

Table 2: Hash-based scheme comparison for 256-bit messages and 256-bit security parameter. Sizes in bytes. $\mathbb{M}$,$\mathbb{F}$ and $\mathbb{H}$ denote MAC, PRF and hash operations respectively. n denotes length of chain-based schemes.

Scheme	$\lvert\sigma\rvert$	$\lvert \mathsf{pk} \rvert$	$\lvert \mathsf{sk} \rvert$	$\mathsf{Sign}()$	$\mathsf{Verify}()$
Short Hash-Based Signatures [29]	$128 + log_2 n$	32	$64(\lceil log_2(n) \rceil + 1)$	$(\lceil log_2(n) \rceil + 3)\mathbb{H} + 3\mathbb{F}$	$\lceil log_2(n) \rceil$
XMSS [22]	2692 (4963)	1504 (68)	64	$747\mathbb{H} + 10315\mathbb{F}$	$83\mathbb{H} + 1072\mathbb{F}$
BPQS [27]	2176	68	64	1073 $\mathbb{H}$	1073 $\mathbb{H}$
SPHINCS [14]	41000	1056	1088	386$\mathbb{F}$, 385 PRGs, 167519 $\mathbb{H}$	14060 $\mathbb{H}$
Our Construction	32(64)	32	32	$\lceil log_2(n) \rceil\ \mathbb{H}$	1 $\mathbb{H}$

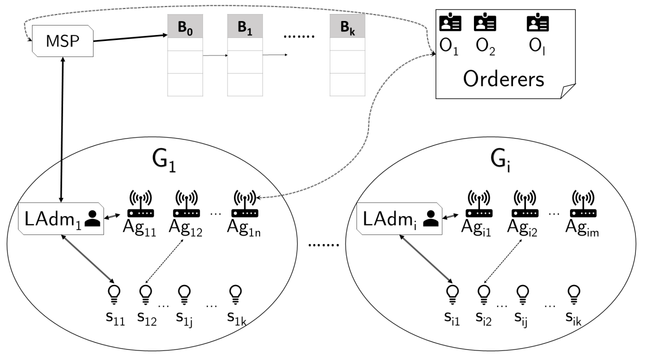

MSP
B_0
B_1
B_k
O_1 O_2 O_l
Orderers
G_1
$LAdm_1$
Ag_{11} Ag_{12} Ag_{1n}
s_{11} s_{12} s_{1j} s_{1k}
G_i
$LAdm_i$
Ag_{i1} Ag_{i2} Ag_{im}
s_{i1} s_{i2} s_{ij} s_{ik}

Figure 3: BBox-IoT construction overview

The caveat in our scheme is that it is susceptible to Man-in-the-Middle attacks. Specifically, an attacker might intercept a signature packet in transit (thus learning the "ephemeral" private key) and replace it with an arbitrary message and signature. Nevertheless such attacks are unlikely to be successful in our setting as discussed later in Section 4.3.

4.2 Overall BBox-IoT Construction

Our BBox-IoT system consists of the following components as shown in Figure 3 illustrating our modifications to the Hyperledger Fabric architecture.

- A (trusted) Membership Service Provider[2] MSP, which resembles a Trusted Party, and is responsible for authorizing participation in the system. The MSP bootstraps the system and forms the genesis block, which contains hardcoded information on its public key and the consensus algorithm. The genesis block also initializes the authorized system participants and the system policy (denoted by Pol), both of which can be changed later.
- A permissioned blockchain BC, which consists of normal "transaction" blocks and special "configuration" blocks.
- A configuration Config for BC, containing membership information for local administrator, orderer and aggregators, as well as system policy data. As in Hyperledger Fabric, Config is stored in the configuration blocks.
- A set of orderer nodes $\mathcal{O} : \{\mathsf{O}_1, \mathsf{O}_2, ..., \mathsf{O}_\ell\}$, responsible for achieving consensus on forming new blocks. These nodes are assumed static, although it can be extended to handle dynamic membership.
- A set of device groups $\mathcal{G} : \{\mathsf{G}_1, \mathsf{G}_2, ..., \mathsf{G}_n\}$. On each group G_i there exist:
  – A local administrator LAdm_i, responsible for its group membership, which includes a set of aggregators and sensors. In order for LAdm_i to add or remove an aggregator in the system must also have consent from the MSP, however he does not need permission to handle sensor membership.
  – A set of aggregators $\mathcal{AG}_i : \{\mathsf{Ag}_{i1}, \mathsf{Ag}_{i2}, ..., \mathsf{Ag}_{im}\}$, which have also the role of *peers* in Hyperledger Fabric. We assume aggregators can perform regular cryptographic operations and aggregate data received from sensors. As discussed in our modified Hyperledger, they also briefly take the role of a "client".
  – A set of sensors $\mathcal{S}_i : \{\mathsf{S}_{i1}, \mathsf{S}_{i2}, ..., \mathsf{S}_{ik}\}$, which are assumed to be resource-constrained devices. These would be the equivalent of *clients* in the original Hyperledger Fabric architecture, but here they are assumed to only broadcast their data to nearby group aggregators, without expecting a confirmation. The only step where interaction occurs is during initial setup, where they exchange their public key and other initialization data with the group administrator. We also assume that sensors can only perform basic cryptographic operations (i.e. hashing), meaning they can't perform public key cryptography operations that use exponentiations.

[2]The MSP also includes the system administrator.

We first describe the initialization process for the system's MSP and genesis block B_0. After generating its keys, MSP bootstraps the system with pre-populated participation whitelists of orderers, local group administrators, and aggregators (denoted by OL, LL and PL respectively) and a pre-defined system policy. Sensors do not need to be tracked from the MSP, as participation authorization for sensors is delegated to the group local administrators. Local administrators control authorization privileges with a respective sensor whitelist denoted by SL, and they also keep a whitelist of group aggregators denoted by AL.

Furthermore, we detail the functionality of reading or updating the system's configuration, including the permissioned participants and the system policy. Orderers and local administrators can only be authorized for participation by the MSP, while aggegators need their local administrator's approval as well. As discussed above, sensor participation is handled by the local administrators, however, group aggregators also keep track of group participation for sensors in a passive manner. The local administrators are also responsible for revoking participation rights for aggregators and sensors belonging in their group. In general, granting or revoking participation privileges is equivalent to adding or removing the participant's public key from the respective whitelist. Note that membership verification can also be handled by accumulators [10, 25] instead of whitelists to achieve lists of constant size, however we keep whitelists for simplicity purposes.

SensorJoin

- Sensor generates a seed uniformly at random, and generates hash chain through OTKeyGen algorithm. (computation is outsourced to a powerful device)
- Sensor stores hash chain "pebbles" in its memory and outputs the last element of the chain as public key to the LAdm

SensorSendData

- Sensor computes signature σ for broadcasted data m using OTSign algorithm
- $\mathsf{S_{ij}}$ broadcasts σ to aggregators in group.
- Each aggregator after verifying the signature through OTVerify, checks if any other aggregator received a conflicting message. It adds the message - signature pair in its local state, pending for blockchain submission.

AggrSendTx

- Aggregator parses its local state for pending blockchain operations as a transaction.
- Aggregator computes signature on transaction and sends it to other aggregators.
- Each aggregator after verifying signature and sender membership in the system, signs the transaction.
- The sending aggregator submits signed transaction to ordering service after reaching necessary number of signatures, as dictated by system policy.
- Each orderer after verifying signatures, runs consensus algorithm which outputs a blockchain update operation.
- The blockchain operation is received by orderers who update the blockchain state.

SensorTransfer

- Aggregator encrypts the state for the sensor under the reveiving aggregator's pk (i.e. the most recent received $\mathsf{sk_i}$) and submits it to the blockchain using AggrSendTx. Sensor is removed from the device group and is transferred to new group.
- Receiving aggregator decrypts state from the blockchain and resumes verification of received data from sensor.

Construction 2: BBox-IoT core algorithms and protocols

Furthermore, on a high-level, sensors "blindly" broadcast their data as signed transactions. Nearby aggregators (belonging to the same device group) receive and verify the data and collect the required amount of signatures from other aggregators in the system (as defined by the system policy), and then submit the signed transaction to the ordering service. The orderers then by running the consensus protocol, "package" the collected transactions to form a blockchain block. Finally, the block is sent back to the aggregators, who as the blockchain "maintainers", append it to the blockchain. The core system functionalities are shown in Construction 2 and we provide a detailed description of all system algorithms and protocols in Appendix D. .

Sensor join: Defined by SensorJoin() protocol between a sensor and a Local administrator. This is the only phase when a sensor is interacting with the system, as the LAdm generates a new hash chain and its associated pebbles in a powerful device. The pebbles are then loaded to the sensor, and LAdm updates the group aggregators with the new sensor's public key.

Sensor broadcast: Defined by SensorSendData() protocol between a sensor and group aggregators. For some data m, the sensor computes the one-time hash-based signature using OTSign() and the signed data m, σ is broadcasted to all group aggregators. If there are any aggregator who receives a different signed message m', σ, the message is discarded, else it remains in the aggregator's pending memory for processing.

Aggregator transaction: Defined by AggrSendTx() protocol between aggregators and orderers. For an aggregator to submit aggregated data to the blockchain, it first needs to collect the needed signatures from other aggregators. Then it submits the signed transaction to the ordering service, which in turn executes the Consensus() algorithm to construct a block with a set of signed transactions. Finally, the block is transmitted to the aggregators, who append the block as the blockchain maintainers.

Sensor transfer: Defined by SensorTransfer algorithm, executed when a sensor is transferred to a new location or device group. The handing-over aggregator saves its state of our signature scheme w.r.t. that sensor and encrypts it on the blockchain under the receiving aggregator's public key. After sensor transfer, the receiving aggregator decrypts that state and resumes message verification.

Optionally in our construction, a symmetric group key $\mathsf{K_G}$ can be shared between each group's local administrator, aggregators and sensors for confidentiality purposes. However, the additional encryption operations have an impact mainly on sensors, which have constrained computational and storage resources. Note that using such key for authentication or integrity would be redundant since these properties are satisfied using public keys existing in the appropriate membership lists and revocation operations can still be performed at an equivalent cost using those lists.

4.3 Security Analysis

THEOREM 1 (INFORMAL). *The construction in Section 4.2 satisfies participant authentication (S-1), sensor health S-2 and device group safety properties (S-3) assuming* (SignGen, Sign, SVrfy) *is an existentially unforgeable under a chosen-message attack signature scheme,* (OTKeyGen, OTSign, OTVerify) *is an unforgeable one-time chain based signature scheme,* MSP *is honest and not compromised and the consensus scheme* (TPSetup, PartyGen, TPMembers, Consensus) *satisfies the consistency property.*

Proof Sketch. We now provide a proof sketch arguing about the security of our scheme.
S-1 Participant Authentication. We require that only authenticated participants can participate in the different functions of our protocol. We argue that if an adversary breaks the participant authentication property then it could break unforgeability of the underlying signature scheme. Specifically:

(1) For property S-1a., in protocol OrdererAdd (coupled with ConfigUpdate) the use of an unforgeable signature scheme guarantees that no one but the MSP can authenticate orderers, while in protocol Consensus the same scheme guarantees that only the authenticated orderers can perform this core functionality. Also recall from the previous property

that an adversary being able to authenticate orderers could break the immutability property.

(2) For property S-1b., in protocol LAdminAdd (coupled with ConfigUpdate) the use of an unforgeable signature scheme guarantees that no one but the MSP can authenticate Local Administrators, while AggrSetup, AggrAdd, AggrUpd, SensorSendData and GroupRevoke guarantee that only the authenticated local administrators can perform these functionalities.

(3) For property S-1c., in protocol AggrAdd (coupled with ConfigUpdate) the use of an unforgeable signature scheme guarantees that no one but the MSP can authenticate Aggregators, while AggrUpd and AggrSendTx the same scheme guarantees that only the authenticated aggregators can perform these functionalities.

S-2 Sensor health. In order for an adversary to impersonate/clone a sensor, it would either have to break the unforgeability of our signature scheme, or launch a MITM attack which is a potential attack vector as discussed in Section 4.1.

As discussed in Section 3.1, we consider jamming attacks at the physical layer outside the scope of this paper. Given the nature of our setting where a sensor's broadcast has typically short range, we consider MITM and message injection attacks hard and unlikely to launch but we still consider them as part of our threat model. Even in these unlikely scenarios, MITM attacks can be easily mitigated in BBox-IoT. A first approach for detecting such attacks is to leverage blockchain properties, where aggregators can compare data received from a sensor in the blockchain level. Our assumption here is that sensor data can be received by more than one aggregators in the vicinity of the sensor which is a reasonable senario for typical dense IoT deployments. If there's even one dissenting aggregator, probably victim of a MITM attack, all the associated data would be considered compromised and disregarded and the operator will be notified of the data discrepancy detected. The above approach while simple, still permits a MITM attacker to "eclipse" a sensor from the system using a jamming attack.

An alternative approach is to make a proactive check in a group level, where each aggregator would verify the validity of its received data by comparing it with other aggregators before even submitting it to the blockchain. In both above strategies, the attacker's work increases significantly because he would need to launch simultaneous MITM attack between the sensor and all aggregators in the vicinity. We adopt the second approach in our Construction in Appendix D.

The above properties and strategies ensure that only data broadcasted by authenticated sensors are accepted by aggregators in SensorSendData.

S-3 Device group safety. An adversary wanting to break device group safety would either have to add or revoke aggregators or sensors in an existing group through AggrAdd, AggrUpd or GroupRevoke thus breaking unforgeability of the signature scheme used in these protocols or interfere with existing authenticated sensors in a group through SensorSendData by breaking unforgeability of the one-time chain based signature scheme. □

Integrity, non-repudiation and data provenance requirements (S-4) are core properties of any digital signature scheme thus directly satisfied in BBox-IoT.

Additionaly we argue that our system is DOS resilient (S-5) in the following scenarios:

- MSP offline or not available: The core system functionality is not affected, although there can be no configuration changes in the system. All algorithms and protocols (except those involving adding or revoking orderers, local administrators or aggregators or those involving system policy changes) perform authentication through the configuration blocks and not the MSP itself.
- Orderers unavailable: Reduces to tolerance properties of consensus algorithm.
- LAdm unavailable: The core system functionality is not affected, although there can be no administrative operations in the respective group.
- Ag unavailable: Transactions are not processed only in the respective groups. However if more than τ aggregators are unavailable as required in AggrSendTx, no transactions can be processed in the whole system.

Also an adversary might attempt to flood an aggregator by broadcasting messages and arbitrary signatures. In this scenario the aggregator would be overwhelmed since by running OTVerify for each message-signature pair separately, it would have to check the signature against all hash chain values up to the first public key. To mitigate this we propose checking only for a few hashes back to the chain (defined by a system parameter "maxVerifications" as shown in Algorithm 2 in the Appendix). This parameter can be set by the local administrator but should be carefully selected. A small value might result in need of frequent re-initializations for the sensors - if a long network outage occurs between a sensor and an aggregator and they lose "synchronization", the local administrator should reinitialize the sensor in the device group. On the other hand, a large value would amplify the impact of DOS attacks.

