Bloom Filter based Collective Remote Attestation for Dynamic Networks

Salvatore Frontera
frontera.1710456@studenti.uniroma1.it
salvatore.frontera@hotmail.com
Sapienza University of Rome
Rome, Italy, IT

Riccardo Lazzeretti
lazzeretti@diag.uniroma1.it
Sapienza University of Rome
Rome, Italy, IT

ABSTRACT

Nowadays, Internet of Things (IoT) devices are widely used in several application scenarios. Due to their cheap structure, they often do not guarantee high security standard, making them prone to hacker attacks. Remote attestation is widely used to verify the configuration integrity on remote devices. Unfortunately, checking the integrity of each single device is impractical, thus several collective remote attestation protocols have been recently proposed to efficiently run attestations in wide device swarms. However, current solutions still have several limitations in terms of network topology, scalability, and efficiency.

This paper presents a new efficient collective remote attestation protocol for highly dynamic networks. Our protocol is implemented according to the self-attestation procedure, where devices iteratively establish a common view of the integrity of the network through a consensus mechanism. Differently from previous protocols, we leverage on Bloom filters, which permits to drastically reduce the message size for communication and to be more flexible with mobile nodes that can also join or leave the swarm. We evaluate our proposal through several simulations and experiments, showing that it outperforms the state of the art.

CCS CONCEPTS

• **Networks** → **Security protocols**; • **Security and privacy** → **Intrusion/anomaly detection and malware mitigation**; **Distributed systems security**; **Mobile and wireless security**.

KEYWORDS

Collective Remote Attestation, Dynamic Networks, Bloom Filter.

ACM Reference Format:
Salvatore Frontera and Riccardo Lazzeretti. 2021. Bloom Filter based Collective Remote Attestation for Dynamic Networks. In *The 16th International Conference on Availability, Reliability and Security (ARES 2021), August 17–20, 2021, Vienna, Austria.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3465481.3470054

ARES 2021, August 17–20, 2021, Vienna, Austria

ACM ISBN 978-1-4503-9051-4/21/08…$15.00
https://doi.org/10.1145/3465481.3470054

1 INTRODUCTION

An IoT environment combines different aspects concerning infrastructures, protocols and communications. It consists of smart devices that generally use embedded and low power systems, such as processors, sensors, and communication hardware. IoT devices need to send, receive and work with data acquired from their environments in order to execute complex tasks. They share collected information by connecting to an IoT gateway or some other devices, where data is either locally analyzed or aggregated and transmitted to cloud services. Nowadays, the application of smart objects is growing exponentially, becoming an integral part of our life.

Despite the benefits of facilitating operations and services, IoT devices present several security flaws. In fact, due to low-cost implementations, IoT systems lacks of security mechanisms [15], making them prone to cyber attacks. IoT devices present several vulnerabilities, which adversaries can exploit with little effort, endangering service integrity, availability, and compromising users' data privacy. The negligence in this field brings to the exposure of sensitive information, ranging from unprotected video streaming like baby monitors [19] to the obtaining of unauthorized emails, passwords, and voice recordings. In order to solve such problems, there exist several protocols [14, 26] concerning the three main IoT security requirements: authentication, confidentiality, and access control.

Proposed in recent years, IoT Remote Attestation (RA) [21] is a protocol that allows a verifier (V) to identify a possible compromised remote platform, called prover (P), from the software point of view. RA allows V, considered a trusted entity, to obtain proof of the integrity of the configuration of an untrusted device, through a challenge-response protocol between them. However, the traditional challenge-response attestation protocols, which are defined between the verifier and a single prover, have problems concerning scalability. Thus, researchers recently started to develop new attestation schemes, named Collective Remote Attestation (CRA), which are capable of performing attestation over a large network of IoT devices. They are based on end-to-end or hop-by-hop communication, allowing a verifier to obtain a unique result from a network of smart devices.

Ideally, a CRA scheme should be scalable in the network size, allow nodes to move, join, leave the network or to go idle, provide strong security assumptions, be efficient in terms of runtime, cheap in terms of power and communication complexity, expressing the collective status of the system as well as the status of every device, etc. However, despite several CRA schemes have been proposed, there still are different challenges for the implementation and we are far from satisfying all these requirements. Since the first CRA attestation schemes, the static scenario has been a common assumption

in the CRA literature [2, 5, 9, 12]. It permits high scalability and efficiency, but assumes that devices of the swarm maintain connectivity during the entire execution of the CRA protocol. This drastically influences the protocols usability in specific environments, such as areas with connectivity problems or motion devices.

In dynamic CRA [4, 13], attestation results needs to be combined with key management, network discovery, and secure and efficient routing, ensuring that compromised devices cannot evade detection during attestation and non-compromised devices are not counted twice. Furthermore, computation and communication costs for the verifier and provers should be minimized, and the swarm burden should be distributed over the network. Recent protocols rely on attestation broadcasting and aggregate each single device attestation [13], at the expense of efficiency, or represent the status of each device in a binary mask [4], which implies that each device must know how many nodes compose the network and the owner previously assign an index to each device.

Contributions. This paper proposes an innovative approach for collective remote attestation of IoT systems. The main goal is to improve the state of the art of CRA schemes in terms of performance and flexibility for dynamic swarms. The paper describes a collective attestation in a dynamic network of devices that communicate with each other in order to understand the status of the swarm. In our model, we assume that nodes can move, join and leave the network, or have intermittent activity, even while participating to the CRA protocol. Differently from previous protocols, our solution estimates the network status, instead of the exact knowledge of each device status. The protocol bases its implementation on two algorithms: Bloom filter and maximum consensus. The first one provides an efficient data structure that allows to collect each result, reducing runtime performance, and to be more scalable in the number of devices, in joining and leaving the network. The second one allows to reach a convergence of the network state through the collection of each attestation proof from devices. From the final collective result a verifier can i) easily check whether the network contains compromised devices by querying any device of the network; ii) estimate the number of compromised devices; iii) check if a specific device is compromised (this is not the main purpose of the protocol and we tolerate errors).

Our experimental results confirm the suitability of our protocol for low-end devices and highly unstructured networks, reaching better performance compared to previous proposals operating in dynamic swarms. In fact it provides: i) flexibility in joining and leaving of devices; ii) scalability regarding the number of devices; iii) high runtime efficiency.

Outline. The rest of this paper is organized as follows: in Section 2, we review related works; in Section 3, we present the system model with the relevant adversary model; preliminary protocol features are outlined in Section 4; then, Section 5 illustrates our proposal; Section 6 shows complexity analysis and simulation results; finally, in Section 7, we provide some conclusions and discuss possible directions for future works.

