Heracles: Scalable, Fine-Grained Access Control for Internet-of-Things in Enterprise Environments

Qian Zhou*, Mohammed Elbadry†, Fan Ye* and Yuanyuan Yang*
*Department of Electrical and Computer Engineering, Stony Brook University
†Department of Computer Science, Stony Brook University
Email: {qian.zhou, mohammed.salah, fan.ye, yuanyuan.yang}@stonybrook.edu

***Abstract*—Scalable, fine-grained access control for Internet-of-Things is needed in enterprise environments, where thousands of subjects need to access possibly one to two orders of magnitude more objects. 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 that achieves robust, fine-grained access control at enterprise scale. Heracles adopts a capability-based approach using secure, unforgeable tokens that describe the authorizations of subjects, to either individual 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, and delegated operations for responsiveness of resource-constrained objects. Extensive security analysis and performance evaluation on a testbed prove that Heracles achieves robust, responsive, fine-grained access control in large scale enterprise environments.**

I. 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 one to two orders of magnitude more smart objects (e.g., a university campus with 100+ buildings each embedded with hundreds 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, 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). 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 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. The cloud will authenticate the subject and check that he has sufficient rights, then notify 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. 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 the 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, distributed execution at enterprise scale. Heracles adopts a capability-based approach where a subject requests secure, unforgeable tokens depicting his access rights to certain objects from the backend. Once the token is obtained, the access no longer involves the backend. The subject includes the token in his commands to the target object, which checks the token and commands, then executes 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 execution at enterprise scale.
- We compare with an alternative approach of ACL based distributed execution, and prove that capability is preferable in enterprise environments due to its higher efficiency and stronger security.
- We offer solutions to two desirable features in enterprise IoT: 1) an attribute-based access strategy for efficient *bulk operations* which control a category of objects using one command; 2) a delegation-based strategy to improve the responsiveness of resource-constrained objects.
- We implement our design, conduct real experiments in a testbed and thoroughly analyze its security to demonstrate its efficiency, scalability and security.

II. Models and Assumptions

Node role. The network consists of three kinds 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 and he 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., 2.7 GHz 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 constitute a *ground network*.

We assume the backend, subject devices and objects are roughly time synchronized (e.g., within tens of seconds). The backend is well protected, and subject devices are reasonably protected (e.g., with OS security mechanisms). We also assume the backend, subject devices and leader objects have enough computing resources to run public-key cryptography algorithms, while member objects may be able to run them only occasionally. Objects may have diverse communication interfaces, e.g., besides WiFi, 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 leaders, have enough energy (e.g., main-powered like light bulbs, wall outlets, surveillance cameras). We do not study energy-saving techniques (e.g., duty cycling) in this paper, but they can be applied orthogonally to battery-powered Things.

Network Scale. The network has an enterprise scale, which has three properties that home environments do not have:

- *Heterogeneous Node Property*: the subjects/objects may be classified into many (e.g., $\sim 10^2$) categories due to their different attributes thus access rights.
- *Huge Node Amount Property*: the subject/object amount is large (e.g., $10^3 \sim 10^4$ subjects, $10^4 \sim 10^5$ objects).
- *Huge Operation Amount Property*: the operation amount is large (e.g., $10^5 \sim 10^6$ operations per day).

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, its *discovery* becomes faster and more robust.

III. 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 Heterogeneous Node 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 quite heterogeneous in positions, thus 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. It is critical such that only legitimate parties can create valid messages to operate objects. *Freshness* is that messages received are generated recently; this prevents replay attacks where adversaries simply record and replay a previously transmitted legitimate message, easy to perform in wireless networks.

Centralized management. The editing of node profile and access right information should be conducted at a single point, including adding/removing a subject/object, or a *category* of subjects/objects sharing certain characteristics, and adding/removing/changing an access right. This centralized strategy makes the system easy to manage: one does not need to make changes in a large amount of nodes one by one.

