SeCaS: Secure Capability Sharing Framework for IoT Devices in a Structured P2P Network

Angeliki Aktypi
Department of Computer Science
University of Oxford
angeliki.aktypi@cs.ox.ac.uk

Kubra Kalkan
Department of Computer Science
Ozyegin University
kubra.kalkan@ozyegin.edu.tr

Kasper B. Rasmussen
Department of Computer Science
University of Oxford
kasper.rasmussen@cs.ox.ac.uk

ABSTRACT

The emergence of the internet of Things (IoT) has resulted in the possession of a continuously increasing number of highly heterogeneous connected devices by the same owner. To make full use of the potential of a personal IoT network, there must be secure and effective cooperation between them. While application platforms (e.g., Samsung SmartThings) and interoperable protocols (e.g., MQTT) exist already, the reliance on a central hub to coordinate communication introduces a single-point of failure, provokes bottleneck problems and raises privacy concerns. In this paper we propose SeCaS, a Secure Capability Sharing framework, built on top of a peer-to-peer (P2P) architecture. SeCaS addresses the problems of fault tolerance, scalability and security in resource discovery and sharing for IoT infrastructures using a structured P2P network, in order to take advantage of the self-organised and decentralised communication it provides. SeCaS brings three main contributions: (i) a capability representation that allows each device to specify what services they offer, and can be used as a common language to search for, and exchange, capabilities, resulting in flexible service discovery that can leverage the properties on a distributed hash table (DHT); (ii) a set of four protocols that provides identification of the different devices that exist in the network and authenticity of the messages that are exchanged among them; and (iii) a thorough security and complexity analysis of the proposed scheme that shows SeCaS to be both secure and scalable.

CCS CONCEPTS

• **Security and privacy → Security protocols**;

KEYWORDS

IoT; DHT; Resource Sharing; Scalability; Fault-Tolerance; Privacy.

ACM Reference Format:
Angeliki Aktypi, Kubra Kalkan, and Kasper B. Rasmussen. 2020. SeCaS: Secure Capability Sharing Framework for IoT Devices in a Structured P2P Network. In *Proceedings of the Tenth ACM Conference on Data and Application Security and Privacy (CODASPY '20), March 16–18, 2020, New Orleans, LA, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3374664.3375739

CODASPY '20, March 16–18, 2020, New Orleans, LA, USA

ACM ISBN 978-1-4503-7107-0/20/03…$15.00
https://doi.org/10.1145/3374664.3375739

1 INTRODUCTION

The achievements of new technological advances and the appealing services they provide result in a proliferation of Internet of Things devices in our life. These smart embedded systems are increasingly ubiquitous and notoriously heterogeneous allowing seamless integration between the physical and the cyber world. The possession of a large variety of such smart appliances by the same owner forms the owner's IoT ecosystem, a personal network where each connected device provides specific services for the user. This ecosystem can enable the available smart devices to unite their strengths and overcome individual weaknesses (e.g., the lack of storage in one device) or provide more advanced functionalities (e.g., smart cameras to add with elderly care). To realise such potential and permit the owners to make the most out of the infrastructure they possess, a secure discovery and access control mechanism that allows device collaboration must be in place [19].

Typically, the mechanisms, which allow devices to collaborate, follow a centralised operation, where a central managerial entity (e.g., a search engine [15], a broker [8], a central hub [6]) controls the information flow and orchestrates the interactions of devices, mediating among their communication [43]. Regardless of their well-established application, the dependency of these centralised mechanisms on the availability of the connection link between the central administrative point and each device makes them vulnerable to single-point failures as when the connection link is down, collaboration among devices cannot be achieved. From a privacy perspective, relying on an external infrastructure that aggregates and controls crucial personal information on users raises significant threats [33]; if the central server is breached the whole system is compromised, whereas users do not control the way their data are handled or by whom they are processed [40]. Also, the fact that the same entity handles all the requests can result in network bottlenecks making this solution not inherently scalable, an essential property of IoT infrastructures that have to accommodate a significant and continuously increasing number of connected devices.

These well-known issues [1, 4, 17] motivate the use of decentralised communication models to be used to allow for all of these endowed components to interact with each other by discovering the resources that are available in the network. In this paper, we study the use of structured peer-to-peer (P2P) overlays in such a use-case. In structured P2P overlays, nodes communicate with each other in a decentralised manner without being obliged to communicate with a hub or a cloud backend, making the system more reliable and robust. The load is uniformly distributed among the network devices themselves; thus all the data are handled locally mitigating personal information leaks. The distributed hash table (DHT)

routing scheme that is followed by the peers provides scalability as queries can be resolved with logarithmic complexity in the size of the network, without creating bandwidth bottlenecks or requiring the nodes to be within distance to directly communicate with each other.

However, employing structured P2P networks to enable IoT device collaboration is not a trivial task [35, 36]. First of all, the exact match query that DHT performs necessitates devices to initiate several communication sessions that are proportional to the resources they search, even if they can be closely related (e.g., they can refer to the possibility to occupy storage but of different space amount). The design of an appropriate expression format that is both (i) compatible with the exact query discovery that DHT performs and (ii) provides discovery flexibility by allowing nodes to search for the capability they want with as little communication cost as possible, is needed.

Enabling device collaboration in a structured P2P manner is also a challenging task from a security point of view. Firstly, the structured P2P networks do not have an administrative entity that can invigilate peers' communication. Also, IoT topologies are highly dynamic due to the mobility of devices and their constrained resources that can lead them to fall in sleep-mode regularly. However, to accomplish such a task it is fundamental to guarantee that only authorised nodes can access the network, that only proper members of the network can access the capabilities of other nodes and that any malicious activity or configuration error can be detected. The heterogeneity of IoT devices concerning their resources further creates the need for the proposed techniques to apply to both powerful and lightweight devices in order to provide scalability.

Our contributions are three-fold:

- We introduce a flexible way of representing device services that we call capabilities, which enables collaboration among peers that are organised in a structured P2P network.
- We design four protocols that achieve an authenticated bootstrapping, an authorised access to the capabilities of nodes and render peers accountable for the messages they exchange; thus allowing them to collaborate securely by discovering and exchanging services.
- We provide a complete security analysis with respect to our threat model for the four protocols and we analyse the complexity of our proposal based on the computational, memory and communication overhead that is introduced to the peers. We prove that our framework can achieve both secure collaboration and good performance for large deployed connected environments.

2 BACKGROUND AND RELATED WORK

In this section, we provide background on DHT schemes, the routing mechanism of structured P2P networks [24], on top of which we construct our proposed communication framework. SeCaS is not tied with any particular structured P2P overlay. For concreteness reasons and without loss of generality we use Chord [31] to indicate examples and underline the performance analysis of our proposal. We briefly elaborate on its design here. Also, in this section we position our paper versus existing work on P2P overlays and other

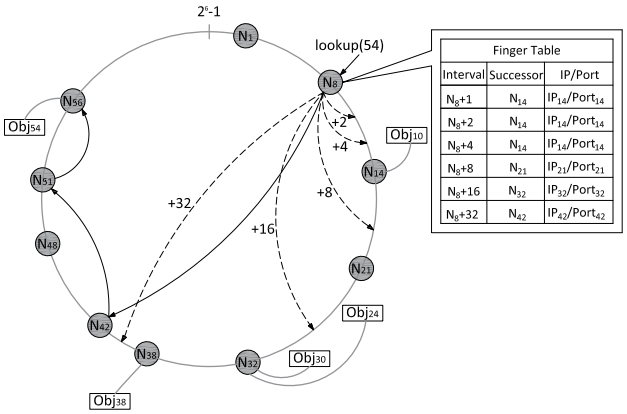

Figure 1: The organisation in Chord, an identifier ring modulo 2^6 consisting of ten nodes that stores five keys. Dashed lines show how the finger table is generated for $Node_8$, whereas solid lines show the path of a lookup query for $Object_{54}$ starting at $Node_8$.

already proposed platforms that enable the discovery and sharing of resources provided by different devices that coexist in the same network.

