Towards Fine-Grained Access Control in Enterprise-Scale Internet-of-Things

Qian Zhou, Mohammed Elbadry, Fan Ye, and Yuanyuan Yang, *Fellow, IEEE*

Abstract—Scalable, fine-grained access control for Internet-of-Things is needed in enterprise environments, where tens of thousands of users need to access smart objects which have a similar or larger order of magnitude. Existing solutions offer all-or-nothing access, or require all access to go through a cloud backend, greatly impeding access granularity, robustness and scale. In this paper, we propose Heracles, an IoT access control system which achieves robust, fine-grained access control and responsive execution at enterprise scale. Heracles adopts a capability-based approach using secure, unforgeable tokens that describe the authorizations of users, to either individuals or collections of objects in single or bulk operations. It has a 3-tier architecture to provide centralized policy and distributed execution desired in enterprise environments. Extensive analysis and performance evaluation on a testbed prove that Heracles achieves fine-grained access control and responsive execution at enterprise scale. Compared with systems using access control list, Heracles eliminates or reduces by 10x–100x the updating overhead under frequent changes of subject memberships and policies. Besides, Heracles achieves responsive execution: it takes 0.57 second to access 18 objects which are scattered 1–9 hops away, and execution on a 1-hop or 2-hop object needs only 0.07 or 0.13 second respectively.

Index Terms—Internet of Things, Security, Access Control

1 Introduction

Access control is a fundamental requirement on Internet-of-Things [1], critical for not only convenience (e.g., lights), but also safety of people and physical assets (e.g., door locks). Most existing smart home products [2] offer coarse grained all-or-nothing access: family members have full rights while others have nothing. This is far from sufficient, especially in an enterprise environment where tens of thousands of subjects (i.e., employees) need to access smart objects which have a similar or larger order of magnitude (e.g., a university campus with tens of buildings each embedded with thousands of IoT devices).

The access control in such enterprise environments must be *fine-grained*. Given the same object, different subjects may have different access rights, or even different degrees of freedom invoking the same function of the object. The available access rights may also depend on the context (e.g., time of the day). For example, only executives may access the door lock, lights, projectors in a VIP meeting room; managers may occupy a conference room for up to half a day, while non-managers can use it for at most two hours. A janitor may enter all these rooms for cleaning before 9 AM, but with no access to IT equipment.

To ease management, many existing solutions [3], [4], [5] use a fully centralized strategy, at the expense of weaker availability and responsiveness. To operate an object, a subject sends a command to the cloud first, which authenticates the subject and confirms that she has sufficient rights, and then notifies the object to execute the command. This strategy places the cloud in the center of the access control loop. It ensures security since the cloud is well protected. However, upon loss of connectivity, nothing is accessible. Besides, the back-and-forth travel to the cloud may add significant latency, adversely impacting responsiveness thus user experience.

What is truly desirable is *centralized policy while distributed execution*. The policy regarding which subjects have what access rights, to what degrees, under what contexts, should be centrally managed. Thus it is convenient to add/remove an employee by changing a few records in a database at the (well-protected) backend, without making changes at a huge amount of objects one by one. The access to objects, however, should be distributed. When invoking a permitted function on an object, a subject should be able to do so via direct connectivity to the object, without detouring to other entities including the backend. This will ensure both availability and responsiveness of command execution.

Unfortunately, such access control for enterprise environments has not been studied in existing work. In this paper, we propose *Heracles*, an access control system that achieves fine-grained access control, centralized policy and distributed execution at enterprise scale. Heracles adopts a capability based approach where a subject requests from the backend secure, unforgeable tokens depicting her access rights to certain objects. Once a token is obtained, the access no longer involves the backend. The subject includes the token in her commands to the target object, which checks the token and the commands before executing the invoked functions. Our contributions are as follows:

- We design a 3-tier IoT access control architecture for enterprise environments, consisting of the backend, resource-rich objects and resource-constrained objects. It supports fine-grained degrees of function invocation, convenient centralized policy management and robust, responsive distributed command execution (i.e. access) at enterprise scale.
- We quantitatively analyze Heracles and an alternative approach of access control list (ACL) based distributed execution. Heracles is much more efficient and scalable under frequent changes of policies

and subject memberships, a common phenomenon in enterprise environments. It eliminates or reduces by 10x–100x the updating overhead in many situations.
- We offer solutions to two features desired by enterprise IoT: 1) an attribute-based access strategy for efficient *bulk operation* which controls a category of objects using a single command; 2) a delegation-based strategy to improve the responsiveness of resource-constrained objects during execution.
- We implement our design, and conduct extensive analysis and experiments on a testbed consisting of a mixture of 18 resource-rich objects and extra constrained objects. The results show that Heracles has secure access control, scalable updating, and agile responsiveness: it takes only 0.57 seconds to access 18 objects which are scattered 1–9 hops away from a subject, and execution on a 1-hop or 2-hop object needs only 0.07 or 0.13 second respectively.

2 Models and Assumptions

Node Type. The network consists of three types of nodes: backend servers, subject devices, and objects. The backend is well protected and run by human administrators. It maintains the profiles of registered subjects (possibly their devices) and objects; it also stores and updates access rights.

A subject is a person who uses a subject device (e.g., smartphone) to interact with objects. We assume the subject device has communication interfaces (e.g., WiFi radios), Internet connectivity to the backend, and reasonable computing/storage resources (e.g., > 2GHz CPU, tens of GBs of storage are common among smartphones). An object is an IoT device, or a "Thing". Objects have different amounts of resources: many are small ones with constrained hardware (e.g., Mica2, Arduino class: smoke/presence/fire detectors, light bulbs), while medium or large ones have space and power for moderate hardware (e.g., Raspberry Pi class: surveillance cameras, coffer makers, air conditioners, wall outlets). In the 3-tier architecture, small ones are *member objects*, medium/large ones are *leader objects*, and they are assigned different responsibilities. Besides, a *target* is the object that a subject attempts to operate, and it can be either a leader or a member one. Subject devices and objects together constitute a *ground network*.

We assume the backend, subject devices and objects are roughly time synchronized (e.g., within tens of seconds). We also assume the backend, subject devices and leader objects have enough computing resources to run public-key cryptographic algorithms (e.g., ECDSA, ECDH), while member objects may be able to run them only occasionally. Objects may have diverse communication interfaces, e.g., besides WiFi and Bluetooth, many IoT devices use ZigBee, Z-Wave, etc. We focus on security design above the network layer, and assume network connectivity exists among all nodes (e.g., via bridging devices with multiple radios), so does multi-hop routing [6], [7] in the ground network.

We assume objects are largely static once installed, thus the topology of the ground network is stable except occasional deployment changes such as addition/removal of objects. A subject device is moving with its owner, thus mobile, but the movement speed is usually slow (e.g., a person walking around). We assume many objects, especially leader ones, have enough energy (e.g., door locks, light bulbs, air conditioners and surveillance cameras are all wall-powered). We do not study energy-saving techniques (e.g., duty cycling) in this paper, but they can be applied orthogonally to battery-powered objects.

Network Scale. We present unique enterprise-scale IoT properties that distinguish enterprise IoT from home-scale IoT, especially on typical scales of several aspects, where $10^i, i \in \mathbb{Z}$ denotes an order of magnitude. E.g., 10^0 means several and 10^3 means thousands. They are intended to provide a rough sense, and not to be interpreted literally; actual systems may have smaller or larger scales.

1) huge subject/object amounts. An enterprise may have $10^4 \sim 10^5$ subjects (e.g., Google has 98K employees). According to our field study (Section 10), even a 2-story building may have ∼2K objects, and there can be many more stories/buildings in a university campus or big company. Thus an enterprise can easily have about $10^4 \sim 10^5$ objects in total. Note that in reality a subject has access rights to only a fraction of all the objects, of which the number is denoted as N, around $10^2 \sim 10^3$.

2) heterogeneous subjects/objects. Subjects can be classified to different categories based on their various attributes (e.g., departments, groups, positions), so are objects, based on device types, installation locations, etc. A subject usually belongs to k ($10^0 \sim 10^1$) subject categories; a subject category may have access rights to c ($10^0 \sim 10^1$) object categories, with n ($10^1 \sim 10^2$) objects in each (e.g., "all the devices in Room X": 10^1; "the lights on the 2nd floor": 10^2). A subject category has m ($10^1 \sim 10^2$) subjects. E.g., "the students in Class A": 10^1; "the employees in Department B": 10^2.

3) possibly frequent subject churns. In enterprises, employee entry/exit or promotion/demotion/rotation happen all the time. They may affect the access rights for subject individuals/categories. Such changes must be effectuated quickly and efficiently on related objects. Otherwise authorized users will fail to access new services timely, while unauthorized users continue to have access to services they are no longer eligible for. E.g., once a subject leaves the enterprise, all the N ($10^2 \sim 10^3$) objects she could access should stop accepting and executing her commands.

Data Caching & Discovery. We assume a data caching and discovery mechanism like PDS [8] exists. Independent data entities (e.g., public key certificates) protected by public-key signatures, are widely propagated and *cached* in the ground network. Due to multiple copies of an entity cached in different nodes, the entity can be *discovered* with higher robustness and responsiveness.

3 Design Goals

Fine-Grained Access Control. The system should be able to specify under what contexts a subject is allowed to invoke on an object what functions with what parameters. This comes from the *heterogeneous subjects/objects* property. Coarse grained all-or-nothing access control works fine for homes, where family members are granted full access rights while strangers nothing. In enterprise environments, however, subjects are heterogeneous (e.g., in departments, positions), thus have diverse responsibilities and access rights. This makes fine access control granularity necessary.

Three security goals should be achieved: *authenticity* is to ensure a party is indeed the claimed one; *integrity* is to ensure messages are not forged or altered by adversaries; *freshness* is that messages received are generated recently, and this prevents replay attacks where adversaries simply record and replay a previously sent legitimate message.