Policy and configuration security (S-6) is ensured by algorithms ConfigUpdate and ReadConfig, as the first algorithm creates a special configuration transaction signed by the MSP and the second returns configuration data originating from such a transaction.

Revocation (S-7) is made possible by NodeRevoke and GroupRevoke (in conjuction with whitelists used throughout all system protocols and algorithms). Also the unforgeability of the underlying signature scheme ensures that only the MSP (and the LAdm respectively only for aggregators and sensors) can revoke these credentials.
Remark. One might suggest to use MACs instead of our proposed signature scheme for sensor authentication. We discuss this in Appendix A.

5 PERFORMANCE EVALUATION & MEASUREMENTS

5.1 The IIoT Setting With Constrained Devices

IIoT environments are complex systems comprising of heterogeneous devices that can be tracked at different organizational layers, namely (a) computational, (b) network, (c) sensor/edge layers [88]. Devices at the higher levels are powerful servers dedicated to the analysis of data, storage, and decision making. They frequently reside outside the factory premises, i.e., in cloud infrastructures. On the other hand, on-site and at the edge layer, a myriad of low-SWaP

Table 3: Classes of Constrained Devices in terms of memory capabilities according to RFC 7228.

Name	RAM	Flash
Class 0	<<10 KiB	<<100 KiB
Class 1	10 KiB	100KiB
Class 2	50KiB	250KiB

devices such as sensors and actuators reside, assigned with the tasks of posting their data or reconfiguring their status based on received instructions. On typical real-life IIoT deployments, the processing speed of such devices ranges from tens (e.g., Atmel AVR family) to hundreds of Mhz (e.g., higher-end models of ARM Cortex M series). Diving even deeper, at the lower end of the spectrum, one may observe sensor-like devices that are severely constrained in memory and processing capabilities.

Such extremely constrained devices have been considered by RFC 7228 [18] which underlines that "most likely they will not have the resources required to communicate directly with the Internet in a secure manner". Thus, the communication of such nodes must be facilitated by stronger devices acting as gateways that reside at the network layer. In Table 3 we provide a taxonomy of constrained devices residing at the edge of IIoT according to RFC 7228.

In this work, we consider a generic IIoT application scenario that involves Class 0 devices which are connected to more powerful IoT gateways in a sensor/gateway ratio of 10:1. The chosen platforms and all experimental decisions were made to provide a realistic scenario under the following assumptions: (a) devices severely constrained in terms of computational power and memory resources (Class 0) and (b) moderately demanding in terms of communication frequency (i.e. transmission once every 10 seconds).

5.2 Evaluation Setup

Our testbed consists of Arduino UNO R3 [1] open-source microcontroller boards equipped with ATmega328P 16 MHz microcontroller and 2KB SRAM fitted with a Bluetooth HC-05 module. These devices are really constrained and they represent the minimum of capabilities in all of IoT sensors utilized in our experimental scenarios (Class 0 in Table 3). For the gateways, we use Raspberry Pi 3 Model B devices equipped with a Quad Core 1.2GHz BCM2837 64bit CPU and 1GB RAM.

We first focus on evaluating our system in a device group level[3]. We use the one-time signature scheme outlined in Construction 1 and SHA256 as the hash function $h()$. The length of the hash chain as defined in section 6.2 sets the upper bound on the number of one-time signatures each sensor $\mathsf{S_i}$ can generate. In the case where the sensor's available signatures are depleted, it would enter an "offline" state and the Local Administrator LAdm would need to manually renew its membership in the system through the $\mathsf{SensorJoin}$ protocol. In a large-scale deployment of our system however, frequent manual interventions are not desirable, so our goal is to pick a sufficiently large n such that the available one-time signatures to the sensor last for the sensor's lifetime. As discussed above and taking similar schemes' evaluations into account [6], we consider a frequency of one (1) signing operation per 10 seconds for simplicity.

Table 4: Evaluation for sensor-aggregator protocol - Average verification times

	T2 (μsec)				T3 (msec)			
maxV	20	22	24	26	20	22	24	26
100	28.12	31.18	31.34	28.95	42.83	42.84	42.91	43.08
500	30.78	31.94	30.31	30.63	51.25	51.23	51.37	51.39
1000	31.39	30.96	31.14	30.74	55.27	55.35	55.36	55.41
2500	30.57	30.97	32.39	30.86	60.61	60.65	60.7	60.78
5000	33.26	31.7	31.66	31.43	64.66	64.74	64.79	64.83
10000	33.34	33.38	33.6	31.41	68.68	68.75	68.78	68.86

We consider sensor lifetimes between 4 months as an lower and 21 years as a upper estimate (as shown in Table 5), which imply a hash chain between 2^{20} and 2^{26} elements respectively.

In the setup phase, we pre-compute the hash-chain as needed by the pebbling algorithm [47] and load the initial pebble values into the sensor. We first measure the actual needed storage on the sensor for various values of n. Note that for $n = 2^{26}$, the lower bound for needed storage using a 256-bit hash function is about $26 \cdot 256 = 832$ bytes of memory. Then we set the sensor device to communicate with the aggregator through Bluetooth in broadcast-only mode and measure the maximum number of signing operations that can be transmitted to the aggregator for various values of n, as well as the verification time needed on the aggregator side since it will need to verify a large number of sensor messages. The fact that we are able to run BBox-IoT on Class 0 devices demonstrates the feasibility of our approach for all low-SWaP sensors.

5.3 Signing and Verification

We run our experiments under different scenarios and multiple times. Our evaluation results, which are shown in Table 5, represent the statistical average across all measurements. Note that for measuring the average signature verification time on the aggregator side, we assume that the aggregator is able to receive all the data broadcasted by the sensor. If a network outage occurs between them (and the sensor during the outage keeps transmitting), the aggregator after reestablishing connection would have to verify the signature by traversing the hash chain back up to the last received secret key, which incurs additional computation time (in Figure 4 we show the associated verification cost in such occasions). As expected, the verification time is relatively constant in all measurements, about 0.031ms on average. This suggests that such an aggregator could still easily handle 10^5 sensors transmitting data for verification (as we considered one transmission every 10 seconds for each sensor).

Table 5, shows that the pebbles data stucture consumes most of the required memory storage in our implementation, while the remaining program requires a constant amount of memory for any number of pebbles. We also observe a slight impact of the number of pebbles on the total verification time, which is mainly affected by the sensor's capability to compute the signature on its message and the next secret key. For example, the sensor needs 50ms to compute the next signature with $n = 2^{26}$ and 49.95ms for $n = 2^{24}$. Also by comparing the total verification time with the signature computation time, we conclude the extra 14.3 msec are needed for transmitting the signature.

[3]Our code is available at https://github.com/PanosChtz/Black-Box-IoT

Table 5: Evaluation for sensor-aggregator protocol (average values for 5000 verifications)

Hash Chain length n	2^{20}	2^{22}	2^{24}	2^{26}
Sensor lifetime for 1sig/10sec (m: months, y: years)	4 m	16 m	5 y	21 y
Pebble Gen time (seconds)	1.62	6.49	24.57	95.33
Verification time per signature (msec)	0.031			
Signature size (bytes)	64+ $\|m\|$			
Total dynamic memory usage (bytes)	1436	1520	1604	1678
Pebble struct memory usage (bytes)	840	924	1008	1082
Program memory usage (bytes)	596			
Signature computation time (msec)	49.82	49.88	49.95	50.00
Average total verification time per signature (msec)	64.15	64.25	64.26	64.32
Communication cost (msec)	14.3			

In Table 4 we provide a series of measurement results for the average verification time of 1 signature on the aggregator. By T2 we denote the verification time of a signature and by T3 the total verification time by an aggregator (we provide detailed algorithms for our measurements in Appendix E.) The average total verification time (denoted by maxV) increases significantly as we require more verification operations from the Arduino device. This happens because of dynamic memory fragmentation as the pebbling algorithm updates the pebble values.

Comparison with ECDSA. We compare our lightweight scheme with ECDSA, which is commonly used in many blockchain applications. We assume IoT data payloads between 50 and 220 bytes, which can accommodate common data such as timestamps, attributes, source IDs and values. In Table 6 we show that our scheme is more efficient compared to ECDSA by 2 and 3 orders of magnitude for signing and verification respectively. Even when considering larger payload sizes which impact hash-based signature operations, our scheme remains much more efficient. However, verification cost for our scheme increases linearly during network outages, and as shown in Figure 4 it might become more expensive than ECDSA when more than 2400 signature packets are lost.

Another metric we consider is energy efficiency, which is of particular importance in IoT applications that involve a battery as power source. Our experiments depicted in Figure 5 show that our ATmega328P microcontroller can perform more than 50x hash-based signing operations compared to the equivalent ECDSA operations for the same amount of power. Finally, while our hash-based signature normally has a size of 64 bytes (as shown in Table 5), we can "compress" consecutive signatures along a hash chain to 32 bytes by only publishing the most recent k_i. The verifier would then generate the previous hash chain values at a minimal computational cost. This makes possible to store more authenticated data in the blockchain, as we show below.

5.4 Consensus Performance

Considering the use-case scenario discussed in Section 5.1, we discuss the performance of our BBox-IoT system as a whole. We show that the most important metric in the system is the transaction throughput which heavily depends on the ability of the SWaP sensors to transmit data in a group setting. Of course, the scalability of the system overall is also directly proportional to the number of system active participants it can support simultaneously.

Sensors. Our measurements indicate that the aggregator - which is a relatively powerful device - is not the bottleneck in the protocol execution. Based on the measurements in Table 5, we can safely assume that a single aggregator can verify over a thousand sensors' data being continuously broadcasted, since the signature computation time by a sensor is three (3) orders of magnitude larger than the verification time by an aggregator. This is still a pessimistic estimation, since we previously assumed that a sensor broadcasts (and signs) data every 10 seconds, which implies that the aggregator can accommodate even more sensors.

Orderers. Since orderers only participate in the consensus protocol to sign blocks, we only need a few orderers such that our system remains resilient to attacks at the consensus level should a subset of orderers become compromised. Orderers can be strategically distributed over a geographical area to minimize the network latency between an aggregator and the ordering service, controlled by the main organization (which also controls the MSP). Evaluations performed in previous works have shown that by having 3 orderers, 3000 transactions/second can be easily achieved using the consensus protocol used in the current version of Hyperledger Fabric (with a potential of further improvement in a future adoption of BFT-SMART), and even considering up to 10 orderers in the system does not greatly affect its performance [7, 77].

Aggregators. The expected number of aggregators in the system depends on the use case as it is expected. As discussed in Section 5.1, where gateways play the role of BBox-IoT aggregators, we consider a sensor/gateway ratio of 10:1 for our evaluation purposes. To our knowledge, no evaluation of Hyperledger Fabric has ever been performed to consider such a great number of peers, which would require a great amount of resources to perform. However, by adopting the evaluation performed in [7] which measured the throughput in terms of number of peers up to 100 (which as discussed, are the aggregators in our system), we can extrapolate this evaluation to the order of thousands, which shows that with the aid of a "peer gossip" protocol, the system remains scalable if the peers are in the same approximate geographical area which implies low average network latency.

Blockchain operations. As discussed, aggregators' role is to aggregate sensor data into blockchain transactions. Assuming that aggregators perform no "lossy" operations (such as averaging techniques), they would just package many collected sensor data along with the respective signatures into a transaction which in turn would be submitted to the ordering service. If we assume as in [7] a block size of 2MB, we can estimate how much signed sensor data a block can hold. Given the discussion in Section 5.3, a Hyperledger block could hold (at most) about 15800 signed sensor data using our hash-based scheme vs. 12700 using ECDSA.

	BBox-IoT		ECDSA	
Message length	Sensor Sign	Aggr Vrfy	Sensor Sign	Aggr Vrfy
50	50.43	0.0339	4200	42.55
100	53.47	0.0349		
150	56.40	0.0357		
202	59.33	0.03687		
218	60.06	0.0369		
Signature size	32		64	

Table 6: Signing and verification costs (in milliseconds) compared with message and signature sizes (in bytes). Note we assume hash-based signatures are aggregated as discussed in Section 5.3. Signer is ATmega328P microcontroller and verifier is RPi 3.

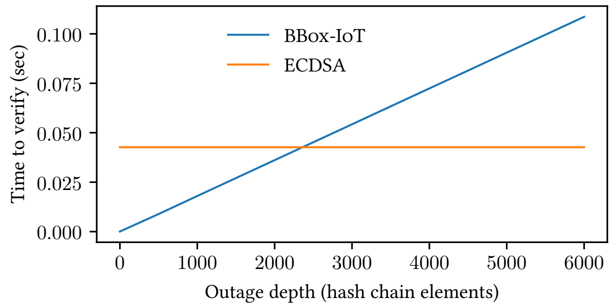


Figure 4: Aggregator verification costs in network outages. BBox-IoT is more expensive when more than about 2400 signature packets are lost.

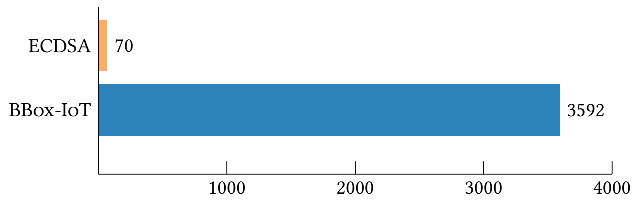


Figure 5: Number of signing operations for a 20mWh battery.

Latency. We also wish to estimate the time from a value being proposed by an aggregator until consensus has been reached on it (assuming the block contains a single transaction). Again we can adopt previous evaluations in Hyperledger Fabric [7], which show an average of 0.5 sec for the complete process. Finally, considering that the previous evaluations mentioned above were all preformed on the original Hyperledger Fabric (while our architecture requires a slight modification as discussed in Section 2.3), for our purposes we assume that the expected performance of aggregators (which are essentially Hyperledger peers also having client application functionalities) is not affected by this additional functionality, since the main affecting factor that can potentially become a bottleneck for the scalability of the whole system is network latency and not computational power.

6 RELATED WORK

We now discuss a number of works that connect IoT to the blockchain setting or works which build cryptographic primitives to optimize different parts of computation for resource-constrained IoT devices. Note that none of these works addresses the problem of authentication for extremely constrained (Class 0) devices.

6.1 IoT and Blockchain

Shafagh et al. [76] presented an architecture aiming to handle IoT data in a decentralized manner while achieving confidentiality, authenticity and integrity. This proposed system defines itself as “IoT compatible" being append-only by a single writer and can be accessed by many readers, and consists of a layered design on top of an existing public blockchain to store access permissions and hash pointers for data, while storing the actual data off-chain using decentralized P2P storage techniques. Other approaches [9, 66, 82] also used a similar "layering" paradigm. While these approaches are simpler than ours, they ultimately rely heavily on the performance and properties of the underlying public blockchain and are not specifically tailored to handle resource-constrained IoT devices.