P2P networks allow for decentralised communication, by enabling nodes to act both as clients and servers. In structured P2P networks, the nodes and the application-specific data are assigned unique identifiers (i.e., $Node_{ID}$ and $Object_{ID}$, respectively) that place them on a large ID space structure. Figure 1, shows how Chord organises the identifiers on a circle modulo 2^m. The routing of messages is based on these unique identifiers. All nodes maintain a routing table (indicated as finger table in Chord) consisting of the $Node_{ID}$ and the communication address (e.g., IP) of several other nodes. Nodes route messages after advising their routing table by following a forward mechanism that leads progressively closer to the identifier that is each time specified. DHT provides the same functionality as a traditional hash table, by storing the correlation between an object and a value, which is location information concerning the nodes on which the data can be found. A defined function maps each object to a different node called the object's responsible node. In Chord, as a responsible node is chosen the first node whose $Node_{ID}$ is equal to or follows the data's identifier in the identifier space.

Our proposal is general and agnostic to the DHT used, which means that any structured P2P network (e.g., CAN [28], Chord [31], Pastry [29], Kademlia [25]) can be employed. SeCaS relies only on a simple DHT abstraction (API) [14, 21] that can provide three primary methods:

- **Join**($\mathbf{ID_{NC}}$, $\mathbf{Node_{BS}}$): This method provides the possibility a new node NC to join the network with the help of a bootstrapping node BS. BS, which already takes part in the overlay, assigns a $Node_{ID}$ to NC, which is used as its identifier in the DHT. BS creates this $Node_{ID}$ according to an identifying

property (ID) of the newcomer (e.g., its IP address). Consequently, with the help of BS and other peers in the network, the NC initialises its routing table, $Table_{ID}$.

- **Store**($\mathbf{Object_{ID}}$, $\mathbf{Node_{ID}}$): This method associates the unique identifier of a peer, $Node_{ID}$ with a specific Object, $Object_{ID}$ in the DHT, and stores it at the responsible node for this specified Object.
- **Lookup**($\mathbf{Object_{ID}}$): This method returns when available the location information (i.e., a list with all the nodes' identifiers, $\{List_{ID}\}$) that is associated with the indicated $Object_{ID}$, which is stored in its responsible node; or an empty list.

A large variety of different applications adopt the distributed operation that P2P overlays provide such as file sharing (e.g., Napster, Freenet), instant messaging (e.g., Skype) and multimedia streaming (e.g., Peertube) [9]. However, each one demands a different configuration as it is bound to the discovery of a specific resource, related to the context of each use case (e.g., a file, a username). The lack of flexibility in the resources that are discovered within each P2P network and the absence of interconnection mechanisms that can enable different P2P deployments to interact between each other prevents the broader and faster adoption of P2P systems in IoT environments. These ecosystems demand interoperable discovery mechanisms to be in place that can support the heterogeneous services, which are provided by the different connected devices.

SeCaS enables a holistic discovery of all the available resources that are registered and are to be shared in an IoT environment. We achieve that by introducing capabilities, a way to represent all the different services that nodes can contribute to the network. In the literature, there have been proposed ways of denoting the competence of devices in IoT ecosystems. In general, semantic ontologies have been introduced to solve interoperability and heterogeneity challenges [7, 34, 37]. Despite the effectiveness of semantic models on providing support for resource and service discovery in the IoT environments, the multiple annotations they use to describe specific services do not fit well with DHT exact terms search. Our hierarchical service representation can associate to each service a unique identifier by hashing. In this way, we enable the use of DHT routing in the context of our application—achieving a cooperative operation of devices. In industry, companies follow a more programmable way to coordinate and control accessories in home automation platforms [6, 30], where devices are abstracted to their underlying commands and attributes. Even if all the above structures achieve a detailed representation of device capabilities, they are targeting services explicitly. Hence, they do not allow the possibility of searching the network in a more advanced way (i.e., starting from a more general to a more specific sought-after service and vice-versa); on the contrary, our proposal benefits from such retrieval flexibility during the search operations.

SeCaS, by using a DHT scheme, achieves scalability and avoids spatial constraints and network flooding, limitations that are encountered in other distributed resource discovery frameworks that use the broadcast communication channel [41] and social links [20] to advertise available services. Compared to other proposals that also rely on a structured P2P overlay to enable resource discovery in IoT environments [2, 12, 27], SeCaS achieves guaranteeing a secure and reliable collaboration among the devices relying just on the DHT structure and without either needing the peers to be organised in different neighbourhoods nor introducing extra organisation layers to the system architecture. To the best of our knowledge, our scheme with the capabilities representation together with the introduced protocols is the first to leverage the use of P2P networks to safely discover and exchange the services provided by a plethora of IoT devices without increasing the deployment cost of the system.

3 CAPABILITIES

The purpose of our framework is to enable devices to work collaboratively in a secure context, by exchanging services. In this section, we elaborate on capabilities, our hierarchical service representation that allows denoting all the different services that nodes can contribute to the network. We start by underlining the objectives that need to be achieved by the proposed structure and then we present our proposal and its advantages.

3.1 Prerequisites

P2P networks have been widely used in different applications that were primarily dedicated to file-sharing by distributing network bandwidth [13]. However, throughout the years they have also enabled the simultaneous exchange of other resources such as processing power [5] and disk storage space [22]. Nowadays, IoT ecosystem presents a significant heterogeneity regarding the abilities of the devices that constitute it. To enable their exchange a mechanism that provides an easy way to represent them must be in place. In particular, this representation must provide a coherent registration and management of the different things that each device can do. It needs to be comprehensive and human-readable and easily extended to adapt any newly introduced device benefits.

To be able to use the DHT and take advantage of the accurate discovery of objects that it provides, the representation mechanism must be able to generate objects by hashing. Furthermore, it needs to allow searching the network in an advanced way, for example, by executing queries that provide the possibility to locate similar objects that fall under the same group. An easy way to achieve that is by using range queries, where the value of the given attribute is between an upper and lower boundary. However, in the DHT, the object to be searched is specified by its unique identifier [3], allowing for only exact-match queries. The challenge that arises is to enable the check of a range by using one and not multiple exact-match queries, as this redundancy increases the communication cost for the nodes.

3.2 Representation mechanism

In our study, we define capabilities as all the different ways in which devices can contribute services in the network [32]. The services can emanate from any resource that the device has, a software (e.g., communication interfaces), a hardware (e.g., memory) or a transducer (e.g., sensor, actuator). For example, an "Amazon Echo" device can access the web through its WiFi interface for a device that only has Bluetooth connectivity, can save data for a peer that is running out of memory or it can produce an alarm sound for a sensor device that does not have a speaker.