Centralized Management. The editing of node profile and access right information should be conducted at a single point (i.e. the backend), including adding/removing a subject/object, a category of subjects/objects sharing certain characteristics, and adding/removing a policy which describes which subject (category) has what access rights to which object (category). This centralized strategy makes the system easy to manage: the administrator does not need to make changes in a large amount of nodes one by one.

Robust & Responsive Execution. If the backend must be involved when subjects execute commands on objects, a total loss of access can happen upon a backend machine failure or a loss of the connection. Despite dedicated maintenance, such failures still occur occasionally in enterprise environments. We need distributed execution such that access is still available upon such failures. Besides, the latency from command issuing by subjects to execution by objects should be small for positive user experience.

Scalable Updating. Any change made on the backend (e.g., policy, subject addition or removal) must be quickly propagated and effectuated on the affected objects, to ensure valid commands are accepted and invalid ones rejected. The overhead is naturally small in smart homes, but large in enterprises due to the *huge subject/object amounts* property and the *frequent subject churns* property, which may lead to more updating failures and delays, compromising the system. The updating must be fast and efficient to make the system secure and scalable.

Non-Goals. We discuss strategies to alleviate the harm of node compromise and denial-of-service attacks which waste system resources by dumping many invalid messages, but complete solutions are out of the scope. Physical level jamming, attacks targeting routing or confidentiality/privacy are not our research topics, neither is trust management.

4 System Overview

There are four main interactions in the system (Fig. 1). We first present the design concerning leader objects only, and introduce that for member objects in Section 7.

1) Commission. To join the system, a subject/object must be registered at the backend out-of-band (e.g., manually by a human administrator), which signs and issues it a private key, a public key certificate (CERT) and a profile (PROF). The subject/object makes its CERT/PROF propagated and cached by nearby objects in the ground network.

2) Discover. The subject device proactively discovers [8] nearby objects by querying their CERTs/PROFs. PROFs contain human-readable descriptions so the subject gains knowledge of which objects provide what functions.

3) Request. The subject sends a signed request (REQ) to the backend, asking for a ticket (TKT)—a token she can use later to invoke certain functions on certain objects. The backend verifies the REQ, examines the access right database, and issues her a signed TKT which carries the requested capabilities and will get expired after some time.

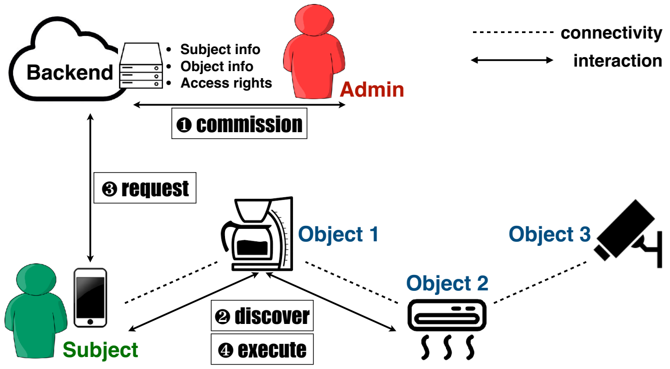


Fig. 1. The backend run by the administrator maintains the profiles and access rights of registered subjects and objects. A subject discovers objects in proximity (e.g., within 2 hops), and requests a ticket describing her access rights to the objects she is interested in operating. She then sends a command to operate the target (e.g., air conditioner); the command carries the ticket to prove its operation authorization.

4) Execute. The subject operates the target by sending a command (CMD), which carries a TKT proving the authorization for its operation. It may be forwarded [6], [7] towards the target by multiple objects. The target checks that the CMD is legitimate and then executes the function specified by the CMD; otherwise it rejects the CMD. A response (RES) is sent back to the subject.

5 Interactions among Nodes

Before presenting the details in the four interactions, we comment a bit more on the backend. It maintains the profiles stating the attributes of every registered subject/object, and subjects' fine-grained access rights to objects.

Fine-Grained Access Constraints. Given the same object, different subjects may be allowed for different functions, or different parameters, time ranges, invocation counts, etc. for the same function. A regular employee can set the thermostat within a normal temperature range, but a repair technician may set extreme temperatures for testing. A janitor may open all locked doors before 8 AM for cleaning, but loses access during business hours. An external UPS driver may get a one-time access token to raise the storage door once to slip in packages. Formally, a constraint is expressed as $(type : \cup item)$, with *type* indicating what to constrain (e.g. parameters) and a union of $items$ together specifying allowed values. An $item$ here is either a set (denoted as $\{x, y, ...\}$, e.g., parameter set {"on", "off"}) or an interval (denoted as $[x\ y]$, e.g., time range [9 17]).

5.1 Commission

A subject must first register at the backend out-of-band. Certain proofs (e.g., government/company issued IDs) may be needed. Then the backend assigns her an ID, a private key, a signed public key certificate (CERT), a signed profile (PROF), together with the backend's public key (K^{pub}_{Admin}). Also, the backend adds the subject's access rights to its

$\mathcal{S} \rightarrow \mathcal{B}ackend : [ID_S,\ \{O, \{F, C\}\},\ LIFE,\ T]SIG_S$
$\mathcal{B}ackend \rightarrow \mathcal{S} : [ID_{TKT},\ ID_{AR},\ ID_S,\ \{O, \{F, C\}\},\ LIFE]SIG_{Admin}$

Fig. 2. $\mathcal{S}$ubject sends a REQ to $\mathcal{B}$ackend and receives a TKT.

$\mathcal{S} \rightarrow \mathcal{T}arget : [ID_{CMD},\ TKT,\ O, F, P,\ T]SIG_S$
$\mathcal{T}arget \rightarrow \mathcal{S} : [ID_{CMD},\ State,\ Data,\ T]SIG_{Target}$

Fig. 3. $\mathcal{S}$ubject sends a CMD to $\mathcal{T}$arget object and receives a RES.

database. After loading such data into her devices (e.g., smartphone, tablet), the subject publicizes her CERT/PROF in the ground network so they are widely cached and can be easily retrieved [8], [9] by other nodes.

An object follows a similar process. Its PROF describes: i) *which*: information like ID, human-readable name, type (e.g., door lock), make/model, version, etc.; ii) *where*: information about its location, e.g., "Light Engineering Building, Floor 2, Room 217" would distinguish those devices in a particular room/building; iii) *functions*: the allowed operations and associated parameters. E.g., a lamp's functions may include "*set_brightness*", with an integer between 1–100.

The content of PROF can be structured (e.g., in JSON, XML) such that it can be queried. One option for the human-readable name is a hierarchical one embedding the object's location, e.g., /StonyBrookUniversity/LightEngineeringBuilding/Floor2/Room217/Light1. Such names can optionally be used to route [7] a command to the target for command execution (Section 5.4).

5.2 Discover

The subject device discovers nearby objects by querying their CERTs/PROFs. PROFs contain descriptions so both the human user and her device gain knowledge of which nearby objects provide what functions. Our design does not enforce any particular discovery mechanism, either an IP-based or a data-centric one works. Data-centric caching and discovery [8], [9] may be preferred for their data acquisition speed and robustness.

5.3 Request

The subject sends a request (REQ) to the backend, asking for a token she can use to invoke certain functions on certain objects. The backend verifies her REQ, examines the access right database to ensure she does have those rights, and sends back a signed ticket (TKT) which describes the requested capabilities.

ID-based & Attribute-based Ticket. Heracles offers both ID-based TKTs and attribute-based TKTs, preferred in different situations to achieve better flexibility or reduce message overhead. An ID-based TKT specifies a set of objects by enumerating their IDs, while an attribute-based one uses *attribute predicates* to describe categories of objects sharing certain characteristics (e.g., $\{type = lamp \wedge floor = 2\}$ means all the lamps on the 2nd floor). An attribute-based TKT is used to achieve efficient bulk operation (i.e., one command controls a large group of objects), which will be introduced in detail in Section 6.

Subject S sends a REQ (Fig. 2) including: 1) ID_S: a unique identity number of S ; 2) O: the object to which S requests her access rights, which is either an object (specified by its identity ID_O) or object category (specified by predicate $Attr_O$). $\{...\}$ denotes a set so multiple object or object categories can be included; 3) F: an O's function to which S requests her access rights; 4) C: a set of constraints (e.g., parameters) on F; 5) $LIFE$: the lifetime by which the TKT expires; 6) T: a timestamp for REQ's freshness. Note that F, C and $LIFE$ are optional in a REQ: if they are left empty, the backend can decide what functions, constraints and lifetime to include in the TKT based on certain rules.

Timestamp T is included for defending against replay attacks. Given the maximum time synchronization error e, the backend keeps the hash codes of all the REQs received in the recent time window e. A REQ is considered fresh if the difference between T and the backend's local time is less than e, and its hash code is not seen in the window. The backend has enough computing/storage resources for that. Other anti-replay mechanisms include: challenge-response, which requires a two-round handshake, significantly increasing the latency; monotonic counters, which require a counter for each subject-object pair, and are much easier to predict than nonces. Thus we choose the combination of timestamps and hash codes for freshness. $[...]SIG_X$ denotes the plaintext in brackets followed by a public-key signature generated by entity X for the content in brackets. SIG_S and SIG_{Admin} protect the authenticity and the integrity of REQ and TKT so they cannot be forged or altered.

Every TKT has an identity ID_{TKT} such that it can be referenced later in command execution (Section 5.4) or ticket revocation (Section 5.5). This improves efficiency and responsiveness. Each access right stored in the backend database also has an identity ID_{AR}, carried by every TKT generated based on this access right. This ID is required for an attribute-based TKT but not for an ID-based one. ID_{AR} is used for efficiently referencing and revoking all the TKTs generated based on the access right (Section 6).

