A Lightweight Block Validation Method for Resource-Constrained IoT Devices in Blockchain-Based Applications

Tam Le
Department of Computer Science and Engineering
Michigan State University
East Lansing, Michigan, USA
letam@msu.edu

Matt W. Mutka
Department of Computer Science and Engineering
Michigan State University
East Lansing, Michigan, USA
mutka@msu.edu

***Abstract*—Secure access control to a wide variety of Internet of Things (IoT) devices has become critical. Blockchain-based access control frameworks are promising technologies to support secure access to IoT devices in pervasive computing applications. However, in most of the proposed solutions, the IoT devices rely on a trusted server to retrieve critical access control data from the blockchains. We propose a method for IoT devices to validate blockchain data without solely being dependent on a central server. In our approach, several witnesses on the network can be selected randomly by the devices to validate access control information. Our method is aided by Bloom filters, which are shown to be lightweight for resource-constrained devices.**

***Index Terms*—IoT, blockchain, access control, Bloom filters.**

I. Introduction

The cost to produce a smart device with connectivity and computing capability has been significantly reduced, making pervasive computing applications grow rapidly in various areas, especially in smart homes and smart cities. As a result, it is critical to have proper access control over these devices. While cloud computing has been supporting Internet of Things (IoT) services with management, data processing and storage, its centralized architecture also raises a concern about security and trustworthiness of cloud providers due to the importance of the IoT data they can access [1] as well as their privileges over the devices. Reliance on cloud systems has posed several problems for smart devices. A recent outage incident with Samsung's SmartThings cloud servers has made users in North America unable to use their smart appliances for hours [2]. In addition, a serious vulnerability due to a flaw in the manufacturer's cloud server architecture has been found in some popular smart cameras [3], which can allow attackers to gain access to all of the cameras. The security issues do not only lie in manufacturers' clouds but also third-party services. For example, IFTTT is a popular service that allows users to define a set of complex rules for their smart devices by acting as a bridge between different services and applications. As a result, a large amount of API keys are stored by IFTTT, which is not only a privacy issue to users but also makes IFTTT a potential target to attackers [4].

Access control to smart devices becomes critical. IoT communities have recently directed their attention to blockchain technology, which is first introduced in the famous cryptocurrency Bitcoin, due to its decentralized/distributed design. Since blockchain can maintain an immutable ledger among untrusted parties, decentralization can be brought to IoT services. Security and privacy also become more transparent to users. Ouaddah et al [5], Huh et al. [6] , Dorri et al. [7] and Andersen et al. [8] have explored blockchain to provide access control policies to IoT devices. In terms of IoT markets, Slock.it [1] is a blockchain infrastructure that allows smart objects and vehicles to be shared or rented. All of the mentioned platforms are based on the same principle: the occurrence of a specific event/transaction on the blockchain needs to be verified before access to an IoT devices can be granted. For example, for Slock.it, a user can rent his/her house by creating a smart contract on a blockchain. By making a payment through the smart contract, the renter can then receive a key/token that later can be used to unlock the house. In this case, the smart lock needs to be able to recognize the transaction between its owner and the renter on the blockchain in order to grant access to the renter. The validation of blockchain data requires nodes to store the entire blockchain, whose size may be a problem. To solve this, Bitcoin also provides Simplified Payment Verification (SPV) [9], which allows nodes that do not have sufficient storage to store only block headers. Although SPV works well for cryptocurrency nodes, it cannot be adapted to IoT devices with very low memory, data storage and bandwidth. Currently in Bitcoin there are over 500 000 blocks with 80 bytes per block header. Ethereum has over 6 000 000 blocks

This material is based upon work supported by the National Science Foundation. Any opinions, findings, conclusions or recommendations reflects the views of the authors and do not necessarily reflect the views of the National Science Foundation.

978-1-7281-0270-2/19/$31.00 ©2019 IEEE

[1] https://slock.it

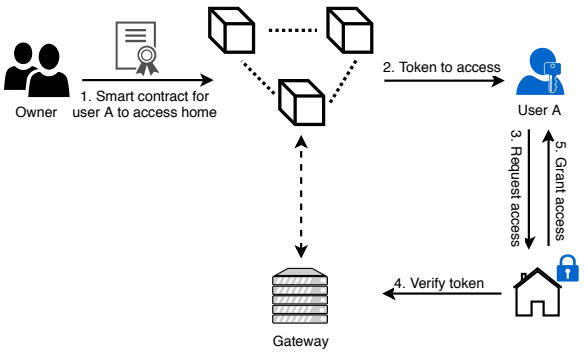


Fig. 1: A use case where an owner transfers the key to his/her house to user A via a smart contract. The door lock confirms the smart contract via a gateway that connects to the blockchain network and grants access to the user accordingly.

with 508 bytes per block header [2]. Hence, block headers require hundreds of megabytes. In addition, although SPV nodes do not store the complete blockchain, they are still peers that receive block headers from others. On the other hand, low-cost devices often have very limited transmission rates and are not capable of maintaining connections and frequently exchanging data with multiple peers. As a result, resource-constrained nodes have no choice but rely on some more powerful nodes that can interact directly with the blockchain. Figure 1 demonstrates the earlier use case of Slock.it where there is a smart home manager that acts as a gateway to inform the smart lock about the contract between the owner and renter.

The use of gateway nodes removes the decentralization benefit of blockchain since if the gateways are compromised, they can feed the IoT devices a false version of the blockchain and give attackers illegal access. Therefore, we propose a validation mechanism that allows IoT devices to validate block headers without relying solely on a single trusted node. Our idea is to have random nodes in the network, which we call *witnesses*, to help the IoT devices confirm incoming block headers. While we still use the capabilities of more powerful nodes as the edge/gateway nodes to relay the main data, the random selection of witnesses would make it difficult for attackers to compromise the edge nodes. It is noteworthy that our mechanism is not to replace, but work on top of blockchain consensus protocols to support low-power devices.