Capabilities, our representation mechanism, are described in plain-text following a hierarchical model depicted in Backus-Naur

form [26] expression illustrated below. The representation of each capability is the name of a service with any number of refinements, separated by dots. The *Service* is used as the group label of each capability and refers to the functionality that is provided by a device acting as a general description of the capability. The *Refinements* are used in order to give more details concerning the different capabilities. If we regard the service as the general group within which a functionality is categorised, we can consider the refinements as the concrete identifiers of the subcategories that constitute the group. Refinements specification walks through the different subgroups of an initial category going from something more general to something more specific. Following that different capabilities have a different number of refinements. For example, `Communication.TCP.HTTPS.TLS12` capability corresponds to the existence of a specific communication interface and the refinement specifies the exact communication module that is provided.

$$Object = hash(Capability)$$

$$\text{where } \langle Capability\rangle ::= \langle Service\rangle \{\text{"."} \langle Refinement\rangle\}$$

The plain-text of the capabilities' representation is hashed by using a collision resistance function creating the *Objects*. For each capability a different number of objects can be generated depending on the specified Refinements that accompany the used Service. In particular, for a capability that has n refinements there will exist $n + 1$ associated objects inserted in the network. After the Objects have been obtained by hashing, a device can invoke the lookup operation by specifying any of the specific objects that is associated with the capability it is looking for.

$$Capability = Service.Ref_1 \dots Ref_n \Rightarrow Objects = \{Object_0, \dots, Object_n\},$$

where

$$\begin{aligned} Object_0 &= hash(Service),\\ Object_1 &= hash(Service.Ref_1),\\ &\vdots\\ Object_n &= hash(Service.Ref_1 \dots Ref_n). \end{aligned}$$

Our proposal provides a convenient way for manufacturers to define new capabilities by introducing new Services or extending the Refinement list for the already created ones. It can also preserve compatibility with the way that capabilities are represented in already implemented protocols for automation control in smart-home environments such as Samsung's SmartThings [30], Google's Weave [18] and Apple's HomeKit [6]. In particular, the different information that they specify for each capability in their description (e.g., commands, attributes, status), can be included as a different refinement in our proposal.

This representation mechanism introduces a lightweight way to generate Objects that incorporate all the necessary information that needs to be defined so that a specific capability can be located efficiently into a broader set; thus compatibility with the DHT is provided, and the execution of exact-match queries is possible.

The hierarchical representation offers a retrieval flexibility during the search operations. If the precise name of a capability is not known (i.e., compatibility issues stem from non-unified naming of capabilities by different vendors) or when the invoked lookup operation of a sought-after capability returns an empty list, nodes can choose to search for a more generalised capability taking advantage of the hierarchy's peel-off property. As a result, we can achieve a reduction of the misleading indicated unavailability cases when not the specified but similar objects exist in the network. For example, if a node searches for `Thermostat.TemperatureState` but this service is stored as `Thermostat.TemperatureMeasurement` by another peer, the requester can peel off the refinement and search only for `Thermostat`. Similarly, `MemoryProvision.Level2` provides a capability of offering storage to another device to save data and the refinement indicates the available space, depicted in different levels that correspond to different storage magnitudes. If the communicated node does not have `Level2` amount of free space, then it can define the level of free space that it has available, for example `Level1`. The above-mentioned described lookup search achieves performing general/range queries by using only one exact-match query; thus, there is no need to perform multiple exact-queries to examine a specific range that increases the bandwidth usage and the communication overhead for the nodes.

The creation of multiple hashed values proportional to the number of defined refinements for each capability renders more fault tolerance to the system. In particular, each capability has associated with it multiple objects. Hence, the disappearance from the network of a capability demands the failure of multiple nodes. Thus, the possibility of unintentional deficient lookup queries is restricted. The proposed representation in conjunction with the Fulfilment protocol presented in section 5.4 achieves the exchange of any capability that can be encountered in an IoT ecosystem, despite their heterogeneity. In particular, both capabilities that demand one communication session (e.g., the download of a software update) or more (e.g., the storage and retrieval of data from the memory of a device) can be satisfied. To carry and transmit all the necessary information that needs to be specified for each capability separately, we harness a *Data* structure. Each time this structure is decomposed into different components (e.g., commands, attributes, methods) and the associated semantics and values vary based on the defined capability and the kind of message (e.g., sending or responding) for which it is used.

4 SYSTEM AND ADVERSARY MODEL

In this section, we introduce the system model, on which we base our work, and we specify the adversarial model against which our framework is secure.

4.1 System Model

In our model, we assume a group of connected devices from lightweight to powerful ones, equipped with a set of different capabilities owned by the same person or organisation. Nodes can directly reach each other to discover and use services, which they have advertised on the system. Potential use case scenarios can be a user's home network or a factory, where several devices are deployed across the same network. We consider a dynamic topology where nodes can join and leave at any time. To join the network, nodes broadcast their presence to discover a bootstrapping node. After joining the overlay, peers follow a DHT routing scheme to communicate.

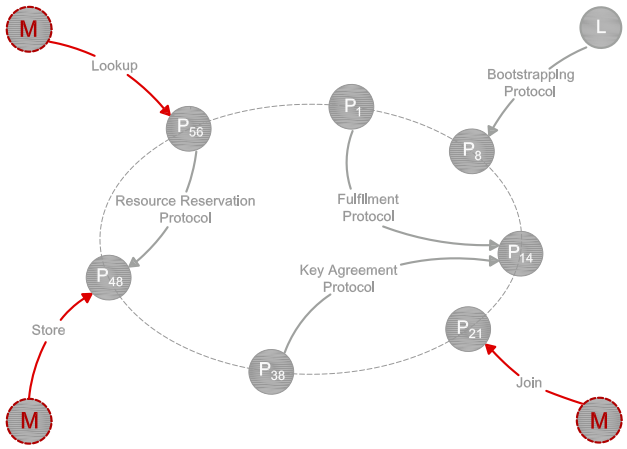

Figure 2: The system and adversary model. Nodes are organised in a structured P2P network, legitimate peers that are already a member in the network are depicted with the letter "P", whereas the legitimate newcomer is denoted with the letter "L"; malicious nodes are shown with the letter "M". Red arrows demonstrate the different attacks that we consider during the communications.

We leverage the decentralised communication of DHT to provide scalability and fault tolerance to the IoT infrastructure; peers can resolve their requests addressing them to other participating nodes rather than to a specialised directory (e.g., a centralised hub) or to all the registered ones (i.e., flooding). Both the underlying DHT's inherent load balancing for keys, by use of consistent hashing and our scheme protocols' random choice over the returned node list, improves fairness among nodes for the provision of services, which means that a single node will not be a repeated bottleneck. We allow for any structured P2P (e.g., CAN, Chord) to be used. Without loss of generality, we choose Chord in our system model depicted in Figure 2. Peers can specify the service they want to utilise by making use of the capabilities representation we specify in section 3. The communication protocols we define in section 5, which determine the way peers interact, guarantee specific security properties throughout their collaboration.

4.2 Threat Model

We assume a Dolev-Yao attacker [16] who fully controls the communication channel, i.e., can eavesdrop, manipulate, replay and intercept all communication and can also non-deterministically inject his messages into the network. For our analysis, we consider an active adversary who aims at manipulating the network without being detected; thus denial-of-service (DoS) and jamming attacks are out-of-scope. The goals of the attacker, depicted in red in Figure 2, are:

(1) to join the network and get the rights of an authenticated node
(2) to manipulate the DHT operations, i.e., initiating fake store operations or responding erroneously to lookup requests
(3) to break the confidentiality and the integrity of the communication among peers that collaborate

Our proposal provides a concrete secure solution that underlines and addresses the guarantees that have to be safeguarded in this new IoT ecosystem, assuming only on a simple DHT API and that an authentication mechanism is in place. We do not make any security assumptions about the DHT functions (i.e., join, store and lookup); as long as those functions are provided, the security properties of SeCaS remain unchanged regardless of the specific type of the underlying peer-to-peer network, which makes the framework more applicable in different scenarios. For our proposal, we made the design choice to adopt an authentication schema which assumes that the network owner possesses a public-private key pair with which she validates out-of-band the identities of legitimate peers. However, other authentication schemas [23] or off-the-shelf authentication protocols such as IPsec or TLS can also be employed. The protocols we specify provide strong accountability for the messages that nodes exchange. In the event of a compromised node, there will be a cryptographic proof of the node's malicious behaviour; thus, the node can be identified and removed from the system out-of-band.

5 FRAMEWORK PROTOCOLS DESCRIPTION

In this section, we present the four protocols that constitute our proposed communication framework. The protocols allow devices to execute an authenticated bootstrapping at any structured P2P network and to check the authenticity of the messages they received from the DHT. Peers that follow the protocols by using the capabilities representation we introduce in section 3 can collaborate in a secure manner by reserving and granting access to others peers capabilities. For all the protocols it is true that if any of the verification steps fails, the protocol terminates with an error.