5.4 Execute

The subject sends a command (CMD) to the target to invoke some function. The CMD might be relayed by multiple objects towards the target using a routing protocol [6], [7]. The target verifies the CMD and if legitimate, it carries out the invoked function; otherwise it rejects the CMD. In both cases a response (RES) is sent back.

ID-based & Attribute-based Command. An ID-based CMD carries an ID-based TKT and specifies target objects with ID enumeration, while an attribute-based one carries an attribute-based TKT and targets object categories using predicates. An attribute-based CMD is used for bulk operation, which will be introduced in detail in Section 6.

Subject S sends a CMD (Fig. 3) including: 1) ID_{CMD}: a random, unique identity number of this CMD; 2) O: the target, expressed as either ID_O or $Attr_O$; 3) F, P: the functions and parameters that S attempts to invoke on O; 4) TKT: the ticket (Fig. 2) proving the authority of S to invoke F, P on O; 5) T: a timestamp for CMD's or RES's freshness; 6) $State$, $Data$: execution error code and return data.

When an object receives a CMD, it will find out if it is a target by comparing its ID (if the CMD is ID-based)

$$\mathcal{B}ackend \rightarrow \mathcal{O} : [\{ID_{TKT}, LIFE\},\ T]SIG_{Admin}$$

Fig. 4. $\mathcal{B}$ackend sends a REV (the 1st form) to $\mathcal{O}$bject, telling it to revoke the ticket with identity ID_{TKT}.

or attributes (if attribute-based, and recall that an object knows its attributes from its PROF) with the CMD's O. The command execution is asynchronous so a subject device does not block on any single CMD. Here the same ID_{CMD} is used in CMD and RES so the subject device knows which RES corresponds to which CMD, and may take further actions for those CMDs getting no RESs (e.g., retransmission). The operation part (O, F, P) must be a subset of the access rights depicted by TKT to pass authorization check conducted by the target.

The freshness of CMD/RES is protected in a similar way to REQ. But here ID_{CMD} effectively serves as a nonce and is kept in the recent time window e. As long as the time synchronization protocol can achieve a reasonably small e (e.g., tens of seconds), the number of remembered ID_{CMD}s will not be many. SIG_S and SIG_{Target} protect the authenticity and the integrity of CMD and RES.

5.5 Ticket Revocation

A subject may lose authorization she once had (e.g., being discharged, moved to a different position). Thus issued TKTs carrying unexpired access rights must be revoked.

To this end, the backend must keep all the TKTs it has issued before their expiration times. Given any change in access rights, it must examine and identify those carrying invalid but unexpired authorizations. It generates a signed ticket revocation message (REV), which can have two forms. The first form includes the IDs and expiration times of all the TKTs to be revoked. The REV is publicized and widely cached among nodes. Objects will add the IDs, expiration times of revoked TKTs to their local ticket revocation lists (TRL). Upon expiration (actually slightly later, at least e after expiration) a revoked TKT's ID will be removed from the TRL. To avoid whole-network propagation of a REV affecting only a few TKTs and objects, the backend may send the REV to those objects and their vicinity only. Any CMD referencing a TKT whose ID is in the TRL will become invalid. The second form is for attribute-based TKTs only, and details can be found in Section 6.

6 Bulk Operation

Bulk operation uses a single command (CMD) to operate a possibly large group of objects with common characteristics. It is common in enterprise IoT. E.g., a student uses one CMD to turn off all the devices in her lab when leaving work, or a manager uses one CMD to trigger all the alarms in the building he is in charge of to notify people to evacuate, or a janitor turns off all the lights on a floor when finishing a night tour. An attribute-based CMD achieves the goal, using two attribute predicates: 1) In the ticket (TKT) referenced by the CMD, one predicate O specifies the object category to which the subject has access rights; 2) In the CMD, the other predicate O specifies the object category that the subject attempts to operate, i.e., the targets.

$$\mathcal{B}ackend \rightarrow \mathcal{O} : [\{ID_{AR}, LIFE\},\ T]SIG_{Admin}$$

Fig. 5. $\mathcal{B}$ackend sends a REV (the 2nd form) to $\mathcal{O}$bject, telling it to revoke all the tickets carrying the access right with identity ID_{AR}.

A primitive predicate is a triple ($attribute$, $operator$, $value$), and possible operators in our system include: $=, \neq, <, >, \leq, \geq, \in$. A complex predicate consists of multiple primitive ones combined in logic AND $\wedge$, OR $\vee$, NOT $\neg$, etc. A simple form is to use logic AND only. E.g., "all windows in Room 217" can be expressed by $\{type = window \wedge room = 217\}$. We implement this design and the support for other forms can be added if necessary.

A bulk operation CMD can be propagated among peer devices directly. This is suitable when targets are within a small or medium scope, e.g., one or a few rooms, floors. Such a CMD is forwarded by an object to its neighboring objects, hop by hop till the CMD reaches every possible target. This P2P strategy does not rely on backend connectivity, and achieves better execution robustness and responsiveness. When target objects are spread over large areas (e.g., in another building), hop-by-hop routing may be slow or even unavailable. In such cases the CMD can be sent via the backend directly to the destination or its vicinity, and then propagated among peers.

Message Overhead. An ID-based CMD can also be used for bulk operation if its TKT enumerates all the target IDs, but it will be short and efficient only when small numbers of objects are included. Since its size grows linearly as more object IDs are enumerated, the TKT may become too large, incurring large overhead and long latency in operation. Besides, to operate a newly added object, a new TKT must be requested to include that object's ID.

On the contrary, an attribute-based one's length only increases with the number of object categories specified, regardless of how many objects inside. Also, it can be used to access new, previously unknown objects. E.g., to access a newly installed light, if ID-based, the subject has to request a new ticket that covers the light's ID; however, if she holds an attribute-based TKT specifying her rights of "operating lights", she does not need to request another ticket.

Ticket Revocation. An attribute-based TKT can be revoked by both forms of revocation messages (REV): when the number of TKTs to be revoked is small, we use the first form (Section 5.5) referencing ID_{TKT}s; when an attribute-based access right is removed from the backend, the number of affected TKTs may be large (e..g, $\sim 10^3$) because the access right may have been requested by many subjects in a category, thus enumerating ID_{TKT}s is inefficient. In this case ID_{AR} is referenced to efficiently revoke all the TKTs carrying the access right.

7 Leader and Member Binding

Due to the abundance of medium or large objects (i.e. leader objects, as defined in Section 2) with sufficient power and resources in enterprise environments, we leverage them to create a hierarchical structure where leader objects form the "backbone" while member objects associate with them as "leaves". The leaders will handle those frequent, compute

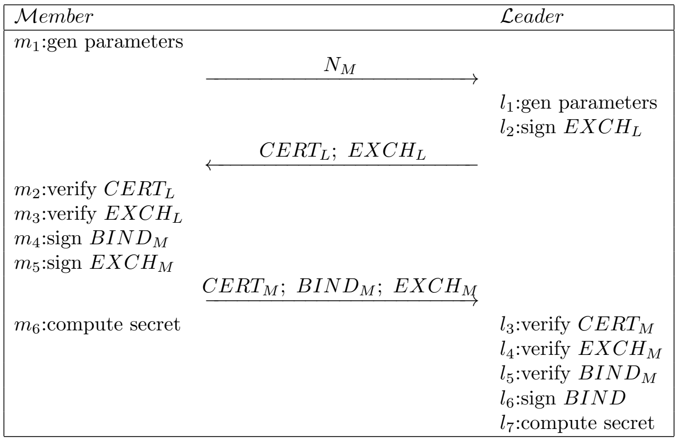


Fig. 6. $\mathcal{M}$ember and $\mathcal{L}$eader establish a shared secret and generate a BIND.

or energy intensive responsibilities (e.g., public-key cryptographic operations, message forwarding) on behalf of their members. A member depends on its leader(s) to receive and verify commands from subjects, and forward responses back to them. This design allows us to leverage more powerful Things to serve less capable ones. The interactions are:

Commission. A member object follows almost the same registration process at the backend as a leader one does, except that its name in the profile (PROF) may not reflect its location. The reason is a member object, usually small and free of wired power supply, has a higher chance of being moved. Thus it is better not to carry its location in its PROF such that the backend does not have to issue a new PROF often. Instead, we obtain its location by checking which leader it is using.

Bind. Each member object must "bind" to at least one leader object. A member object broadcasts messages seeking leaders from one-hop neighbors, and leader objects that are willing to accept more members will respond. The member object chooses one or multiple as its pre-leader(s) (e.g., based on RSSI) and starts to establish a shared secret and generate a binding notification (BIND). The BIND reveals the member object's location: it tells which leader(s) the member object associates with, thus should be used as the destination when sending commands to operate the member. Its format is $[[ID_{BIND}, L, M, LIFE, V]SIG_M]SIG_L$, where ID_{BIND}, L, M, $LIFE$, V denote the BIND's ID, leader's ID, member's ID, BIND's expiration time and version number. An unexpired BIND will be overridden by another BIND with the same L and M but a higher version. It is generated and signed by the member, then sent to, signed and publicized by the leader. This nested double signing prevents a leader or member from unilaterally publicizing a forged bilateral relationship.

Our message flow of shared secret establishment and BIND generation is given in Fig. 6, and it is inspired by the design of TLS handshake [10]. ECC-based TLS supports multiple key exchange algorithms, with many parameters configurable (e.g., elliptic curves, point formats). By fixing the key exchange algorithm at ephemeral ECDH and other parameters (e.g., curve is *secp224r1*), we reduce the number of messages to three, while generating BIND meanwhile.

$$\mathcal{L} \rightarrow \mathcal{M} : [ID_{CMD},\ F, P,\ T]MAC_{L,M}$$
$$\mathcal{M} \rightarrow \mathcal{L} : [ID_{CMD},\ State,\ Data,\ T]MAC_{L,M}$$

Fig. 7. $\mathcal{L}$eader sends an adapted CMD to $\mathcal{M}$ember and receives an adapted RES.