2 RELATED WORKS

Collective Remote Attestation protocols [3] have recently been proposed in order to efficiently verify groups of devices. In general these protocols are implemented with the presence of a Trusted Execution Environment (TEE), where the code executors have high levels of trust through the isolation. The presence of minimal secure hardware components in the TEE, such as Read Only Memory (ROM), a Memory Protection Unit (MPU), and a reliable clock, is sufficient to guarantee the right execution of the protocol, permitting to ignore threats from the rest of the device. From SEDA [5], the first CRA protocol proposed in literature, other enhanced approaches were porposed. Similarly to SEDA, most of them perform attestations over a spanning tree with the verifier as root and therefore can operate only on static networks, which limits the usability in several practical scenarios. Recently, SALAD [13] and PADS [4] proposed innovative CRA protocols for highly dynamic swarm topologies. SALAD provides an innovative aggregation protocol for networks with intermittent connectivity. All devices within communication range mutually attest each others' software integrity and then broadcast attestation results through the network, gradually expanding their view of the network state until almost all devices share the same result. Unfortunately, SALAD is computational expansive, making the proposal potentially unsuitable for large networks, wherein collective remote attestation requires minutes. PADS has similar result distribution approach with respect to SALAD, but it turns the attestation into a consensus problem, where provers share attestation results after a self-attestation procedure, and iteratively converge to a "snapshot" of the network state. Verifier can obtain the collective status of the network by querying any device in the network. Network status is represented through a bitmap where a couple of bits is assigned to each device and different values represents different status: healthy, compromised, or unknown. Due to this static bitmap, the network owner must reconfigure all the network devices each time that a node must be added or removed. However, differently from SALAD, PADS requires few seconds for the collective attestation of the network. Thus, we just compare our protocol with PADS.

3 SYSTEM MODEL

The general system model for CRA schemes [3] contains several heterogeneous devices from a software and hardware configuration point of view, connected in a network. Such devices are able to identify and communicate with their direct neighbours and can randomly move within the area. Because of this dynamic scenario, they are seldom tightly-coupled within a networking infrastructure, which consequently reduce connections and successful packets exchange. In fact devices cannot always guarantee their availability, due to intermittent connectivity or completely absence of communication. This is the case of large groups of embedded devices, such as industrial control systems, IoT devices in smart environments, drones, robots, etc., that can compose a network of either static or dynamic topology.

As described in Figure 1, there are four main entities considered in the system model:

- Network Owner (O): the entity for network setup and maintenance, providing the necessary cryptographic material and credential for attestation.
- Verifier (V): the entity that acts as an attestation checker, on request of the owner. Generally, it executes the attestation

process sending an attestation request to a device, and collects the attestation result as an aggregated response.
- Prover (P): each device to be checked by the verifier. It computes the proof of integrity with respect to its correct configuration. At the end of the protocol, any device is considered healthy whether using the correct configuration, or compromised.
- Aggregator (A): a device that relays messages among entities in a network and aggregates responses from neighbours in the topology. Whenever possible, it guarantees better coverage of the network and better performance. In our decentralized environment, provers also act as aggregators within the network.

Adversary Model. The adversaries considered for this protocol are the following:

- Communication adversary: an adversary that can obtain full control of the communication channels among provers and Verifier. Possible attacks can be: replay attestation reports used for the communication among provers, forge attestation reports to edit their values, and drop attestation reports from provers.
- Software adversary: an adversary that can run malicious code or firmware on a device, exploiting software vulnerabilities, or read unprotected regions and manipulate software state.
- Mobile software adversary: an adversary able to compromise the software configuration of a device and then to eliminate the malware used in order to be not visible to attestation protocols. Moreover, the adversary changes location continuously to evade detection and can perform malicious activities during two consecutive attestation periods.

Key management. In this work, similarly to PADS [4] and many other CRA protocol, we rely on symmetric encryption. It enables the use of a master key k_{att}, generated by O, shared by all the devices in the swarm, and stored in a hardware-protected storage. This choice can be weak in case of the physical attacker presence. In fact, if a physical adversary captures a device, the presence of a single secret key compromises the entire network, since the adversary is able to forge every message. In order to mitigate this condition, the protocol can be easily modified to have an individual private/public key pair per each device, with a consequently increase of the cost performance.

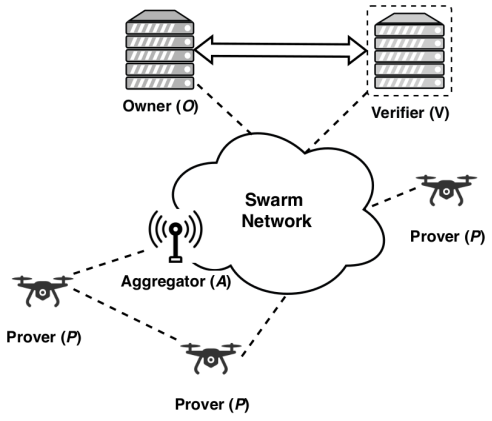


Figure 1: Swarm Network model [3].

4 PRELIMINARIES

4.1 Requirements

Hardware. Our protocol needs specific hardware components to guarantee the correctness of self-attestation and messages exchanged during consensus. Our minimal requirements are the same of other preceding protocols in the literature. Thus, we base them on SeED [9], the protocol that firstly introduce self-attestation of the devices evaluated into a TEE, and PADS [4]. The implementation design of this proposal needs an enhanced version of TrustLite [11], which is equipped with a simple Memory Protection Unit (MPU), to ensure access limitation to data, and a secure boot, to guarantee the authenticity and confidentiality of both data and code isolated components. The integrity of the secure boot code is also guaranteed by storing it in read-only memory (ROM).

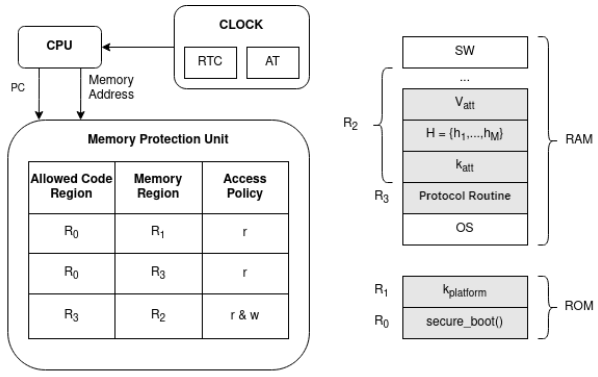


Figure 2: Protocol implementation based on SeED [9], TrustLite [11], and PADS [4].