5.1 Bootstrapping Protocol

One device with C number of capabilities, henceforth referred to as Alice, that has already obtained her certificate (i.e., she is a legitimate node) and wants to join the overlay, follows the Bootstrapping protocol, depicted in Figure 3. After picking a random nonce N_A, Alice initiates the communication by broadcasting her certificate, the freshly picked nonce N_A and her signature on the wireless channel, waiting to hear back responses from neighbouring nodes. Upon receiving the broadcast message, one of the devices already a member of the overlay, henceforth referred to as Bob, authenticates Alice by examining the received certificate and checking the integrity of the nonce based on the provided signature. Subsequently, Bob picks a nonce N_B and replies to Alice by providing his certificate, his node unique identifier in the DHT and the freshly picked nonce N_B that he signs together with the nonce that was indicated in Alice's initiated message. Alice authenticates Bob based on his certificate and she inspects the integrity and freshness of his response based on the provided signature and the included nonces. She responds back to Bob, by signing the nonce that he picked together with his node identifier. Bob after receiving Alice's signature knows that he really talks to Alice, thus, he is safeguarded from the unnecessary execution of the Join operation that is invoked immediately afterwards. Thereafter the two devices execute together the Join

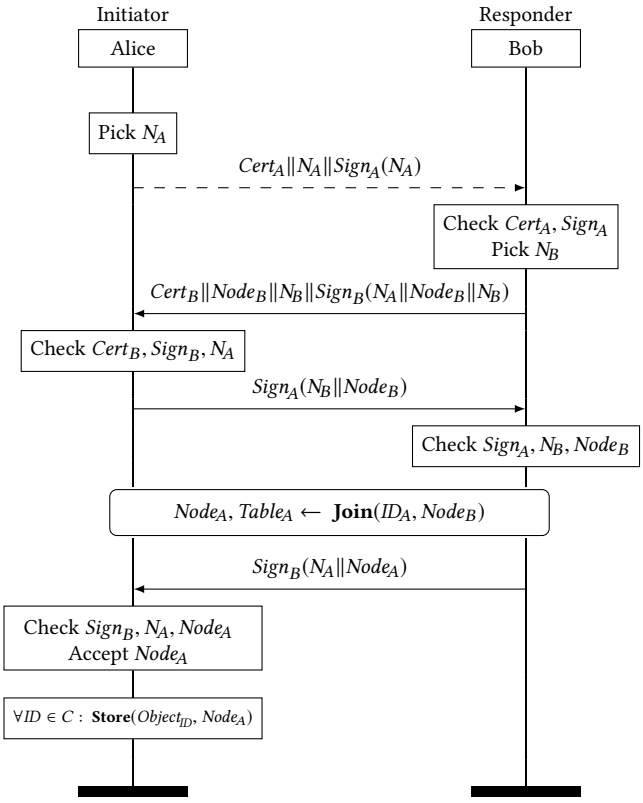

Figure 3: Bootstrapping Protocol. A node that wants to be registered in the network initiates a broadcast message waiting a response from a SeCaS peer. Nodes obtain their certificates out-of-band.

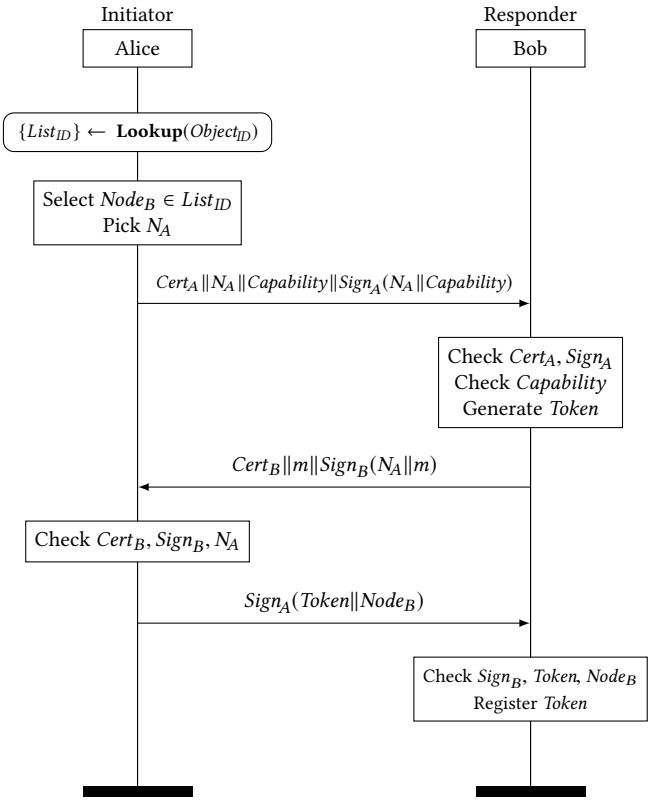

Figure 4: Resource Reservation Protocol. The initiator who needs a capability invokes the lookup operation from the DHT specifying its $Object_{ID}$. Then she selects arbitrarily one of the available peers from the return list and verifies if he has the sought-after service. If the service is available, she reserves it.

operation provided by the DHT, which returns the unique identifier that is assigned to Alice and her initialised routing table. Finally, Bob signs Alice's identifier together with the nonce that she chose. After receiving Bob's signature, Alice accepts the node identifier that she was assigned $Node_A$, as valid and she then invokes the *Store* operation in order to update the records of each responsible node of the C capabilities that she can contribute to the network. At the end of the protocol, a legitimate and alive node Alice has successfully joined the network and the network is both acquainted of her presence and of the capabilities that she can provide.

5.2 Resource Reservation Protocol

When a node, referred to as Alice searches for a capability in the overlay, she initiates the Resource Reservation protocol, illustrated as Figure 4. At first, Alice invokes the *Lookup* operation of the DHT specifying the unique identifier, $Object_{ID}$ of the capability that she is looking for. The invoked operation returns a list with all the peers, which have denoted that can provide the requested service. However, as the DHT provides no security, the obtained list is considered to be potentially tampered. After obtaining the list, Alice selects arbitrarily one peer, called Bob to which she sends a signed capability request message that contains her certificate, a freshly picked nonce, N_A and the queried capability, $Capability$. Bob checks the authenticity of the message by examining the received certificate and signature and inspects if he can provide the requested service. If this holds, he generates a $Token$ (i.e., a random number) that he associates with Alice's $Node_{ID}$ and the sought-after $Capability$. Subsequently, he replies to Alice's request by providing his certificate and a message m, that includes an acknowledgement and the freshly generated token, $m = Ack||Token$. He also signs message m together with the nonce, N_A that was chosen by Alice. When Alice receives Bob response, she checks its authenticity by inspecting the included certificate and the provided signature and assures that it corresponds to her initial request based on the returned nonce. She then signs the provided token together with Bob's node identifier and sends it to him. Finally, after checking the received signature Bob registers the token as issued. In case Bob cannot provide the requested service, he replies to Alice's request by providing his

certificate and a message m, that includes a negative acknowledgement, $m = Nack$. He also signs message m together with the nonce, N_A that was chosen by Alice, and the protocol terminates. If the sought-after capability is not currently available in the network (e.g., Bob was the only one who could provide it), Alice can take advantage of the peel-off property of the proposed capability representation to look for other capabilities that are related to the sought after one. At the end of this protocol, the requester node Alice reserves the capability from another peer Bob who has this capability in the network. The nodes that are looking for a specific capability after executing this protocol will know if such capability is really registered in the network and they will have also identify the node that is able and committed to providing it. Also, the contacted peer will be aware of the interest of another node for its specific service and will have reserved it for as long as the provided token is in effect (e.g., for a time interval T).