Member object $\mathcal{M}$ starts by sending a nonce N_M. After receiving it, leader object $\mathcal{L}$ generates a key exchange message $EXCH_L = N_L, P_L^{pub}, (N_M, N_L, P_L^{pub})SIG_L$, where N_L and P_L^{pub} denote the leader's nonce and key exchange material (an ECDH public parameter generated by $\mathcal{L}$, Step l_1). N_M, N_L are used in challenge-response, for freshness. $SIG_X(...)$ is a signature signed by entity X for the content in parentheses. SIG_L is generated (Step l_2) to authenticate the ECDH key exchange material. $\mathcal{L}$'s public key certificate $CERT_L$ and $EXCH_L$ are sent to $\mathcal{M}$.

After receiving them, $\mathcal{M}$ first verifies the signature in $CERT_L$ (Step m_2). If valid, it uses $\mathcal{L}$'s public key to verify SIG_L in $EXCH_L$ (Step m_3). If valid, it generates and signs a BIND (Step m_4). Also, it generates $EXCH_M = P_M^{pub}, (*)SIG_M$, where P_M^{pub} denotes the member's key exchange material and $*$ denotes all the messages sent and received so far. $(*)SIG_M$ is generated (Step m_5) to protect authenticity and integrity. Note that the P_M^{pub} used here is generated beforehand (Step m_1) to speed up the binding process. $CERT_M, BIND_M, EXCH_M$ are all sent to $\mathcal{L}$, and $\mathcal{M}$ starts computing the ECDH shared secret.

Similarly, $\mathcal{L}$ verifies the signature in $CERT_M$, and then $(*)SIG_M$. After that it verifies the signature of $BIND_M$, and if valid, it appends its own signature to make it doubly signed. At last, it starts computing the ECDH shared secret. Now both the member and the leader have the shared secret, and they use a key derivation function (e.g., HKDF [11]) to convert the secret to a session key for their future conversations. Member objects launch this handshake periodically (e.g., a few times a day) or when their leaders change, to update session keys and binding relationships.

Discover & Request. A leader publicizes its members' CERTs/PROFs in the ground network. Then discovering a member object and requesting a ticket (TKT) for it becomes exactly the same as dealing with a leader object.

Execute. When a leader receives a command (CMD), it will find out if it or its member is a target by comparing their IDs (if the CMD is ID-based) or attributes (if attribute-based) with the CMD's O. If the target is its member, it will check if the CMD is legitimate and if so, send to the member an adapted CMD (Fig. 7) with the same ID_{CMD}, F, P, T, protected by a message authentication code (MAC) generated from their session key. The MAC ensures authenticity and integrity, and the freshness check is done similarly. The leader replaces the public-key signature with a MAC because it has sufficient resources/power to finish such compute/energy intensive tasks at reasonable speed but its members do not. In this way a member only needs to verify MACs, which incurs much less time and energy.

8 Security Analysis

Threat Model. We assume the backend is trustworthy and well-protected. Also, communication between the backend and subject/object devices is secure.

We assume breaking the cryptographic algorithms (e.g., ECDSA, ECDH) are computationally infeasible when long enough keys are used (e.g., 128-bit). Attackers can capture, inject, modify and replay messages sent over the communication channel. **Sources.** Attackers may be *external*—they are not registered at the backend thus have no backend-signed public keys, or *internal* ones that are registered but go rogue. **Roles.** Attackers may behave passively as eavesdroppers, or actively to impersonate subjects or objects and interact with benign nodes. **Targets.** Attackers may target authenticity, integrity, freshness, and availability.

Like TLS and many other algorithms, the security of ours is on the premise that secret information is kept to its owner, and is computationally infeasible to compromise, and cannot be obtained from sources outside of the channel. However, in reality it can, e.g., attackers have military computing resources, or they leverage malware or social engineering to steal private keys from users/devices. Resisting those attacks is out of the scope. Our analysis below shows that: 1) our system resists well to attacks from external adversaries that target authenticity, integrity and freshness; 2) it does not address attacks from internal adversaries or DoS, but we discuss strategies which possibly alleviate the harm.

Discover. External leader, member objects may pose as benign ones by propagating profiles (PROF), waiting for subjects to discover and later execute commands (CMD) on them. Because they do not have properly signed PROFs, it is easy to detect and drop them. Internal ones, however, are able to entice subjects to operate them, thus collecting information about the subjects' locations, operation behaviors, etc. Such privacy issues are beyond the scope.

Bind. An external leader object may cajole benign member objects into choosing it as their leader and then manipulate them. But it has no private key or public key certificate (CERT) assigned by the administrator, and cannot accomplish the handshake for shared secret establishment and binding notification (BIND) generation. For the same reason, an external member object will fail in finding a leader. To the contrary, a malicious internal leader object is able to recruit benign members. A member object can have multiple leaders (only one is active at a time) and change the active one from time to time, reducing the probability of accepting malicious CMDs. Similarly, a malicious internal member object can associate with benign leaders, but it cannot cause much harm beyond itself. Besides, a malicious internal leader object may publicize fake BINDs, but our double signing strategy foils that.

Request. An external subject device cannot succeed in requesting tickets (TKT) due to the lack of a valid private key thus valid signatures. A replayed request will fail due to the protection of timestamp and hash code. A malicious internal subject device can sign properly, thus request TKTs successfully. We may use extra mechanisms (e.g., operation behavior analysis) on the backend to detect compromised subject devices. Once detected, the subject device will not be issued any new TKT by the backend, and the TKTs it has obtained will be revoked.

Execute. An external subject device's forged/altered CMDs will not get accepted by leader objects due to the protection of signatures, neither will its replayed ones because we have timestamp and nonce jointly for resistance. The node may keep sending invalid CMDs to waste resources of benign nodes. To mitigate this harm, we may ask intermediate relaying nodes to examine CMD integrity/freshness (originally such checks are conducted by the target only). This *en-route check* drops an invalid CMD before it travels far, reducing the attack range.

An external node may mimic a leader object. Its CMDs to member objects will be found illegitimate for either wrong message authentication codes or being obsolete. As for DoS attacks, the malicious leader object may send large amounts of invalid CMDs to member objects around, attempting to drain their batteries. A member object may regard being awakened too often as abnormality and report it to the administrator, who will take further countermeasures. Similarly, an external member object will fail in making its forged/altered/replayed responses (RES) accepted by a leader object. Note that usually a leader object has sufficient energy from wired power supply and does not have the dead battery problem, but a similar detection strategy can be applied to notify the administrator.

A malicious internal subject device could get its CMDs executed, attacking authenticity and integrity successfully. Faced with such situations, the backend can issue subject devices TKTs of constrained access rights and short lifetimes to alleviate the damage to some degree. The attacker, though having compromised the subject's identity, can only exert the access rights offered by the TKTs stored in the device. Thus the less capable the TKTs are, the less harm the attacker can do. The attacker may try requesting more TKTs, but as mentioned, the backend may detect and reject it.

If a leader object gets compromised, all of its members will be indirectly compromised and execute the attacker's CMDs. But as mentioned, a member object may keep switching from one leader to another, reducing the amount of malicious CMDs it receives. As for a malicious internal member object, it is under control of the attacker. Possibly, its leader may detect its abnormality, e.g, finding it does not follow a legitimate CMD, and then inform the administrator.

9 Updating Overhead Analysis

When the administrator adds/removes a subject/object individual/category, or edits a policy, the change needs to be immediately synchronized to the affected objects on the ground. Otherwise authorized users will fail to access new services timely, while unauthorized users continue to have access to services they are no longer eligible for. We define *updating overhead* as the number of objects that need to be notified immediately. We conduct quantitative analysis and find that compared with an ACL based strategy, our capability based design is able to: i) eliminate the updating overhead when adding a subject, policy or subject/object category; ii) reduce the overhead by **10x–100x** when removing a subject or ID-based policy; iii) reduce the overhead slightly in other cases. As a result, Heracles achieves efficient updating, thus scalable and secure enterprise IoT access control. We outline how ACL and Heracles work differently before presenting overhead comparison:

ACL. In an ACL based system, each object stores an access control list [12] specifying by which subjects it can be accessed, and what functions (i.e. access rights) are allowed.

TABLE 1
The number of affected objects when adding/removing a subject/object/policy. Compared to an ACL based strategy, Heracles reduces updating overhead: to 0 ; by 10x–100x ; slightly , depending on which updating operation the administrator performs.

	Add 1 Subject	**Rmv 1 Subject**	**Add 1 Object**	**Rmv 1 Object**	**Add 1 Policy**	**Rmv 1 Policy**
ACL, ID	N $(10^2 \sim 10^3)$	N $(10^2 \sim 10^3)$	1	1	1	1
Heracles, ID	0	$\theta N \approx [0.01, 0.1]N$	1	1	0	$\theta \approx [0.01, 0.1]$
ACL, attribute	0	kcn $(10^2 \sim 10^3)$	1	1	n $(10^1 \sim 10^2)$	n $(10^1 \sim 10^2)$
Heracles, attribute	0	$\eta kcn \approx [0.01, 0.1]kcn$	1	1	0	$\Theta n < n$

TABLE 2
The number of affected objects when adding/removing a subject/object category.

	Add 1 SubjectCateg	**Rmv 1 SubjectCateg**	**Add 1 ObjectCateg**	**Rmv 1 ObjectCateg**
ACL, attribute	cn $(10^1 \sim 10^3)$	cn $(10^1 \sim 10^3)$	n $(10^1 \sim 10^2)$	n $(10^1 \sim 10^2)$
Heracles, attribute	0	$\Theta cn < cn$	0	$\Theta n < n$

An ID-based ACL specifies subjects by enumerating their IDs, e.g., the access right part is $\{ID_S, \{F, C\}\}$, where ID_S, F, C are a subject's identity, allowed functions, constraints. To execute a command (CMD) on an object, a subject needs to prove her identity (e.g., using her signature), and the object will accept it only if her ID is in the list and the functions/constraints match. An attribute-based ACL describes subject categories using predicates, i.e., $\{Attr_S, \{F, C\}\}$. And in command execution, a subject needs to prove her attributes (e.g., by attaching her profile (PROF) with commands). Only if her attributes match the predicates ($Attr_S$) may the command be accepted.