Execution availability and responsiveness. If the backend is needed during command execution, a total loss of access can happen upon machine/network failures in/to the backend. Despite dedicated maintenance, such failures still occur occasionally in enterprise environments. We need distributed execution such that access is still available upon such failures. Also, the latency from command issuing by subjects to execution by objects should be small for positive user experience.

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.

IV. 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 VII.

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.

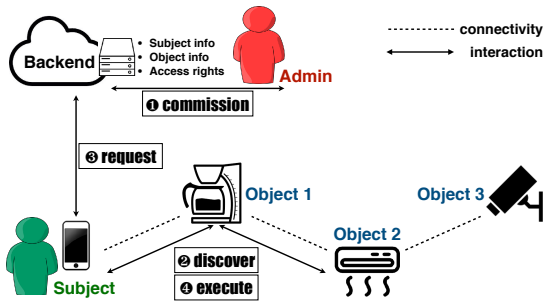


Fig. 1. The backend run by the administrator maintains the profiles and access rights of registered subjects and objects. A subject discovers objects around him (e.g., within 2 hops), requests a ticket covering the needed access rights, and sends a command to operate the target (e.g., air conditioner).

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 tokens he may use later to invoke certain functions on certain objects. The backend authenticates his REQ, examines the access right database, and issues a signed ticket (TKT) carrying the requested capabilities.

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 executes the invoked function; otherwise it rejects the CMD. A response (RES) is sent back to the subject.

V. 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 $item$s 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]).

A. 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 him 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 database. After loading such data into his devices, the subject publicizes his 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., /UniversityX/EngBldg/Floor2/Room217/Light1. Such names can optionally be used to route [7] a command to the target for command execution (Section V-D).

B. Discover

The subject device discovers nearby objects by querying their CERTs/PROFs. PROFs contain descriptions so both the human and his 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] can be chosen for their data acquisition speed and robustness.

C. Request

The subject sends a request (REQ) to the backend, asking for tokens he can use to invoke certain functions on certain objects. The backend authenticates his REQ, examines the access right database to ensure he does have those rights, and issues a signed ticket (TKT) carrying the requested capabilities.

ID-based and 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 a category of objects sharing certain characteristics (e.g., all lamps on Floor 2).

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

Fig. 2. Subject sends a REQ to the backend and gets a TKT.

S (Subject) sends a REQ (Fig. 2) including: 1) ID_S: a unique identity number of S; 2) O: the object(s) to which S requests his access rights, either an object set (specified by their identities ID_O) or an object category (specified by attribute predicates $Attr_O$); 3) F: a set of functions on O to which S requests his 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.

T is included for defending against replay attacks. Given the maximum time synchronization error e, the backend keeps the

hash codes of all 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 thus 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 a public-key signature generated by X for content in brackets. SIG_S and SIG_{Admin} protect 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 V-D) or ticket revocation (Section V-F) – without presenting the whole TKT. This improves efficiency and responsiveness. An access right stored in the backend 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 referencing and revoking all TKTs depicting a certain access right efficiently (Section VI).

D. 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 and attribute-based command. An ID-based CMD carries an ID-based TKT and targets a set of objects, while an attribute-based one carries an attribute-based TKT and targets a category of objects. An attribute-based CMD is used for a *bulk operation* (see details in Section VI).

$S \rightarrow Target : [ID_{CMD}, TKT, O, F, P, T]SIG_S$

$Target \rightarrow S : [ID_{CMD}, State, Data, T]SIG_{Target}$

Fig. 3. **S**ubject sends a CMD to the target and gets a RES.

S (Subject) 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 and 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) 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, and computing a hash value is no longer needed. As long as the time synchronization protocol can achieve a reasonable e (e.g., tens of seconds), the number of remembered ID_{CMD}s will not be many. SIG_S and SIG_{Target} protect the integrity of CMD and RES.