SeED extends TrustLite with a Real-Time Clock (RTC), which is write-protected and not modifiable via software, and an Attestation Trigger (AT), which updates and monitors the value of a timer in order to start the attestation procedure. The implementation is described in Figure 2. All the parameters required for attestation (secret seed V_{att}, set of configurations $H = \{h_1, h_2, ..., h_M\}$ and secret attestation key k_{att}) are stored in a writable memory, and are read and write protected by MPU rule, ensuring the exclusive access only to our protocol code. The integrity of our protocol code is ensured by secure boot.

Communication protocol. All the communications in this protocol are carried out over the IEEE 802.15.4 protocol, which is the main standard protocol for IoT devices [27]. This protocol defines a standard for low rate wireless personal area networks (LR-WPANs), implementing a communication based on low data rate, low cost wireless networking, and low power consumption. IEEE 802.15.4 provides a maximum data rate of 250 Kbps, communication range of around 75 m, and a frame size of 127 bytes, including the header required for the layers.

4.2 Bloom filter

A Bloom filter [6] is a space-efficient probabilistic data structure that is used to check whether an element is a member of a set. They are used in several applications and, among them, demonstrated their utility in network security for preventing distributed denial of service [18], deep packet inspection [7], traffic mapping [23], etc., and is also one of the main component of our protocol.

With this data structure, false-positive matches are possible, but it is not possible to have false-negatives. In fact, a search for an element that is not in the filter can give an incorrect answer, but it always retrieves a positive result in case of its presence in the filter. In the standard Bloom filter implementation, elements can be added to the set, but not removed, increasing the probability of false positives as long as items are inserted. A Bloom filter is a bit array of m bits, which initially are all set to 0. In order to insert an item in the filter, k different hash functions are evaluated on some item identifier, each one mapping the item to one of the m array positions, generating a uniform random distribution. Therefore, to add an element, Bloom filter must do an insertion for each of the k hash functions, setting k array positions to 1. Typically, k is a constant which depends on the desired false error rate. At the same time, m is proportional to k and the number of elements to be added. Of course, in a general approach, the number of hash functions must be limited in order to avoid computational delays for each insertion, but at the same time optimized with respect to the false positive probability. The hash functions used in a Bloom filter should be independent and uniformly distributed.

To test if a component is in the set, its identifier must be input to the hash functions to get k bit positions:

- If any of the bits at these positions is 0, the element definitely is not in the set.
- If all of them are 1, then the element may be in the set or we have a false positive, due to insertions of other elements that collide with the results of the hash functions.

An example is represented in Figure 3. Given a filter populated with $x,\ y,\ z$, by checking if some elements are in the Bloom filter, obviously we obtain that $x,\ y,\ z$ are in the filter, while the element w has not been inserted. On the other side, the element u is a false-positive in the set.

Performance. Bloom filters have great advantages in terms of space with respect to other data structures, such as trees, hash tables, or simple arrays, that require storing at least the data items themselves. On the other side, Bloom filters do not really store items. In fact, they save only the element reference with respect to the output of each hash function used. Moreover, they also have the property of fixing the time cost for adding or checking an element, which requires for both $O(k)$, completely independent of the number of items already in the set.

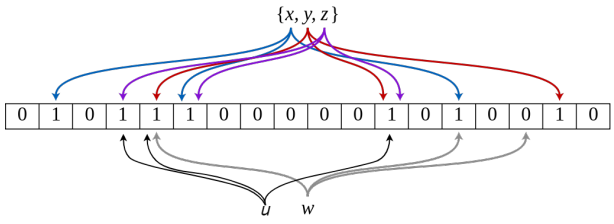


Figure 3: Bloom filter example. On the top, $x,\ y,\ z$ represent items in the filter. On the bottom $u,\ w$ represent items to be searched, where w correctly results not in the filter, while u is a false positive.

4.3 Maximum consensus

Consensus protocols [17] overcome the need for distributed and fault-tolerant computation and information exchange algorithms. A maximum consensus protocol allows broadcasting the devices knowledge among devices efficiently, converging to a state equal to the maximum of devices' input.

In this paper we rely on asynchronous consensus protocols, where nodes periodically broadcast their status within their communication range. In practice, any node receives information from the neighbours, updates its status, and broadcasts the new state to its neighbours in the next step. The state of all the network nodes will converge to the same consensus after a certain number of steps. In [8, 24], authors demonstrate that distributed consensus protocols converge in a finite amount of steps T, which depends on the connectivity of the graph.

The input of the device i to the maximum consensus protocol is represented by x_i^0. Starting from x_i^0, the device generates a sequence of observation states $\{x_i^t\}_{t=0}^{T}$, through which at every step t the node i updates its status by computing $x_i^{t+1} = \max_{j \in N_i \cup \{i\}} x_j^t$, where N_i represents the neighbours of the node i. According to [16], the state of each device converges to $\max_{i \in \text{Network}}(x_i^0)$.

5 NEW APPROACH FOR AN EFFICIENT ATTESTATION

We here present a new protocol, based on PADS [4], that implements a collective attestation in a dynamic scenario, where each device periodically broadcasts its knowledge of the network status and aggregates the results to reach a bitwise maximum consensus of the overall network. Differently from previous works, the proposal uses a data structure called Bloom filter, that permits to reduce the packet size and consequently the time of coverage of the network.

In this proposal, a verifier wants to check the status of the network concerning the number of compromised devices, tolerating a compromise state for only a small part of the network. Hence, our protocol is used in order to verify whether the number of compromised devices is below a given a threshold, which can be linked to the number of positions set in the Bloom filter. If the protocol outputs a value below a certain percentage of compromised devices, then the swarm can carry on its tasks safely. Otherwise, it reaches a condition where the swarm is probably not reliable and collective attestation returns a compromised state.

The main advantage is that, differently to PADS, device can dynamically join and leave the swarm, without the necessity of changing the protocol's configuration. Moreover the use of the Bloom filter reduces the message size significantly. However, due to the Bloom filter usage, it is impossible to understand with certainty which devices are compromised, because we cannot exclude collisions, but V can assert that a specific device (maybe an important

central node) is healthy at the self-attestation timestamp if at least one of the corresponding positions on the Bloom filter are not set.

Another problem is that our protocol does not consider the possibility that compromised devices evade detection by not communicating with the network (such as for packets dropped or delayed by adversaries). In this case, the verifier does not know if there are hidden compromised devices not partecipating to the CRA protocol. Nevertheless, this simple attack to the protocol does not influence the correctness of swarm operations. In fact, we assume that healthy devices and aggregators do not interact with devices that are not sharing their attestation knowledge.