Capability. In a capability [12] based system, a subject holds a ticket (TKT) stating her access rights to certain objects. The ticket is signed by a trusted common authority (e.g., the administrator) so nobody can forge it. An ID-based TKT specifies objects by enumerating their IDs, while an attribute-based one uses attribute predicates (Section 6). In either case the subject sends a TKT together with her CMD, and the object verifies the TKT to find out if the subject is authorized to access it.

9.1 Symbol Definition and Magnitudes

We first introduce as follows the symbols we use in updating overhead expressions. Explanation for the magnitude values of N, k, c, n, m can be found in Section 2.

1) object amounts. An enterprise may have $10^4 \sim 10^5$ objects in total. In reality a subject has access rights to only a fraction of all the objects, of which the number is denoted as N in an ID-based system, around $10^2 \sim 10^3$.

2) categories. A subject usually belongs to k $(10^0 \sim 10^1)$ subject categories; each subject category may have access rights to c ($10^0 \sim 10^1$) object categories, with n $(10^1 \sim 10^2)$ objects in each. Thus, in an attribute-based system, the number of objects a subject can access is kcn, whose order of magnitude is consistent with N in an ID-based system. A subject category has m $(10^1 \sim 10^2)$ subjects.

Note that one ID-based policy specifies one subject's access rights to one object; one attribute-based policy relates one subject category and one object category, thus n objects.

3) policy service rates. A policy is called *in service* if there exists a TKT which was generated based on it and is not expired yet. For simplicity, we assume a subject requests dozens of (10^1) ID-based TKTs and a few (10^0) attribute-based TKTs per day, with each having a 1-day lifetime; her N ID-based access rights have an independent identical chance to be requested (if ID-based), so are her kc attribute-based access rights (if attribute-based).

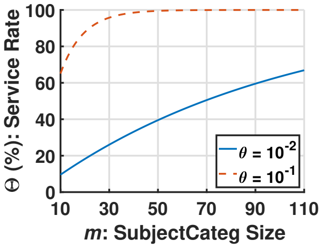


Fig. 8. The impact of subject category size on attribute-based policy service rate Θ.

Since an ID-based policy serves exactly one subject, the policy is in service if and only if that subject has requested a TKT for access rights specified by it that day. So ID-based policy service rate $\theta = 10^1/N = 10^{-2} \sim 10^{-1}$.

However, an attribute-based policy serves a subject category which has m subjects. The policy is in service as long as *at least one* subject among the m has requested a TKT generated based on it that day. The probability that one subject requests it is $\eta = 10^0/kc = 10^{-2} \sim 10^{-1}$, and attribute-based policy service rate $\Theta = 1 - (1 - \eta)^m$. Fig. 8 shows that Θ is less than but close to 100% most of the time, unless the access right is for unpopular IoT service (e.g., $\theta = 10^{-2}$) and there are not many subjects (e.g., $m < 100$) in the subject category served by the policy.

9.2 Updating Overhead Comparison

We present the updating overhead of four strategies (Tab. 1 and Tab. 2): ACL and Heracles (capability based), with each having an ID-based variant and an attribute-based variant.

1) ACL, ID-based. An ID-based ACL system has the largest updating overhead. When a subject joins/leaves, the backend needs to immediately contact all the N $(10^2 \sim 10^3)$ objects that she can access, to add/remove her access rights to/from their ACLs. When an ID-based policy gets added/removed by the administrator, the corresponding object always needs to be notified.

2) Heracles, ID-based. Heracles adopts capability based access control, and has remarkably smaller updating overhead than ID-based ACL facing most operations. First, there is no overhead when a subject/policy is added because now it is the subject's duty to request her tickets when needed. This overhead shift mitigates not only the workload but also the difficulty of updating, because: 1) a subject consciously requests only the access rights she is about to use, while in ACL all objects must be notified regardless of whether they are to be accessed; 2) subjects share the updating work from the backend and each handles her own part, making it more robust against network failures—the backend cannot do much when finding some objects unreachable, but a human subject can respond flexibly, e.g., try alternative network connections.

Second, the updating overhead is one or two orders of magnitude smaller than ID-based ACL when a subject/policy is removed. It is because an updating message will be sent to an object, telling it to revoke a subject's ticket, only if the subject requested a ticket targeting that object early that day. If a policy is not in service (i.e., no ticket based on it was requested), the corresponding object does not need to be contacted when the subject or the policy is removed. As introduced, an ID-based policy has a small service rate θ ($10^{-2} \sim 10^{-1}$), so the overhead is reduced by 10x–100x.

3) ACL, attribute-based. Similar to ID-based ACL, an attribute-based ACL system has large overhead. Compared with ID-based ACL, its main advantage is when a subject is added, no objects need to be notified. This is because the ACLs stored by objects specify authorized subjects using predicates instead of ID enumeration; later it finds a subject authorized if the subject presents a valid profile showing that her attributes meet the ACL predicates. But like ID-based ACL, when a subject leaves, all the kcn ($10^2 \sim 10^3$) objects she could access must be updated to reject her access attempts in the future. As for policy updating, note that an attribute-based policy relates one object category, thus n ($10^1 \sim 10^2$) objects should be notified to update their ACLs when the policy is added or removed.

4) Heracles, attribute-based. Heracles supports both ID-based and attribute-based access, and it outperforms attribute-based ACL in updating efficiency. First, it eliminates the overhead when a subject/policy is added because the burden has shifted to the subject/backend to obtain/issue tickets properly. Second, similarly, one or two orders of magnitude fewer objects need to be notified to revoke a subject's tickets when the subject is removed, because she only holds tickets for ηkcn ($\eta = 10^{-}2 \sim 10^{-}1$). objects, among all the kcn ($10^2 \sim 10^3$) ones she could access.

Third, when removing an attribute-based policy, Heracles always has smaller overhead than attribute-based ACL although the improvement may be slight. An attribute-based policy relates n objects, which need to be notified for ticket revocation only if the policy is in service (i.e., there exist unexpired tickets generated based on it). Thus, the expected value of updating overhead is Θn, compared to ACL's n. Though each subject has a small probability η to put a policy in service, an attribute-based policy can be requested by any of the m ($10^1 \sim 10^2$) subjects, and the probability that at least one of them puts the policy in service can be high (i.e. $\Theta \approx 100\%$, then Heracles and ACL have similar overhead), especially when m is large. Limiting the size of a subject category (i.e. m) makes updating overhead smaller. E.g., when $m = 10$, $\Theta = 10\% \sim 65\%$ (Fig. 8).

Adding/Removing Categories. Tab. 2 shows the overhead when adding/removing a subject/object category, and these operations are for attribute-based systems only. As is seen, Heracles eliminates the overhead when adding a category, and reduces the overhead slightly when removing one. For both attribute-based ACL and Heracles, the overhead of adding/removing a subject category is c ($10^0 \sim 10^1$) times of that of adding/removing a policy in that system, because a subject category has access rights to c object categories. The overhead of adding/removing an object category is the same as adding/removing a policy, because each attribute-based policy relates exactly one object category.

Adding/Removing Objects. In any of the four strategies, when an object is added/removed, only that object needs to be notified, so the overhead is always 1. Strictly speaking, Heracles does not reduce the overhead in these two cases, but the overhead is originally small enough, and can hardly be further reduced to 0.

9.3 Impact of Updating Overhead on Security

Either ACL updating (if ACL based) or ticket revocation (if using Heracles) messages need to reach the affected objects immediately, otherwise valid subjects' commands will be rejected while invalid ones' commands accepted. Larger overhead in synchronizing access control related information to objects results in higher vulnerability, because more objects to notify inevitably leads to more updating failures/delays, due to network failures, processing/transmission delay, etc. Heracles greatly reduces the number of objects to notify, thus chances of failures, achieving much stronger security.

Delay-Tolerant Updating. Besides the updating we have analyzed so far which needs immediate completion, there may be other updating activities. E.g., in an ACL based system, objects periodically (e.g., every day) check ACL updates from the backend; in Heracles, subjects request needed tickets beforehand. Such updating activities (in both ACL systems and Heracles) are scattered throughout all the subjects/objects and time of the day; additionally, they are delay-tolerant. They are different from the updating incurred by the administrator's operation on the backend database, which contacts objects in *burst mode* and is delay sensitive. So they are less an issue for system burden and security, thus not considered in our analysis.

10 Experimental Evaluation

We have implemented our prototype including four components of Heracles: the backend, subject devices, leader objects and member objects. The backend program runs in a server machine. We use a Samsung Galaxy S8 (2.3GHz Quad + 1.7GHz Quad CPU, 4GB RAM) as our subject device, and deploy 18 leader objects in a large room to construct a group network with diameter up to 9 hops, where each leader object runs on a Raspberry Pi 3 (1.2GHz Quad CPU, 1GB RAM). Besides, we use Arduino Mega 2560 (16MHz Clock + 8KB RAM) as resource-constrained member objects. Different radios can be used, as long as network connectivity and routing exist, and we use WiFi on our testbed: Pi has a built-in WiFi module; the Arduino Mega 2560 is equipped with an ESP8266 module for WiFi communication.