5.3 Key Agreement Protocol

Two peers jointly agree on a secret key following the Key Agreement protocol depicted as Figure 5. The protocol is based on the Diffie-Hellman algorithm. The predefined base g is a primitive root in the group of integers modulo p. Alice, who initiates the protocol, chooses a secret integer a and picks a random nonce N_A. She then sends to Bob her certificate, $g^a \bmod p$ and the freshly picked nonce N_A. In her message, she also includes her signature for the $g^a \bmod p$, N_A and Bob's identifier. Bob upon receiving the message checks the correspondence of the indicated certificate and the provided signature and also inspects the node identifier of the message's receiver assuring that this message is addressed to him. If the check is successful, Bob chooses, in turn, a secret integer b and constructs the shared secret key $K_{AB} = (g^a)^b \bmod p$. He sends back to Alice his certificate and $g^b \bmod p$ that he signs together with the nonce N_A that she picked at the beginning. As soon as Alice receives Bob's response, she checks the correspondence of the indicated certificate with the provided signature and the returned nonce, and she then calculates the shared secret key $K_{AB} = (g^b)^a \bmod p$. If the protocol terminates, it guarantees that the secret key is only known to Alice and Bob. The resulting secret will be used in subsequent communication between the devices, enabling them to authenticate each other and exchange capabilities over a confidential secret channel. The protocol takes into consideration the storage consumption at each node and the dynamic character of the network. By following it, nodes can establish keys with any of the peers registered in the system if they choose to do so; thus the number of keys that each node has to store in its memory is not in direct proportion to the network size.

5.4 Fulfilment Protocol

When a node wants to make use of a peer's commitment to the provision of a capability, it initiates the Fulfilment protocol delineated in Figure 6. Primarily, the initiator, Alice sends the possessed token that has been obtained from the execution of the Resource Reservation protocol, a data structure that forms her request and her node identifier. The message is sent encrypted together with its MAC, by using the secret key that is established from the completion of the Key Agreement protocol between Alice and the responder,

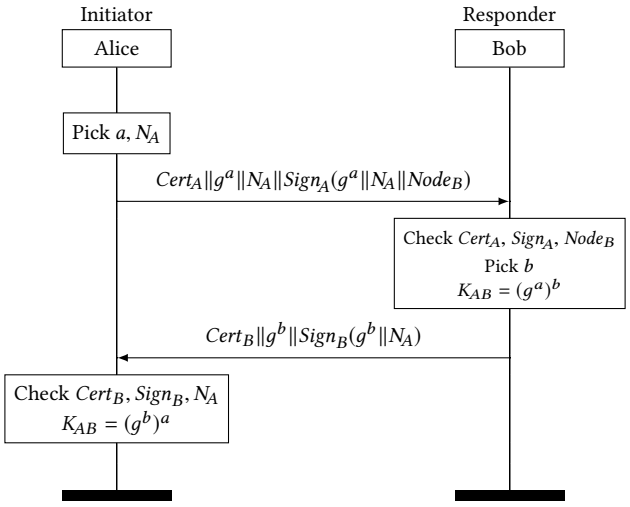

Figure 5: Key Agreement Protocol. Two peers that want to establish a shared secret key follow an authenticated Diffie-Hellman key algorithm.

Bob. The responder upon receiving the node's request retrieves their shared secret key, checks its integrity based on the provided MAC and then decrypts it. Subsequently, he ensures that she did not create the message in the past by inspecting the sender's included node identifier. Then, he examines if the specified token is still valid, if it is associated with the initiator's $Node_{ID}$ and if it is in accordance with the capability $Object_{ID}$ to which the defined data structure refers. If the checking is successful, he provides the specified capability. Hence, he responds back by encrypting the initially provided token, a data structure that contains his respond, his node identifier and a new generated token in case of a long lasting capability. This token will be used in a subsequent execution of the Fulfilment protocol to refer back to a capability that is still under sharing. Finally, the responder includes the MAC of his encrypted response that will then be used from the initiator for checking the integrity of the replied message. The protocol provides access control to the nodes' capabilities. The responder is able to control which device is going to be the beneficiary of the service provided with the help of the token. The initiator at the end of the protocol's execution will have taken advantage of the other peer's capability and will have an acknowledgement for the provided service. All the communication is encrypted providing confidentiality to the communication between the collaborative parties.

6 SECURITY ANALYSIS

Based on the threat model that we describe in section 4.2 we now evaluate the security of the protocols of our framework by discussing the likelihood of an adversary breaking the security guarantees of each protocol to achieve his attacking goals. We will base our analysis on the following four main assumptions:

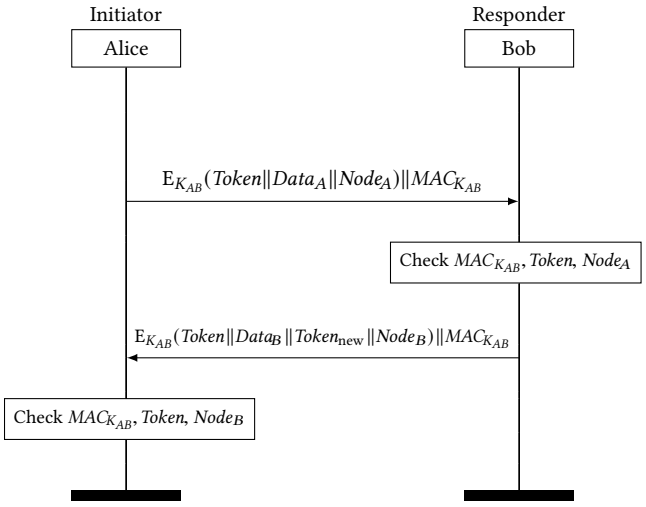

Figure 6: Fulfilment Protocol. The responder grants access to the initiator to one of his services based on the indicated token. The nodes are authenticated based on the secret key they share.

(1) The signature scheme used by the protocol participants and the CA is secure, i.e., it is not possible to forge a signature without access to the private key.
(2) The same nonce is only picked twice with negligible probability.
(3) The same token is only picked twice by the same device with negligible probability.
(4) Each device registers only one token for any given capability each time it is requested, along with the $Node_{ID}$ of the node to whom it was issued. The tokens are maintained for a time T (enough for nodes to execute the protocols in Figures 5 and 6). After this time the tokens are discarded.

6.1 Bootstrapping Protocol

The Bootstrapping protocol, Figure 3 provides two guarantees, one for Alice and one for Bob.

Guarantee 1. *If the protocol completes successfully, Alice is assigned a $Node_{ID}$ sent by Bob (a valid node), and she correctly joins the overlay network.*

Proof. For an adversary to break this guarantee and falsely convince Alice that she has been assigned a valid $Node_{ID}$, the adversary would have to successfully send Message 4, as this is the only way for Alice to get the $Node_{ID}$ that is signed by Bob. The adversary only has two options to send Message 4. He can either craft the message or replay a previously captured message. To craft Message 4 the adversary has to produce a signature over the message content that matches a certificate signed by the CA. This means he must either forge the signature or obtain the private keys. The adversary cannot forge the signature by assumption 1, so assuming the private keys are kept private, the adversary cannot craft the message. For replay to work, the adversary has to make sure that the content of Message 4 expected by Alice, corresponds to one of the messages available to her. Because the nonce N_A selected by Alice is part of the signature of Message 4, this means that the adversary either has to force Alice to choose a nonce that corresponds to one of his captured messages, or predict the choice and force Bob to construct a valid message ahead of time. The adversary cannot influence Alice's choice of N_A and by assumption 2 previous nonces (picked by Alice or other nodes) will only be useful with negligible probability. This means he would have to predict the choice and make Bob generate Message 4 ahead of time. However, Bob will only send Message 4 (containing N_A) in response to a signed response (Message 3), and we prove subsequently that the adversary cannot forge such responses. □

Guarantee 2. *For Bob, this protocol guarantees that the join request is fresh and originated from Alice (a valid node).*

Proof. For an adversary to break this guarantee and falsely convince Bob that he is Alice, the adversary would have to send Message 3 successfully. The adversary has two options: he can either create or replay Message 3. To create Message 3, the adversary has to produce a signature over the message content, which is not possible according to assumption 1. To replay Message 3, the adversary has to control Bob's choice of N_B, or predict it and force Alice to generate Message 3. However, these options are not viable for similar reasons to those explained above related to the replay of Message 4. □

6.2 Resource Reservation Protocol

The Resource Reservation protocol, Figure 4 has two guarantees, one for Alice and one for Bob.