We base the protocol evaluation on the ability of covering the area of reachable provers, using the following definition defined in PADS:

Definition 5.1. Coverage: We say that at time t our proposal has coverage $c_X^t = Y$, if at least a portion $X \in [0, 1]$ of the provers in G hold information of at least a portion $Y \in [0, 1]$ of the reachable provers population.

Intuitively, increasing the level of coverage, the protocol requires more update steps, increasing consequently the time required for the execution. From this definition, in order to measure the runtime, we derive the notion of Minimum Coverage Time (MCT), defined in [4]:

Definition 5.2. Minimum Coverage Time: given a desired coverage level $c_X = Y$, t_m is the minimum amount of time necessary to reach c_X Formally: $t_m = \arg\min_t \ c_X^t$.

5.1 Protocol description

Our approach, similarly to PADS [4], comprises four phases: initialization, self-attestation, consensus, and verification. We remember that the four protocols are evaluated inside a Trusted Execution Environment.

Initialization. The initialization sets the main parameters for the attestation. Our protocol assumes for simplicity that nodes in the network share the symmetric key k_{att}. Similar to previous works, each prover is provisioned with a set $H = \{h_1, h_2, ..., h_M\}$ of known good configurations defined by hash values, which can be either static or securely updated. Moreover, provers are characterized by a shared secret seed V_{att}. Seed allows to autonomously calculate a pseudorandom sequence of self-attestation times, hiding next attestations to the attackers. Another possible idea is to store the time sequence of self-attestation on secure memory, but this increases the memory overhead of each device. Initialization also sets the parameter of the Bloom filter, i.e. its bitsize m and the number k of hash functions to evaluate during self-attestation.

Self-Attestation. The self-attestation operation is computed in a certain point of time, defined by T_{att}. It is generated every time by the device's secure clock using a pseudorandom number generator initialized with the secret seed and triggered by a secure clock's function schedule. At the time T_{att}, each prover simultaneously executes a self-attestation procedure. In fact, the prover computes its measurement value h and checks by own its correctness against the set of right configurations H. The result is a binary value: 1 if the configuration is a correct one, and 0 otherwise. Differently from PADS, where each device has two bits exclusively assigned to share its status, the prover initializes the Bloom filter $x_i^0 = \{0\}^m$, i.e. generates a null sequence of m zeros and whether the output of self attestation is 0, the prover evaluates k distinct hash functions from its unique id in input to determine which positions must be set to 1 in the Bloom filter. The unique id can be for example the MAC address, the GPS location, an identifier assigned by the owner, etc. We rely on non-cryptographic hash functions, such as MurmurHash [25], computed on some unique identifier of the device (such as the IP address and MAC address).

Consensus. In this phase, provers periodically update their knowledge about the network status, using a modified version of the distributed maximum consensus described before, where the maximum is bitwise computed. Each prover maintains its status of the whole network, corresponding to its current knowledge of the network health, saving the Bloom filter in its read and write protected memory. Each prover updates it iteratively every time it receives a valid message from its neighbours. In the meanwhile it periodically broadcast the Bloom filter (x_i^t) to allow other devices update their status. Each message is characterized by x_i^t, attestation Time T_{att}, current broadcast timestamp T, and a MAC taken over them for message authentication. We rely on SHA-1 for MAC, considered a good balance for collision-avoidance and performance. Note that, a compromised device transmitting a fake message can be identified from a wrong MAC. Every other prover receiving a broadcast message, after verifying the MAC, checks whether T is in the current interval, in order to prevent replay attacks, and finally executes the maximum consensus considering the following equation:

$$\begin{aligned} x_i^{t+1} &= \text{maxatt}(x_i^t, \{x_j^t\}_{P_j \in N_{i,t} \cap R_t}), \\ \text{s.t., } x_i^{t+1}[l] &= \max(x_i^t[l], \{x_j^t[l]\}_{P_j \in N_{i,t} \cap R_t}), \ l = 0, \ldots, m, \end{aligned} \tag{1}$$

where $N_{i,t}$ is the set of neighbors of the prover P_i at time t and R_t is the set of active provers. In practice, for each position l of the Bloom filter, the prover P_i computes the bitwise maximum between the filter he computed in the previous step and the filters received by the neighbors. Note that, due to the usage of the Bloom filter with a combination of zeros and ones, the bitwise maximum can be easily computed with an OR operator. Furthermore, in order to guarantee the correctness of the consensus results, it is assumed that x_i^t is stored in an access-restricted area of the prover, protected by MPU inside the device.

Verification. In the verification phase, the verifier collects the estimation of the network status x^t by contacting a prover, randomly chosen from the network with respect to some condition, such as the nearest prover or the first message obtained. The verifier executes this final step in any moment after a reasonable amount of time T_{MAX}, which could be pre-established or randomly chosen by V. The final verification step is similar to receive a message broadcasted by a prover during consensus. In fact, the verifier checks the MAC of the message, and whether T resides within the valid time interval $[T_{att}, T_{MAX}]$. If any of the above checks fail, the verifier considers the device to be compromised, discards the attestation result, and query any other device. Otherwise it obtains an estimation of a Bloom filter where the bits of all the compromised devices in the swarm are set to 1. A null x^t indicates the absence

of compromised device in the network. Otherwise, by the number of bits set to 1, V can estimate the number of compromised devices. According to [22], the number of items in the Bloom filter, i.e. the number of compromised devices, is

$$n_c \approx \frac{m}{k} \ln\left(1 - \frac{X}{m}\right),$$

where m is the bitlength of the filter, k is the number of hashes, and X is the number of bits set to 1. Whether interested to the status of a given device, V can check device correctness by evaluating MurmurHashes with respect to device identifier. If at least one of the bits is null, V can deduce that the device is healthy, otherwise the device could be compromised. Indeed this result is not really reliable. A compromised device could result healthy because not participating to the CRA protocol or because the protocol has not propagated its status to the node contacted by the verifier. Moreover, an healthy device could be considered compromised due to collisions in Bloom filter insertions.

5.2 Security

In this section, we discuss the security of our proposal against the main adversary models, previously described in Section 3. In a collective attestation protocol, the goal of an adversary is generally to change the configuration of swarm devices, modifying the firmware or data, and hide their malicious operation from the verifier. In order to achieve this, the attack should be executed for a non-negligible time, in which the adversary can restore the prover to its original state within a negligible period of time.

We consider a collective attestation protocol for dynamic networks to be secure if it is computationally infeasible for a polynomial attacker to induce the verifier to accept a fake attestation result. It follows that this proposal is a secure collective attestation protocol for dynamic networks, if:

- the adopted pseudo-random number generator is cryptographically secure;
- the MAC, e.g., a Hash-based MAC (HMAC), in use is selective forgery resistant;
- hash function is collision-resistant.