The rest of the paper is organized as follows. Section II is an overview of our protocol. Section III discusses the details of each step of the protocol. We provide a probabilistic security analysis in section IV and performance evaluation in section V. Related work is addressed in section VI. Finally, the paper is concluded by section VII.

II. Protocol Overview

A. System model

We consider 3 parties in our protocol:

- *Device* D: a resource-constrained device that may not directly interact with the blockchain network.
- *Gateway server* S: to transfer any necessary messages from the blockchain network to D.
- *Witnesses*: regular peers on the blockchain network that regularly broadcast a lightweight *"confirmation"* message that confirms one or multiple blocks.

The device D will accept a new block header from S if it receives confirmations for the same block from a set of witnesses that is selected randomly to prevent corruption. Our protocol employs a probabilistic approach based on Bloom filters [10], which have been known to be space and time efficient for membership testing. Therefore, it can be lightweight enough for resource-constrained devices. A Bloom filter $\mathcal{B}_X$ is characterized by 3 parameters: the size m_X, the number of bits used per item k_X and the fill rate f_X, which is the ratio between the number of bits that are set to 1 and m_X. Specifically, the following 3 filters are used in our protocol:

- *Confirmation filter* ($\mathcal{B}_c$): this is the main content of the confirmation messages sent by witnesses. $\mathcal{B}_c$ stores individual block confirmations between a witness and each of the devices it is going to serve. Each individual confirmation is inserted to $\mathcal{B}_c$ using k_c bits.
- *Selection filter* ($\mathcal{B}_s$): a random filter generated by each device that represents the condition to select witnesses. A witness is selected if its ID has k_s bits included in $\mathcal{B}_s$.
- *Whitelist filter* ($\mathcal{B}_w$): a pre-selection filter to narrow down the set of devices that a witness can serve. A device is available to a witness and thus can be added to the witness's confirmation filter if its ID has k_w bits in $\mathcal{B}_w$.

It is noteworthy that among the 3 filters, only $\mathcal{B}_c$ actually represents a set of items (i.e. individual confirmations) while $\mathcal{B}_s$ and $\mathcal{B}_w$ are just arrays of bits that describe the condition for selection. Additionally, we use a fixed fill rate of set bits to avoid the use of filters with all/most of 1s. Therefore in case of $\mathcal{B}_c$, if there are not enough bits after inserting all necessary individual confirmations, the witnesses can add more random bits to reach the fill rate.

Figure 2 describes the main steps of our protocol to verify new blocks at a device. Each verification round starts at step 0 when device D informs its server S the selection filter $\mathcal{B}_s$ for the upcoming block. When the block arrives, each witness computes the whitelist $\mathcal{B}_w$ to determine the set of devices available to them and sends out confirmation filters $\mathcal{B}_c$ accordingly. S then selects only confirmations from witnesses that satisfy the condition $\mathcal{B}_s$ and relays them to D together with the new block header. The block is accepted by D if it is confirmed by η witnesses. If D does not have enough space to store the header after verifying it using the selected confirmations, D can "stamp" the header and sends it back to S for storage. The stamped header can be a message authentication code (e.g. HMAC) computed using the device's own secret to prevent the server from altering the header.

[2]From https://blockchain.com and https://ethstats.net, as of Aug 2018

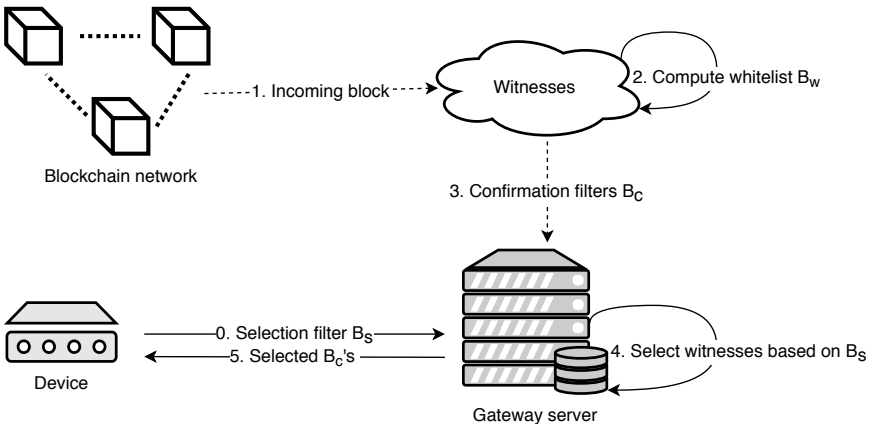


Fig. 2: Each validation round starts at step 0 when a device sends a random condition represented by selection filter $\mathcal{B}_s$ to its server to select witnesses for the next incoming block.

B. Threat model

As discussed earlier, a node needs at least the block headers in order to verify a transaction. It also needs to be sure that the headers are valid. While consensus protocols (e.g. proof of work) can provide agreement on the current state of the blockchain, a resource-constrained device with limited bandwidth may not be able to handle all of the peer-to-peer message exchange and communication during the consensus phase. As a result, a gateway between the devices and blockchain network cannot be avoided completely. We consider an extreme scenario where D has very limited resource, hence it needs to frequently receive not only new block headers but also confirmation messages from the gateway S. It should be emphasized that the gateway is not obligatory if the device has reasonable storage and networking capabilities.

We assume that S can be compromised and create false blocks and confirmations to D. Our assumption includes the possibility of a man-in-the-middle attack, which can also make D receive false information.