Guarantee 3. *At the end of the protocol, Alice obtains a token which proves that she has reserved a resource from Bob (a valid node).*

Proof. For an adversary to break this guarantee and falsely convince Alice that she has received a valid token, the adversary has to send Message 2 successfully. The adversary only has two options to send Message 2. He can either create the message or replay a previously captured message. To create the message, the adversary has to produce a signature over the message, which is not possible according to assumption 1. To replay, the adversary has to control Alice's choice of N_A, or predict it and force Bob to generate Message 2, nevertheless for similar reasons to the ones explained in section 6.1 for the Bootstrapping Protocol, this cannot be achieved. □

Guarantee 4. *For Bob, the protocol guarantees that the request is fresh and really from Alice, i.e., that it has not been changed or submitted by someone else.*

Proof. For an adversary to make Bob register a token as "issued", he must confirm the token with Message 3 (in Figure 4). The adversary only has two options to send Message 3. He can either craft the message or replay a previously captured message. To craft Message 3 the adversary has to produce a signature over the message which is not possible according to assumption 1. For replay to work, the adversary has to control the choice of the token, or predict it and force Alice to generate Message 3. With the help

of assumption 3 and similar reasons of the replay of messages as above, these options are not viable. □

6.3 Key Agreement Protocol

The Key Agreement protocol, Figure 5 provides a guarantee that holds for both Alice and Bob.

Guarantee 5. *If the protocol is completed successfully, a secret key is created which is only known by Alice and Bob.*

Proof. For an adversary to break this guarantee he has two options: he can create the secret key by using the information obtained from Message 1 and Message 2, or he can control the choice of a or b. To obtain the key, he gets the information g^a from Message 1 and g^b from Message 2. To craft the key from this information he needs to solve the discrete logarithm problem which is computationally infeasible. Assuming the adversary cannot influence Alice's choice of a or Bob's choice of b the adversary cannot create the shared key between Alice and Bob. □

6.4 Fulfilment Protocol

The Fulfilment protocol, Figure 6 provides two guarantees, one for Alice and one for Bob.

Guarantee 6. *If the protocol completes, Alice will successfully access one of Bob's resources.*

Proof. In order for an adversary to break this guarantee, the adversary would have to successfully send Message 2 to take advantage of the provided capability. The adversary can either craft the message, or replay a previously captured message. To craft Message 2 the adversary has to encrypt the message content; thus he has to obtain the secret key K_{AB} that is shared between Alice and Bob. However, as we proved in the Key Agreement Protocol, the adversary cannot obtain the secret key, K_{AB}. To replay the message, the adversary needs to make the content of the message acceptable by Alice. According to assumption 3, the same token cannot be applicable twice for the same device. Thus, Alice will understand from the token number that it is a replay message. However, there can be a case that Alice has reserved in the past a capability for Bob with the same token and they have also executed the fulfilment protocol, the adversary will be able to capture the encryption of the provided token and perform a replay attack for Message 1 (reflection attack). However, $Node_B$ in the message will reveal that this message is not coming from Bob. □

Guarantee 7. *For Bob, the protocol guarantees that he shares with Alice the resource for which he earlier provided a token.*

Proof. Alice cannot cheat and take advantage of another resource other than what she reserved based on the assumption 4. For an adversary to trick Bob to share his capability with him will have to send Message 1 successfully. The adversary can either craft or replay the message. To craft Message 1 the adversary has to encrypt the message content; thus he has to obtain the secret key K_{AB} which we proved that it is not possible in the previous section 6.3. To replay the message, the adversary needs to make the content of the message acceptable by Alice. According to assumption 3, the token number will reveal that it is a replay message. However, there can be a similar case explained above, which can cause a reflection attack. However, $Node_A$ in Message 1 casts out this threat. □

7 COMPLEXITY ANALYSIS

In this section, we provide a complexity analysis of the SeCaS framework that runs on top of a structured P2P network. To avoid duplicating existing work that implements DHT [3, 12, 27] in IoT environments, we focus on the performance delta (i.e., computational overhead, storage consumption, communication cost) that nodes need to sustain when they participate in DHT and when they follow our proposal. In our analysis, we use as a benchmark the Chord [31] protocol, for concreteness and consistency reasons. However, the referred performance is followed by the majority of the structured P2P networks.

7.1 Computational Overhead

Recent studies have shown that it is feasible to apply public key cryptography to sensor networks by using the right selection of algorithms and associated parameters, optimization, and low power techniques [38]. Following that, we base our analysis on the number of demanding cryptographic operations (i.e., exponentiations, signature generations and verifications) that nodes need to perform.

Apart from the hashing operation, which does not impose a significant computational burden to the nodes, Chord and in general the DHT does not demand any other cryptographic operation. Hence, based on our assumption the computational overhead for the nodes in Chord is zero.

SeCaS framework provides the security guarantees that we present in section 6, by using different cryptographic operations. Starting with the Bootstrapping protocol, both Alice and Bob have to perform two signature generations over the messages they initiate. Also, both of them have to verify signatures three times to authenticate their communicating party based on their provided certificate or to check the authenticity of the received messages. Hence, the total computational cost of the Bootstrapping protocol is four signature generations and six signature verifications. Analogously, we compute the total introduced overhead for the Resource Reservation protocol and the Key Agreement protocol, which follows a Diffie-Hellman key exchange algorithm. The Fulfilment protocol demands only symmetric encryption-decryption and message authentication code (MAC) verifications; thus following our initial assumption, its computational cost is zero. Table 1 summarises our results.

7.2 Storage Consumption

To avoid a linear search of the network [42], nodes in Chord maintain a routing table (denoted as finger table) containing at maximum m entries, where m is the number of bits of the identifiers. A finger table entry includes the finger interval and both the Chord identifier and the communication address of the relevant node; thus for storing their finger table nodes are burdened to reserve $O(m)$ space in their memory.

In the DHT, nodes are responsible for storing application-specific data related to a number of different objects that are registered in

Table 1: Complexity overview of the computational overhead, the storage consumption and the communication cost that nodes need to sustain when following the Chord P2P protocol and the ones introduced on top of them by SeCaS Framework.

	Computational Overhead		Storage Consumption		Communication Cost	
Protocol	Alice	Bob	Chord	Introduced	Chord	Introduced
Bootstrapping	$2s + 3v$	$2s + 3v$	$O(m) + O(r)$	$4 \cdot k_a$	$O((\log n)^2) + O(\log n)$	3
Resource Reservation	$2s + 2v$	$1s + 3v$	–	$(g + m + t) \cdot n_t$	$O(\log n)$	n_t
Key Agreement	$1s + 2v + 2e$	$1s + 2v + 2e$	–	$(m + k_s) \cdot (n_t + n_s - (n_t \cap n_s))$	–	$n_t + n_s - (n_t \cap n_s)$
Fulfilment	0	0	–	$f(Data)$	–	2

the network. In Chord, assuming h registered objects in a network with n number of nodes, each node is responsible for $r = h/n$ objects. Each new entry causes the table and also the space that it occupies to increase linearly and in direct proportion to the number of inputs. Hence, the nodes are burdened to reserve $O(r)$ space in their memory for the data that they are responsible for.