This approach adopts similar PADS conditions to perform the self-attestation and consensus. Thus, we can refer to [4, 9] for a security proof of self-attestation and consensus. In the following subsections, we discuss the security concerning a communication adversary, a software adversary, and a mobile adversary.

Communication Adversary. Given full control over communication channels between verifier and prover and among provers, the adversary either tries to forge the messages exchanged, or tries to reuse an old good attestation report, making a replay attack. In our case, both the attacks will fail, since:

- the probability for verifier or a prover to accept a message with a forged MAC is negligible for a polynomial number of steps of MAC operation;
- the freshness of every received message is guaranteed by the time of attestation T_{att} and the timestamp T.

Software Adversary. A software adversary can try different approaches to remain undetected: edit the process responsible for attestation, extract prover's authentication key or other security parameters, and handle provers' clock in order to obtain future measurements. However, these attacks are infeasible for a polynomial attacker, since:

- the integrity of the measurement code is protected through Read-only Memory (ROM);
- k_{att} is protected by appropriate rules in Memory Protections Unit (MPU);
- each prover has a Real-Time Clock (RTC), which is not accessible by any software.

Moreover, similarly to the self-attestation procedure, the consensus code is protected by secure boot, so it cannot be accessible.

Mobile Software Adversary. Mobile Software Adversary scenario assumes that an adversary compromises a prover before the next attestation protocol, convincing the verifier to output a wrong configuration state and being hidden from the protocol, possible for example going out from the communication range. In this way, the adversary evades detection and restores the prover to its original state. Since the adversary can restore a prover in a negligible amount of time, it has to know the exact attestation time in order to execute this attack. However, the mobile adversary does not have the possibility to get the needed information, due to the presence of MPU protection, and the probability of predicting is negligible with respect to the pseudo-random number generator.

6 SIMULATIONS AND RESULTS

This section provides the results of the simulations carried out for testing our proposal in different setups. We analyze the communication, memory, and energy costs required by our protocol, and we compare the results with PADS to better outline the improvements respect to the literature. Other CRA protocols are not considered in our comparisons because applicable only to static networks. To the best of our knowledge, the only other CRA protocol for dynamic networks is SALAD [13], but it uses asymmetric cryptography to provide increased security, in spite of runtimes which are orders of magnitude higher than PADS. We implement our protocol by using the INET framework, differently from PADS [4], where MiXiM [1] is used. To provide a fair comparison, we also re-implement and evaluate PADS.

We simulate dynamic networks of small-medium sizes, starting from 128 devices. Due to high computational requirements in simulation, we limit the number of devices to 2048. We adopt a fixed rate for broadcasting the Bloom filter, i.e., 500 ms. Furthermore, we consider all the provers, either good or compromised, to participate in the attestation process. As outlined, compromised device has few advantages to go offline, because if they do not participate to CRA protocol, they should be cut off also by other network activities.

We rely on the IEEE 802.15.4 protocol, as outlined in Section 4.1. However our protocol can work also with other protocols such as Bluetooth, LoRaWan, etc. The frame size is often insufficient for the transmission of the whole message. Therefore, in several protocol executions the messages exceed the packet size, requiring message fragmentation. This brings to an increase in the number of packets, and thus an increase in runtime simulations that is not linear in the number of bits of the Bloom filter.

We evaluate the performance through experiments in which provers move according to a random path, making network connections dynamic and causing disconnections. We randomly deploy them over a simulated area of size proportional to the number of devices, ranging from 125×125 m^2 for 128 devices to 500×500 m^2 for 2048 device.

We measure the runtime of our proposal as the average MCT in 50 runs of each simulation with different target coverage levels, i.e., 85%, 90%, and 95%. For simplicity, we do not implement any single operation in OMNeT++, but we focus on the implementation of the consensus and verification phase, which are the crucial parts to evaluate runtimes. We use benchmarks to evaluate the complexity of the protocol (according to [5] and [9]), in which there are estimation times of various operations of interest, and we set time delays to estimate the self attestation procedure, considering all the devices already inizialized. We consider 48 ms as the time necessary to compute HMAC and hash operations [9], and 187 ms as the overhead of every self-attestation operation. As described in [9], the latter is approximated by the generation of 32 random bits, and the calculation of a hash over the amount of memory to be attested. Moreover, self-attestation eventually computes also the insertions in the Bloom filter, with at most 96 ms for each device. In fact, as described in [10], in the case of usage of more than 2 hash functions, as in the Bloom filter population, we can leverage on a standard technique based on double hashing (e.g. $g_i(x) = h_1(x) + ih_2(x)$) that permits to simulate additional hash functions without any loss of false positive probability. However, since we do not have measures of MurmurHash [25], we upperbound its runtime with the one of SHA-1.

In our experiments, we vary the number of compromised devices in percentage and the tolerated false-positive probability in order to examine the Bloom filter dimension and consequently evaluate how packet fragmentation affects protocol complexity.

6.1 Protocol complexity

We analyze the complexity of the proposed protocol in terms of memory, energy and time required.

Memory overhead. Considering scenarios where all the provers share the same symmetric attestation key k_{att}, the memory overhead required by our protocol depends on different aspects: the attestation key k_{att}, the number of allowed configurations in H, and the size of the Bloom filter m. Bloom filter size depends on the settings chosen for the simulation, i.e. the number of provers to be inserted and the probability of false positives (always considering optimal the number of hash functions). Thus, considering each element of H of 160 bits (20 bytes), we define the memory overhead as $k_{att} + m + 160 \times |H|$ bits.

Communication overhead. The communication overhead is relative to consensus phase, where each prover broadcasts and receives messages periodically. The message is composed by Bloom filter (m bits), the attestation time T_{att} (32 bits), the timestamp T (32 bits), and one HMAC (160 bits). In total each brodcasted message is $m + 224$ bits long.

The size of the message depends on the bitlength of the Bloom filter, and impacts on the number of packets to be sent on the communication link. In low power settings, in order to have better performance, it is preferable to limit the number of packets to be transmitted for each message. Moreover, as we will see in Section 6.3, increasing the fragmentation increases drastically the runtime of the protocol, and consequently the time required for reaching a certain coverage.

Energy Consumption. We estimate the energy consumption for an update step in the consensus protocol and for the computation of the main cryptographic operations, similarly to [4].