TABLE I: Notations

m_X	Size of Bloom filter X.
f_X	Fill rate of Bloom filter $\mathcal{B}_X$, i.e. the fraction of bit 1s in the filter.
k_X	Number of bits used per item in Bloom filter $\mathcal{B}_X$.
N_W	Total number of witnesses in the network.
N_D	Total number of devices in the network.
n_c	Capacity of confirmation filter $\mathcal{B}_c$
η	Required number of selected witnesses.

III. Protocol details

A. Witness advertisement

Any node that wants to become a witness can advertise itself by posting its ID to the blockchain. To encourage nodes to be witnesses, the devices can pay a small reward to every selected witness. On the other hand, witnesses also need to make a deposit along with the advertisement. The deposit amount will be the initial weight of each witness to prevent a Sybil attack and can be returned when the witness decides to retire. It is assumed that the devices also publish their IDs during some bootstrap stage. We use public keys as the IDs, thus throughout the paper ID/public key may be used interchangeably.

It is important that when a device is newly launched, it has no knowledge of any witness. Therefore, we suppose at this bootstrap stage, the device would require a trusted server or some designated witnesses to download previous block headers in order to synchronize with the network. As long as the bootstrap is monitored, the device can receive new blocks reliably with the support of new witnesses.

B. Witness selection

1) Simple selection: The selection of witnesses takes place off-chain and only involves the device D and server S. First D selects a random seed to generate a condition for choosing witnesses. S then picks up a list of witnesses that match the condition and sends it back to D. For each selected witness, S also attaches the block that includes the witness's advertisement transaction as a proof that the witness is valid.

The random condition is represented by a Bloom filter $\mathcal{B}_s$ with a fill rate f_s, thus a witness is selected if its ID matches k_s bits of the filter. Therefore, the probability of being selected of each witness is

$$p_s = f_s^{k_s} \tag{1}$$

Let N_W be the total number of witnesses in the network, n_W be the number of witnesses selected by $\mathcal{B}_s$ and η be the number of witnesses that the devices requires to accept a new block. Since each witness is selected with the same probability p_s, n_W follows a Binomial distribution $B(N_W, p_s)$. Hence, the probability that less than η witnesses are selected is

$$Pr(n_W \leq \eta) = \binom{N_W}{\eta} \times {p_s}^{\eta} \times (1-p_s)^{N_W-\eta} \tag{2}$$

Since there should be at least η selected, with a given value of N_W, η and f_s, D can adjust k_s to make the probability

negligible. Figure 3 plots the CDF as a function of k_s with $\eta = 16$ and N_W from 1000 to 10000.

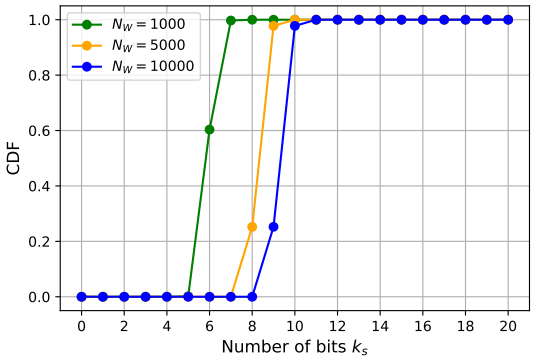


Fig. 3: Probability that there are less than $\eta = 16$ witnesses selected as a function of k_s.

Based on the figure, one can choose $k_s < 6$ to maintain a negligible probability as the network grows.

To prevent S from selecting witnesses in its favor, in each round of selection, D generates a new filter $\mathcal{B}_s$ and a random number r using a secret seed. A witness with ID $\mathcal{W}$ is identified by a codeword $\mathcal{H}(r, \mathcal{W})$ where $\mathcal{H}$ is a pseudo-random function. There are various ways to generate a random filter. A possible method is described by procedure GenSelectionFilter() in Algorithm 1, which repeatedly inserts the hashes computed by a secret s held by the device and a random number to the filter until it reaches the fill rate f_s .

2) Witness selection based on weights: By the above selection, witnesses are selected with equal probability $p_s = {f_s}^{k_s}$, hence it is vulnerable to a Sybil attack when an adversary can create as many witnesses as they want and increase their probability of being chosen. Let $W = \sum_{i=1}^{N} w_i$ be the cumulative weight of all witnesses where w_i is the weight of witness i, a weight-based selection can be implemented by treating each unit of w_i as a sub-witness, hence there will be W sub-witnesses in total and equation (2) can be changed to

$$Pr(n_W \leq \eta) = \binom{W}{\eta} \times {p_s}^{\eta} \times (1 - p_s)^{W-\eta} \qquad (3)$$

Procedure SelectByWeight() in Algorithm 1 describes the weight-based selection, where a witness $\mathcal{W}$ with w units of weight is simulated as w sub-witnesses, i.e. it has w chances to be selected.

It is noteworthy that for such weight-based selection, a device also needs to update the witnesses' weights correctly. As addressed in section III-A, the witnesses deposit an amount as the initial weight and are rewarded for each time they are selected. All of these events are recorded on the blockchain as transactions. Therefore, a witness can include in its confirmation message a flag that indicates the number of reward transactions existing in the new block. When validating the block, D can receive these transactions from its server, validate them using Merkle proofs [11] as in Bitcoin's SPV method [9] and then update the weights accordingly. Like the stamped block headers, if D does not have enough space to maintain the weights locally, it can also stamp and store them on the server instead.