Dorri, Kanhere, and Jurdak [32] considered a “local" private blockchain maintained by a capable device, managed by the on-site owner and containing the local IoT device transactions. These lower-tier elements would be overlaid by a shared blockchain that can handle hashed data originating from the local blockchain and stored in a cloud storage service, and can enable access to local data. The above approach also offers confidentiality and integrity for submitted data and is suitable for resource-constrained IoT devices, however it is more complex than BBox-IoT and requires managing and replicating data over several points in the system.

More recently, AlTawy and Gong [5] presented a blockchain-based framework in the supply chain setting using RFIDs. This model considered blockchain smart contracts interacting with an overlay application on the RFID readers and a centralized server that handles membership credentials. This framework offers anonymity for the participating entities, which prove their membership in zero-knowledge, while their anonymity remains revocable by the server. It also provides confidentiality for its transactions and enforces a notion of “forward secrecy" which enables future product owners in the supply chain to access its entire history. BBox-IoT differs from the above work in several ways, since it is tailored to handle resource-constrained devices. Our work does not have confidentiality or anonymity as a main goal, although it can be added as an option using symmetric keys. We also do not require any smart contract functionality from the blockchain, and we operate exclusively in the permissioned setting.

IoTLogBlock [72] shares a common goal with our work: enabling the participation of low-power devices in a distributed fashion, and similarly uses Hyperledger as a “cloud service” in a IoT setting. The crucial difference with our work, is that IoTLogBlock is evaluated on a Class 2 device using ECDSA signatures, which are far more expensive than our proposed hash-based signature and could not have been supported at all by a Class 0 device, while having much larger power consumption (Fig 5). Our proposed signature scheme is a key component for efficient implementations of blockchain-based systems in the IIoT setting.

Several more approaches have been presented which augmented an IoT infrastructure with a blockchain, focusing on providing two-factor authentication [87], managing or improving communication among IoT devices [64, 75], implementing a trust management system in vehicular networks [91], providing edge computing services [90], data resiliency [57], providing secure and private energy trade in a smart-grid environment [3] and implementing a hierarchical blockhain storage for efficient industrial IoT infrastructures [84] and all of which are orthogonal to our work. We point the reader to [4, 35] for extensive reviews on the related literature.

6.2 Hash-based Signatures

Early works such as Lamport's One-Time Signatures (OTS) [53] allowed the use of a hash function to construct a signature scheme. Apart from being one-time however, this scheme suffered from large key sizes. Utilizing tree-based structures such as Merkle trees [62], enabled to sign many times while keeping a constant-sized public key as the Merkle root. Winternitz OTS and later WOTS+ [21][42] introduced a way of trading space for computation for the underlying OTS, by signing messages in groups. XMSS [22] further optimized the Merkle tree construction using Winternitz OTS as an underlying OTS. Other works such as HORS [73] enabled signing more than once, and more recently SPHINCS and SPHINCS+ [14, 15] enabled signing without the need to track state. Using HORS [73] as a primitive combined with a hash chain, Time Valid One-Time Signature (TV-HORS) [85] improves in signing and verification computational efficiency, but assuming "loose" time synchronization between the sender and the verifier. All of the above scheme families while only involving hash-based operations, still incur either large computational and/or space costs, and cannot be implemented in Class 0 resource-constrained devices we consider. Follow-up work exists for implementing SPHINCS on resource-constrained devices [44] which we discuss later in this section and compare in Appendix B.5.

The TESLA Broadcast Message Authentication Protocol [69, 70] follows a "one-way" chain-based approach for constructing a hash-based message authentication scheme. Based on a "seed" value, it generates a one-way chain of n keys, which elements are used to generate temporal MAC keys for specified time intervals. The protocol then discloses each chain element with some time delay Δ, then authenticity can be determined based on the validity of the element in the chain as well as the disclosure time. The "pebbling" algorithms [47, 92] enable logarithmic storage and computational costs as discussed in Section 4.1. Its main drawback however is that it also requires "loose" time synchronization between the sender and the receiver for distinguishing valid keys. In an IoT setting this would require the frequent execution of an interactive synchronization protocol, since IoT devices are prone to clock drifting [34, 79]. Also we assume in Section 3 that IoT devices function in a broadcast-only mode, which would not allow the execution of such interactive protocol in the first place. Furthermore, TESLA introduces a "key disclosure delay" which might be problematic in certain IoT applications, and gives up the non-repudiation property of digital signatures.

Several modifications and upgrades to the TESLA protocol have been proposed, with most of them maintaining its "key disclosure delay" approach which is also associated with the loose time synchronization requirement [41, 60]. A notable paradigm is the "hierarchical" (or two-dimensional) one-way chain structure, where the elements of a "primary" hash chain serve as seeds for "secondary" chains in order to reduce communication costs. [41] includes several such proposals. For instance, its Sandwich-chain uses two separate one-way chains. The first one-way chain is used as a "primary" chain, which generates intermediate "secondary" chains using the elements of the second one-way chain as salts. However to maintain efficiency, it still assumes some weak time synchronicity between the signer and the verifier by disclosing each element of the "primary" chain with some time delay (else the verifier in case of a network outage would have to recompute all the previous secondary chains as well which would defeat its efficiency gains). More importantly however, this construction has much larger storage requirements than ours. In the same work, the Comb Skipchain construction is asymptotically more efficient in signing costs than our scheme and does not require time synchronicity, but has worse concrete storage requirements which are prohibitive for low-end IoT devices, and still suffers from delayed verification. This work includes other interesting modifications such as the "light" chains where the secondary chains are generated using a lower security parameter and a standard one-dimensional TESLA variant which does not require a MAC.

6.3 Cryptographic Operations in IoT

In the context of improving cryptographic operations in the IoT setting, Ozmen and Yavuz [68] focused on optimizing public key cryptography for resource-constrained devices. This work exploited techniques in Elliptic Curve scalar multiplication optimized for such devices and presented practical evaluations of their scheme on a low-end device. Even though the device used in this work is can be classified as a Class 1 or Class 2 device, our construction signing is more efficient both in terms of computation cost and storage by at least an order of magnitude.

As discussed above, Hülsing, Rijneveld and Schwabe [44] showed a practical evaluation of the SPHINCS hash-based signature scheme [14] on a Class 2 device. At first glance this implementation could also serve our purposes, however our proposed construction, while stateful, is much cheaper in terms of runtime, storage and communication costs, without such additional assumptions. We directly compare with their scheme in Appendix B.5.

Kumar et al. [52] propose an integrated confidentiality and integrity solution for large-scale IoT systems, which relies on an identity-based encryption scheme that can distribute keys in a hierarchical manner. This solution also uses similar techniques to our work for signature optimization for resource-constrained devices, however, it requires synchronicity between the system participants. Portunes [56] is tailored for preserving privacy (which is not within our main goals in our setting), and requires multiple rounds of communication (while we consider a "broadcast-only" setting)

Wander et al. [83] quantified the energy costs of RSA and Elliptic Curve operations as public key cryptography algorithms in resource-constrained devices. In a similar context, Potlapally et al. [71] performed a comprehensive analysis of several cryptographic algorithms for battery-powered embedded systems. However as discussed in Section 6.2, we consider hash-based algorithms that are lighter and more efficient.

Finally we mention an extensive IoT authentication survey [33]. In this work, our authentication scheme is comparable to [11] which utilizes hashing for one-way authentication in a distributed architecture, however our scheme is more storage-efficient, suited for low-SWaP (Class 0) sensors.

7 CONCLUSIONS

In this paper we designed and implemented BBox-IoT, a blockchain inspired approach for Industrial IoT sensors aiming at offering a transparent and immutable system for sensing and control information exchanged between IIoT sensors and aggregators. Our approach guarantees blockchain-derived properties to even low-Size Weight and Power (SWaP) devices. Moreover, BBox-IoT acts as a "black-box" that empowers the operators of any IoT system to detect data and sensor tampering ferreting out attacks against even SWaP devices. We posit that enabling data auditing and security at the lowest sensing level will be highly beneficial to critical infrastructure environments with sensors from multiple vendors.

Finally, we envision that our approach will be implemented during the sensor manufacturing stage: having industrial sensors shipped with pre-computed pebbles and their key material labeled using QR-code on the sensor body will allow for a seamless and practical deployment of BBox-IoT.

ACKNOWLEDGMENTS

Foteini Baldimtsi and Panagiotis Chatzigiannis were supported by NSA 204761.

REFERENCES

[1] 2019. Arduino Uno Rev3. https://store.arduino.cc/usa/arduino-uno-rev3

[2] 2020. Cisco Annual Internet Report . https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html

[3] N. Z. Aitzhan and D. Svetinovic. 2018. Security and Privacy in Decentralized Energy Trading Through Multi-Signatures, Blockchain and Anonymous Messaging Streams. *IEEE Transactions on Dependable and Secure Computing* 15, 5 (Sep. 2018), 840–852. https://doi.org/10.1109/TDSC.2016.2616861

[4] M. Ali, M. Vecchio, M.Pincheira, K. Dolui, F. Antonelli, and M. Rehmani. 2019. Applications of Blockchains in the Internet of Things: A Comprehensive Survey. *IEEE Communications Surveys and Tutorials* 21, 2 (2019), 1676–1717. https://doi.org/10.1109/COMST.2018.2886932

[5] Riham AlTawy and Guang Gong. 2019. Mesh: A Supply Chain Solution with Locally Private Blockchain Transactions. *PoPETs* 2019, 3 (2019), 149–169. https://doi.org/10.2478/popets-2019-0041

[6] Dorian Amiet, Andreas Curiger, and Paul Zbinden. 2018. FPGA-based Accelerator for SPHINCS-256. *IACR TCHES* 2018, 1 (2018), 18–39. https://doi.org/10.13154/tches.v2018.i1.18-39 https://tches.iacr.org/index.php/TCHES/article/view/831.

[7] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich, S. Muralidharan, C. Murthy, B. Nguyen, M. Sethi, G. Singh, K. Smith, A. Sorniotti, C. Stathakopoulou, M. Vukolic, S. Weed Cocco, and J. Yellick. 2018. Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains. *CoRR* abs/1801.10228 (2018). http://arxiv.org/abs/1801.10228

[8] P. L. Aublin, S. B. Mokhtar, and V. Quéma. 2013. RBFT: Redundant Byzantine Fault Tolerance. In *2013 IEEE 33rd International Conference on Distributed Computing Systems*. 297–306. https://doi.org/10.1109/ICDCS.2013.53

[9] L. Bai, M. Hu, M. Liu, and J. Wang. 2019. BPIIoT: A Light-Weighted Blockchain-Based Platform for Industrial IoT. *IEEE Access* 7 (2019), 58381–58393. https://doi.org/10.1109/ACCESS.2019.2914223

[10] Foteini Baldimtsi, Jan Camenisch, Maria Dubovitskaya, Anna Lysyanskaya, Leonid Reyzin, Kai Samelin, and Sophia Yakoubov. 2017. Accumulators with Applications to Anonymity-Preserving Revocation. Cryptology ePrint Archive, Report 2017/043. http://eprint.iacr.org/2017/043.

[11] Omaimah Omar Bamasag and Kamal Youcef-Toumi. 2015. Towards Continuous Authentication in Internet of Things Based on Secret Sharing Scheme. In *Proceedings of the 10th Workshop on Embedded Systems Security, WESS 2015, Amsterdam, The Netherlands, October 8, 2015*, Stavros A. Koubias and Thilo Sauter (Eds.). ACM, 1. https://doi.org/10.1145/2818362.2818363

[12] Shehar Bano, Alberto Sonnino, Mustafa Al-Bassam, Sarah Azouvi, Patrick McCorry, Sarah Meiklejohn, and George Danezis. 2017. Consensus in the Age of Blockchains. *CoRR* abs/1711.03936 (2017). arXiv:1711.03936 http://arxiv.org/abs/1711.03936

[13] Jonathan Bell, Thomas D. LaToza, Foteini Baldimtsi, and Angelos Stavrou. 2017. Advancing Open Science with Version Control and Blockchains. In *12th IEEE/ACM International Workshop on Software Engineering for Science, SE4Science@ICSE 2017, Buenos Aires, Argentina, May 22, 2017*. IEEE, 13–14. https://doi.org/10.1109/SE4Science.2017.11

[14] Daniel J. Bernstein, Daira Hopwood, Andreas Hülsing, Tanja Lange, Ruben Niederhagen, Louiza Papachristodoulou, Michael Schneider, Peter Schwabe, and Zooko Wilcox-O'Hearn. 2015. SPHINCS: Practical Stateless Hash-Based Signatures. In *EUROCRYPT 2015, Part I (LNCS, Vol. 9056)*, Elisabeth Oswald and Marc Fischlin (Eds.). Springer, Heidelberg, 368–397. https://doi.org/10.1007/978-3-662-46800-5_15

[15] Daniel J. Bernstein, Andreas Hülsing, Stefan Kölbl, Ruben Niederhagen, Joost Rijneveld, and Peter Schwabe. 2019. The SPHINCS$^+$ Signature Framework. In *ACM CCS 2019*, Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz (Eds.). ACM Press, 2129–2146. https://doi.org/10.1145/3319535.3363229

[16] Alysson Neves Bessani, João Sousa, and Eduardo Adílio Pelinson Alchieri. 2014. State Machine Replication for the Masses with BFT-SMART. In *44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2014, Atlanta, GA, USA, June 23-26, 2014*. IEEE Computer Society, 355–362. https://doi.org/10.1109/DSN.2014.43

[17] Arjun Nitin Bhagoji, Supriyo Chakraborty, Prateek Mittal, and Seraphin B. Calo. 2019. Analyzing Federated Learning through an Adversarial Lens. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA (Proceedings of Machine Learning Research, Vol. 97)*, Kamalika Chaudhuri and Ruslan Salakhutdinov (Eds.). PMLR, 634–643. http://proceedings.mlr.press/v97/bhagoji19a.html

[18] C. Bormann, M. Ersue, and A. Keranen. 2014. *Terminology for Constrained-Node Networks*. RFC 7228. RFC Editor.

[19] Mic Bowman and Camille Morhardt. 2018. Blockchain Must Adapt to Build Trust in the Internet of Things. https://www.coindesk.com/blockchain-must-adapt-build-trust-internet-things/ Retrieved June 24, 2018.

[20] Kyle Boyer, Laura Brubaker, Kyle Everly, RIchard Herriman, Paul Houston, Sean Ruckle, Rory Scobie, and Ian Ulanday. 2017. A distributed sensor network for an off-road racing vehicle. International Foundation for Telemetering.

[21] Johannes Buchmann, Erik Dahmen, Sarah Ereth, Andreas Hülsing, and Markus Rückert. 2011. On the Security of the Winternitz One-Time Signature Scheme. Cryptology ePrint Archive, Report 2011/191. http://eprint.iacr.org/2011/191.