Let $\mathcal{E}_{send}, \mathcal{E}_{recv}, \mathcal{E}_{hmac}, \mathcal{E}_{filter}$ and $\mathcal{E}_{att}$ denote, respectively, the energy required to send a byte, receive a byte, calculate one HMAC, compute Bloom filter insertion, which depends on the number of hashes, and perform self-attestation. At timestamp t each transmitting prover sends a message which content and size are provided in previous paragraph. Thus, we define the estimation of the energy consumption for sending and receiving a single message as:

$$\mathcal{E}_{send}^{Prover_i} = \mathcal{E}_{send} \times \left(\frac{224 + m}{8}\right);$$

$$\mathcal{E}_{receive}^{Prover_i} = \mathcal{E}_{receive} \times \left(\frac{224 + m}{8}\right).$$

Defining n as the number of steps executed by each device for the consensus, we estimate the energy required by our protocol considering that each prover shares its Bloom filter at fixed time intervals $(t_1, t_2, ..., t_n)$. From the implementation point of view, the number of time intervals is the same for all devices, but they transmit in different moments. In fact, each device broadcasts in random instants of each interval in order to increase the possible packet reception. Considering $N_{i,t}$ the set of neighbors of the prover at time t and R_t the active provers, a prover computes an HMAC, broadcasts a packet, receives a number of broadcast messages from its neighbors $N_{i,t}$, and verifies the HMAC associated with them. We estimate the energy required by our protocol for a prover as:

$$\mathcal{E}^{Prover_i} = \mathcal{E}_{att} + \mathcal{E}_{filter} + \mathcal{E}_{max}.$$

where $\mathcal{E}_{max}$ is the energy necessary to compute the maximum consensus and is equal to:

$$\mathcal{E}_{max} = \sum_{t=t_1}^{t_n} \left(\mathcal{E}_{hmac} + \mathcal{E}_{send}^{Prover_i} + \left(\mathcal{E}_{hmac} + \mathcal{E}_{receive}^{Prover_i}\right) \times |N_{i,t} \cap R_t|\right).$$

6.2 Simulations

We compute the number of fragments transmitted in each consensus step according to the maximum packet size of the standard IEEE 802.1.54 protocol. We consider different possible protocol parameter configurations, shown in Table 1. According to the network size, a target percentage of compromised devices, and a specific probability of false positives, i.e. the probability that a given healthy device the Verifier considers to be compromised by checking its relative bits in the Bloom filter, we derive the optimal Bloom filter bitlength m and the number of Hashes k [20]. We also derive the number of fragments necessary to broadcast the consensus status.

As we can see in the table, reducing the value of false-positive probability or increasing the number of compromised devices to be inserted, the number of bits of the Bloom filter increases.

Table 1: Parameter configuration.

Network size	128	256	512	1024	2048
Number of compromised devices n_c	7	13	26	52	103
Number of bits required m	68	125	250	499	988
Number of hash functions k	7	7	7	7	7
Fragments required	1	1	1	2	2

(a) 5% of compromised devices with 1% probability of false positives

Network size	128	256	512	1024	2048
Number of compromised devices n_c	7	13	26	52	103
Number of bits required m	29	70	139	309	611
Number of hash functions k	3	4	4	4	4
Fragments required	1	1	1	1	2

(b) 5% of compromised devices with 5% probability of false positives

Network size	128	256	512	1024	2048
Number of compromised devices n_c	13	26	52	103	205
Number of bits required m	125	250	499	988	1965
Number of hash functions k	7	7	7	7	7
Fragments required	1	1	2	2	3

(c) 10% of compromised devices with 1% probability of false positives

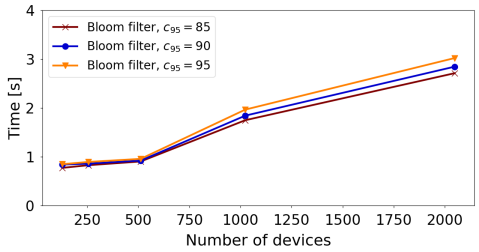
Network size	128	256	512	1024	2048
Number of compromised devices n_c	13	26	52	103	205
Number of bits required m	70	139	309	611	1246
Number of hash functions k	4	4	4	4	4
Fragments required	1	1	1	2	2

(d) 10% of compromised devices with 5% probability of false positives

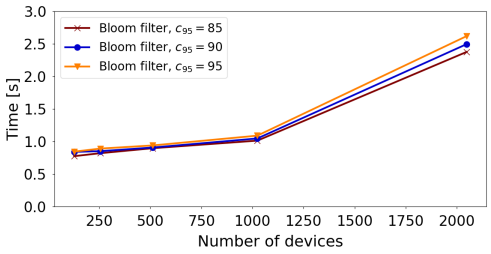
6.3 Runtime complexity

This section focuses on the results of some of the simulations described before. We compare the runtimes of our proposal with the different parameter configurations identified in Section 6.2.

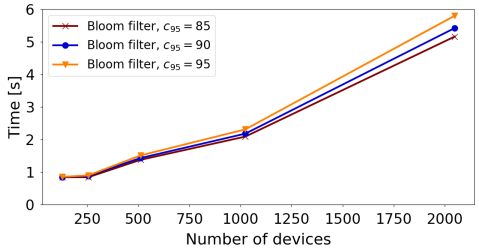
As we can see in Figure 4, there are not significant differences between 85% ($C_{95} = 85$), 90% ($C_{95} = 90$), and 95% ($C_{95} = 95$) of coverage for 95% of network devices, which are more perceptible as the number of devices increases. The performance principally changes with the number of packets required for each message. Figures 4a and 4c show a similar behavior up to 256 and 1024 devices respectively, in which they require the same number of packets to send their messages. Performances change in tests with 512 and 2048 devices in the network. In fact, to tolerate 5% of compromised devices, we require one packet for 512 devices and two packets for 2048, while, to tolerate 10% of compromised devices, we need two packets for 512 and three packets for 2048, increasing the runtime required for Figure 4c, which takes about 500 ms more for 512 devices and about three seconds more for 2048. Figure 4b and Figure 4d present a similar behaviour, where the only difference among them is when we simulate 1024 devices, which displays a significant performance gap. At the same time, the 2048 devices setting is of particular interest. The simulations have the same number of packets required but a significant difference in message size (about 70 bytes). In this case, the graphs highlight a runtime difference between them of about 200 ms. On the other side, Figure 4b and Figure 4d highlight lower performance differences in message whose size is within the same number of packets. In this case, we have



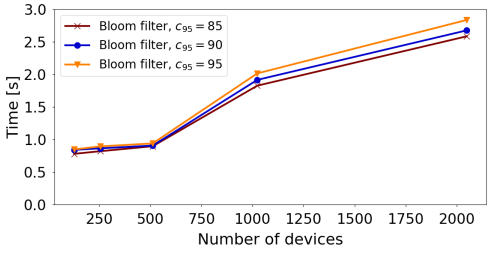
(a) 5% of compromised devices with 1% probability of false positives