E. Comparison with existing work

Distributed execution. Our design is in contrast to cloud centric approaches [3], [4], [5] where the backend is needed in command execution. In such systems a machine/network failure results in total loss of access, and it has much more serious impact in enterprise environments than homes because of the former's Huge Operation Amount Property. E.g., in a university campus, even a one-hour network fault in one building would cause thousands of access fail. We have tickets carry the requested authorizations, thus 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 first ticket request involves back-and-forth communication to the backend. Subsequent commands are sent directly to objects, which greatly improves responsiveness.

Capability-based. Some existing work [10], [11] adopts distributed execution but is based on ACL. Others [12], [13], [14] use capability but lack insights on the tradeoffs with ACL. Here we prove capability is preferred to ACL in enterprise environments for its higher efficiency and security. Many times, a synchronization message must be sent to each affected object immediately after the administrator edits the backend database. The message may tell the object to add/remove certain access rights in its ACL (if ACL based), or to revoke certain tickets (if capability based), and we define *sync overhead* as the number of affected objects, which should be minimized to ensure fast convergence and compliance after such changes, otherwise denied or compromised access may happen.

Compared with ACL, capability is able to eliminate sync overhead in many cases, reduce overhead by one to two orders of magnitude, or at least keep comparable overhead in other cases. In contrast, most administrator operations lead to large sync overhead in an ACL system. Due to space limit, we briefly summarize that a capability one: 1) eliminates overhead in subject/object/access right addition. E.g., upon a subject who has access rights to N ($10^2 \sim 10^3$) objects is added to the database, all N objects in an ACL system need to be notified immediately and update their ACLs. While in a capability system, they do not need to do anything. The subject will discover available objects and request only access rights he has and is about to use on demand; 2) reduces overhead by one to two orders of magnitude in subject/ID-based access right removal. In ACL systems all affected objects must remove respective ACL entries, while in capability ones, only a small number of objects that have unexpired tickets containing removed rights must be notified, which is usually a small

fraction ($10^{-2} \sim 10^{-1}$); 3) keeps comparable overhead facing object/attribute-based access right removal.

A capability system is more efficient and secure due to its remarkably smaller sync overhead. In most cases ACL needs many more objects to be contacted by the backend within a short time. This inevitably leads to more failed or delayed updating, thus denied or compromised access: those for addition operations make subjects' authorized operations rejected, and those for removal operations make subjects' revoked operations accepted.

Local discovery. Some smart home products [15] rely on the backend to give the subject a list of all installed objects and provided functions. Huge Node Amount Property of enterprise environments makes it infeasible and unnecessary to know all the objects. Instead, the subject is interested in mostly those around him. Caching and discovery mechanisms (especially data-centric ones [8]) can find them out quickly and robustly.

F. Ticket Revocation

A subject may lose authorization he once had (e.g., being discharged, moved to different positions). Thus outstanding TKTs carrying unexpired access rights must be revoked.

To this end, the backend must keep all outstanding 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 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, see details in Section VI.

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

Fig. 4. The backend sends a REV (the 1st form) to **O**bject.

VI. Bulk Operations

A 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 devices in his lab when leaving work, or a manager uses one CMD to trigger all alarms in his building to notify people to evacuate, or a janitor turns off all 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.

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 neighbor 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. Thus 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 target IDs, but it is short, efficient, only when including small numbers of objects. 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, when a new object is added, a new TKT must be requested to include its ID. On the contrary, an attribute-based one has a fixed size and can be used to access new, previously unknown objects.

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 V-F) 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 revoke all TKTs carrying the access right.

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

Fig. 5. The backend sends a REV (the 2nd form) to **O**bject.

VII. Leader and Member Binding

Due to the abundance of medium or large objects 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 or energy intensive responsibilities (e.g., public-key cryptography, 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 register process at the backend as a leader one, 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.

$M \rightarrow L : N_M$

$L \rightarrow M : CERT_L; EXCH_L$

$M \rightarrow L : CERT_M; EXCH_M; BIND_M$

Fig. 6. **M**ember and **L**eader establish a shared secret and generate a BIND.

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 [16]. 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., ECDSA on curve *secp224r1* for signing), we reduce the number of messages to three, while generating BIND concurrently.