Algorithm 1 GenSelectionFilter() is to generate a random $\mathcal{B}_s$ with given m_s, f_s and a secret s. SelectByWeight() is to select a witness with ID $\mathcal{W}$ and weight w, k_s is the number of bits needed to match with Bloom filter $\mathcal{B}_s$.

procedure GENSELECTIONFILTER(s, m_s, f_s)
  $\mathcal{B}_s \leftarrow empty$
  $i \leftarrow 0$
  $r \leftarrow rand()$
  $k = \lfloor \frac{hashlen}{log_2 m_s} \rfloor$
  while $\mathcal{B}_s.CountSetBits() < m_s \times f_s$ **do**
    $l \leftarrow m_s \times f_s - \mathcal{B}_s.CountSetBits()$
    if $r \geq k_s$ **then**
      Insert $\mathcal{H}(i, r, s)$ to $\mathcal{B}_s$ using k bits
    else
      Insert $\mathcal{H}(i, r, s)$ to $\mathcal{B}_s$ using l bits
    end if
    $i \leftarrow i + 1$
  end while
  return $\mathcal{B}_s$
end procedure
procedure SELECTBYWEIGHT($\mathcal{W}, w, k_s, \mathcal{B}_s$)
  $i \leftarrow 0$
  $r \leftarrow rand()$
  while $i < w$ **do**
    $hash \leftarrow \mathcal{H}(r, \mathcal{W}, i)$
    if $hash$ has k_s bits match with $\mathcal{B}_s$ **then**
      return i
    else
      $i \leftarrow i + 1$
    end if
  end while
  return -1
end procedure

C. Confirmation messages

A witness confirms a new block header by broadcasting a confirmation message. Since we assume that the server S will deliver any message from the blockchain to their devices, the messages need to be authenticated to make sure that they come from valid witnesses. Although it would be ideal to use signed messages, verifying signatures with public key cryptography could result in high computation overhead at the resource-constrained devices, especially when the rate of incoming blocks becomes high. On the other hand, symmetric key approaches are more lightweight, however they are not suitable for message broadcasting/multicasting as it would require a lot of bandwidth for witnesses to send one-to-one confirmations to all of their devices. Therefore, to save the bandwidth as well as reduce the computation overhead, we have the witnesses to include all of the individual confirmations to a Bloom filter and broadcast it as a single confirmation message.

Recall that both witnesses and devices have their public keys on the blockchain, hence each pair of witness and device can compute a Diffie-Hellman (DH) shared key s by themselves. Let $\mathbf{s} = \{s_i\}$ be the set of DH keys for all of devices witness $\mathcal{W}$ would like to serve. For each block header B_j, $\mathcal{W}$ computes a codeword $c_{ij} = \mathcal{H}(s_i, B_j)$ where $\mathcal{H}$ is a pseudo-random function and insert those values to confirmation filter $\mathcal{B}_c$, which will be broadcast then together with the ID of $\mathcal{W}$ as a confirmation message.

Algorithm 2 To generate a confirmation message $\mathcal{B}_c$ with parameters for a new block B using a set of shared keys $\mathbf{s}$

procedure GENCONFIRMATION($k_c, B, \mathbf{s}$)
  $\mathcal{B}_c \leftarrow \emptyset$
  for each $s_i \in \mathbf{s}$ **do**
    $c \leftarrow \mathcal{H}(s_i, B)$
    Insert c to $\mathcal{B}_c$ using k_c bits
  end for
  return $\mathcal{B}_c$
end procedure

To validate a new block, a device D checks if the block header is included in every $\mathcal{B}_c$ sent by its selected witnesses using the same shared DH keys. Due to the limited resource for both memory storage and computation, it is not practical to have the device store keys for all of the witnesses nor recompute the shared keys for every validation. Therefore, the keys can be encrypted and then stored on the server S instead once it is computed by D.

With a fixed fill rate f_c, the false positive rate, i.e. the probability that a block header can be falsely recognized in a single filter is ${f_c}^{k_c}$. As a block header needs confirmations from η witnesses, the probability that a device may accept a false block is exponentially reduced to

$$p = {f_c}^{k_c \times \eta} \tag{4}$$

Since the number of devices that can be included in $\mathcal{B}_c$ is limited by k_c, we can select more witnesses to maintain a reasonable false positive rate while still keeping k_c low to increase the capacity of $\mathcal{B}_c$. In the lower part of Figure 4, we plot n (denoted by ■) with respect to $p = 2^{-128}$, which we suppose should be the minimum threshold to maintain reasonable security (since this is also the probability of finding a 128-bit key). On the other hand, with pre-defined (m_c, k_c, f_c), according to [12], the capacity of $\mathcal{B}_c$ can be approximated by

$$n_c = -\frac{m_c}{k_c} ln\,(1 - f_c) \tag{5}$$

In the same figure, we also plot n_c (denoted by •) for a 1024-bit filter (i.e. $m_c = 1024$). As shown in the figure, a fill rate of 50% and 7-9 bits used seem to provide a good balance between n_c and η, since we would not need too many witnesses but also want to include to $\mathcal{B}_c$ as many devices as possible.

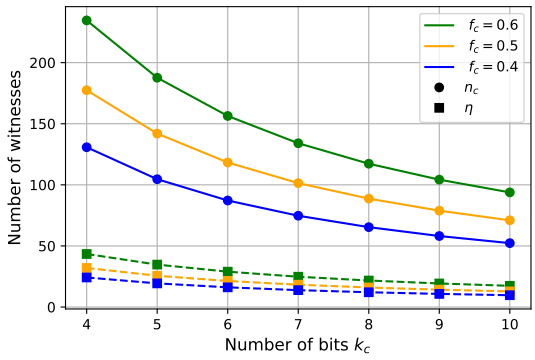


Fig. 4: This figure plots the size of witness set (η) with respect to $m_c = 1024$ and the number of devices that can fit to a $\mathcal{B}_c$ (n_c) with respect to false positive rate $p = 2^{-128}$.