We evaluate message overhead of commands, time cost of cryptographic operations (e.g., signature signing/verification), binding latency, and command execution latency. Note that evaluation results like latency depend significantly on factors like the radio, background traffic intensity, routing protocol and cryptographic algorithms/parameters/libraries chosen in implementation. Thus the performance numbers should not be interpreted literally, but rather revealing the likely ranges and magnitudes. Also, though our design targets enterprise IoT, our testbed consisting of 18 leader objects (and extra member objects) which constitute a 9-hop network is sufficient for testing command execution latency. In reality there will be more objects, but hop count mostly would not be larger, because most users will be controlling IoT devices in proximity. For rare cases of distant targets (> 10 hops), commands can be routed via the backbone Internet for better resilience and shorter latency.

10.1 Command Message Overhead

By comparing the length of CMDs which are either ID- or attribute-based in two real cases (Student Case and Administrator Case), we prove that the two are preferable in different scenarios: ID-based CMDs are more efficient for small amounts of objects in various categories (Student Case like), while attribute-based ones excel in bulk operation that targets large amounts of objects in a few categories (Administrator Case like).

Field Study. Types and amounts of the objects in the two cases are from a field study on our engineering building on campus, which has two floors, with 32 offices/labs on the 1st floor, and 36 on the 2nd. A medium office/lab is used as a representative which has 6 ceiling lights, 8 desk lamps, 5 computers, 1 door, 3 windows, 1 alarm and 6 other devices. Totally, there could be approximately 30 objects each room and 2040 objects in this building, excluding those in restrooms, lobbies or corridors.

Student Case. In this scenario, a graduate student requests a TKT in the morning for certain objects installed in her lab, for the functions she will probably use this day. There are 8 objects included: 2 ceiling lights, 2 desk lamps, 1 door, 1 window, 1 coffee maker, and 1 air conditioner. This TKT is a representative covering a few objects in quite different categories, and later the subject will use it to operate a single object at a time.

Administrator Case. In this scenarios, an administrator of a building requests a TKT for all the 408 lights and 68 alarms in the building. This TKT is a representative covering great amounts but limited categories of objects and will be used for bulk operation. E.g., he uses it to trigger all the alarms and turn on all the lights to evacuate people from the building when an emergency occurs.

Message Overhead. Fig. 9 (a) shows the length of ID- and attribute-based CMD for both cases. First, in Student Case, an ID-based CMD has 312 B while an attribute-based one has 346 B. The former is shorter because this case has only 8 objects, thus simply enumerating their IDs cost fewer bytes. Second, in Administrator Case, an ID-based CMD has 2,128 B while an attribute-based one needs only 256 B, which is 12% of the former. It is because Administrator Case has 476 objects but only 2 types (lights and alarms), thus 2 predicates are enough to specify all of them, reducing the length of an ID-based CMD by one order of magnitude. The length of attribute-based CMD only increases with the number of object categories, regardless of how many objects need to be covered.

10.2 Cryptographic Operation Time Cost

We test the computation time of operations related to public-key signatures and message authentication codes on subject devices, leader objects and member objects, using cryptographic libraries AndroidOpenSSL (default library of Android), Java Cryptography Architecture [13] (built-in piece of Java), and micro-ecc [14] (small and fast implementation of ECDSA/ECDH for 8-bit processors using C language) respectively. Exactly which signature algorithm and key size to pick is orthogonal to our design. We choose ECC in our implementation because compared with RSA, it offers similar security strength at a smaller key size [10], [15]. We choose ecliptic curve *secp224r1* because: i) it achieves high enough strength (112-bit, comparable to RSA 2048) with short computation time; ii) it is supported by the libraries across all the three platforms (Android, Pi and Arduino).

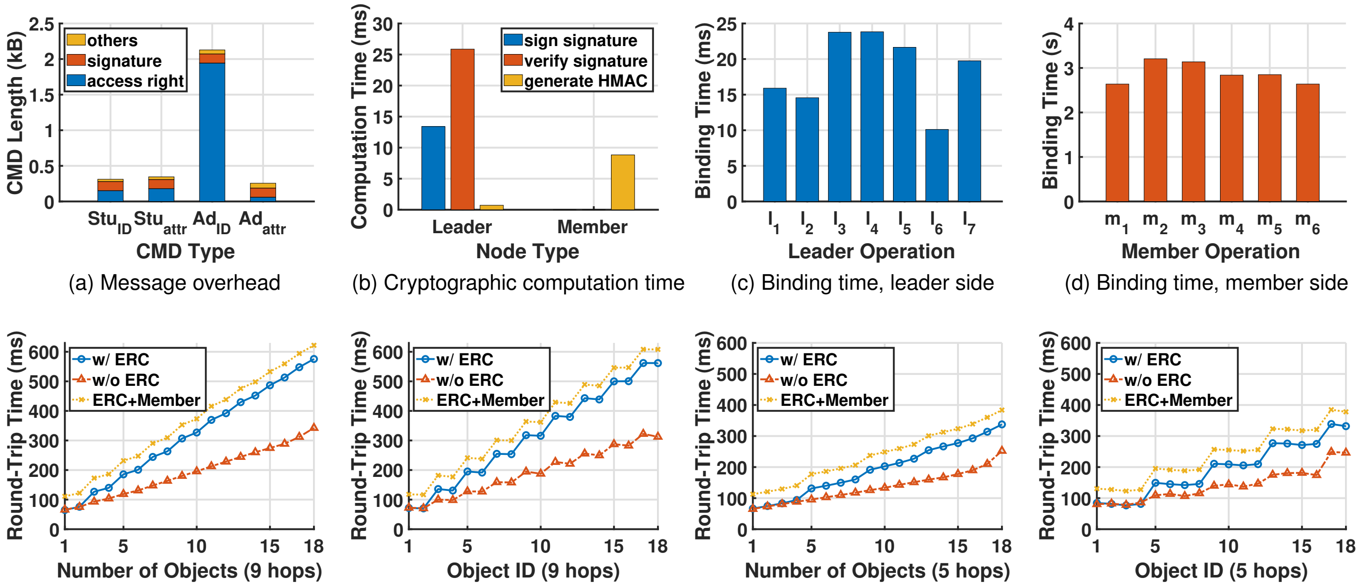
Signature. First, a subject device needs only about 2 ms for signature signing/verification, due to the device's rich computing resource. As for objects, as shown in Fig. 9 (b), on a leader object it takes 13.4 ms for signing and 25.9 ms for verification. On a member object signing/verification costs 2.8/3.2 seconds, which are two orders of magnitude larger than those on a leader, thus not presented in the same figure. So far, we confirm that such public-key operations, when performed on member objects, will lead to significant latency. Thus they should only be conducted occasionally on member objects (e.g., in binding, which happens at most a few times a day) instead of too frequently (e.g., command execution), to avoid sluggish user experience.

RSA vs. ECC. RSA 2048 and ECDSA 224 have comparable security strength, but the former is much less efficient in both computation and signature length: i) though RSA 2048 has fast signature verification (9.9 ms on a leader object), its signing time is 265.0 ms, about 20x as long as ECDSA 224; ii) an RSA 2048 signature has 256 B while ECDSA 224 needs only 56 B. We use ECDSA 224 in our implementation.

HMAC. On the other hand, hash-based message authentication code [16] (HMAC) is significantly less expensive and can be generated at high speed by even a resource-constrained member object, using 8.8 ms. And a leader object spends only 0.7 ms. Thus, it is a good choice to use signatures for authenticity/integrity protection on interactions between subject devices and leader objects, and use HMACs between leader and member objects. Heracles uses this strategy: a leader object verifies a CMD's signature signed by a subject, and sends its member an adapted CMD which uses an HMAC in place of the signature.

10.3 Binding Latency

We evaluate the time cost by a member to bind to a leader, and its composition. Binding latency is the time from the



(a) Message overhead (b) Cryptographic computation time (c) Binding time, leader side (d) Binding time, member side

(e) Execution latency, 9-hop (f) Impact of hop counts, 9-hop (g) Execution latency, 5-hop (h) Impact of hop counts, 5-hop

Fig. 9. Performance evaluation on message overhead, cryptographic computation time, binding latency, and command execution latency.

member's sending the first message (N_M) till both the member and the leader finish computing the shared secret using ECDH algorithm. The overall latency is 14.8 seconds.

Fig. 9 (c) (d) show the time cost of every time-consuming operations (either ECDSA or ECDH operations) in binding stage: 6 operations (m_1 to m_6) are performed by a member object, and 7 operations (l_1 to l_7) by a leader. Details can be found in Section 7. Again, we see that a leader operation costs at most 23.8 ms, while a member operation consumes at least 2.6 seconds and up to 3.2 seconds.

The overall binding latency mainly consists of: i) time cost of 5 public-key operations on the member T_m; ii) time cost of 2 public-key operations on the leader T_l; iii) transmission time of the three messages (95.6 ms). $T_m = \sum_{i=2}^{6} t(m_i) =$ 14,670.3 ms; $T_l = \sum_{i=1}^{2} t(l_i) =$ 30.5 ms. Note that m_1 does not contribute to the binding latency because the member can perform it in advance, and save the generated ECDH public parameter for later binding use. This reduces the binding time by 2.6 seconds at the expense of 56 B storage. Also, l_3 to l_7 (together costing 99.1 ms) do not contribute to the binding latency either because they are performed concurrently with m_6 (costing 2.6 seconds). Since m_6 is slow, it is completed much later than l_7, and it determines when the binding process is finished. We see that 99% of the binding latency comes from the member's public-key operations due to its constrained computing resource.

Such binding latency (14.8 seconds) is acceptable, since session keys and binding notifications are updated infrequently (usually once or a few times a day). Also, it is not performed on demand of subjects' commands, thus there is no stringent requirement of fast completion.

10.4 Command Execution Latency

We test the time difference from a CMD's issuing to its RES's returning and verification completion, for ID-based CMDs (Student Case, a single target) and attribute-based CMDs (Administrator Case, bulk operation). The experiments show that Heracles achieves responsive execution: a bulk operation CMD takes 0.57 second to control 18 objects which are scattered 1–9 hops away from a subject; mostly a target is nearby, i.e. 1 or 2 hops away, and execution of an ID-/attribute-based CMD on such an object costs only 0.07 second (1 hop) or 0.13 second (2 hops).