(b) 5% of compromised devices with 5% probability of false positives



(c) 10% of compromised devices with 1% probability of false positives

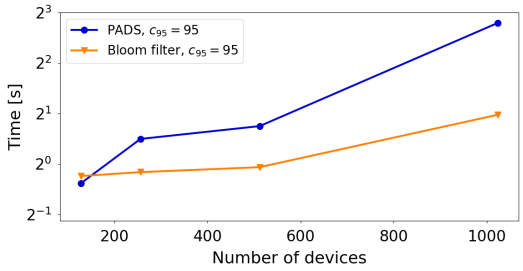


(d) 10% of compromised devices with 5% probability of false positives

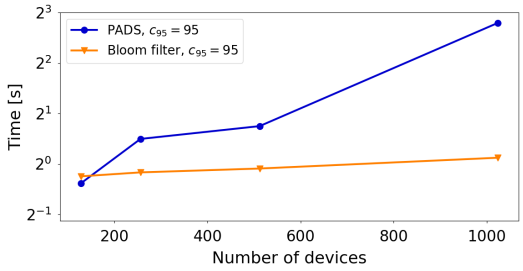
Figure 4: Average MCT of the protocol in function of the size of the network, percentage of compromised devices, probability of false positives, and coverage.

only variations in terms of milliseconds since the IEEE 802.15.4 data rate (up to 250 Kbps) reduces the message size gap.

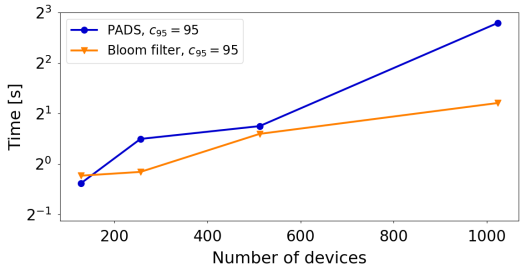
Comparison with the state of the art. We compare our protocol performance with PADS, with respect to 95% of the population that reaches 95% of coverage ($c_{95} = 95$). We provide the comparisons in logarithmic scale in Figure 5. We underline that, due to the



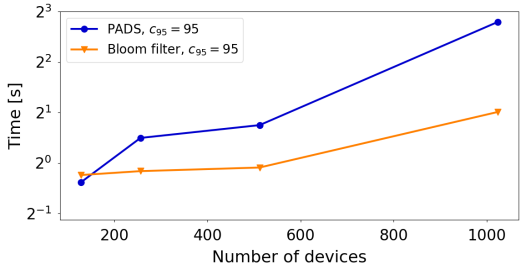
(a) PADS compared to 5% of compromised devices with 1% probability of false positives



(b) PADS compared to 5% of compromised devices with 5% probability of false positives



(c) PADS compared to 10% of compromised devices with 1% probability of false positives



(d) PADS compared to 10% of compromised devices with 5% probability of false positives

Figure 5: Protocol comparison with PADS.

high computational power requirements, our PADS implementation reaches up to 1024 devices. So we compare our protocol from 128 to 1024 devices.

Our results show that our approach outperforms PADS in most of the setting defined. We can observe that PADS is preferable in small swarms, i.e., 128 devices, because the Bloom filter requires more hashing operations to store the internal state. There are also situations where the performances are similar, such as in Figure **??**. In this condition, both our proposal and PADS have similar runtime in 512 devices simulation, since the message of our protocol, of 499 bits, requires two packets in order to be transmitted.

7 CONCLUSIONS

We have presented a new flexible, secure, and efficient protocol for collective remote attestation in highly dynamic low-power devices networks. This protocol leverages on the Bloom filter data structure in order to represent the state of the network, transforming the collective attestation into a maximum consensus problem. It permits to improve the performance and the flexibility respect to recent works in literature based on dynamic swarm scenario, allowing a dynamic change of network devices in the attestation and reducing the message size for communication between devices. The protocol simulation shows that it is possible to attestate thousands of devices in few seconds.

While our approach gives advantages concerning the runtime execution and dynamic presence of devices, there are significant limitations to be considered. One of the main problem, shared with the most of CRA protocol, is the Time of Check to Time of Use attack [21], since the protocol cannot detect devices compromised during the consensus phase. Another significant limitation is represented by the weakness for a physical attacker. In fact, our proposal considers a software-only attacker, and a single physically compromised device is sufficient to reveal the secret key shared in the network, enabling the attacker to inject easily fake attestation reports. However, due to the usage of the maximum consensus approach, it is not feasible to change bit values of Bloom filter, if previously propagated by healthy provers in the network. Moreover, like PADS, our proposal is more resilient to physical attacks then previous works. In fact, information is not processed along a tree, and this limits the propagation of malicious information. A single physically compromised device could change the attestation report of a whole subtree of devices. Instead, in this protocol the attacker might physically compromise all the network. A possibility to reduce the adversary forgery is the public-key cryptography implementation, which remove the possibility of obtaining the secret key of the whole network. However, such implementation increases the computational costs of each device. Another limitation with respect to PADS [4] is represented by the fact that, due to its probabilistic concept, the Bloom filter data structure does not permit to certainly have the specific node status, especially if in order to reduce the size of the Bloom filter, we tolerate a high percentage of collisions. In this case, we cannot even consider a possible device search by the verifier. In order to do that, we have to reduce drastically such probability, which proportionally increases the size of the Bloom filter, reducing the usefulness of the protocol.

7.1 Future works

Our protocol can be improved in several directions. First of all, the protocol assumes the usage of symmetric key cryptography, which limits its resilience against physical attackers. CRA protocols would benefit from new mechanism that permits attestations with different communication keys without losing performance. There is also the need to understand how many devices participate in

the CRA protocol. In fact, from the Bloom filter we can derive the number of compromised protocols, but neither the number of inactive devices (healthy or compromised), nor the total number of devices in the network is known.

Our protocol could be also modified in order to identify the location of compromised devices. A possible choice is to create a protocol where this approach is used with Bloom filters for location. In this way, hashes are not evaluated on devices identifier, but on areas identifiers, so that the verifier can locate the areas where compromised devices are. Assuming that in a small area we have a limited number of devices, their attestation is so fast that the network can be considered to be static. Thus, the verifier can run a different attestation protocol which permits to exactly identify the compromised devices.

The latter idea is the first step toward the development of a CRA framework that can auto-configure and choose which protocol use according to the state of the network. The goal of such a framework is that it configures parameters (i.e., the Bloom filter size) according to the number of devices in the network, or also switches to a different CRA protocol, for example from this proposal to PADS, when the number of compromised devices is over a given threshold and there is the necessity to identify them, or to a spanning-tree based CRA protocol, when verifying only a small sub-network.