D. Whitelist filter

Since n_c would be much smaller than the total number of devices in the whole network, witnesses may try to maximize their profit by sending as many confirmations as possible. To prevent the network from being flooded with too many messages, we use a second filter, denoted by $\mathcal{B}_w$, as a "whitelist" that is announced at the beginning of each round. Unlike $\mathcal{B}_s$ which is created by the devices, when a new block is received, each witness computes its own whitelist based on its ID and the new block hash. A device can be added to the witness's confirmation filter $\mathcal{B}_c$ if its ID has k_w bits in $\mathcal{B}_w$. The generation of $\mathcal{B}_w$ is described in Algorithm 3.

Algorithm 3 To compute the whitelist filter $\mathcal{B}_w$ with parameters (m_w, f_w, k_w) for witness $\mathcal{W}$ and new block B

procedure GENWHITELIST($m_w, f_w, k_w, B, \mathcal{W}$)
  $\mathcal{B}_w \leftarrow \emptyset$
  $i \leftarrow 0$
  while $\mathcal{B}_w.CountSetBits() < m_w \times f_w$ **do**
    $r \leftarrow m_b \times f_w - \mathcal{B}_w.CountSetBits()$
    if $r \geq k_w$ **then**
      Insert $\mathcal{H}(i, B, \mathcal{W})$ to $\mathcal{B}_w$ using k_b bits
    else
      Insert $\mathcal{H}(i, B, \mathcal{W})$ to $\mathcal{B}_w$ using r bits
    end if
    $i \leftarrow i + 1$
  end while
  return $\mathcal{B}_w$
end procedure

Assuming every device has the same probability p_w to be included in the whitelist, the expected number of devices that are available to a witness is $p_w \times N_D$, where N_D is the total number of devices. Hence, with a certain n_c, p_w can be chosen such that

$$p_w = \frac{n_c}{N_D} \tag{6}$$

Since the whitelist is also a Bloom filter, it is characterized by the tuple (m_w, f_w, k_w). A witness can only pass if it has k_w bits in $\mathcal{B}_w$, the whitelist probability is

$$p_w = {f_w}^{k_w} \tag{7}$$

Putting (4) - (7) together, the condition to choose the parameters of the whitelist filter will be

$${f_w}^{k_w} = \frac{n_c}{N_D} = -\frac{m_c}{N_D \times k_c} ln\,(1 - f_c) \tag{8}$$

It is important that with the whitelist, the expected number of witnesses that can serve a certain device is reduced to $\frac{n_c}{N_D} \times N_W$. Hence, if the number of devices is much larger than the number of witnesses, i.e. $\frac{N_D}{N_W} > n_c$, most of the devices will have no witness available to go to the selection stage. However, since we are using weighted witnesses, by dividing the weight unit, we can increase the number of sub-witnesses and narrow down the ratio. In practice, we believe that the gap between the two numbers would not be too big, as nodes are always incentivized to become witnesses. Additionally, devices with the same owner may also share a group ID on the blockchain instead of individual ones.

E. Network dynamics

1) Not enough confirmations: Since our method is probabilistic, there is still a small probability that less than η witnesses satisfy the condition. Another possibility that there are not enough confirmations is when forks happen, i.e. more than one valid block are produced nearly at the same time, which can result in multiple versions of the blockchain among nodes. To deal with such cases, instead of confirming 1 block at a time, the witnesses can compute their codewords for the confirmation filter $\mathcal{B}_c$ using the n most recent blocks in their main chain [3]. Hence, if a device does not get enough confirmations for a new block, it can delay the validation until the next round, when it can receive more confirmations.

2) Changes in N_W and N_D: In order to determine N_W and N_D, we require both witnesses and devices to register their IDs on the blockchain. A witness leaving can be detected by a retirement transaction that allows the witness to withdraw its deposit. Hence, like regular miners, the witnesses are incentivized to be online most of the time.

Unlike the witnesses, IoT devices can be intermittently disconnected. In such case, the gateway can save all of the confirmations from the witnesses that pass the first whitelist $\mathcal{B}_w$, so that when the device is back, it can resume the validation process by randomly selecting a subset of saved confirmations.

F. Example of parameter selection

Suppose the network consists of 10 000 witnesses with equal weight and 10 000 devices, i.e. $N_W = N_D = 10\,000$. First for $\mathcal{B}_c$, we choose $m_c = 1024, f_c = 0.5$ and $k_c = 8$. Hence, by Eq. (4), to achieve the false positive rate $p = 2^{-128}$, we would need confirmations from $\eta = 16$ witnesses. By Eq. (5), the estimated capacity of $\mathcal{B}_c$ is $n_c = 88$.

To have at maximum 88 out of 10 000 devices pass the whitelist $\mathcal{B}_w$, by Eq. (8) we choose $f_w = 0.5$ and $k_w = 7$. With such parameters, the expected number of witnesses remains for each device after the whitelist stage is $0.5^7 \times N_W = 78$. To have at least $\eta = 16$ witnesses selected out of 78 witnesses, a device can choose $k_s = 2$ and $f_s = 0.5$.

G. Operations on IoT devices

To verify a confirmation filter $\mathcal{B}_c$ from a witness $\mathcal{W}$, a device needs to perform the following steps:

1) Check if it matches $\mathcal{W}$'s whitelist.

2) Retrieve the shared key with $\mathcal{W}$ from server S, compute the corresponding codeword for the header and check if it is included in $\mathcal{B}_c$.

Additionally, if the block contains advertisements of new witnesses, the device also needs to verify these advertisements using Merkle proof and calculate new shared keys. We describe the details of all these steps in Algorithm 4.