The hierarchical representation that SeCaS introduces for representing capabilities associates multiple objects with each capability based on the number of refinements that it has. Despite introducing a higher load to the total network, the load balancing feature that is inherent in DHTs such as in Chord promotes the uniform distribution of the extra objects among all the nodes in the network. By having all the nodes contributing evenly in the maintenance of the network, we reduce the collateral damage of the failure of each node in the total network operation.

In addition to these two tables, the certificate-based authentication mechanism which we adopt for SeCaS schema here requires each participant node to store a few further properties: the public key of the owner (PK_{CA}), a pair of public-private keys (PKi, SKi) and its validated identity in the form of a certificate ($Cert_i = (Sign_{CA}(ID_i, PK_i), ID_i, PK_i)$). The amount of storage that these properties occupy on each node's memory depends on the selected cryptographic asymmetric algorithm that is used for generating key pairs and signing messages, and the selected key-length. For example, in RSA the signature size depends on the key size, the RSA signature size is equal to the length of the modulus in bytes. This means that for a k_a-bit key, the resulting signature will be exactly k_a-bit long (e.g., 2048-bit key length will result in a signature of 256 bytes). Taking the bit length of the chosen key k_a as constant, storing the four preliminary values requires $4 \cdot k_a$ bits of memory space on each node.

Further to these values, in the proposed framework each node needs to store two more tables. In the first table, each node has to maintain the tokens that provide authorised access to its capabilities. Each entry in this table has to indicate the string value of the capability, the unique identifier of the node that has reserved it and the related token. Assuming a g-bit string, an m-bit unique identifier for the DHT and a t-bit token in use, the amount of memory that this table requires is $(g + m + t) \cdot n_t$ bits, where n_t is the number of nodes that each node serves.

The second table, has to save the shared secret symmetric keys which the node establishes with other peers from the network. Each entry in this table will include the unique identifier of the cooperative node and the established symmetric key. The total amount of memory that this table demands depends on the bit length of the established symmetric key (e.g., 256-bit for AES algorithm [39]) and it grows linearly whenever a new key is established. Nodes share a secret key with all the nodes with which they collaborate. Considering the establishment of a k_s-bit symmetric key between the nodes, the storage of this table reserves $(m + k_s) \cdot (n_t + n_s - (n_t \cap n_s))$ bits on the memory of each node, where n_s is the number of nodes that serves this node.

Nodes are further burdened to use space in their memory depending on the *Data* structure of the capability that is exchanged each time. The Fulfilment protocol, Figure 6 enables capability exchange by making use of this structure, which carries and transmits all the necessary information (e.g., commands, attributes, methods) for each capability.

7.3 Communication Cost

To maintain a correct mapping in the DHT when a node n_i joins the network, the responsibility of the identifier space is going to be updated, so as the newcoming node to be assigned certain identifiers previously assigned to other peers. This initiates a message exchange procedure in the network to update the routing invariants of the affected nodes. In Chord, such a join procedure demands $O((\log n)^2)$ messages, in order to make sure certain identifiers previously assigned to n's successor to become assigned to n.

To locate both resources and peers across the network, nodes in the DHT use their routing table. The communication complexity achieves a balanced trade-off between the cost of maintaining a full directory and the network delay of storing one neighbour per node. The level of routing complexity in a stable n-node Chord overlay requires $O(\log n)$ hops. Hence, both the store and lookup operations demand $O(\log n)$ number of messages if recursive routing [10] is applied or $O(2 \log n)$ in case of iterative routing [11].

In SeCaS, both the Bootstrapping protocol (Figure 3) and the Fulfilment protocol (Figure 6) of our scheme, add a constant number of three and two messages, respectively, to the nodes that are communicating in the structured P2P network. The number of executions of the Resource Reservation protocol (Figure 4) and the Key Agreement protocol (Figure 5), as already explained in the previous subsection 7.2, depends on the number of nodes that the node serves and the number of the nodes that serve it; thus the introduced communication cost from these protocols will be of n_t and $n_t + n_s - (n_t \cap n_s)$ messages, respectively.

In SeCaS, nodes' liveliness is inspected by using cryptographic nonces and not other freshness mechanisms (e.g., timestamps); thus

no time synchronisation is demanded among the different devices in the network. The communication overhead that is generated in order the devices to acquire the necessary properties (e.g., their certificate) for being authenticated and is not a burden that our proposal imposes but is linked with the authentication schema that SeCaS uses; thus we do not include it in our calculations.

8 CONCLUSION

In this paper, we introduce SeCaS, a framework that enables efficient and secure capability sharing between IoT devices. Our framework takes advantage of the self-organised and decentralised communication provided by a structured P2P network and is built to utilise the search capabilities such networks already possess. SeCaS ensures a number of security properties including only letting authorised nodes join the network and search for capabilities; message freshness; authentication of nodes and messages; and accountability to aid in troubleshooting in case of configuration errors or malicious behaviours. The security properties hold regardless of the exact type of underlying P2P network, on the only condition that three functionalities – Join, *Store*, and *Lookup* – are provided. We do not make any security assumptions about these functions, nor does it matter how they are implemented. The adversary is considered to have full control over the data returned by these functions. Despite this strong adversary model, we provide a complete decentralised secure capability discovery and resource sharing network solution. This is accomplished by first introducing a flexible way to represent device services that we call capabilities. Capabilities act as a common language to allow nodes to exchange, and search for, services without the need for a central hub. We design four protocols that utilise the underlying P2P network as a transport layer and together implement the framework. These protocols guarantee that only authorised nodes can access the network and only proper members of the network can access the capabilities of other nodes. We provide a complete security analysis for all four protocols, with respect to our threat model. We additionally analyse the computational overhead, as well as the memory and communication complexity of our proposal showing that our framework scales well and does not impose a considerable cost to the participating nodes. Both our security and complexity analyses prove SeCaS to be a scalable architecture that provides fault-tolerance and addresses privacy concerns, suggesting it as a feasible alternative to the established IoT collaboration approaches that are either centralised or use flooding mechanisms.

ACKNOWLEDGMENTS

We would wish to thank the UK EPSRC and British Telecommunications who have funded this research through a partial PhD studentship in Cyber Security for EU Candidates and a Russel Studentship, respectively.

