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 are 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 more 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, 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 offer coarse-grained all-or-nothing access: the home owner has full access 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 (xx cite) 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 tens of thousands 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 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 authenticates the token and command, then executes invoked functions. We make the following contributions in this work:

- We design a 3-layer 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 on objects, convenient centralized policy management and robust, responsive distributed execution at enterprise scale.
- We compare with an alternative approach of ACL-based distributed exeuction, 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* that control a group 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 categories. The network consists of three categories 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., a 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 quad-core processor and tens of GB 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 and 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* while medium/large ones *leader objects*, and are assigned different responsibilities. Besides, a *target* is the object that a subject wants 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). We also assume the backend is well protected, and subject devices are reasonably protected (e.g., with OS security mechanisms). 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. 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).

We assume objects are largely static once installed. Thus the topology of the object network is stable except occasional deployment changes such as addition/removal of objects. A subject device is 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, coffee makers). Although we do not study duty cycling techniques [1] in this paper, they can be applied orthogonally to save energy for battery powered Things.

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

- *Heterogeneous Node Property*: subjects/objects may be classified into many (e.g., $\sim 10^2$) groups 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/discovery mechanism like PDS [2] exists. Independent data entities, those (e.g., certificates) protected by public key signatures, are widely propagated and *cached* in the network. Due to multiple copies cached in different nodes, its *discovery* becomes faster and more robust compared to always retrieved from the backend.

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 which functions with what parameters. This comes from Heterogeneous Node Property. Coarse-grained all-or-nothing access control works fine for homes, where family members have full access rights and strangers have 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. This is necessary such that it is indeed the subject that is authorized to invoke respective functions, and indeed on targeted objects. *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 send again a previously transmitted legitimate message, easy to perform in wireless networks.

Centralized management. Editing of node profile and access right information should be conducted at a single point, which includes adding/removing a subject/object, creating/deleting a *subject/object group* in which subjects/objects share 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 huge 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 when there are machine/network failures in/to the backend. Despite dedicated maintenance, such failures can still occur occasionally in enterprise environments. Therefore, 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 denial-of-service attacks and node compromise, but complete solutions are out of the scope. Neither are attacks targeting routing or confidentiality/privacy, or trust management. Besides, "scene" operations, which invoke a different command to each object in a collection simultaneously, is not among the goals. [xx wonder we may discuss how we can support scene easily if needed - I think that's true. then we can remove this sentence.]

IV. System Overview

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

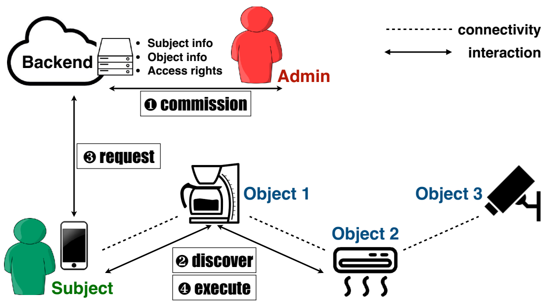


Fig. 1. The backend run by the administrator maintains the profiles and access rights of registered subjects/objects. A subject discovers objects around him (e.g., within 2 hops), requests a ticket covering the needed access rights, and sends a command to operate the target (e.g., the air conditioner).

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 [2] 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 requested capabilities.

4) Execute. The subject sends a signed command (CMD) for operating the target. The CMD carries a TKT containing required capabilities. It may be forwarded towards the target by multiple objects. The target verifies the CMD is legitimate and executes the invoked function; otherwise it rejects the CMD. In both cases 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 profiles stating the attributes of every registered subject/object, and subjects' access rights to objects. An access right is fine-grained in constraints on legitimate functions, parameters, contexts, etc. (xx wonder havent we talked about this in those 4 steps, what's new?)

Fine-grained access constraints. Typical constraints between a subject/object pair include valid functions, parameters, time ranges, invocation counts, etc. Given the same object, different subjects may be allowed for different functions, or different parameters, time ranges 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 door locks 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]).

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), and a signed profile (PROF). The backend's public key (K^{pub}_{Admin}) is also given. 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 [][] (xx also cite yaodong's caching work) by other nodes.

An object follows similar process. Its PROF describes: i) *who*: information like ID, human-readable name, category (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 $1 - 100$, and "*set_color*", with three integers each in $0 - 255$ for R/G/B.

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 for routing a command to the target later (see Section ... xx finish and make this clear, could cite exiting work).

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 [][] 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 an object by enumerating its ID, while an attribute-based one uses *attribute predicates* to describe 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 S 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 (specified by an object identity ID_O) or an object category (specified by an attribute predicate $Attr_O$); 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 the REQ's freshness.