Algorithm 4 Verification of a new block header B using a set of witnesses and their corresponding confirmations $\{B_{c_i}, \mathcal{W}_i\}$.

procedure VERIFYBLOCK($B, \{\mathcal{B}_{c_i}, \mathcal{W}_i\}$)
  for each $\mathcal{B}_{c_i}, \mathcal{W}_i$ **do**
    $\mathcal{B}_{w_i} \leftarrow GenWhitelist(B, m_w, f_w, k_w, \mathcal{W}_i)$
    if $D \notin \mathcal{B}_{w_i}$ **then**
      return false
    else
      Retrieve s_i from S
      $c_i = \mathcal{H}(s_i, B)$
      if $c_i \notin \mathcal{B}_{c_i}$ **then**
        return false
      end if
    end if
  end for
  if B has tx_{adv} **then**
    for each tx_{adv_i} **do**
      $UpdateWitness(tx_{adv_i})$
    end for
  end if
end procedure

procedure UPDATEWITNESS(B, tx_{adv})
  $\pi \leftarrow RetrieveMerkleProof(B, tx_{adv})$
  if $VerifyTx(tx_{adv}, \pi) == true$ **then**
    $\mathcal{W} \leftarrow RetrieveWitnessID(tx_{adv})$
    $s = DiffieHellman(\mathcal{W}, D)$
    Send $Encrypt(s)$ to S
  end if
end procedure

IV. SECURITY ANALYSIS

As addressed in Section II-B, we assumes that the gateway server S can deliver false information to device D. Since the

[3] As a result, the codeword c in Algorithm 2 can be computed as $c = \mathcal{H}(s_i, B_{i-n}|...B_{i-1}|B_i)$ where $\{B_{i-n}, \ldots, B_i\}$ are the n most recent blocks.

condition to select witnesses is generated by D itself, in order to make D accept a false block, S must have the selected witnesses include the false block in their confirmation filters $\mathcal{B}_c$. Recall that a shared key between D and the witnesses is used to generate $\mathcal{B}_c$, a successful attack depends on whether S can compromise the witnesses or not.

If S cannot compromise the selected witnesses, it is unable to know the shared keys. Therefore, it may have a false block accepted in 2 ways:

1) S alters all of the selected $\mathcal{B}_c$'s and finds a false block header that matches all of the fake filters. Since S does not know the secret shared keys, it is unable to know what bits are set for the block header, therefore the probability that S could find η $\mathcal{B}_c$'s that includes the header is $\binom{m_c}{k_c}^{-\eta}$. For a 1024-bit filter with $k_c = 8$ and $\eta = 16$, the probability is negligible.

2) S keeps the true selected $\mathcal{B}_c$'s but is able to find a false positive block header that matches all of the $\mathcal{B}_c$'s. We have shown in section III-C that by properly choosing parameters for $\mathcal{B}_c$ as well as the required number of witnesses η, the probability that S can successfully modify all of the selected $\mathcal{B}_c$ can be as low as 2^{-128}.

If all of the selected witnesses are compromised by S [4], they can send confirmations for the false block. In order to be selected, a witness $\mathcal{W}$ has to pass both the whitelist $\mathcal{B}_w$ and selection filter $\mathcal{B}_s$. For simplicity, we assume that all witnesses have equal weight. Thus, they have the same probability p to pass both filters. Since $\mathcal{B}_s$'s parameters are chosen based on $\mathcal{B}_w$, p can be calculated as follows:

$$\begin{aligned} p &= Pr(\text{pass } \mathcal{B}_s | \text{pass } \mathcal{B}_w) \times Pr(\text{pass } \mathcal{B}_w) \\ &= {f_s}^{k_s} \times {f_w}^{k_w} \end{aligned}$$

Assuming S can hold X out of N_W witnesses, the probability that exactly η compromised witnesses can be selected is

$$p_{compromised} = \binom{X}{\eta} \times p^{\eta}(1-p)^{X-\eta} \tag{9}$$

In table II we calculate $p_{compromised}$ for the example in section III-F, i.e. $\eta = 16, N_W = 10000$ and $p = 0.5^2 \times 0.5^7$. As shown in the table, unless S can acquire more than 50% witnesses, the compromised probability is quite small.

TABLE II: Probability that the whole set of selected witnesses is from an adversary given $n_w = 16, N_W = 10000, p = 0.5^9$

% of compromised witnesses	$p_{compromised}$
10%	2.7746×10^{-10}
20%	2.7345×10^{-6}
30%	0.0002594
40%	0.0037
50%	0.01872
60%	0.049205
70%	0.082288
80%	0.098874
90%	0.092296

V. Evaluation

As our method is based on random selection, it is important to evaluate how the confirmations are propagated through the network and if a node can receive a sufficient number of confirmations on time in order to carry out the selection. Therefore, we use a simulator to simulate a Bitcoin network in different conditions, i.e. the rate of block generation and block size. In addition, to show the feasibility of our mechanism, we also conduct an experiment using a low-power device.

A. Simulation to evaluate confirmation throughput

We use a customized version of the Bitcoin Simulator [13] on ns-3 to evaluate the throughput of the confirmation messages. We suppose that the witnesses can form an overlay network between themselves for faster communication. Using an overlay network on top of the existing blockchain is not new. For instance, Bitcoin's Lightning network [14] is an overlay network that supports instant payments and improves transaction throughput. Therefore, we simulate a network that consists of 100 witnesses that are also Bitcoin nodes. Each confirmation message is 192 bytes, including a 1024-bit confirmation filter $\mathcal{B}_c$ and the witness's ID and signature, which is 32 bytes each. Each witness sends out a confirmation after validating a new block. The confirmations are propagated by gossiping, i.e. nodes relay the messages if they have not heard of them before. For block generation, there are additionally 16 miners to produce new blocks. However, the miners are not witnesses, since they are only to trigger the selection process. Each simulation runs for 200 consecutive blocks.