We deploy 18 leader objects and test two network topologies. In the first network, objects are 1–9 hops away from the subject device, with 2 objects at each hop (i.e., Object 1, 2 are 1-hop away, Object 3, 4 are 2-hop away, and so on). In the second network, objects are 1–5 hops away from the subject device: there are 4 objects at hop 1–4 respectively, and Object 17, 18 are 5 hops away from the subject.

The latency mainly results from: i) the subject device's signing a CMD and verifying the target's RES; ii) the target's verifying the CMD and signing an RES; iii) messages' transmission time which depends on background traffic intensity and can vary with environments and time. Besides, en-route check can be optionally enabled to make intermediate nodes between the subject device and the target object verify CMDs and RESs before forwarding them. This is a useful feature in help alleviate DoS attacks which keep flooding fake messages in the network (Section 8).

10.4.1 Subject-to-Leader Commands

As shown in Fig. 9 (e), in the 9-hop network, a bulk operation CMD has short latency no matter en-route check is enabled or not: when en-route check is on, it takes 575.6 ms to operate 18 objects; when it is off, it takes 342.4 ms. The former costs slightly longer because each intermediate node in the routing path verifies the signature of CMDs going towards the targets and the returning RESs before forwarding them. According to research on usability engineering [17], if response time is below 1 second, the user's flow of thought will stay uninterrupted. Thus our bulk operation achieves good responsiveness.

Fig. 9 (f) presents the impact of hop counts on latency: the ladder-shaped curves show that command execution on objects at the same hop cost similar time, and the time increases fairly linearly with hop counts. In the vast majority of cases, IoT users are controlling nearby objects which are 1 or 2 hops away: bulk operation execution on a 1-hop object needs only 70.3 ms no matter en-route check is on or off, because there is no forwarding node between the subject and 1-hop targets thus no en-route check performed; execution on a 2-hop object costs 130.6 ms when en-route check is on, and 98.5 ms when it is off. Such short latency has the user feel that the operation is completed instantaneously [17]. Controlling an object which is 9 hops away needs 561.7/312.4 ms when en-route check is on/off.

The 5-hop network test (Fig. 9 (g) (h)) shows similar trends. Both cases have responsive execution and cost shorter time than 9-hop: it costs 337.5 ms to finish bulk operations on 18 objects with en-route check, and 252.3 ms without en-route check. Execution on objects which are 5 hops away cost 331.8/246.4 ms when check is on/off.

Besides, for a target of the same hop count, an ID-based CMD for Student Case costs similar time as an attribute-based CMD for Administrator Case due to their similar message length (Fig. 9 (a)). So its figures are omitted here.

By comparing the results of this new testbed (18 objects, up to 9 hops) with those of our old one (5 objects, up to 3 hops) in [18], we observe a similar trend that the time cost increases linearly with hop count which is within 9. We do not explore the case of operating objects at larger hops because we believe it is rare, and we suggest in that case infrastructures should participate in such that hop-by-hop routing does not need to go beyond 9 hops. More discussion can be found in Section 12.

10.4.2 *Subject-to-Member Commands*

Strategy 1: Indirect Subject-to-Member Commands. A target may also be a member object. In this strategy it is associated with a leader beforehand: the leader verifies/signs the signatures of CMDs/RESs from/to the subject on behalf of the member, while the leader-member CMDs/RESs are secured using HMACs. The latency between a leader's issuing a CMD with HMAC to its member and the leader's receiving, verifying the returning RES is 46.2 ms. In detail, 1.4 ms is cost by the leader in generating HMAC for CMD and verifying HMAC of RES from the member; 17.7 ms is cost by the member in verifying HMAC of CMD from the leader and generating HMAC for RES. Message transmission costs 27 ms, about 59% of the overall latency.

Let's define an i-hop member object as a member whose leader is i-hop away from the subject ($1 \leq i \leq 9$), then the overall latency of operating it is always 46.2 ms larger than operating its leader. Recall that in Heracles, leader objects form the "backbone" for message forwarding and member objects are "leaves". Thus, a command targeting an i-hop member is realized with one targeting an i-hop leader object, followed by a 1-hop leader-to-member command which costs 46.2 ms. Fig. 9 (e) to (h) show that in the most time consuming case (i.e., control a member object when en-route check is enabled), it costs 621.8 ms to operate 18 objects in the 9-hop network, and 383.7 ms in the 5-hop network. Good enough responsiveness is still achieved.

Strategy 2: Direct Subject-to-Member Commands. If a member interacts directly with a subject device via CMDs/RESs secured by signatures, it does not need a leader. However, the execution latency would be significantly prolonged due to a member's poor computing performance. According to our experiments, a member needs 3.2 seconds to verify the signature in a subject CMD, and 2.8 seconds to sign the RES it generates. Thus, even operating a single 1-hop member object costs at least 6.0 seconds, about **10x** as long as controlling 18 leader objects spread in 9 hops using Strategy 1, let alone operating many multi-hop member targets. This strategy not only causes unacceptably long delay to users, but also quickly drains the energy of member objects most of which are battery-powered.

There is a symmetric-key alternative way for subject-member end-to-end protection: a member establishes a symmetric key with every subject it needs to interact with, and they use HMACs in their CMDs/RESs. In this way the member also does not need a leader, and has short execution latency at the same time. However, compared with Strategy 1 in which a member needs to establish a key with only its leader, this strategy has expensive key establishment and maintenance overhead, and is not fit for enterprise scales. Besides, en-route check is unavailable unless all the forwarding nodes also share the symmetric key.

11 Related Work

ACL and capability are two common forms of access control [12], and their differences in computer systems are analyzed in [19]. Access control policies include discretionary, mandatory, and role based access control. Attribute based access on encrypted data in cloud [20] is explored using attribute based encryption [21]. Our system adopts capability based access control for its efficient updating.

Exiting smart home products have mostly all-or-nothing access control [2], [3], [22] that family members can access everything and outsiders nothing. Recent work provides access control based on subject-object pairs using hierarchical data names [23], or extensions on time by abstracting smart objects as peripherals to a computer [24]. They are intended for traditional computer systems/cloud, targeting small scale homes, or providing coarse grained and basic ACL based access control. Our system achieves fine grained access control, which is necessary for enterprise environments where users and devices are both heterogeneous.

Many approaches [3], [4], [5] use centralized execution strategies for secure access, and all access must go through the cloud for enforcing authorization policies. They have weak availability: a machine/network failure results in total loss of access. This causes much more serious impact in enterprise environments than homes due to the former's *huge subject/object amounts* property (Section 2). Kerberos [25], which has been widely adopted by industry, realizes distributed authentication by granting parties tickets that prove their identities. It does not deal with access rights. We have tickets carry the requested authorizations, thus when the backend is unavailable, a subject can continue operating objects till the tickets expire (e.g., a few hours), hopefully by then the network/server failure has been resolved. Only the ticket request operation involves back-and-forth communication to the backend. Subsequent commands are sent directly to objects, which greatly improves responsiveness.

There are a few capability based IoT access control designs [26], [27], [28], but they lack deep justification proving capability's advantage over ACL at enterprise scale. We are the first one to perform quantitative analysis and comparison on ACL based systems and capability based systems under enterprise IoT contexts, and we prove that a capability based strategy significantly reduces the overall updating overhead and results in higher system scalability and stronger security. Also, those designs are ID-based while our system additionally supports attribute capability-based access control which achieves efficient bulk operation and smaller requesting overhead. E.g., to access a newly installed light, if ID-based, the subject has to request a new ticket that covers the light's ID; however, if she holds an attribute-based TKT specifying her rights of "operating lights", she does not need to request another ticket. Besides, none of the existing work offers complete design, implementation and evaluation as we do.

12 Discussion

Impact of Leader Update on Binding. When a leader object leaves, its members should get new leaders. As mentioned in Section 7, a member object can establish symmetric keys with multiple leaders beforehand, but is bound to one at a time. When the current leader fails, it can quickly switch to another which it shares a key with, without performing the slow key establishment process. Besides, we may limit the number of members that one leader can accept, and spread binding load more evenly among leader objects, restricting the binding updating overhead upon a leader's leaving.

Bulk Operation. We use a testbed of 18 objects, up to 9 hops to evaluate bulk operation. First, considering that mainstream radios (e.g., WiFi, Bluetooth, ZigBee) have reasonably far transmission distances (e.g., dozens of meters) while IoT commands mostly target objects around the user, e.g., within her building, 9 hops are more than enough. When commands occasionally do need to go further, hop-by-hop routing might be slow or fragile; in that case they can be sent via infrastructures (e.g., access points, cables) to the destinations or their vicinity, and then propagated among peers. In this way, hop-by-hop routing does not need to go beyond 9 hops. Second, in reality, there can be more objects at each hop, using broadcast can make them receive commands quickly regardless of the number of objects. Thus, we believe our evaluation is reasonable.

Bulk Operation Command Routing. A bulk operation CMD is usually propagated with a scope control mechanism to avoid blind flooding. One solution is to use filters based on object locations. E.g., a CMD with predicate $\{type = lamp \wedge floor = 2\}$ targets the objects on the second floor only, and an object should not forward it to objects out of the scope (e.g., Floor 1, 3). It is easy to realize if an object maintains the location information of its neighbors (e.g., location based names in data-centric networks [8]).

Confidentiality. Our design protects authenticity, integrity and freshness but does not ensure TKT/CMD content confidentiality. The content is not encrypted, thus adversaries may find out one's access rights, intended operations, which could be sensitive. Given that each subject/object has a public-private key pair, establishing symmetric keys to encrypt conversations is feasible. We leave the complete solution as future work.