[22] Johannes A. Buchmann, Erik Dahmen, and Andreas Hülsing. 2011. XMSS - A Practical Forward Secure Signature Scheme Based on Minimal Security Assumptions. In *Post-Quantum Cryptography - 4th International Workshop, PQCrypto 2011*, Bo-Yin Yang (Ed.). Springer, Heidelberg, 117–129. https://doi.org/10.1007/978-3-642-25405-5_8

[23] Joseph Bugeja, Paul Davidsson, and Andreas Jacobsson. 2018. Functional Classification and Quantitative Analysis of Smart Connected Home Devices. In *2018 Global Internet of Things Summit, GIoTS 2018, Bilbao, Spain, June 4-7, 2018*. IEEE, 1–6. https://doi.org/10.1109/GIOTS.2018.8534563

[24] Christian Cachin and Marko Vukolic. 2017. Blockchain Consensus Protocols in the Wild. *CoRR* abs/1707.01873 (2017). arXiv:1707.01873 http://arxiv.org/abs/1707.01873

[25] Jan Camenisch and Anna Lysyanskaya. 2002. Dynamic Accumulators and Application to Efficient Revocation of Anonymous Credentials. In *CRYPTO 2002 (LNCS, Vol. 2442)*, Moti Yung (Ed.). Springer, Heidelberg, 61–76. https://doi.org/10.1007/3-540-45708-9_5

[26] Miguel Castro and Barbara Liskov. 1999. Practical Byzantine Fault Tolerance. In *OSDI '99* (New Orleans, Louisiana, USA). USENIX Association, Berkeley, CA, USA. http://dl.acm.org/citation.cfm?id=296806.296824

[27] Konstantinos Chalkias, James Brown, Mike Hearn, Tommy Lillehagen, Igor Nitto, and Thomas Schroeter. 2018. Blockchained Post-Quantum Signatures. Cryptology ePrint Archive, Report 2018/658. https://eprint.iacr.org/2018/658.

[28] Nikos Chondros, Konstantinos Kokordelis, and Mema Roussopoulos. 2012. On the Practicality of Practical Byzantine Fault Tolerance. In *ACM/IFIP/USENIX 13th International Middleware Conference (Lecture Notes in Computer Science, Vol. 7662)*, Priya Narasimhan and Peter Triantafillou (Eds.). Springer, 436–455. https://doi.org/10.1007/978-3-642-35170-9_22

[29] Erik Dahmen and Christoph Krauß. 2009. Short Hash-Based Signatures for Wireless Sensor Networks. In *CANS 09 (LNCS, Vol. 5888)*, Juan A. Garay, Atsuko Miyaji, and Akira Otsuka (Eds.). Springer, Heidelberg, 463–476.

[30] George Danezis and Sarah Meiklejohn. 2016. Centrally Banked Cryptocurrencies. In *NDSS 2016*. The Internet Society.

[31] Hasan Derhamy, Jens Eliasson, Jerker Delsing, and Peter Priller. 2015. A survey of commercial frameworks for the internet of things. In *ETFA 2015*. IEEE.

[32] Ali Dorri, Salil S. Kanhere, and Raja Jurdak. 2016. Blockchain in internet of things: Challenges and Solutions. *CoRR* abs/1608.05187 (2016). arXiv:1608.05187 http://arxiv.org/abs/1608.05187

[33] Mohammed El-hajj, Ahmad Fadlallah, Maroun Chamoun, and Ahmed Serhrouchni. 2019. A Survey of Internet of Things (IoT) Authentication Schemes. *Sensors* 19, 5 (2019), 1141. https://doi.org/10.3390/s19051141

[34] A. Elsts, X. Fafoutis, S. Duquennoy, G. Oikonomou, R. J. Piechocki, and I. Craddock. 2018. Temperature-Resilient Time Synchronization for the Internet of Things. *IEEE Trans. Industrial Informatics* 14, 5 (2018), 2241–2250. https://doi.org/10.1109/TII.2017.2778746

[35] Mohamed Amine Ferrag, Makhlouf Derdour, Mithun Mukherjee, Abdelouahid Derhab, Leandros A. Maglaras, and Helge Janicke. 2019. Blockchain Technologies for the Internet of Things: Research Issues and Challenges. *IEEE Internet Things J.* 6, 2 (2019), 2188–2204. https://doi.org/10.1109/JIOT.2018.2882794

[36] Michael J. Fischer, Nancy A. Lynch, and Mike Paterson. 1985. Impossibility of Distributed Consensus with One Faulty Process. *J. ACM* 32, 2 (1985), 374–382. https://doi.org/10.1145/3149.214121

[37] Juan A. Garay and Aggelos Kiayias. 2020. SoK: A Consensus Taxonomy in the Blockchain Era. In *CT-RSA 2020 (LNCS, Vol. 12006)*, Stanislaw Jarecki (Ed.). Springer, Heidelberg, 284–318. https://doi.org/10.1007/978-3-030-40186-3_13

[38] O. Garcia-Morchon, R. Rietman, S. Sharma, L. Tolhuizen, and J.L. Torre-Arce. 2015. A Comprehensive and Lightweight Security Architecture to Secure the IoT Throughout the Lifecycle of a Device Based on HIMMO. In *ALGOSENSORS 2015* (Patras, Greece). Springer-Verlag, Berlin, Heidelberg, 112–128. https://doi.org/10.1007/978-3-319-28472-9_9

[39] Christian Gorenflo, Stephen Lee, Lukasz Golab, and Srinivasan Keshav. 2019. FastFabric: Scaling Hyperledger Fabric to 20, 000 Transactions per Second. *CoRR* abs/1901.00910 (2019). arXiv:1901.00910 http://arxiv.org/abs/1901.00910

[40] Tianyu Gu, Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg. 2019. Badnets: Evaluating backdooring attacks on deep neural networks. *IEEE Access* 7 (2019), 47230–47244.

[41] Yih-Chun Hu, Markus Jakobsson, and Adrian Perrig. 2005. Efficient Constructions for One-Way Hash Chains. In *ACNS 05 (LNCS, Vol. 3531)*, John Ioannidis, Angelos Keromytis, and Moti Yung (Eds.). Springer, Heidelberg, 423–441. https://doi.org/10.1007/11496137_29

[42] Andreas Hülsing. 2013. W-OTS+ - Shorter Signatures for Hash-Based Signature Schemes. In *AFRICACRYPT 13 (LNCS, Vol. 7918)*, Amr Youssef, Abderrahmane Nitaj, and Aboul Ella Hassanien (Eds.). Springer, Heidelberg, 173–188. https://doi.org/10.1007/978-3-642-38553-7_10

[43] Andreas Hülsing, Lea Rausch, and Johannes Buchmann. 2017. Optimal Parameters for XMSS^{MT}. Cryptology ePrint Archive, Report 2017/966. http://eprint.iacr.org/2017/966.

[44] Andreas Hülsing, Joost Rijneveld, and Peter Schwabe. 2016. ARMed SPHINCS - Computing a 41 KB Signature in 16 KB of RAM. In *PKC 2016, Part I (LNCS, Vol. 9614)*, Chen-Mou Cheng, Kai-Min Chung, Giuseppe Persiano, and Bo-Yin Yang (Eds.). Springer, Heidelberg, 446–470. https://doi.org/10.1007/978-3-662-49384-7_17

[45] Hyperledger. 2018. Hyperledger Architecture Volumes 1 and 2. https://www.hyperledger.org/wp-content/uploads/2017/08/Hyperledger_Arch_WG_Paper_1_Consensus.pdf

[46] Hyperledger. 2018. HyperLedger Fabric Documentation: Consensus Algorithm, Release 0.6. https://fabricdocs.readthedocs.io/en/origin-v0.6/FAQ/consensus_FAQ.html

[47] Markus Jakobsson. 2002. Fractal hash sequence representation and traversal. In *Information Theory, 2002.* IEEE, 437.

[48] Panos Kampanakis and Scott Fluhrer. 2017. LMS vs XMSS: A comparison of the Stateful Hash-Based Signature Proposed Standards. Cryptology ePrint Archive, Report 2017/349. http://eprint.iacr.org/2017/349.

[49] Fatma Karray, Mohamed Wassim Jmal, Alberto García Ortiz, Mohamed Abid, and Abdulfattah Mohammad Obeid. 2018. A comprehensive survey on wireless sensor node hardware platforms. *Comput. Networks* 144, 89–110. https://doi.org/10.1016/j.comnet.2018.05.010

[50] Jonathan Katz and Yehuda Lindell. 2014. *Introduction to Modern Cryptography, Second Edition* (2nd ed.). Chapman & Hall/CRC.

[51] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. 2017. Ouroboros: A Provably Secure Proof-of-Stake Blockchain Protocol. In *CRYPTO 2017, Part I (LNCS, Vol. 10401)*, Jonathan Katz and Hovav Shacham (Eds.). Springer, Heidelberg, 357–388. https://doi.org/10.1007/978-3-319-63688-7_12

[52] Sam Kumar, Yuncong Hu, Michael P. Andersen, Raluca Ada Popa, and David E. Culler. 2019. JEDI: Many-to-Many End-to-End Encryption and Key Delegation for IoT. *CoRR* abs/1905.13369 (2019). arXiv:1905.13369 http://arxiv.org/abs/1905.13369

[53] Leslie Lamport. 1979. *Constructing Digital Signatures from a One-way Function.* Technical Report SRI-CSL-98. SRI International Computer Science Laboratory.

[54] Leslie Lamport. 1981. Password Authentication with Insecure Communication. *Commun. ACM* 24, 11 (Nov. 1981), 770–772. https://doi.org/10.1145/358790.358797

[55] Leslie Lamport, Robert Shostak, and Marshall Pease. 1982. The Byzantine Generals Problem. *ACM Trans. Program. Lang. Syst.* 4, 3 (July 1982), 382–401. https://doi.org/10.1145/357172.357176

[56] Hongyang Li, György Dán, and Klara Nahrstedt. 2014. Portunes: Privacy-preserving fast authentication for dynamic electric vehicle charging. In *2014 IEEE International Conference on Smart Grid Communications, SmartGridComm 2014, Venice, Italy, November 3-6, 2014.* IEEE, 920–925. https://doi.org/10.1109/SmartGridComm.2014.7007766

[57] X. Liang, J. Zhao, S. Shetty, and D. Li. 2017. Towards data assurance and resilience in IoT using blockchain. In *MILCOM 2017 - 2017 IEEE Military Communications Conference (MILCOM).* 261–266. https://doi.org/10.1109/MILCOM.2017.8170858

[58] Li Lin, Xiaofei Liao, Hai Jin, and Peng Li. 2019. Computation offloading toward edge computing. *Proc. IEEE* 107, 8 (2019), 1584–1607.

[59] Yehuda Lindell, Anna Lysyanskaya, and Tal Rabin. 2004. On the Composition of Authenticated Byzantine Agreement. Cryptology ePrint Archive, Report 2004/181. http://eprint.iacr.org/2004/181.

[60] Donggang Liu and Peng Ning. 2003. Efficient Distribution of Key Chain Commitments for Broadcast Authentication in Distributed Sensor Networks. In *NDSS 2003.* The Internet Society.

[61] Jing Liu, Yang Xiao, and Jingcheng Gao. 2014. Achieving Accountability in Smart Grid. *IEEE Systems Journal* 8, 2 (2014), 493–508. https://doi.org/10.1109/JSYST.2013.2260697

[62] Ralph C. Merkle. 1988. A Digital Signature Based on a Conventional Encryption Function. In *CRYPTO'87 (LNCS, Vol. 293)*, Carl Pomerance (Ed.). Springer, Heidelberg, 369–378. https://doi.org/10.1007/3-540-48184-2_32

[63] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. 2016. The Honey Badger of BFT Protocols. In *ACM CCS 2016*, Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi (Eds.). ACM Press, 31–42. https://doi.org/10.1145/2976749.2978399

[64] D. Miller. 2018. Blockchain and the Internet of Things in the Industrial Sector. *IT Professional* 20, 3 (May 2018), 15–18. https://doi.org/10.1109/MITP.2018.032501742

[65] Satoshi Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system," http://bitcoin.org/bitcoin.pdf.

[66] O. Novo. 2018. Blockchain Meets IoT: An Architecture for Scalable Access Management in IoT. *IEEE Internet of Things Journal* 5, 2 (April 2018), 1184–1195. https://doi.org/10.1109/JIOT.2018.2812239

[67] Diego Ongaro and John K. Ousterhout. 2014. In Search of an Understandable Consensus Algorithm. In *2014 USENIX Annual Technical Conference, USENIX ATC '14, Philadelphia, PA, USA, June 19-20, 2014.*, Garth Gibson and Nickolai Zeldovich (Eds.). USENIX Association, 305–319. https://www.usenix.org/conference/atc14/technical-sessions/presentation/ongaro

[68] Muslum Ozgur Ozmen and Attila A. Yavuz. 2017. Low-Cost Standard Public Key Cryptography Services for Wireless IoT Systems. In *Proceedings of the 2017 Workshop on Internet of Things Security and Privacy, IoT S&P@CCS,*, Peng Liu, Yuqing Zhang, Theophilus Benson, and Srikanth Sundaresan (Eds.). ACM, 65–70. https://doi.org/10.1145/3139937.3139940

[69] A. Perrig, R. Canetti, J. D. Tygar, and D. Song. 2000. Efficient authentication and signing of multicast streams over lossy channels. In *IEEE S & P 2000.* 56–73. https://doi.org/10.1109/SECPRI.2000.848446

[70] Adrian Perrig, Ran Canetti, J. D. Tygar, and Dawn Song. 2002. The TESLA Broadcast Authentication Protocol.