After receiving the member's nonce N_M, the leader generates an exchange message (EXCH): $[N_M, N_L, KM_L]SIG_L$, where N_L, KM_L denote the leader's nonce and key material (an ECDH public key). Then the member sends its EXCH, $[N_L, KM_M, ID_{BIND}]SIG_M$, together with a BIND_M (a BIND signed by M only). Both EXCHs are signed for protecting authenticity, integrity, and the sender's CERT is attached such that the receiver can verify the signature. N_M, N_L are used in challenge-response, for freshness. The leader and the member use each other's key material to compute the shared secret, and use a key derivation function (e.g., HKDF [17]) to convert the secret to a session key suitable for use in authentication. Besides, the leader signs the BIND_M to get a complete BIND. Member objects launch this handshake periodically (e.g., a few times a day) and 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 much more resources to conduct those compute and energy intensive work (i.e., verifying public-key signatures). The member only needs to verify MACs, which incurs much less time and energy.

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

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

Fig. 7. **L**eader sends an adapted CMD to **M**ember and gets a RES.

VIII. Security Analysis

We show in this section how our system reacts to possible attacks in all interactions but commissioning (it is assumed to be a secure out-of-band process). The system resists well to attacks from external adversaries that target authenticity, integrity, freshness. Besides, we discuss strategies to alleviate the harm of node compromise and DoS attacks which flood invalid messages, but complete solutions are out of the scope.

We classify attacks based on the malicious node's source, role and target: 1) source: the malicious node can be from *external*, or it is a once benign node in the network but now compromised (e.g., a smartphone is stolen or its private key gets leaked), which we call *internal* attacks; 2) role: the malicious node may behave as a subject device, leader object, member object; 3) target: the possible security properties to attack include authenticity, integrity, freshness, availability.

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 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 new TKTs, 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 checking* 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, 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.

IX. Experimental Evaluation

We have implemented our prototype including three components of Heracles: the backend, subject devices, leader objects. The backend program runs in a server machine. We use Google Nexus 6 (2.7 GHz CPU, 3 GB RAM) as subject devices, and deploy 5 leader objects in a large room to construct a group network with diameter of 3 hops, each leader object emulated by a Raspberry Pi 2 (900 MHz CPU, 1 GB RAM).

Different radios can be used, as long as network connectivity and routing exist. The subject device requests tickets (TKT) from the backend over TCP, while communicating with a one-hop leader object (for object discovery, command execution) over UDP unicast due to its lower overhead. As for interactions between leader objects, UDP unicast is used for delivering ID-based commands (CMD) and broadcast is used for attribute-based CMDs. All responses (RES) come back over unicast.

We evaluate signature operation time cost, TKT/CMD message overhead, command execution latency. Note that evaluation results like latency depend significantly on factors like the radio, routing protocol and cryptography algorithm/library chosen in implementation. Thus the performance should not be interpreted literally, but rather revealing the likely ranges or magnitudes. In fact, we are in progress of upgrading our testbed to the next version with less latency and overhead. We will briefly describe the efforts and some prelim results. Although our design targets enterprise IoT, the small testbed can still tell us the latency and message overhead for command execution. This is because most users will be controlling devices nearby, thus a CMD seldom travels more than a few hops. For rare cases of distant targets ($\sim$ 10 hops), the CMD can be routed via the backbone Internet for better resilience.