REFERENCES

[1] Omar Alrawi, Chaz Lever, Manos Antonakakis, and Fabian Monrose. 2019. Sok: Security evaluation of home-based iot deployments. In *IEEE S&P*. 208–226.
[2] Joao Alveirinho, Joao Paiva, Joao Leitao, and Luis Rodrigues. 2010. Flexible and efficient resource location in large-scale systems. In *Proceedings of the 4th International Workshop on Large Scale Distributed Systems and Middleware*. ACM, 55–60.
[3] João Pedro Fernandes Alveirinho. 2010. *Resource Location in P2P Systems*. Master's thesis. Instituto Superior Técnico.
[4] Mahmoud Ammar, Giovanni Russello, and Bruno Crispo. 2018. Internet of Things: A survey on the security of IoT frameworks. *Journal of Information Security and Applications* 38 (2018), 8–27.
[5] David P Anderson, Jeff Cobb, Eric Korpela, Matt Lebofsky, and Dan Werthimer. 2002. SETI@ home: an experiment in public-resource computing. *Commun. ACM* 45, 11 (2002), 56–61.
[6] Apple. 2018. *HomeKit*. https://developer.apple.com/documentation/homekit
[7] Garvita Bajaj, Rachit Agarwal, Pushpendra Singh, Nikolaos Georgantas, and Valérie Issarny. 2017. 4W1H in IoT semantics. *IEEE Access* (2017).
[8] Andrew Banks and Rahul Gupta. 2014. MQTT Version 3.1. 1. *OASIS standard* 29 (2014). http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.pdf
[9] John F Buford and Heather Yu. 2010. Peer-to-peer networking and applications: synopsis and research directions. In *Handbook of Peer-to-Peer Networking*. Springer, 3–45.
[10] Farida Chowdhury and Md. Sadek Ferdous. 2017. Performance analysis of R/Kademlia, Pastry and Bamboo using recursive routing in mobile networks. *International Journal of Computer Networks & Communications (IJCNC)* 9, 5 (2017).
[11] Farida Chowdhury, Jamie Furness, and Mario Kolberg. 2017. Performance analysis of structured peer-to-peer overlays for mobile networks. *International Journal of Parallel, Emergent and Distributed Systems* 32, 5 (2017), 522–548. https://doi.org/10.1080/17445760.2016.1203917 arXiv:https://doi.org/10.1080/17445760.2016.1203917
[12] Simone Cirani, Luca Davoli, Gianluigi Ferrari, Rémy Léone, Paolo Medagliani, Marco Picone, and Luca Veltri. 2014. A scalable and self-configuring architecture for service discovery in the internet of things. *IEEE Internet of Things Journal* 1, 5 (2014), 508–521.
[13] Bram Cohen. 2008. The BitTorrent protocol specification.
[14] Frank Dabek, Ben Zhao, Peter Druschel, John Kubiatowicz, and Ion Stoica. 2003. Towards a common API for structured peer-to-peer overlays. In *Peer-to-Peer Systems II – International Workshop on Peer-to-Peer Systems (IPTPS)*. Springer, 33–44.
[15] Soumya Kanti Datta, Rui Pedro Ferreira Da Costa, and Christian Bonnet. 2015. Resource discovery in Internet of Things: Current trends and future standardization aspects. In *World Forum on Internet of Things (WF-IoT)*. IEEE, 542–547.
[16] Danny Dolev and Andrew Yao. 1983. On the security of public key protocols. *IEEE Transactions on information theory* 29, 2 (1983), 198–208.
[17] Earlence Fernandes, Jaeyeon Jung, and Atul Prakash. 2016. Security analysis of emerging smart home applications. In *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE, 636–654.
[18] Google. 2018. *Weave*. https://nest.com/weave/
[19] Juan A Holgado-Terriza and Sandra Rodríguez-Valenzuela. 2011. Services composition model for home-automation peer-to-peer pervasive computing. In *Federated Conference on Computer Science and Information Systems (FedCSIS)*. IEEE, 529–536.
[20] Dimitris N Kalofonos, Zoe Antoniou, Franklin D Reynolds, Max Van-Kleek, Jacob Strauss, and Paul Wisner. 2008. Mynet: A platform for secure p2p personal and social networking services. In *International Conference on Pervasive Computing and Communications (PerCom)*. IEEE, 135–146.
[21] Brad Karp, Sylvia Ratnasamy, Sean Rhea, and Scott Shenker. 2004. Spurring adoption of DHTs with OpenHash, a public DHT service. In *International Workshop on Peer-to-Peer Systems*. Springer, 195–205.
[22] John Kubiatowicz, David Bindel, Yan Chen, Steven Czerwinski, Patrick Eaton, Dennis Geels, Ramakrishna Gummadi, Sean Rhea, Hakim Weatherspoon, Westley Weimer, et al. 2000. Oceanstore: An architecture for global-scale persistent storage. In *ACM SIGARCH Computer Architecture News*, Vol. 28. ACM, 190–201.
[23] Chaohao Li, Xiaoyu Ji, Xinyan Zhou, Juchuan Zhang, Jing Tian, Yanmiao Zhang, and Wenyuan Xu. 2018. HlcAuth: Key-free and Secure Communications via Home-Limited Channel. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*. ACM, 29–35.
[24] Eng Keong Lua, Jon Crowcroft, Marcelo Pias, Ravi Sharma, and Steven Lim. 2005. A survey and comparison of peer-to-peer overlay network schemes. *IEEE Communications Surveys & Tutorials* 7, 2 (2005), 72–93.
[25] Petar Maymounkov and David Mazieres. 2002. Kademlia: A peer-to-peer information system based on the xor metric. In *International Workshop on Peer-to-Peer Systems*. Springer, 53–65.
[26] Daniel D McCracken and Edwin D Reilly. 2003. Backus-naur form (bnf). *Encyclopedia of Computer Science* (2003), 129–131.
[27] Federica Paganelli and David Parlanti. 2012. A DHT-based discovery service for the Internet of Things. *Journal of Computer Networks and Communications* 2012 (2012).
[28] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. 2001. *A scalable content-addressable network*. Vol. 31. ACM.
[29] Antony Rowstron. 2001. Pastry: Scalable, distributed object location and routing for large-scale, persistent peer-to-peer storage utility. In *IFIP/ACM International Conference on Distributed Plarforms*.
[30] Samsung. 2018. *SmartThings*. https://smartthings.developer.samsung.com/develop/api-ref/capabilities.html
[31] Ion Stoica, Robert Morris, David Karger, M Frans Kaashoek, and Hari Balakrishnan. 2001. Chord: A scalable peer-to-peer lookup service for internet applications. *ACM SIGCOMM Computer Communication Review* 31, 4 (2001), 149–160.

[32] Jack Sturgess, Jason R C Nurse, and Jun Zhao. 2018. A capability-oriented approach to assessing privacy risk in smart home ecosystems. In *Living in the Internet of Things: Cybersecurity of the IoT Conference*. IET.

[33] Yuan Tian, Nan Zhang, Yueh-Hsun Lin, XiaoFeng Wang, Blase Ur, XianZheng Guo, and Patrick Tague. 2017. SmartAuth: User-Centered Authorization for the Internet of Things. In *USENIX Security Symposium*. USENIX Association, 361–378.

[34] Matú Tomlein and Kaj Grønbæk. 2016. Semantic Model of Variability and Capabilities of IoT Applications for Embedded Software Ecosystems. In *Working IEEE/IFIP Conference on Software Architecture (WICSA)*. IEEE, 247–252.

[35] Guido Urdaneta, Guillaume Pierre, and Maarten Van Steen. 2011. A survey of DHT security techniques. *ACM Computing Surveys (CSUR)* 43, 2 (2011), 8.

[36] Dan S Wallach. 2003. A survey of peer-to-peer security issues. In *Software Security—Theories and Systems*. Springer, 42–57.

[37] Wei Wang, Suparna De, Ralf Toenjes, Eike Reetz, and Klaus Moessner. 2012. A comprehensive ontology for knowledge representation in the internet of things. In *International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*. IEEE, 1793–1798.

[38] Yong Wang and Jason Nikolai. 2017. Key Management in CPSs. *Security and Privacy in Cyber-Physical Systems: Foundations, Principles and Applications* (2017), 117–136.

[39] Wikipedia contributors. 2018. Key size — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/w/index.php?title=Key_size&oldid=834490386. [Online; accessed 27-July-2018].

[40] Wikipedia contributors. 2019. Facebook-Cambridge Analytica data scandal — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/w/index.php?title=Facebook\T1\textendashCambridge_Analytica_data_scandal&oldid=892459300. [Online; accessed 16-April-2019].

[41] David J Wu, Ankur Taly, Asim Shankar, and Dan Boneh. 2016. Privacy, discovery, and authentication for the internet of things. In *European Symposium on Research in Computer Security*. Springer, 301–319.

[42] Jun Xu, A. Kumar, and Xingxing Yu. 2004. On the fundamental tradeoffs between routing table size and network diameter in peer-to-peer networks. *IEEE Journal on Selected Areas in Communications* 22, 1 (Jan 2004), 151–163. https://doi.org/10.1109/JSAC.2003.818805

[43] Fen Zhu, Matt W Mutka, and Lionel M Ni. 2005. Service discovery in pervasive computing environments. *IEEE Pervasive computing* 4 (2005), 81–90.