[71] Nachiketh R. Potlapally, Srivaths Ravi, Anand Raghunathan, and Niraj K. Jha. 2003. Analyzing the energy consumption of security protocols. In *Proceedings of the 2003 International Symposium on Low Power Electronics and Design, 2003, Seoul, Korea, August 25-27, 2003*, Ingrid Verbauwhede and Hyung Roh (Eds.). ACM, 30–35. https://doi.org/10.1145/871506.871518

[72] Christos Profentzas, Magnus Almgren, and Olaf Landsiedel. 2019. IoTLogBlock: Recording Off-line Transactions of Low-Power IoT Devices Using a Blockchain. In *44th IEEE Conference on Local Computer Networks, LCN 2019, Osnabrueck, Germany, October 14-17, 2019*, Karl Andersson, Hwee-Pink Tan, and Sharief Oteafy (Eds.). IEEE, 414–421. https://doi.org/10.1109/LCN44214.2019.8990728

[73] Leonid Reyzin and Natan Reyzin. 2002. Better than BiBa: Short One-Time Signatures with Fast Signing and Verifying. In *ACISP 02 (LNCS, Vol. 2384)*, Lynn Margaret Batten and Jennifer Seberry (Eds.). Springer, Heidelberg, 144–153. https://doi.org/10.1007/3-540-45450-0_11

[74] R. Rodrigues, B. Liskov, K. Chen, M. Liskov, and D. Schultz. 2012. Automatic Reconfiguration for Large-Scale Reliable Storage Systems. *IEEE Trans. Dependable Sec. Comput.* 9, 2 (2012), 145–158. https://doi.org/10.1109/TDSC.2010.52

[75] M. Samaniego and R. Deters. 2017. Internet of Smart Things - IoST: Using Blockchain and CLIPS to Make Things Autonomous. In *IEEE ICCC 2017.* 9–16. https://doi.org/10.1109/IEEE.ICCC.2017.9

[76] Hossein Shafagh, Lukas Burkhalter, Anwar Hithnawi, and Simon Duquennoy. 2017. Towards Blockchain-based Auditable Storage and Sharing of IoT Data. In *Proceedings of the 2017 on Cloud Computing Security Workshop* (Dallas, Texas, USA) *(CCSW '17).* ACM, New York, NY, USA, 45–50. https://doi.org/10.1145/3140649.3140656

[77] J. Sousa, Al. Bessani, and M. Vukolic. 2018. A Byzantine Fault-Tolerant Ordering Service for the Hyperledger Fabric Blockchain Platform. In *DSN 2018*. IEEE Computer Society, 51–58. https://doi.org/10.1109/DSN.2018.00018

[78] P. Tenti, H. K. M. Paredes, and P. Mattavelli. 2011. Conservative Power Theory, a Framework to Approach Control and Accountability Issues in Smart Microgrids. *IEEE Transactions on Power Electronics* 26, 3 (March 2011), 664–673. https://doi.org/10.1109/TPEL.2010.2093153

[79] Francisco Tirado-Andrés, Alba Rozas, and Álvaro Araujo. 2019. A Methodology for Choosing Time Synchronization Strategies for Wireless IoT Networks. *Sensors* 19, 16, 3476. https://doi.org/10.3390/s19163476

[80] Meltem Sönmez Turan, Kerry A McKay, Çağdaş Çalık, Donghoon Chang, and Larry Bassham. 2019. Status report on the first round of the NIST lightweight cryptography standardization process. (2019).

[81] Marko Vukolic. 2015. The Quest for Scalable Blockchain Fabric: Proof-of-Work vs. BFT Replication. In *IFIP WG 11.4 International Workshop, iNetSec 2015 (Lecture Notes in Computer Science, Vol. 9591)*, Jan Camenisch and Dogan Kesdogan (Eds.). Springer, 112–125. https://doi.org/10.1007/978-3-319-39028-4_9

[82] J. Wan, J. Li, M. Imran, and D. Li. 2019. A Blockchain-Based Solution for Enhancing Security and Privacy in Smart Factory. *IEEE Transactions on Industrial Informatics* (June 2019). https://doi.org/10.1109/TII.2019.2894573

[83] Arvinderpal Wander, Nils Gura, Hans Eberle, Vipul Gupta, and Sheueling Chang Shantz. 2005. Energy Analysis of Public-Key Cryptography for Wireless Sensor Networks. In *3rd IEEE International Conference on Pervasive Computing and Communications (PerCom 2005), 8-12 March 2005, Kauai Island, HI, USA*. IEEE Computer Society, 324–328. https://doi.org/10.1109/PERCOM.2005.18

[84] Gang Wang, Zhijie Shi, Mark Nixon, and Song Han. 2019. ChainSplitter: Towards Blockchain-Based Industrial IoT Architecture for Supporting Hierarchical Storage. In *IEEE International Conference on Blockchain, 2019*. IEEE, 166–175. https://doi.org/10.1109/Blockchain.2019.00030

[85] Qiyan Wang, Himanshu Khurana, Ying Huang, and Klara Nahrstedt. 2009. Time Valid One-Time Signature for Time-Critical Multicast Data Authentication. In *INFOCOM 2009. 28th IEEE International Conference on Computer Communications, Joint Conference of the IEEE Computer and Communications Societies, 19-25 April 2009, Rio de Janeiro, Brazil*. IEEE, 1233–1241. https://doi.org/10.1109/INFCOM.2009.5062037

[86] Gavin Wood. 2018. Ethereum: A secure decentralized generalised transaction ledger. https://ethereum.github.io/yellowpaper/paper.pdf Accessed: 2020-07-18.

[87] L. Wu, X. Du, W. Wang, and B. Lin. 2018. An Out-of-band Authentication Scheme for Internet of Things Using Blockchain Technology. In *2018 International Conference on Computing, Networking and Communications (ICNC)*. 769–773. https://doi.org/10.1109/ICCNC.2018.8390280

[88] Yulei Wu, Hong-Ning Dai, and Hao Wang. 2020. Convergence of Blockchain and Edge Computing for Secure and Scalable IIoT Critical Infrastructures in Industry 4.0. *IEEE Internet of Things Journal* (2020).

[89] Karl Wüst and Arthur Gervais. 2017. Do you need a Blockchain? Cryptology ePrint Archive, Report 2017/375. http://eprint.iacr.org/2017/375.

[90] Z. Xiong, Y. Zhang, D. Niyato, P. Wang, and Z. Han. 2018. When Mobile Blockchain Meets Edge Computing. *IEEE Communications Magazine* 56, 8 (August 2018), 33–39. https://doi.org/10.1109/MCOM.2018.1701095

[91] Z. Yang, K. Yang, L. Lei, K. Zheng, and V. C. M. Leung. 2019. Blockchain-Based Decentralized Trust Management in Vehicular Networks. *IEEE Internet of Things Journal* 6, 2 (April 2019), 1495–1505. https://doi.org/10.1109/JIOT.2018.2836144

[92] Dae Hyun Yum, Jae Woo Seo, Sungwook Eom, and Pil Joong Lee. 2009. Single-Layer Fractal Hash Chain Traversal with Almost Optimal Complexity. In *CT-RSA 2009 (LNCS, Vol. 5473)*, Marc Fischlin (Ed.). Springer, Heidelberg, 325–339. https://doi.org/10.1007/978-3-642-00862-7_22

[93] Han Zou, Yuxun Zhou, Jianfei Yang, and Costas J Spanos. 2018. Towards occupant activity driven smart buildings via WiFi-enabled IoT devices and deep learning. *Energy and Buildings* 177 (2018), 12–22.

A ON MACS FOR SENSOR AUTHENTICATION

One might suggest using MAC authentication in our scheme instead of one-time hash based signatures, which might be slightly more efficient in terms of computation cost for generating a signature, are simpler in usage and do not expire. The question of whether its preferable using symmetric primitives in resource-constrained IoT devices instead of public key cryptography has been raised in academic works [68], and several motivations to provide efficient public key cryptography techniques in such devices were outlined, most of which are also applicable to our system as follows.

Firstly, signatures provide non-repudiation, which as discussed previously is a needed security property S-4. Although a way to achieve non-repudiation through MACs could be to use a separate MAC key for each sensor, each key would need to be shared with each group aggregator separately since they should be all able to verify data from all sensors in the group. This would increase the attack surface since an attacker compromising any aggregator could also send bogus data for all sensors. Also considering that aggregators might have to verify data from a great number of sensors, our hash-based verification cost (which involves one hash operation) is cheaper than one MAC operation. Although for sensors a MAC operation is cheaper than a hash-based signature, as we show in section 5 a hash-based signature which involves a few hashes and a Quicksort operation is still relatively efficient even for the weakest types of sensors.

Secondly, our chain hash-based scheme has a built-in "replay protection" against an attacker, since that signature is by definition valid for one time only. A MAC scheme would require extra layers of protection (nonces and/or timers) against replay attacks.

Lastly, by using our hash-based signature scheme we enable public verifiability of signed sensor data on the blockchain, even by entities not authorized to participate in the system.

B CHAIN-BASED HASH SIGNATURES

B.1 Digital Signatures.

A digital signature scheme consists of the following algorithms [50]:

- $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{SignGen}(1^\lambda)$: Outputs a pair of keys $(\mathsf{pk}, \mathsf{sk})$.
- $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, m)$: Takes as input a private key sk and a message m and outputs a signature σ.
- $\mathsf{SVrfy}(\mathsf{pk}, m, \sigma) := b$: Takes as input a public key pk, a message m and a signature σ, and outputs a bit b where $b = 1$ indicates successful verification.

A digital signature is considered secure if an adversary $\mathcal{A}$ cannot forge a signature on a message even after adaptively receiving signatures on messages of its choice. To formalize the security definition we first describe the following experiment $\mathsf{SigForge}(\lambda)$:

(1) $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{SignGen}(1^\lambda)$
(2) $\mathcal{A}$ on input (pk) queries the signing oracle polynomial number of times q. Let $Q : [m_i, \sigma_i]_{i=1}^q$ be the set of all such queries.
(3) $\mathcal{A}$ outputs (m^*, σ^*).
(4) $\mathcal{A}$ wins if $\mathsf{SVrfy}(m^*, \sigma^*) = 1$ where $m^* \notin [m_i]_{i=1}^q$ and $\mathsf{SigForge}$ outputs "1", else it outputs "0".

Definition 1. *A digital signature scheme is existentially unforgeable under an adaptive chosen-message attack, if for all PPT $\mathcal{A}$,* $\Pr[\mathsf{SigForge}(\lambda) = 1]$ *is negligible in λ.*

B.2 One-time signatures

A digital signature scheme that can be used to sign only one message per key pair is called a one-time signature (OTS) scheme.

Definition 2. *A one-time digital signature scheme is existentially unforgeable under an adaptive chosen-message attack, if for all PPT $\mathcal{A}$ and for $q \leq 1$,* $\Pr[\mathsf{SigForge}(\lambda) = 1]$ *is negligible in λ.*

B.3 Hash functions.

An (unkeyed) hash function $y := h(m)$ on input of a message m outputs a digest y. A cryptographic hash function is considered secure if the probability to find collisions is negligible (i.e. it is *collision resistant*). More formally, we consider the following experiment $\mathsf{HashColl}$[50]:

(1) $\mathcal{A}$ picks values $x, x' \in \{0,1\}^*$ s.t. $x \neq x'$.
(2) $\mathcal{A}$ wins if $h(x) = h(x')$ and $\mathsf{HashColl}$ outputs "1".

Definition 3. *A hash function $h() : \{0,1\}^* \rightarrow \{0,1\}^\lambda$ is collision resistant if for all PPT $\mathcal{A}$,* $\Pr[\mathsf{HashColl} = 1]$ *is negligible.*

A weaker notion for security of a hash function is preimage-resistance. We consider the following experiment $\mathsf{PreIm}(\lambda, y)$:

(1) $\mathcal{A}$ is given $y \in \{0,1\}^\lambda$
(2) $\mathcal{A}$ outputs x.
(3) $\mathcal{A}$ wins if $h(x) = y$. If $\mathcal{A}$ wins PreIm outputs "1", else it outputs "0".

Definition 4. *A hash function $h() : \{0,1\}^* \rightarrow \{0,1\}^\lambda$ is preimage resistant if $\forall$ ppt $\mathcal{A}$ and $\forall y \in \{0,1\}^\lambda$,* $\Pr[\mathsf{PreIm}(\lambda, y) = 1]$ *is negligible in λ.*

Corollary 1. *A collision resistant hash function is also preimage resistant.*

B.4 Definition and Security proof

We first define the API of a chain based signature for a fixed number of messages n.

- $(\mathsf{pk}, \mathsf{sk_n}, s_0) \leftarrow \mathsf{OTKeyGen}(1^\lambda, n)$: Outputs a pair of keys $\mathsf{pk}, \mathsf{sk_n}$ and an initial state s_0, where $\mathsf{pk} = h^n(\mathsf{sk_n})$ and $h()$ is a collision resistant hash function.
- $(\sigma, \mathsf{sk_i}, s_i) \leftarrow \mathsf{OTSign}(\mathsf{sk_{i-1}}, m, s_{i-1})$: Takes as input the system state s_{i-1}, a private key $\mathsf{sk_{i-1}}$ and a message m, generates a signature σ and updates the signer's private key to $\mathsf{sk_i}$ where $\mathsf{sk_{i-1}} = h(\mathsf{sk_i})$ and his state to s_i where $i \leq n$.
- $\mathsf{OTVerify}(\mathsf{pk}, m, \sigma) := b$: Takes as input a public key pk, a message m and a signature σ, and outputs a bit b where $b = 1$ indicates successful verification.

To formalize security for chain-based signatures with length of chain n, we describe the following experiment $\mathsf{OTSigForge}(\lambda, n)$:

(1) $(\mathsf{pk}, \mathsf{sk_n}, s_0) \leftarrow \mathsf{OTKeyGen}(1^\lambda, n)$
(2) $\mathcal{A}$ on input (pk, n) makes up to $q \leq n$ queries to the signing oracle. Let $Q : [m_i, \sigma_i]_{i=1}^q$ the set of all such queries where m_i is the queried message and σ_i is the signature returned for m_i.

(3) $\mathcal{A}$ outputs (m_{q+1}, σ_{q+1}).
(4) $\mathcal{A}$ wins if $\mathsf{OTVerify}(\mathsf{pk}, m_{q+1}, \sigma_{q+1}) := 1$ and $h^i(\mathsf{sk_i}) \neq \mathsf{pk}$ $\forall i \leq q$ where $\mathsf{OTSigForge}$ outputs "1", else it outputs "0".

Note in the above experiment by $h^i(\mathsf{sk_i}) \neq \mathsf{pk}$ $\forall i \leq q$ we restrict $\mathcal{A}$ from winning the game by reusing a secret key $\mathsf{sk_i}$ existing in the chain up to distance q from the public key pk.

Definition 5. *A chain-based one-time digital signature scheme is existentially unforgeable under an adaptive chosen-message attack, if* $\forall$ *ppt* $\mathcal{A}$, $\Pr[\mathsf{OTSigForge}(\lambda, n) = 1]$ *is negligible in* λ.

Given the formal definition above we now prove the security of Construction 1.

Theorem 2. *Let* $h : \{0,1\}^* \rightarrow \{0,1\}^\lambda$ *be a preimage resistant hash function. Then Construction 1 is an existentially unforgeable chain-based one-time signature scheme.*

Proof. Let $\mathcal{A}$ be an adversary who wins the $\mathsf{OTSigForge}$ game described in Section 6.2 and therefore can forge signatures using the above scheme with non-negligible probability $p(\lambda)$. That is, $\exists$ $\mathcal{A}$ which after performing q queries $\{m_i\}_{i=1}^q$ where $q \leq n$, can output a signature σ_{q+1} for a message m_{q+1} where $\mathsf{OTVerify}(\mathsf{pk}, m_{q+1}, \sigma_{q+1}) = 1$ and $h^i(\mathsf{sk_i}) \neq \mathsf{pk}$ $\forall i \leq q$.