Service Discovery. In the current system, subjects see the same profiles (PROF) of an object even though they have very different access rights to the object. If a PROF contains sensitive information (e.g., functions for VIPs' exclusive use), it should not be disclosed to subjects without appropriate levels. In the future we will make PROFs customized such that subjects discover different versions for the same object and gain only the knowledge allowed.

Physical Contact Attacks. An attacker may gain physical contact with objects, and launch attacks such as rebooting the target to purge its records of ID_{CMD}s and replay CMDs whose timestamps are still within time synchronization error e. To address this problem, we may require any object not to accept CMDs upon power up until e time later. Then the replayed CMDs will be rejected for their obsolete timestamps. A full solution to physical contact based attacks goes beyond the scope of this paper.

En-Route Check. Leader objects can conduct en-route check to alleviate denial-of-service attacks that flood large numbers of fake messages in the network. Under normal conditions when attacks do not happen, en-route check can be disabled to save computation, energy and time. If a target leader detects attacks, it may broadcast an alert message notifying other leader objects in vicinity to switch on en-route check, and with possible hints on what to check (e.g., TKT integrity, CMD freshness).

13 Conclusion

In this paper, we describe the design, implementation and evaluation of Heracles, which achieves fine-grained access control, efficient updating, and responsive command execution for enterprise scale IoT. Heracles uses secure, unforgeable tokens to describe the authorizations granted to a subject, which are used to access objects robustly and quickly, without involving the cloud. Besides, it supports responsive operations on resource-constrained objects and efficient bulk operations. Our analysis and performance evaluation show that it is secure, and has scalable updating (eliminates or reduces by 10x–100x the updating overhead in many situations), and responsive execution (0.07 s to access a 1-hop object; 0.57 s to access 18 multi-hop objects).

Acknowledgment

This work is supported in part by the National Science Foundation under grant numbers CCF 1652276, CNS 1513719 and CNS 1730291.

References

[1] J. Greenough and J. Camhi, "The Internet of Things: examining how the IoT will affect the world," BI Intelligent, Tech. Rep., 2015.

[2] B. Ur, J. Jung, and S. Schechter, "The current state of access control for smart devices in homes," in *Workshop on Home Usable Privacy and Security (HUPS)*, 2013.

[3] SmartThings, “SmartThings Classic Developer Documentation,” https://buildmedia.readthedocs.org/media/pdf/smartthings/latest/smartthings.pdf.

[4] Amazon, “AWS IoT Developer Guide,” https://docs.aws.amazon.com/iot/latest/developerguide/iot-dg.pdf.

[5] IBM, “Meet Watson: the platform for cognitive business,” http://www.ibm.com/watson/ .

[6] C. Perkins, E. Belding-Royer, and S. Das, “Ad hoc on-demand distance vector (aodv) routing,” Tech. Rep., 2003.

[7] A. Hoque, S. O. Amin, A. Alyyan, B. Zhang, L. Zhang, and L. Wang, “Nlsr: named-data link state routing protocol,” in *Proceedings of the 3rd ACM SIGCOMM workshop on Information-centric networking*. ACM, 2013, pp. 15–20.

[8] X. Song, Y. Huang, Q. Zhou, F. Ye, Y. Yang, and X. Li, “Content centric peer data sharing in pervasive edge computing environments,” in *Distributed Computing Systems (ICDCS), 2017 IEEE 37th International Conference on*. IEEE, 2017, pp. 287–297.

[9] Y. Huang, X. Song, F. Ye, Y. Yang, and X. Li, “Fair caching algorithms for peer data sharing in pervasive edge computing environments,” in *Distributed Computing Systems (ICDCS), 2017 IEEE 37th International Conference on*. IEEE, 2017, pp. 605–614.

[10] S. Blake-Wilson, B. Moeller, V. Gupta, C. Hawk, and N. Bolyard, “Elliptic curve cryptography (ecc) cipher suites for transport layer security (tls),” 2006.

[11] H. Krawczyk and P. Eronen, “Hmac-based extract-and-expand key derivation function (hkdf),” 2010.

[12] R. S. Sandhu and P. Samarati, “Access control: principle and practice,” *IEEE communications magazine*, vol. 32, no. 9, pp. 40–48, 1994.

[13] Oracle, “Java Cryptography Architecture Reference Guide,” https://docs.oracle.com/javase/7/docs/technotes/guides/security/crypto/CryptoSpec.html.

[14] Kenneth MacKay, “micro-ecc,” https://github.com/kmackay/micro-ecc.

[15] M. Qu, “Sec 2: Recommended elliptic curve domain parameters,” 1999.

[16] M. Bellare, R. Canetti, and H. Krawczyk, “Keying hash functions for message authentication,” in *Annual international cryptology conference*. Springer, 1996, pp. 1–15.

[17] J. Nielsen, *Usability engineering*. Elsevier, 1994.

[18] Q. Zhou, M. Elbadry, F. Ye, and Y. Yang, “Heracles: Scalable, fine-grained access control for internet-of-things in enterprise environments,” in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 2018, pp. 1772–1780.

[19] M. S. Miller, K.-P. Yee, J. Shapiro *et al.*, “Capability myths demolished,” Technical Report SRL2003-02, Johns Hopkins University Systems Research Laboratory, 2003. http://www. erights. org/elib/capability/duals, Tech. Rep., 2003.

[20] S. Yu, C. Wang, K. Ren, and W. Lou, “Achieving secure, scalable, and fine-grained data access control in cloud computing,” in *Infocom, 2010 proceedings IEEE*. Ieee, 2010, pp. 1–9.

[21] V. Goyal, O. Pandey, A. Sahai, and B. Waters, “Attribute-based encryption for fine-grained access control of encrypted data,” in *Proceedings of the 13th ACM conference on Computer and communications security*. Acm, 2006, pp. 89–98.

[22] Apple, “Homekit,” https://developer.apple.com/homekit/.

[23] W. Shang, Q. Ding, A. Marianantoni, J. Burke, and L. Zhang, “Securing building management systems using named data networking,” *IEEE Network*, vol. 28, no. 3, pp. 50–56, 2014.

[24] C. Dixon, R. Mahajan, S. Agarwal, A. Brush, B. Lee, S. Saroiu, and P. Bahl, “An operating system for the home,” in *Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, 2012, pp. 337–352.

[25] B. C. Neuman and T. Ts’o, “Kerberos: An authentication service for computer networks,” *IEEE Communications magazine*, vol. 32, no. 9, pp. 33–38, 1994.

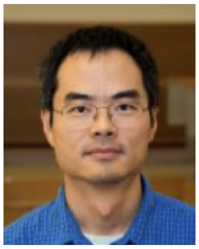
[26] J. L. Hernández-Ramos, A. J. Jara, L. Marin, and A. F. Skarmeta, “Distributed capability-based access control for the internet of things,” *Journal of Internet Services and Information Security (JISIS)*, vol. 3, no. 3/4, pp. 1–16, 2013.

[27] P. N. Mahalle, B. Anggorojati, N. R. Prasad, and R. Prasad, “Identity authentication and capability based access control (iacac) for the internet of things,” *Journal of Cyber Security and Mobility*, vol. 1, no. 4, pp. 309–348, 2013.

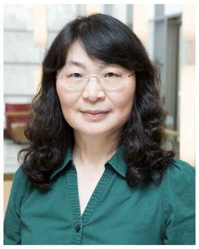
[28] S. Gusmeroli, S. Piccione, and D. Rotondi, “A capability-based security approach to manage access control in the internet of things,” *Mathematical and Computer Modelling*, vol. 58, no. 5, pp. 1189–1205, 2013.

Qian Zhou received the BE degree from Beihang University (previously known as Beijing University of Aeronautics and Astronautics). He is working toward the PhD degree in the ECE Department, Stony Brook University. His research mainly focuses on enterprise-scale Internet of Things (or cyber-physical systems), particularly in security & privacy and networking aspects.

Mohammed Elbadry received the BS and MS degrees from Stony Brook University. He is working toward the PhD degree in the ECE Department, Stony Brook University. His research interests include wireless networking and sensing, and IoT security.



Fan Ye received the BE and MS degrees from Tsinghua University, and the PhD degree from the Computer Science Department, UCLA. He is an associate professor with the ECE Department, Stony Brook University. He has published more than 90 peer reviewed papers that have received more than 12,000 citations according to Google Scholar. His research interests include mobile sensing systems, with applications in location based services and healthcare, Internet-of-Things, and wireless and sensor networks.



Yuanyuan Yang received the BEng and MS degrees in computer science and engineering from Tsinghua University, Beijing, China, and the MSE and PhD degrees in computer science from Johns Hopkins University, Baltimore, Maryland. She is currently a SUNY Distinguished Professor in the Department of Electrical & Computer Engineering and Department of Computer Science at Stony Brook University, New York, which she joined in 1999. She has been on leave since 2018 serving as a program director in the National Science Foundation’s Directorate of Computer and Information Science and Engineering (CISE). Her research interests include parallel/distributed computing, cloud computing, edge computing and mobile computing. She has published over 440 scientific papers in leading refereed journals and conferences. She is also an inventor/co-inventor of seven U.S. patents in the area of interconnection networks. She is currently the Editor-in-Chief for IEEE Transactions on Cloud Computing. She served as the Associate Editor-in-Chief for IEEE Transactions on Cloud Computing from 2016-2019, and the Associate Editor-in-Chief for IEEE Transactions on Computers from 2013-2014. She is currently an Associate Editor for ACM Computing Surveys and an Associate Editor for IEEE Transactions on Parallel and Distributed Systems. She served as an Associated Editor for IEEE Transactions on Computers from 2006-2012, IEEE Transactions on Parallel and Distributed Systems from 2001-2005 and Journal of Parallel and Distributed Computing from 2003-2017. She is an IEEE Fellow.