We assume that the 100 witnesses send confirmations to the same set of devices and each of them is also a server to deliver the confirmations to some devices. Therefore, we measure the total number of confirmations each witness receives from the other witnesses for each new block. To account for delay, we only consider confirmations that are received before a new block arrives. Figure 5a plots the average percentage of confirmations received per block with the block interval varied from 3 seconds to 5 minutes. The block size is fixed to 500 KB for every interval. As shown in the figure, the higher the block rate, the less confirmations arrive on time. In addition, the throughput also has more volatility, as described by the error bars, when blocks arrive fast. This is expected as a long block interval would allow nodes to have more time to receive the confirmations. Moreover, since we set a fixed block size, it would take nodes a similar amount of time to receive a complete block and validate it before they can send out confirmations regardless of block intervals. In Figure 5b, we plot the delay to receive all the "on-time" confirmations. As the interval is longer, the delay becomes stable at around 1 minute. Since the block size is the same for every interval, based on Figure 5a, it can be inferred that this is also the delay for nodes to receive more than 90% of confirmations.

To evaluate the impact of block size, in our second experiment, the block size is set to follow the real distribution of

[4]Here we focus on the edge node S, but the analysis can be applied to any adversary.

Bitcoin's network, which is estimated by data from a Bitcoin explorer website [13]. Following such distribution, the block size decreases as the block rate becomes faster. The mean block size for each interval is provided in Table III. For instance, the mean block size when the block interval is 3 seconds is only 2.4 KB. Therefore, blocks can be propagated and validated faster. As a result, as shown in Figure 6, even when the block interval is short, nodes can still receive nearly all of the confirmations on time. The delay is always within the block intervals as well.

TABLE III: Block size according to Bitcoin block distribution

Mean block interval	Mean block size (KB)
3 seconds	2.4
15 seconds	12.6
30 seconds	23.8
1 minute	50.2
2 minutes	105.9
3 minutes	149.6
4 minutes	213.8
5 minutes	264.0

B. Performance on IoT devices

We conduct an experiment to evaluate the performance on a low-power device, which is an Arduino MKR1000 with 32KB RAM. It is connected via Wi-fi to a Dell laptop that acts as the server. Since we assume that the selected confirmations and new block headers are relayed to the device via its server, no blockchain network activities except block rate may affect the performance of the device. Therefore, we have the server simulate the witnesses and subscribe to the Bitcoin explorer website blockchain.info via websocket to receive new block headers instead of connecting to a real blockchain network. As shown in table IV, since it takes only 19 ms to verify one confirmation filter, then if 16 witnesses are required, the total time to verify a new block header is 304 ms. This is significantly smaller compared to the use of signatures, which could take 1.4 second for a single verification.

The most time-consuming operation in our mechanism is to compute a shared key between the device and a witness. Therefore, if the new block consists more than one new witness advertisement, the witness update process can take up to several seconds. However, the computation of shared keys can be delayed until the witnesses are actually selected to confirm new blocks.

Although the simulation shows that most confirmations can be received on time even when the block interval is short, we suggest that for blockchains with high block rates and devices that are accessed frequently, it is better to perform validation for every n blocks instead of one block at a time, as discussed in section III-E1, to reduce the workload on the devices.

VI. Related work

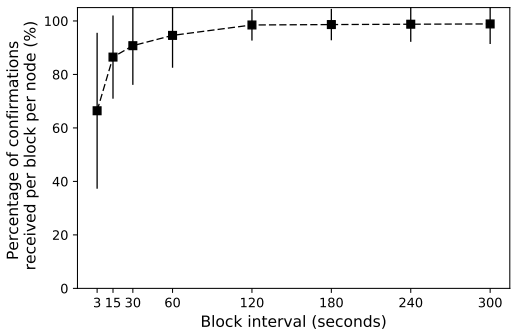
Access control is one of the research domains for blockchain applications in IoT environments that heavily uses the data immutability feature of blockchains. ControlChain [15] is a blockchain-based architecture for access authorizations to IoT resources. It supports relationship establishment between users and devices as well as rule and context definition. According to the authors, the resource-limited IoT devices can receive updates from ControlChain and blockchains in general with the support of other devices with the same owner, which means some trusted nodes are still needed. In addition, there is also no evaluation on real IoT devices. In [16], Dukkipati et al. propose a framework that applies the attribute-based access control model to blockchain. The framework is validated using a use case in smart cities. However, the authors did not address how the IoT devices can be brought to the scenario. On the other hand, we are particularly focused on how low-powered devices can access the blockchains.

TABLE IV: Performance on Arduino MKR1000

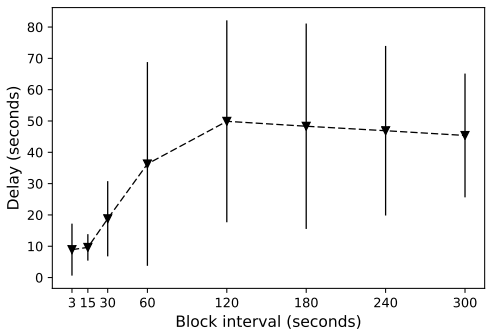
Action	Sub action	Time
Update witness	Verify Merkle proof	14 ms.
	Compute one shared key	460 ms
Verify block	Verify one $\mathcal{B}_c$	19 ms
Verify signature		1451 ms

FairAccess [5] and WAVE [8] are blockchain-based access control frameworks for IoT where permissions to IoT devices can be transferred between users. Both frameworks use Raspberry Pi as the IoT devices in their proof-of-concept. However, Raspberry Pi can be on the top-tier in terms of computation power while we consider less powerful devices. For smart home applications, Dorri et al. [7] propose a solution that does not use proof-of-work to avoid heavy computation. However, the smart home devices are not actually members of the blockchain network. Instead, there is an overlay network that consists of so-called smart home managers that manage transactions to and from the smart homes. Like other previously mentioned approaches, the use of smart home managers as central servers implies trust from the smart home devices.