Then, an algorithm $\mathcal{B}$ running the PreIm experiment would use $\mathcal{A}$ to break preimage resistance of h as follows: On input (λ, y), $\mathcal{B}$ would generate a hash chain of length n with seed y as $(y, h(y), ..., h^n(y))$ and forward $(h^n(y), n)$ to $\mathcal{A}$. Then $\mathcal{A}$ makes up to $q \leq n$ queries to $\mathcal{B}$. When $\mathcal{A}$ queries for m_i (where i denotes the query number), $\mathcal{B}$ returns $\sigma_i = h(m_i || h^{n-i+1}(y)) || h^{n-i}(y)$ to $\mathcal{A}$. If $q = n$ and $\mathcal{A}$ does not output a forgery, $\mathcal{B}$ returns $\perp$ and starts over. If $\mathcal{A}$ eventually outputs a forgery (m^{q+1}, σ^{q+1}) to $\mathcal{B}$ and $q < n$, then $\mathcal{B}$ returns $\perp$ as output of PreIm experiment and starts over, else if $q = n$, $\mathcal{B}$ would parse $\sigma^{n+1} = \sigma^A || \sigma^B$ and return σ^B. Assuming a uniform probability distribution of the number of queries q, $\Pr[\mathsf{PreIm}(\lambda, y) = 1] = \frac{\Pr[\mathsf{OTSigForge}(\lambda, n) = 1]}{n} = \frac{p(\lambda)}{n}$ which is non-negligible.

□

B.5 Evaluation comparison with modified SPHINCS

As discussed in section 6.2, the modified SPHINCS scheme tailored for resource-constrained devices [44] could be a candidate scheme for our system. Here we make a direct comparison between modified SPHINCS and our proposed scheme for our system's purposes.

Assuming a hash chain length of 2^{26} elements (which as discussed is only exhausted after 21 years assuming generation of one signature every 10 seconds), a signature generation only requires 27 hashing operations in the worst case, which according to our measurements on a 8-bit 16Mhz CPU Arduino device outlined in Section 5.3, would only need 50 ms on average. On the other hand, modified SPHINCS' evaluation performed on a resource-constrained device (32-bit 32Mhz Cortex M3 which is more powerful than our Arduino Uno R3) needs 22.81 seconds for signature generation. Also our signature size (excluding the payload) is only 64 bytes for the signature and the program storage requirement 1082 bytes, while modified SPHINCS generates a 41KB signature, streamed serially. Our only additional requirement is an initial precomputation phase using a powerful device, which will have to pre-compute the 2^{26} hash chain elements and then send the "pebbles" to the resource-constrained device.

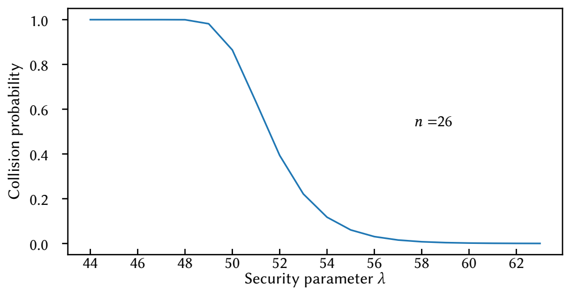


Figure 6: Collision probability for hash chain length 2^{26}.

B.6 Collision probability analysis

Although we assumed a collision resistant hash function in our hash-based signature construction, given the length of the hash chain (typical length 2^{26}) there is an increased likelihood of a collision along that chain through the birthday paradox (especially for lower levels of security where the output size of the hash function is small), which would result in "cycles" of hashes. If such cycles occur, an adversary could then trivially break the security of our scheme and sign bogus sensor data.

Assume a hash chain of length 2^n and a security parameter λ. From the birthday paradox, the probability of a collision on the hash chain is approximated by $p = 1 - e^{\frac{-2^n(2^n-1)}{2^{\lambda+1}}}$. In Figure 6 we show that given a chain length of 2^{26} as previously discussed, the output size of the hash function $h()$ should be at least 64, and SHA256 which we used in our evaluations satisfies these requirements.

Nevertheless, if birthday attacks become an issue for a small security parameter, we can apply the technique from [41] where the chain index is used as salt to prevent such attacks for a small overhead in cost. However since we show that the birthday attack is negligible, we prefer to keep the costs as low as possible.

C CONSENSUS

C.1 Definitions

Definition 6. *Let parties* $[P_i]_{i=1}^n$*, each having a view of the blockchain* $\mathsf{BC}(i)$*, and receive a common sequence of messages in rounds* $[1..j..]$ *A protocol solves the ledger consensus problem if the following properties hold:*

i. Consistency: An honest node's view of the blockchain on some round j *is a prefix of an honest node's view of the blockchain on some round* $j+\ell, \ell > 0$*, or* $\mathsf{BC}(i)_j || \mathsf{B_{j+1}} || ... || \mathsf{B_{j+\ell}} = \mathsf{BC}(i')_{j+\ell}$*,* $\forall P_i, P_{i'}, j, \ell$*.*
ii. Liveness: An honest party P_i *on input of an operation (or transaction)* tx*, after a certain number of rounds will output a view of the blockchain* $\mathsf{BC}(i)$ *that includes* tx*.*

The protocol can be augmented with the existence of a TP assigning membership credentials (which requires an additional trusted setup phase) resulting in an *Authenticated* ledger consensus protocol [55][59]. Such a protocol consists of the following algorithms:

(1) $(\mathsf{pp}) \leftarrow \mathsf{TPSetup}(1^{\lambda})$: A trusted party TP generates the system paremeters pp.
(2) $(\mathsf{pk_i}, \mathsf{sk_i}) \leftarrow \mathsf{PartyGen}(\mathsf{pp})$: Each party i generates a public-private key pair.
(3) $\mathsf{TPMembers}(\widehat{[\mathsf{pk}]}) := [\mathsf{pk}]$: The TP chooses the protocol participants from a list of public keys $\widehat{[\mathsf{pk}]}$, and outputs a list of authenticated participant public keys $[\mathsf{pk}]$.
(4) $\mathsf{Consensus}([[\mathsf{pk_j}]_{j=1}^{n}, \mathsf{sk_i}, \mathsf{st_i}, \mathsf{BC}(i)]_{i=1}^{n}) := \mathsf{BC}'$: All system participants with state $\mathsf{st_i}$ and a view of the blockchain $\mathsf{BC}(i)]_{i=1}^{n}$), agree on a new blockchain BC'.

C.2 Byzantine Generals Problem

One of the first studies attempting to achieve agreement on a distributed synchronous system was proposed by Leslie Lamport in 1982 [55]. It proposed using either an "Oral message" solution to achieve binary consensus among $n = 3f + 1$ nodes, where a leader was sending a proposed binary value in a fully connected network, then the honest nodes would propagate that value using the same protocol acting as leaders themselves. Using authentication further improves the resilience of the protocol to $n = 2f + 1$, as Byzantine faults can be tolerated under the assumption that dishonest parties cannot forge the leader's signature. In both cases however, the communication complexity is $O(n^2)$, which is not considered scalable.

C.3 PBFT

The Practical Byzantine Fault Tolerance consensus protocol [26] was an exemplary protocol for many consensus protocols to follow. It assumes a partially asynchronous setting (i.e. offering *eventual synchrony*), where a message is assumed to arrive to the destination node in the network after some unknown but bounded time t. In each round, a leader orders messages and propagates them to all nodes in the network in three phases. The leader is defined by a sequence of "views", and the backup nodes can propose a leader (or view) change if he has faulty behavior (timeout exceeded). The protocol assumes $n = 3f + 1$ faulty nodes, and has communication complexity $O(n^2)$, which again is in practice not scalable for more than 20 nodes [81]. It also assumes static membership of participating nodes.

C.4 PBFT with dynamic clients

Chondros et al. [28] suggest modifying the original PBFT protocol by adding "Join" and "Leave" system requests. The "Join" request, equivalent to a "client" request in PBFT, would trigger the execution of the original PBFT protocol between the primary and the replicas, and replying back with a challenge to prevent DoS attacks. The client wishing to join, replies with a message containing its credentials and a nonce, which are then associated in an array containing the protocol participants. The protocol is based on fully synchronous assumptions. While the protocol offers dynamic membership, it has high communication overhead as PBFT, therefore it is not a suitable candidate for our BBox-IoT system.

C.5 System membership tracking consensus

Rodrigues et al. [74] propose a "Membership service" or a "Configuration management service" for achieving dynamic membership in the permissioned consensus setting. Following a "transaction endorsement - consensus nodes decoupling" paradigm, only a chosen subset of the participating nodes would execute a BFT-family protocol, while also being responsible for handling "add" or "remove" requests, pinging (or probing) for failed or crashed nodes and removing them after some time (or epochs), and keeping track of the epochs. Since these are only a small subset of the total nodes, the overall system remains scalable. Adding/removal is signed by a TP, which is compatible with our BBox-IoT architecture.

C.6 RSCoin

Designed as a cryptocurrency framework, RSCoin [30] deviates from the common practice of decentralizing monetary supply, which found in other cryptocurrencies. In RSCoin's architecture, a Trusted Party (the Bank) delegates its authority for validating transactions to known and semi-trusted "mintettes", which in turn are each responsible for interacting with a subset of the cryptocurrency's addresses, forming "shards". A basic consensus protocol based on Two-Phase Commit is executed between a group of mintettes and a client, which involves collecting signatures from the majority of mintettes responsible and then sending back the transaction to be included in a block. Also, since there is no direct communication between all mintettes, but they rather communicate indirectly through the clients, the communications complexity is very low which enables high scalability and performance. RSCoin could also potentially be used as a "consensus" replacement for BBox-IoT, however it would require extensive modifications because of its cryptocurrency-oriented architecture.

C.7 Hyperledger Frameworks

In the following paragraphs we summarize the properties of additional Hyperledger frameworks [45]:

- **Hyperledger Indy** uses the *Redundant Byzantine Fault Tolerance (RBFT)* consensus algorithm [8], which is voting-based. As its name suggests, it is based on the PBFT consensus algorithm, modified to execute many protocol instances in parallel for improving performance in the case of a faulty leader. It provides Byzantine fault tolerance and reaches consensus very fast, however the time scales with the number of nodes ($O(n^2)$ as in PBFT). An additional requirement is that the nodes must be totally connected.
- **Hyperledger Iroha** uses the *Sumeragi* consensus algorithm, based on a reputation system. As with RBFT, it provides Byzantine fault tolerance reaching consensus in very short time, but that time scales with the number of nodes, and the nodes must be totally connected.
- **Hyperledger Sawtooth** uses the novel *Proof of Elapsed Time (PoET)* consensus algorithm, which is lottery-based. It can be categorized in the "Proof-of-X" consensus family, by replacing proof of computational work with a proof of elapsed time, using trusted hardware (Intel SGX enclave). Each protocol participant would request a wait time from their trusted hardware, and the shortest would be elected

as the leader, by providing a proof that he indeed had the shortest wait time and the new block alongside the proof was not broadcasted until that time had expired [12]. While the algorithm is scalable and Byzantine fault tolerant, there is a possibility of delayed consensus due to forks. Also this algorithm would not suit to our BBox-IoT system, since we do not require our blockchain "maintainers" to have trusted hardware capabilities.

C.8 Additional Consensus properties

We outline the following additional consensus properties, which are desirable in our setting but not strictly required:

i. Byzantine Fault Tolerant: As previously discussed, the crash tolerant model of consensus does not take Byzantine behavior of nodes into account [24]. Although our system considers the consensus algorithm running among a closed, controlled set of nodes, it might be depoyable in a more uncontrolled environment, where Byzantine behavior is possible (Byzantine consensus)[55].
ii. No synchronicity assumptions: Due to [36], which excludes deterministic protocols from reaching consensus in a fully asynchronous system, to achieve synchronicity, we would have to use a randomized protocol, else we will have to assume "eventual synchrony" (i.e. protocol finishes within a fixed but unknown time bound) [63].
iii. Incentive-compatible: Protocol keeps nodes motivated to participate in the system and follow its rules [12].
iv. Minimal setup assumptions: No need for a trusted setup phase (as required by Authenticated Byzantine agreement discussed above).
v. Weaker adversarial model: While most classical consensus protocols require a 33% adversarial model (which we believe should be sufficient for our purposes), some protocols have a weaker requirement of 49% adversarial power. However this usually comes at the cost of sacrificing Byzantine tolerance (as we discussed above) or scalability in terms of operations per second [65].

C.9 An instantiation for Consensus algorithm

In the generic construction of our scheme, we assumed a "pluggable" consensus algorithm, decoupled from our construction, similar to the original Hyperledger architecture. Recall that this algorithm, which is executed among all orderers O_i, on input of a blockchain BC and some orderer state $\mathsf{st_i}$, outputs an agreed new updated blockchain BC'. Here we provide a concrete instantiation of a consensus algorithm for the modified Hyperledger used in BBox-IoT that matches the PBFT consensus protocol [26] as follows (note though that PBFT would not satisfy all of the required system properties as discussed in section 2.1):

(1) O_i parses TXL from its $\mathsf{st_{O_i}}$ extracting a set of transactions $\{\mathsf{tx}_i\}$.
(2) O_i based on the current BC and $\{\mathsf{tx}_i\}$ constructs a new block $\mathsf{B_i}$ which would create $\mathsf{BC}||\mathsf{B_i} \rightarrow \mathsf{BC}'$.
(3) O_i computes $\sigma := \mathsf{Sign}(\mathsf{sk_{O_i}}, \mathsf{B_i})$. and sends σ to all orderers in O (equivalent to "pre-prepare" phase in PBFT).
(4) All the other orderers $\mathsf{O}_x \in O$ parse $(\mathsf{OL_{BC}})$ from the output of $\mathsf{ReadConfig}(\mathsf{BC})$. Check if $\mathsf{pk_{O_i}} \in \mathsf{OL_{BC}}$ and $\mathsf{SVrfy}(\mathsf{pk_{O_i}}, \sigma, \mathsf{B_i}) = 1$. Then it verifies that the proposed block was formed correctly (i.e., it is a valid extension of the current blockchain BC). If all verifications pass, it computes $\sigma_x := \mathsf{Sign}(\mathsf{sk_{O_x}}, \mathsf{B_i})$ and sends σ_x to all orderers in O (equivalent to "prepare" phase in PBFT).
(5) Each $\mathsf{O}_x \in O$ (including O_i) checks if $\mathsf{SVrfy}(\mathsf{pk_{O_x}}, \sigma_x, \mathsf{B_i}) = 1$. If it collects sufficient number of signatures (specific to each consensus protocol) it computes $\sigma'_x := \mathsf{Sign}(\mathsf{sk_{O_x}}, \mathsf{B_i}, 1)$ and sends σ'_x to all orderers in O (equivalent to PBFT "commit" phase).
(6) Each $\mathsf{O}_x \in O$ (including O_i) checks if $\mathsf{SVrfy}(\mathsf{pk_{O_{x''}}}, \sigma'_x, \mathsf{B_i}, 1) = 1$. If it receives sufficient number of signatures (specific to each consensus protocol) it updates it state to $\mathsf{st_{O_x}}'$ and outputs "1". It outputs "0" in all other cases.

The above instantiation satisfied the basic consensus properties in Definition 6.