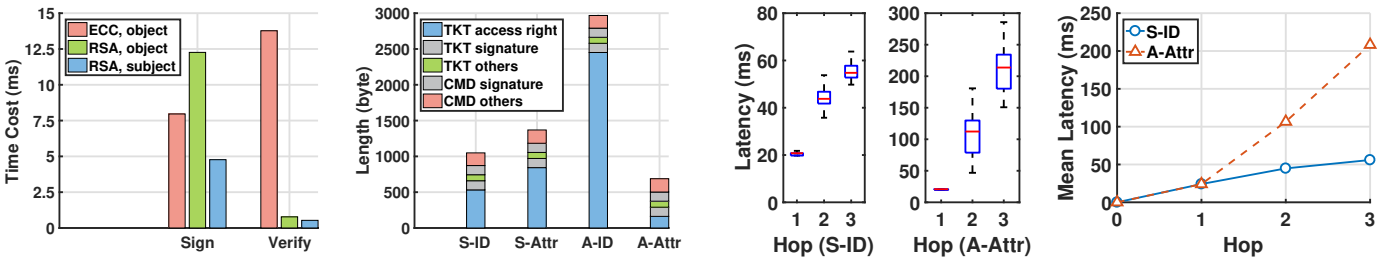
A. Signature operation time cost

We test RSA/ECDSA signing/verification time on leader objects and subject devices, using cryptography library Crypto++ [18] and AndroidOpenSSL respectively. Each test message is 1 KB, a common CMD length. Compared with RSA, ECC offers similar security at smaller key size [19]. Fig. 8 (a) shows the time cost of ECC160 and RSA1024, which have similar strength, and we notice RSA has acceptably slower signing but significantly faster verification than ECC. ECC160 on subject devices is not supported by the library thus not shown. Other libraries for Android are found less efficient. E.g, Spongy Castle [20] costs 51 ms for ECC160 verification.

Exactly which signature algorithm and key size to pick is orthogonal to our design, but in the testbed we choose RSA1024 for its fast verification. This feature is beneficial to en-route checking, where a CMD is signed once but verified for multiple times. Though RSA1024 leads to a 88-byte longer signature than ECC160, that extra overhead is less remarkable considering that a CMD is usually close to 1 KB or longer.

B. Ticket/Command message overhead

By comparing the length of TKTs/CMDs which are either ID- or attribute-based in two real cases, we prove the two are preferable in different scenarios: ID-based TKTs/CMDs are more efficient in scenarios with small amounts of objects in various categories, while attribute-based ones work better for bulk operations that target large amounts of objects in a few categories. The types and amounts of all objects below are from a field study on our engineering building on campus, which has two floors, with 32 offices/labs on the first, and 36



(a) Signature operation time cost (b) Message overhead (c) Execution latency (d) Mean execution latency

Fig. 8. **S-ID**: Student Case, ID-based; **S-Attr**: Student Case, attribute-based; **A-ID**: Admin Case, ID-based; **A-Attr**: Admin Case, attribute-based.

on the second. 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 are approximately 30 objects each room and 2040 objects in this building, excluding those in restrooms, lobbies or corridors.

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

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

Fig. 8 (b) shows the length of ID- and attribute-based TKT/CMD for both cases. $len(CMD_{Attr})/len(CMD_{ID})$ is 130% in Student Case and 23% in Admin Case. Except A-Attr, the access right (i.e. $O, \{F, C\}$, denoted as AR) part is the largest component in a TKT/CMD and it affects the overhead. $len(AR_{Attr})/len(AR_{ID})$ is 158%, 6% in Student, Admin Case. Student Case has only 8 objects, thus simply enumerating their IDs leads to a shorter O and AR. Admin Case has 476 objects but only 2 types (lights, alarms), and it is more efficient to describe O with attribute predicates.

C. Command execution latency

We test the time difference between a CMD's issuing and its RES's receiving for ID-based CMD in Student Case and attribute-based CMD in Admin Case. The latency mainly results from CMD signing (by subject device), CMD/RES transmission and RES signing (by target). Other time cost like signature verification, message encoding/decoding is short.

As is shown in Fig. 8 (c) (d), the execution latency of ID-based CMD increases fairly linearly with hop counts, while that of attribute-based CMD rises faster (but linearly) after the 1st hop. This is because UDP broadcast is used in implementation for attribute-based CMD propagation among objects, and it is slower than unicast used by subject-object communication (the 1st hop) and ID-based CMD propagation among objects. Also note that the time cost of an attribute-based CMD has larger fluctuation because a broadcast packet will be hold by the access point (AP) till the current Beacon Interval runs out and then forwarded. The latency can be reduced by setting the interval smaller. In Student Case, the access to a 3-hop target can be accomplished within 55 ms. In Admin Case, an attribute-based bulk operation costs about 208 ms (mainly due to the Beacon Interval) to operate all targets within 3 hops. Both cases have good responsiveness.