To the best of our knowledge, we only found Slock.it to share the same idea of having supporting nodes to deliver blockchain data to IoT devices without a trusted server. Particularly, Slock.it introduces a network of nodes called INCUBED that sends signed block hashes to the IoT devices. The nodes are incentivized by making a deposit which they can lose if they are found to cheat by other nodes. Although our approach is similar, we do not use signed messages, thus the computation overhead can be reduced. Moreover, we randomize the node selection process to prevent malicious parties from bribing or attacking witness nodes. Algorand [17] is a consensus protocol that also has a random selection of a committee for each new block where nodes can select themselves with a verifiable proof. However, the voting process in Algorand requires not only public key verification but also strong communication between nodes to receive the votes, for which resource-constrained devices may not be capable.

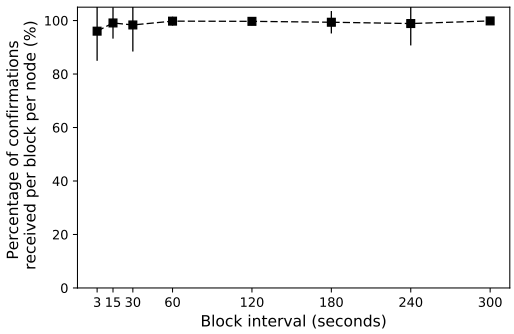


(a) Percentage of confirmations

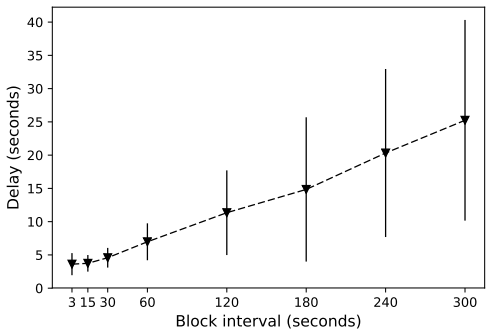


(b) Confirmation delay

Fig. 5: Average percentage of confirmations per block and confirmation delay with block size = 500 KB.



(a) Percentage of confirmations



(b) Confirmation delay

Fig. 6: Average percentage of confirmations per block and confirmation delay when block size varies depending on block rate.

VII. Conclusion

We propose a lightweight method for resource-constrained IoT devices to validate new blocks with the help of witnesses from the blockchain network. Using Bloom filters, the IoT devices can perform validation without complex computation. It is shown by both our simulation and experiment on a low-power device that our method is achievable.

References

[1] A. Reyna, C. Martín, J. Chen, E. Soler, and M. Díaz, "On blockchain and its integration with iot. challenges and opportunities," *Future Generation Computer Systems*, 2018.

[2] "Samsungs SmartThings smart home hub has been down since yesterday for some users," https://www.theverge.com/circuitbreaker/2018/3/13/17115624/samsung-smartthings-outage-over-14-hours-update, 2018, [Online; accessed 19-Aug-2018].

[3] K. Lab, "Somebodys watching! When cameras are more than just smart," https://ics-cert.kaspersky.com/reports/2018/03/12/somebodys-watching-when-cameras-are-more-than-just-smart/, 2018, [Online; accessed 19-Aug-2018].

[4] E. Fernandes, A. Rahmati, J. Jung, and A. Prakash, "Decoupled-ifttt: Constraining privilege in trigger-action platforms for the internet of things," *arXiv preprint arXiv:1707.00405*, 2017.

[5] A. Ouaddah, A. Abou Elkalam, and A. Ait Ouahman, "Fairaccess: a new blockchain-based access control framework for the internet of things," *Security and Communication Networks*, vol. 9, no. 18, 2016.

[6] S. Huh, S. Cho, and S. Kim, "Managing iot devices using blockchain platform," in *19th International Conference on Advanced Communication Technology (ICACT)*. IEEE, 2017.

[7] A. Dorri, S. S. Kanhere, R. Jurdak, and P. Gauravaram, "Blockchain for iot security and privacy: The case study of a smart home," in *PerCom Workshops*. IEEE, 2017.

[8] M. P. Andersen, J. Kolb, K. Chen, G. Fierro, D. E. Culler, and R. A. Popa, "Wave: A decentralized authorization system for iot via blockchain smart contracts," 2017.

[9] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[10] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Commun. ACM*, vol. 13, no. 7, pp. 422–426, Jul. 1970.

[11] R. C. Merkle, "Protocols for public key cryptosystems," in *1980 IEEE Symposium on Security and Privacy*. IEEE, 1980, pp. 122–122.

[12] S. J. Swamidass and P. Baldi, "Mathematical correction for fingerprint similarity measures to improve chemical retrieval," *Journal of chemical information and modeling*, vol. 47, no. 3, 2007.

[13] A. Gervais, G. Karame, K. Wst, V. Glykantzis, H. Ritzdorf, and S. Capkun, "On the security and performance of proof of work blockchains," in *Proceedings of the 23nd ACM SIGSAC Conference on Computer and Communication Security (CCS)*. ACM, 2016.

[14] J. Poon and T. Dryja, "The bitcoin lightning network: Scalable off-chain instant payments," *See https://lightning. network/lightning-network-paper. pdf*, 2016.

[15] O. J. A. Pinno, A. R. A. Gregio, and L. C. E. D. Bona, "Controlchain: Blockchain as a central enabler for access control authorizations in the iot," in *IEEE GLOBECOM 2017*, 2017.

[16] C. Dukkipati, Y. Zhang, and L. C. Cheng, "Decentralized, blockchain based access control framework for the heterogeneous internet of things," in *Proceedings of the Third ACM Workshop on Attribute-Based Access Control*, ser. ABAC'18. New York, NY, USA: ACM, 2018.

[17] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, "Algorand: Scaling byzantine agreements for cryptocurrencies," in *Proceedings of the 26th Symposium on Operating Systems Principles*. ACM, 2017.