D CONSTRUCTION ALGORITHMS

For our construction we assume an existentially unforgeable signature scheme ($\mathsf{SignGen}$, Sign, SVrfy) and an unforgeable one-time chain based signature scheme as defined in Section 6.2 ($\mathsf{OTKeyGen}$, OTSign, $\mathsf{OTVerify}$). We also assume an authenticated blockchain consensus scheme ($\mathsf{TPSetup}$, $\mathsf{PartyGen}$, $\mathsf{TPMembers}$, $\mathsf{Consensus}$) satisfying the properties outlined in Section 2.1.

(1) $\mathsf{SystemInit}(1^\lambda, \mathsf{LL}, \mathsf{OL}, \mathsf{PL}, \mathsf{Pol})$ lets the MSP to initialize the BBox-IoT system. The initialization is optionally based on predetermined initial system participants, where $\mathsf{LL}, \mathsf{OL}, \mathsf{PL}$ are lists containing public keys for Local administrators, orderers and peers respectively, as well as a preselected system policy Pol.
  (a) MSP sets as pp the system parameters for the signature and the hash function, as well as the consensus algorithm by running $\mathsf{TPSetup}$.
  (b) Computes a random key pair $(\mathsf{pk_{MSP}}, \mathsf{sk_{MSP}}) \leftarrow \mathsf{SignGen}(1^\lambda)$.
  (c) Assembles and outputs the genesis block $\mathsf{B_0}$ (serving as the initial configuration block) by copying $\mathsf{pk_{MSP}}$, pp and $[\mathsf{LL}, \mathsf{OL}, \mathsf{PL}, \mathsf{Pol}]$ from the algorithm inputs.
  (d) Initializes empty lists in MSP memory $[\mathsf{LL_{MSP}}, \mathsf{OL_{MSP}}, \mathsf{PL_{MSP}}, \mathsf{Pol}, \mathsf{oper}]$ where oper denotes a pending revoke operation list.
  (e) Copies Pol to $\mathsf{Pol_{MSP}}$.
  The genesis block $\mathsf{B_0}$ (as the blockchain BC in general) is public, while the rest of the outputs remain private to MSP. For the following algorithms and protocols we assume that the security parameter and the system parameters are a default input.
(2) $\mathsf{ConfigUpdate}(\mathsf{BC}, \mathsf{sk_{MSP}}, \mathsf{st_{MSP}})$ enables MSP to read system configuration information from its memory that is pending to be updated, and construct a new configuration block to make the new system configuration readable and valid in the blockchain by all system participants.
  (a) MSP parses $\mathsf{st_{MSP}}$ as $[\mathsf{LL_{MSP}}, \mathsf{OL_{MSP}}, \mathsf{PL_{MSP}}, \mathsf{Pol_{MSP}}]$.

Algorithm	Adversarial model	Byzantine tolerant	Dynamic membership	Scalable	DoS resistant	Notes
PBFT	$3f+1$	✓	✗	✗	Semi	Hyperledger Fabric v0.6
Kafka	$2f+1$	✗	✓	✓	✗	Hyperledger Fabric v1.4
BFT-SMaRt	$3f+1$	✓	✓	Semi	✓	
Nakamoto consensus	$2f+1$	✓	Permissionless	✓	✓	

Table 7: Consensus algorithm comparison

Table 8: Hash-based schemes concrete comparison, 256-bit security

Scheme	Stateful	Public key (bytes)	Secret key (bytes)	Signature (bytes)	Sign (msec)	Verify (msec)	Remarks
XMSS	Yes	68		4963	610	160	Cortex M3 32MHz 32-bit 16KB RAM [43, 48]
SPHINCS	No	1056	1088	41000	18410	513	Cortex M3 32MHz 32-bit 16KB RAM [44]
Our scheme	Yes	32	32	32 (64)	52	0.035	ATmega328P 16MHz 8-bit 2KB RAM

(b) Assembles a configuration update transaction $\mathsf{tx}_u = [\mathsf{LL_{MSP}}, \mathsf{OL_{MSP}}, \mathsf{PL_{MSP}}, \mathsf{Pol_{MSP}}]$.
(c) Parses $\mathsf{ReadConfig(BC)}$ as $\mathsf{PL_{BC}}$.
(d) Sends signed transaction $\sigma_{\mathsf{MSP}}(\mathsf{tx}_u)$ to all $\mathsf{Ag_i} \in \mathsf{PL_{BC}}$.

Since Pol does not apply to transactions signed by MSP, the configuration update transaction is promptly appended to BC by all aggregators, resulting in public output BC'.

(3) $\mathsf{PolicyUpdate(st_{MSP}, Pol)}$ enables MSP to update system policy parameters. On input of a new system policy Pol, MSP copies it to $\mathsf{st_{MSP}[Pol]}$, overwriting the previous policy. The algorithm outputs the new updated $\mathsf{st_{MSP}}'$.

(4) $\mathsf{ReadConfig(BC)}$ can be run by any system participant to recover the current system configuration.
(a) Parses BC as a series of blocks $\mathsf{B_i}$.
(b) From the set of blocks marked as "configuration" blocks where $\mathsf{B_i}[type = "C"]$, selects the block $\mathsf{B_c}$ with the greatest height c.
(c) Parses and outputs $\mathsf{B_c}$ as $([\mathsf{LL_{BC}}, \mathsf{OL_{BC}}, \mathsf{PL_{BC}}], \mathsf{Pol_{BC}})$.

(5) $\mathsf{OrdererSetup()}$ is run by an orderer O_i initializing its credentials and state. It computes and outputs signing keys as $(\mathsf{pk_{O_i}}, \mathsf{sk_{O_i}}) \leftarrow \mathsf{SignGen}(1^\lambda)$ and initializes an signed transaction list in memory $\mathsf{st_{O_i}}[TXL]$.

(6) $\mathsf{OrdererAdd}\{\mathsf{O}_i(\mathsf{pk_{O_i}}, \mathsf{sk_{O_i}}) \leftrightarrow \mathsf{MSP}(\mathsf{pk_{MSP}}, \mathsf{sk_{MSP}}, \mathsf{st_{MSP}}[\mathsf{OL_{MSP}}], \mathsf{BC})\}$ is an interactive protocol between an orderer O_i and the system MSP in order to add that orderer in the system:
(a) O_i first creates a physical identity proof π, then submits π and $\mathsf{pk_{O_i}}$ to MSP.
(b) MSP verifies π. Then it parses $(\mathsf{OL_{BC}})$ from the output of $\mathsf{ReadConfig(BC)}$. Check that $(\mathsf{pk_{O_i}} \notin \mathsf{OL_{MSP}}) \wedge (\mathsf{pk_{O_i}} \notin \mathsf{OL_{BC}})$. If all verifications hold, add $\mathsf{pk_{O_i}}$ to its local orderer list $\mathsf{OL_{MSP}}$ and return "1" to O_i, else return "0" with an error code.

(7) $\mathsf{LAdminSetup()}$ is an algorithm run by a LAdm to initialize its credentials and state and create a new device group G. A Local Administrator computes and outputs signing keys as $(\mathsf{pk_{LAdm_i}}, \mathsf{sk_{LAdm_i}}) \leftarrow \mathsf{SignGen}(1^\lambda)$. Allocates memory for storing group aggregators' and sensors' public keys as $\mathsf{st_{LAdm_i}}[\mathsf{AL}, \mathsf{SL}]$.

(8) $\mathsf{LAdminAdd}\{\mathsf{LAdm_i}(\mathsf{pk_{LAdm}}, \mathsf{sk_{LAdm}}) \leftrightarrow \mathsf{MSP}(\mathsf{pk_{MSP}}, \mathsf{sk_{MSP}}, \mathsf{st_{MSP}}[\mathsf{LL_{MSP}}], \mathsf{BC})\}$ is an interactive protocol between a local group administrator $\mathsf{LAdm_i}$ and MSP in order to add $\mathsf{LAdm_i}$ in the system.
(a) $\mathsf{LAdm_i}$ creates a physical identity proof π, then submits π and $\mathsf{pk_{LAdm}}$ to MSP.
(b) MSP verifies π. Then it parses $(\mathsf{LL_{BC}})$ from the output of $\mathsf{ReadConfig(BC)}$. Check that $(\mathsf{pk_{LAdm_i}} \notin \mathsf{LL_{BC}}) \wedge (\mathsf{pk_{LAdm_i}} \notin \mathsf{LL_{MSP}})$. If all verifications hold, add $\mathsf{pk_{LAdm_i}}$ to $\mathsf{LL_{MSP}}$ in $\mathsf{st_{MSP}}$ and return "1" to $\mathsf{LAdm_i}$, else return "0" with an error code.

(9) $\mathsf{AggrSetup}\{\mathsf{LAdm_i}(\mathsf{pk_{LAdm_i}}, \mathsf{sk_{LAdm_i}}, \mathsf{st_{LAdm_i}}) \leftrightarrow \mathsf{Ag_{ij}}()\}$ is an interactive protocol between an $\mathsf{LAdm_i}$ and an aggregator $\mathsf{Ag_{ij}}$ wishing to join group $\mathsf{G_i}$.
(a) $\mathsf{Ag_{ij}}$ computes signing keys as $(\mathsf{pk_{A_{ij}}}, \mathsf{sk_{A_{ij}}}) \leftarrow \mathsf{SignGen}(1^\lambda)$ and initializes pending and write transaction sets $\mathsf{pset_i} \rightarrow \emptyset, \mathsf{txset_i} \rightarrow \emptyset$ in its $\mathsf{st_{Ag_{ij}}}$.
(b) $\mathsf{Ag_{ij}}$ creates a physical identity proof π, then submits π and $\mathsf{pk_{A_{ij}}}$ to $\mathsf{LAdm_i}$.
(c) $\mathsf{LAdm_i}$ verifies $(\mathsf{pk_{A_{ij}}} \notin \mathsf{AL})$ and π. If these verifications hold, it invokes $\mathsf{AggrAdd}(\mathsf{pk_{A_{ij}}})$ with MSP. If MSP outputs "1", it adds $\mathsf{pk_{A_{ij}}}$ to AL, sends an updated copy of AL to all $\mathsf{Ag_{ij}} \in \mathsf{AL}$ and $\mathsf{pk_{LAdm_i}}$ to $\mathsf{Ag_{ij}}$. In all other cases it returns "0".
(d) $\mathsf{Ag_{ij}}$ copies $\mathsf{pk_{LAdm_i}}$ in its memory in $\mathsf{st_{Ag_{ij}}}$.

(10) $\mathsf{AggrAdd}\{\mathsf{LAdm_i}(\mathsf{pk_{LAdm_i}}, \mathsf{sk_{LAdm_i}}, \mathsf{pk_{A_{ij}}}) \leftrightarrow \mathsf{MSP}(\mathsf{pk_{MSP}}, \mathsf{sk_{MSP}}, \mathsf{st_{MSP}}[\mathsf{PL_{MSP}}], \mathsf{BC})\}$ is an interactive protocol between a local administrator $\mathsf{LAdm_i}$ wishing to add an aggregator to the system and MSP.
 (a) $\mathsf{LAdm_i}$ computes $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk_{LAdm_i}}, \mathsf{pk_{A_{ij}}})$. Send σ to MSP.
 (b) MSP computes $\mathsf{SVrfy}(\mathsf{pk_{LAdm_i}}, \mathsf{pk_{A_{ij}}}, \sigma) := b$. Checks that $(\mathsf{pk_{LAdm_i}} \in \mathsf{LL_{MSP}}) \wedge b \wedge (\mathsf{pk_{A_{ij}}} \notin \mathsf{PL_{MSP}}) == 1$. If the verification holds, it parses $(\mathsf{PL_{BC}})$ from the output of $\mathsf{ReadConfig}(\mathsf{BC})$. Check that $(\mathsf{pk_{A_{ij}}} \notin \mathsf{PL_{BC}})$. If the verification holds, add $\mathsf{pk_{A_{ij}}}$ to $\mathsf{PL_{MSP}}$ and returns "1" to $\mathsf{LAdm_i}$. It returns "0" in all other cases.

(11) $\mathsf{AggrUpd}\{\mathsf{LAdm_i}(\mathsf{sk_{LAdm}}, \mathsf{pk_S}) \leftrightarrow \mathsf{Ag_{ij}}(\mathsf{st_{Ag_{ij}}}[\mathsf{CL}])\}$ is an interactive protocol between $\mathsf{LAdm_i}$ and an aggregator $\mathsf{Ag_{ij}}$, both belonging to Group i. It is used when $\mathsf{LAdm_i}$ wants to add a sensor public key $\mathsf{pk_S}$ to $\mathsf{Ag_{ij}}$ and update its sensor list CL.
 (a) $\mathsf{LAdm_i}$ computes $\sigma := \mathsf{Sign}(\mathsf{sk_{LAdm_i}}, \mathsf{pk_S})$. Send σ to $\mathsf{Ag_{ij}}$.
 (b) $\mathsf{Ag_{ij}}$ computes $\mathsf{SVrfy}(\mathsf{pk_{LAdm_i}}, \mathsf{pk_S}, \sigma) := b$. Checks that $(\mathsf{pk_S} \notin \mathsf{st_{Ag_{ij}}}) \wedge b == 1$. If the verification holds, it adds $\mathsf{pk_S}$ to CL[4] and returns "1" to $\mathsf{LAdm_i}$. It returns "0" in all other cases.

(12) $\mathsf{SensorJoin}\{\mathsf{LAdm_i}(\mathsf{pk_{LAdm_i}}, \mathsf{sk_{LAdm_i}}, \mathsf{st_{LAdm_i}}[\mathsf{SL}]) \leftrightarrow \mathsf{S_{ij}}(n)\}$ is an interactive protocol between $\mathsf{LAdm_i}$ of Group i and a sensor $\mathsf{S_{ij}}$ wishing to join the system.
 (a) $\mathsf{S_{ij}}$ using the one-time signature scheme described in Section 6.2:
  (i) Samples $k \leftarrow (1^{\lambda})$. and stores it in $\mathsf{st_{S_{ij}}}$.
  (ii) Runs $(\mathsf{pk_{S_{ij}}}, \mathsf{sk_{S_{ij}}}, \ell = 1) \leftarrow \mathsf{OTKeyGen}(1^{\lambda}, n)$[5]
  (iii) Stores $\ell = 1$ to $\mathsf{st_{S_{ij}}}$ where ℓ denotes the current "index" in the hash chain.
  (iv) Creates a physical identity proof π
  (v) Sends $(\pi, \mathsf{pk_{S_{ij}}})$ to $\mathsf{LAdm_i}$
 (b) $\mathsf{LAdm_i}$ checks $\mathsf{Vrfy}(\pi) \wedge (\mathsf{pk_{S_{ij}}} \notin \mathsf{SL})$ $\wedge \mathsf{AggrUpd}(\mathsf{sk_{LAdm_i}}, \mathsf{pk_{S_{ij}}}) == 1\ \forall \mathsf{Ag_{ij}} \in \mathsf{G_i}$. $\mathsf{LAdm_i}$ adds $\mathsf{pk_{S_{ij}}}$ to SL, else it outputs "0".