D. Testbed upgrade in progress

We briefly describe our testbed upgrading efforts for faster, more efficient cryptography libraries and networking protocols, and the details will be reported in our future work. 1) We expand the number of leader objects from 5 to 20, each emulated by a Raspberry Pi 3. Every subject device and leader object has simultaneous AP connectivity (to the Internet and backend) and WiFi Direct connectivity (for peer object discovery, command relay). In this way subject devices and leader objects can all interact directly, while being monitored and controlled by the backend simultaneously. 2) We implement member objects using Arduino Mega 2560 and apply a lighter-weight cryptography library micro-ecc [21]. A member object uses a low power radio (e.g., Bluetooth) to talk with a leader one, and it takes about 17 s for them to finish binding when using curve *secp224r1* for ECDH and ECDSA. The time cost is acceptable, since session keys and binding notifications are updated infrequently (usually once or a few times a day).

X. Related Work

ACL and capability are two common forms of access control [22], with their differences in computer systems analyzed in [23]. Access control policies include discretionary, mandatory, role based ones. Attribute based access on encrypted data in cloud [24] is explored using attribute based encryption [25].

Exiting smart home products have mostly all-or-nothing access control [2], [3], [15]. Recent work provides access control based on subject-object pairs using hierarchical data names [10], or extensions on time by abstracting smart objects as peripherals to a computer [26]. They are intended for traditional computer systems/cloud, targeting small scale homes, or providing coarse grained, basic ACL based access control.

Many approaches [3], [4], [5] use centralized execution strategies for secure access, and all access must go through the cloud for enforcing authorization policies, at the expense of weaker availability and responsiveness. Kerberos [27], 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.

There are a few capability-based IoT access control designs [12], [13], [14], but they lack deep justification proving capability's advantage over ACL at enterprise scale. Also, they do not support efficient bulk operations. Besides, none of them offers complete design, implementation and evaluation.

XI. Discussion

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 attribute predicate $\{type = lamp \wedge floor = 2\}$ targets the objects on Floor 2 only, and an object should not forward the CMD 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]).

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.

In the current system, subjects see the same PROF of an object even though they have very different access rights. 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.

Leader objects can conduct en-route checking to alleviate DoS attacks that flood fake messages. Under normal conditions when attacks do not happen, en-route checking can be disabled to save computation, energy and time. If a target leader detects attacks, it may send an alarm message notifying other leader objects in vicinity to switch on en-route checking.

XII. Conclusion

In this paper, we describe the design, implementation, evaluation of Heracles, which achieves efficient, robust, fine-grained access control for enterprise scale IoT. Heracles uses secure, unforgeable tokens to describe the authorizations granted to a subject. Besides, it supports responsive operations on resource-constrained objects and efficient bulk operations. Our analysis and performance evaluation prove its robustness, responsiveness and fine granularity in enterprise environments.

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 Developer Documentation," https://media.readthedocs.org/pdf/smartthings/latest/smartthings.pdf.

[4] Amazon, "AWS IoT Developer Guide," http://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] 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.

[11] N. Ye, Y. Zhu, R.-C. Wang, and Q.-m. Lin, "An efficient authentication and access control scheme for perception layer of internet of things," 2014.

[12] 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.

[13] 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.

[14] 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.

[15] Apple, "Homekit," https://developer.apple.com/homekit/.

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

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

[18] *Crypto++*, https://www.cryptopp.com.

[19] M. Qu, "Sec 2: Recommended elliptic curve domain parameters," 1999.

[20] *Spongy Castle*, http://rtyley.github.io/spongycastle/.

[21] *micro-ecc*, https://github.com/kmackay/micro-ecc.

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

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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.