ACKNOWLEDGMENTS

This work has been partially supported by La Sapienza University of Rome within the Bando Ricerca 2017, project *Security and Privacy of Biometrics for Mobile Authentication (SPoB-MA)* – Protocol n. RM11715C7878B045 and within the Bando Ricerca 2020, project *Secure ANd PrivatE Information sharing (SANPEI)* – Protocol n. RM120172B4CC529E. We are also grateful to Dr. M. Ambrosin and Dr. Md M. Rabbani for the useful talk we had.

REFERENCES

[1] 2011. MiXiM framework for Omnet++. http://mixim.sourceforge.net/

[2] M. Ambrosin, M. Conti, A. Ibrahim, G. Neven, A.-R. Sadeghi, and M. Schunter. 2016. SANA: secure and scalable aggregate network attestation. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. 731–742.

[3] M. Ambrosin, M. Conti, R. Lazzeretti, M. Rabbani, and S. Ranise. 2020. Collective Remote Attestation at the Internet of Things Scale: State-of-the-art and Future Challenges. *IEEE Communications Surveys & Tutorials* 22, 4 (2020), 2447–2461.

[4] M. Ambrosin, M. Conti, R. Lazzeretti, Md M. Rabbani, and S. Ranise. 2018. PADS: practical attestation for highly dynamic swarm topologies. In *International Workshop on Secure Internet of Things (SIoT)*. IEEE, 18–27.

[5] N. Asokan, F. Brasser, A. Ibrahim, A. Sadeghi, M. Schunter, G. Tsudik, and C. Wachsmann. 2015. SEDA: Scalable embedded device attestation. In *ACM SIGSAC Conference on Computer and Communications Security*. 964–975.

[6] B.H. Bloom. 1970. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM* 13, 7 (1970), 422–426.

[7] S. Dharmapurikar, P. Krishnamurthy, T.S. Sproull, and J.W. Lockwood. 2004. Deep packet inspection using parallel bloom filters. *IEEE Micro* 24, 1 (2004), 52–61. https://doi.org/10.1109/MM.2004.1268997

[8] A.G. Dimakis, S. Kar, J.M.F. Moura, M.G. Rabbat, and A. Scaglione. 2010. Gossip algorithms for distributed signal processing. *Proceedings of the IEEE* 98, 11 (2010), 1847–1864.

[9] A. Ibrahim, A. Sadeghi, and S. Zeitouni. 2017. SeED: secure non-interactive attestation for embedded devices. In *ACM Conference on Security and Privacy in Wireless and Mobile Networks*. 64–74.

[10] A. Kirsch and M. Mitzenmacher. 2008. Less hashing, same performance: Building a better Bloom filter. *Random Structures & Algorithms* 33, 2 (2008), 187–218.

[11] P. Koeberl, S. Schulz, A. Sadeghi, and V. Varadharajan. 2014. TrustLite: A security architecture for tiny embedded devices. In *European Conference on Computer Systems*. 1–14.

[12] F. Kohnhäuser, N. Büscher, S. Gabmeyer, and S. Katzenbeisser. 2017. Scapi: a scalable attestation protocol to detect software and physical attacks. In *Proceedings of the 10th ACM Conference on Security and Privacy in Wireless and Mobile Networks*. 75–86.

[13] F. Kohnhäuser, N. Büscher, and S. Katzenbeisser. 2018. SALAD: Secure and lightweight attestation of highly dynamic and disruptive networks. In *Asia Conference on Computer and Communications Security*. 329–342.

[14] M.b. Mohamad Noor and W.H. Hassan. 2019. Current research on Internet of Things (IoT) security: A survey. *Computer Networks* 148 (2019), 283–294. https://doi.org/10.1016/j.comnet.2018.11.025

[15] N. Neshenko, E. Bou-Harb, J. Crichigno, G. Kaddoum, and N. Ghani. 2019. Demystifying IoT security: an exhaustive survey on IoT vulnerabilities and a first empirical look on internet-scale IoT exploitations. *IEEE Communications Surveys & Tutorials* 21, 3 (2019), 2702–2733.

[16] R. Olfati-Saber and R.M. Murray. 2004. Consensus problems in networks of agents with switching topology and time-delays. *IEEE Transactions on automatic control* 49, 9 (2004), 1520–1533.

[17] R. Olfati-Saber and J.S. Shamma. 2005. Consensus filters for sensor networks and distributed sensor fusion. In *IEEE Conference on Decision and Control*. IEEE, 6698–6703.

[18] R. Patgiri, S. Nayak, and S.K. Borgohain. 2018. Preventing ddos using bloom filter: A survey. *arXiv preprint arXiv:1810.06689* (2018).

[19] M. Stanislav and T. Beardsley. 2015. Hacking IoT: A case study on baby monitor exposures and vulnerabilities. *Rapid7 Report* (2015).

[20] D. Starobinski, A. Trachtenberg, and S. Agarwal. 2003. Efficient PDA synchronization. *IEEE Transactions on Mobile Computing* 2, 1 (2003), 40–51.

[21] R.V. Steiner and E. Lupu. 2016. Attestation in wireless sensor networks: A survey. *ACM Computing Surveys (CSUR)* 49, 3 (2016), 1–31.

[22] S.J. Swamidass and P. Baldi. 2007. Mathematical correction for fingerprint similarity measures to improve chemical retrieval. *Journal of chemical information and modeling* 47, 3 (2007), 952–964.

[23] H. Wu, H.C. Hsiao, and Y.C. Hu. 2014. Efficient large flow detection over arbitrary windows: An algorithm exact outside an ambiguity region. In *Proceedings of the 2014 Conference on Internet Measurement Conference*. 209–222.

[24] V. Yadav and M.V. Salapaka. 2007. Distributed protocol for determining when averaging consensus is reached. In *Allerton Conference on communication, control, and computing*. 715–720.

[25] F. Yamaguchi and H. Nishi. 2013. Hardware-based hash functions for network applications. In *IEEE International Conference on Networks (ICON)*. IEEE, 1–6.

[26] Z.-K. Zhang, M.C.Y. Cho, C.W. Wang, C.W. Hsu, C.K. Chen, and S. Shieh. 2014. IoT security: ongoing challenges and research opportunities. In *2014 IEEE 7th international conference on service-oriented computing and applications*. IEEE, 230–234.

[27] J. Zheng and M.J. Lee. 2006. A comprehensive performance study of IEEE 802.15.4. *Sensor network operations* 4 (2006), 218–237.