(13) $\mathsf{SensorSendData}\{\mathsf{S_{ij}}(\mathsf{pk_{S_{ij}}}, \mathsf{sk_{S_{ij}}}, m, \mathsf{st_{S_{ij}}}) \leftrightarrow \mathcal{AG}_x(\mathsf{st_{Ag}}[\mathsf{CL}, \mathsf{pset}, \mathsf{txset}])\}$ is a protocol between sensor $\mathsf{S_{ij}}$ broadcasting data and a subset of aggregators $\mathcal{AG}_x \subseteq \{\mathcal{AG}_i\}$ (where $\{\mathcal{AG}_i\}$ is the aggregator set in $\mathsf{G_i}$).
 (a) For sending data m, $\mathsf{S_{ij}}$ computes $(\sigma, \mathsf{sk_S}, \mathsf{st_{S_{ij}}}') \leftarrow \mathsf{OTSign}(\mathsf{sk_S}, m, \mathsf{st_{S_{ij}}})$
 (b) $\mathsf{S_{ij}}$ broadcasts σ to $\mathcal{AG}_x$.
 (c) $\mathsf{Ag_k}$ runs $\mathsf{OTVerify}(\mathsf{pk_{S_{ij}}}, m, \sigma) := b$.[6] If $b == 1$ it runs $\mathsf{AggrAgree}$ with all other aggregators in the group. If no "alarm" message m^A from some other aggregator is received within some time δ, it adds m, σ to $\mathsf{pset_i}$. If at least one "alarm" message is received, it outputs $\perp$.

(14) $\mathsf{AggrAgree}\{\mathsf{Ag_k}(\mathsf{sk_{A_k}}, \mathsf{AL}, \mathsf{pk_{S_{ij}}}, m, \sigma) \leftrightarrow \mathcal{AG}([\mathsf{sk_{A_i}}, \mathsf{pk_{S_{ij}}}, m'])\}$ is a protocol between an aggregator in $\mathsf{G_i}$ and all other aggregators in the group. The purpose of this protocol is to detect any MITM attacks, and verifies that no aggregator in the group has received any message m', σ' from $\mathsf{pk_{S_{ij}}}$ where $m \neq m'$[7].
 (a) $\mathsf{Ag_k}$ for payload $\mu = (\mathsf{pk_{S_{ij}}}, m, \sigma)$ computes $s = \mathsf{Sign}(\mathsf{sk_{A_k}}, \mu)$ using an EU-CMA signature scheme and sends s, μ to all $\mathsf{Ag_i} \in \mathcal{AG}$.
 (b) Each $\mathsf{Ag_i}$ checks if it received a message m' with signature σ from sensor $\mathsf{pk_{S_{ij}}}$ where $m \neq m'$. If there's no such message, it outputs $\perp$. Else it sends an "alarm" message m^A and respective signature s to $\mathsf{Ag_k}$ and keeps a record in its log.

(15) $\mathsf{AggrSendTx}\{\mathcal{AG}([\mathsf{pk_{A_i}}, \mathsf{sk_{A_i}}, \mathsf{st_{Ag_i}}, \mathsf{BC}]) \leftrightarrow \mathcal{O}(\mathsf{pk_{O_j}}, \mathsf{sk_{O_j}}, \mathsf{st_j})\}$ is an interactive protocol between all aggregators $\mathsf{Ag_i} \in \mathcal{AG}$ and all orderers $\mathsf{O}_j \in \mathcal{O}$. It is initiated when an aggregator wishes to submit a transaction for validation in the system and eventually store it in the blockchain.
 (a) An $\mathsf{Ag_i} \in \mathcal{AG}$ parses $\mathsf{pset_i}$ in $\mathsf{st_{Ag_i}}$ as a set of transactions $\{\mathsf{tx}\}$.
 (b) $\mathsf{Ag_i}$ samples a nonce $n \leftarrow (1^{\lambda})$ and appends it to $\{\mathsf{tx}\}$.
 (c) $\mathsf{Ag_i}$ computes $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk_{A_i}}, \{\mathsf{tx}\})$. Send σ to all other $\mathsf{Ag_j} \in \mathcal{AG}_x$.
 (d) Each $\mathsf{Ag_j}$, parses $(\mathsf{PL_{BC}})$ from the output of $\mathsf{ReadConfig}(\mathsf{BC})$. Computes $\mathsf{SVrfy}(\mathsf{pk_{A_i}}, \{\mathsf{tx}\}, \sigma) := b$. If $(\mathsf{pk_{A_i}} \in \mathsf{PL_{BC}}) \wedge$ $b == 1$, compute
 $\sigma_j \leftarrow \mathsf{Sign}(\mathsf{sk_{A_j}}, \{\mathsf{tx}\})$. Send σ_j to $\mathsf{Ag_i}$.
 (e) $\mathsf{Ag_i}$ parses $\mathsf{ReadConfig}(\mathsf{BC}) \rightarrow \mathsf{Pol_{BC}} \rightarrow \tau$ where τ the minimum required number of signatures for a transaction to be submitted on the blockchain, as defined by policy Pol.
 (f) If $|\{\sigma_j\}| > \tau$, select a reachable orderer O, send $\{\sigma_j\}$, copy $\{\mathsf{tx}\} \rightarrow \mathsf{txset}$ and set $\mathsf{pset} \rightarrow \emptyset$.
 (g) The orderer O parses $\mathsf{st_j} \rightarrow TXL$, $\{\mathsf{tx}\} \rightarrow n$, $\mathsf{ReadConfig}(\mathsf{BC}) \rightarrow \mathsf{PL_{BC}}, \mathsf{Pol_{BC}}$ and $\mathsf{Pol_{BC}} \rightarrow \tau$ then checks:
  (i) $|\{\sigma_j\}| > \tau$ and $n \notin TXL$
  (ii) Compute $\mathsf{SVrfy}(\mathsf{pk_{A_j}}, \{\mathsf{tx}\}, \sigma_j) := b, \forall j$ then $\prod b_j == 1$
  (iii) $\prod_j (\{\mathsf{pk_{A_j}}\} \in \mathsf{PL_{BC}}) == 1$
  If the checks are valid, stores $|\{\sigma_j\}|$ in its $\mathsf{st_{O_i}}$ and replies "1" to $\mathsf{Ag_i}$ as a confirmation, else it replies "0".
 (h) If O_i has created a new block containing ordered transactions, it runs $\mathsf{Consensus}$ to update the blockchain.
 (i) If $\mathsf{Consensus}$ succeeds, it runs $\mathsf{UpdateBC}$ with all aggregators to update to the new BC'.

(16) $\mathsf{Consensus}([[\mathsf{pk_{O_j}}]_{j=1}^{n}, \mathsf{sk_{O_i}}, \mathsf{st_i}, \mathsf{BC}]_{i=1}^{n}) := \mathsf{BC}'$
The exact protocol functionality is described in the system parameters pp[8] and follows the definition provided in Section 2.1. In general this protocol is executed among all orderers $\mathsf{O}_i \in \mathcal{O}$ where they agree on a new updated blockchain BC'. In Appendix C.9 we provide a concrete instantiation of a consensus algorithm for our construction.

[4] $\mathsf{LAdm_i}$ should run the protocol with every aggregator in the group, however we present this with one aggregator for simplicity.
[5] This step is typically computed by a powerful device.
[6] To avoid redundancy, the protocol can be improved by deterministically defining a "responsible" aggregator for each transaction as discussed previously in this section.
[7] This protocol does not require that all other aggregators in the group are reachable, therefore it does not require a reply from all aggregators to complete.
[8] This is equivalent to Hyperledger's "pluggable" consensus, which is defined in the genesis block.

(17) $\mathsf{UpdateBC}\{\mathsf{O}_i(\mathsf{pk}_{\mathsf{O_i}}, \mathsf{sk}_{\mathsf{O_i}}, \mathsf{st}_{\mathsf{O_i}}, \mathsf{BC}') \leftrightarrow \mathcal{AG}([\mathsf{pk}_{\mathsf{A_x}}, \mathsf{sk}_{\mathsf{A_x}}, \mathsf{st}_{\mathsf{Ag_x}}, \mathsf{BC}])\}$ is initiated by an orderer O_i to append a new block in the blockchain.
   (a) O_i parses its $\mathsf{st}_{\mathsf{O_i}}$ to retrieve the agreed blockchain update signature set $\{\sigma'_x\}$
   (b) O_i computes $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}_{\mathsf{O_i}}, (\mathsf{B_i}, \{\sigma'_x\}))$ where $\mathsf{BC}' := \mathsf{BC}||\mathsf{B_i}$ and sends σ to all $\mathsf{Ag_x} \in \mathcal{AG}$.
   (c) Each $\mathsf{Ag_x}$ computes $\mathsf{SVrfy}(\mathsf{pk}_{\mathsf{O_i}}, \mathsf{B_i}||\{\sigma'_x\}, \sigma) := b$ and checks if $b == 1$. Then it parses σ as a transaction set $\{\mathsf{tx}\}$ and removes these from $\mathsf{txset_x}$. Then it updates BC to BC', else it outputs $\perp$.

(18) $\mathsf{NodeRevoke}(\mathsf{pk_i}, \sigma, \mathsf{st}_{\mathsf{MSP}}, \mathsf{BC})$ is initiated by MSP to revoke credentials of any system participant.
   MSP parses $\mathsf{ReadConfig}(\mathsf{BC})$ as $[\mathsf{LL}_{\mathsf{BC}}, \mathsf{OL}_{\mathsf{BC}}, \mathsf{PL}_{\mathsf{BC}}]$. It verifies σ (if the remove operation was initiated by a LAdm) and checks if $\mathsf{pk_i}$ exists in $[\mathsf{LL}_{\mathsf{BC}}, \mathsf{OL}_{\mathsf{BC}}, \mathsf{PL}_{\mathsf{BC}}]$ or in its $[\mathsf{LL}_{\mathsf{MSP}}, \mathsf{OL}_{\mathsf{MSP}}, \mathsf{PL}_{\mathsf{MSP}}] \in \mathsf{st}_{\mathsf{MSP}}$ in case participation privileges for $\mathsf{pk_i}$ have not yet been updated on the BC through $\mathsf{ConfigUpdate}$. If it finds a match in the blockchain lists, it creates a remove operation $R := (\mathsf{pk_i}, \text{“}rm\text{”})$ and adds R to oper, else if it finds a match in its state lists it removes it from the respective list, else it outputs $\perp$. If $\mathsf{pk_i} \in \mathsf{PL}_{\mathsf{BC}} \vee \mathsf{pk_i} \in \mathsf{PL}_{\mathsf{MSP}}$, it also informs $\mathsf{LAdm_i}$.

(19) $\mathsf{GroupRevoke}(\mathsf{pk_i}, \mathsf{st}_{\mathsf{LAdm}}[\mathsf{AL}, \mathsf{SL}])$ is initiated by a Local Administrator to revoke credentials of an aggregator or sensor in its group. LAdm checks if $\mathsf{pk_i} \in [\mathsf{AL}, \mathsf{SL}]$. If it finds a match and $\mathsf{pk_i} \in \mathsf{AL}$, it $\mathsf{LAdm_i}$ computes $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}_{\mathsf{LAdm_i}}, \mathsf{pk}_{\mathsf{A_i}}, \text{”}R\text{”})$. Sends $(\sigma, \mathsf{pk}_{\mathsf{A_i}}, \text{”}R\text{”})$ to MSP. On receiving successful removal from MSP (after it invokes $\mathsf{NodeRevoke}$), LAdm removes $\mathsf{pk_i}$ from AL. If $\mathsf{pk_i} \in \mathsf{SL}$, it invokes $\mathsf{AggRevokeSensor}$ with all $\mathsf{Ag} \in \mathsf{AL}$. After successful completion, it removes $\mathsf{pk_i}$ from SL.

(20) $\mathsf{AggRevokeSensor}\{\mathsf{LAdm_i}(\mathsf{sk}_{\mathsf{LAdm}}, \mathsf{pk_S}) \leftrightarrow \mathsf{Ag_{ij}}(\mathsf{st}_{\mathsf{Ag_{ij}}}[\mathsf{CL}])\}$ is initiated by a Local Administrator as a subroutine of $\mathsf{GroupRevoke}$ to revoke credentials of a sensor in its group.
   (a) $\mathsf{LAdm_i}$ computes $\sigma := \mathsf{Sign}(\mathsf{sk}_{\mathsf{LAdm_i}}, \mathsf{pk_S}\ \text{“}R\text{”})$. Send σ to $\mathsf{Ag_{ij}}$.
   (b) $\mathsf{Ag_{ij}}$ computes $\mathsf{SVrfy}(\mathsf{pk}_{\mathsf{LAdm_i}}, \mathsf{pk_S}, \sigma) := b$. Checks that $(\mathsf{pk_S} \notin \mathsf{st}_{\mathsf{Ag_{ij}}}) \wedge b == 1$. If the verification holds, it removes $\mathsf{pk_S}$ from CL and returns “1" to $\mathsf{LAdm_i}$. It returns “0" in all other cases.

E EVALUATION DETAILS

Algorithms 1 and 2 show the pseudocode for our evaluations on the sensor and aggregator side respectively. We denote by timer1 the signature computation time, by timer2 the verification time and by timer3 the total verification time, as previously shown in Table 5.

Algorithm 1 Sensor send data

```
1:  tempkey ← $k_0$
2:  initPebbles()
3:  while True do
4:      m ← readSensor()
5:      output.type ← “payload"
6:      output.data ← m
7:      transmit(output)
8:      T1.start()
9:      hashedData ← $h$(m||tempkey)
10:     output.type ← “hash"
11:     output.data ← hashedData
12:     transmit(output)
13:     tempkey ← computePebbles() {as in [47]}
14:     output.type ← “secretKey"
15:     T1.end()
16:     output.data ← tempkey
17:     transmit(output)
18: end while
```

Algorithm 2 Aggregator receive data

```
1: publickey ⟵ $k_0$
2: verifications ⟵ 0
3: while verifications < maxVerifications do
4:     check1 ⟵ False
5:     check2 ⟵ False
6:     read ⟵ input()
7:     if read.type = “payload" then
8:         T3.start()
9:         m ⟵ read.data
10:    else if read.type = “hash" then
11:        $s_1$ ⟵ read.data
12:    else if read.type = “secretKey" then
13:        $s_2$ ⟵ read.data
14:        tempkey ⟵ $s_2$
15:        $i$ ⟵ 0
16:        T2.start()
17:        while $i$ < maxVerification ∧ doWhile = True do
18:            if $h$(tempkey) = publickey then
19:                check1 ⟵ True
20:                if $h$(m||publickey) = $s_1$ then
21:                    check2 ⟵ True
22:                    publickey ⟵ secretkey
23:                end if
24:                doWhile = False
25:            else
26:                tempkey ⟵ $h$(tempkey)
27:                i++
28:            end if
29:        end while
30:        if check1 ∧ check2 = True then
31:            print(“Payload m is valid")
32:            verifications++
33:            T2.end()
34:        else
35:            print("Verification failed")
36:        end if
37:        T3.end()
38:    end if
39: end while
```