Note that F, C and $LIFE$ can be optional in a REQ. The backend can decide what functions, constraints and lifetime to include in the TKT based on certain policy rules (e.g., granting all allowed access rights).

T is included for defending against replay attacks. The backend will accept a REQ only if its local clock and T are within some known maximum time synchronization error (e.g., t seconds). The backend needs to remember the timestamps received within recent time window t to reject those replayed. [...]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 altered or forged.

Every TKT has an identity ID_{TKT} such that it can be referenced later in command execution (see Section V-D) or ticket revocation (see Section V-F) – without presenting the whole TKT. This improves efficiency and responsiveness.

ID_{AR} is the identity of the access right stored in the backend and based on which this TKT was generated. 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 carrying a certain access right (see Section VI) efficiently. (xx need to explain the format of ID_{AR})

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 (xx cite ip/data centric ones). 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 single object, while an attribute-based one carries an attribute-based TKT and targets a group of objects. An attribute-based command is used for a *bulk operation* (see details in Section VI).

S (Subject) sends a CMD (Fig. 3) including: 1) ID_{CMD}: a unique identification number of the CMD; 2) O: the target, expressed as either ID_O or $Attr_O$; 3) F, P: functions and parameters that S attempts to invoke on O; 4) TKT: the ticket (see Fig. 2) proving the authority of S to invoke F, P on O; 5) T: a timestamp for the CMD's freshness. The CMD is signed by S to prove the authenticity.

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

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

Fig. 3. Subject sends a CMD to the target and gets a RES.

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 the CMD is attribute-based, and recall that an object knows its attributes from its PROF) with the CMD's O. The command execution is asynchronous such that 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 granted by TKT to pass authorization check conducted by the target. SIG_S and SIG_{Target} protect the integrity of CMD and RES so they cannot be altered or forged.

The freshness is ensured by both T and ID_{CMD}. Given the maximum time synchronization error e (xx make symbol consistent, t was used before?), a leader keeps all ID_{CMD}s it has received in recent time window e. A CMD is considered fresh if the difference between T and the local time is less than e, and its ID was not seen in recent time window. As long as the time synchronization protocol can achieve a reasonable e (e.g., a few minutes (xx you said tens of seconds before - make it consistent)), the number of remembered ID_{CMD}s will not be too many. Other mechanisms for freshness include: challenge-response, which requires two rounds handshake thus significantly increasing the latency; monotonic counters, which require a counter for each pair of subject-object, and are much easier to predict compared to nonces. Thus we choose the combination of timestamp and random command ID (which effectively serves as a nonce).

E. Comparison with existing work

Distributed execution. Our design is in contrast to cloud centric approaches [3] 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 due to 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 command executions fail. The tickets carry requested authorizations, thus a subject can continue to operate objects until the expiration (e.g., a few hours) of tickets, hopefully by then the network or server failure has been resolved. Only the first ticket request involves back-and-forth communication to the backend. Subsequent commands are sent directly to object, thus also greatly reducing the latency and improving responsiveness.

Capability-based. Some existing work [][] adopts distributed execution but is based on ACL. Others [4], [5] use capability but lack insights on the tradeoffs with ACL. Here we prove capability is preferred to ACL in enterprise environments for its better scalability and security. Many times, a synchronization message must be sent to each affected object immediately after the administrator changes the backend database (including add/remove subject/object/access right). The message may tell the object to add/remove certain access rights in its ACL (in ACL systems), or to revoke certain credentials (mostly (xx mostly not only?) in capability systems), 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 or 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 requests only access rights he does have and is about to use on demand; 2) reduces overhead by one or 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 TKTs 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/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 [6] 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) are effective to find them out efficiently, quickly and robustly.

F. Ticket Revocation

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

To this end, the backend must keep all outstanding tickets 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 tickets to be revoked. The REV is publicized and widely cached among nodes. Objects will add the IDs, expiration times of revoked tickets to their local ticket revocation lists (TRL). Upon expiration (actually slightly later, at least e after expiration) a revoked ticket's ID will be removed from the TRL. To avoid whole-network propagation of a REV affecting only a few tickets and objects, the backend may send the REV to those objects and their vicinity only. Any command referencing a ticket whose ID is in the TRL will become invalid. The second form is for efficiently revoking all attribute-based tickets carrying a certain access right, which will be explained in Section VI.

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

Fig. 4. Revocation message (the 1st form)

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) referred 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 combine multiple primitive predicates in logic AND. We implement this design and the support for other forms can be added if necessary. E.g., "all the windows in Room 217" can be expressed by $\{type = window \wedge room = 217\}$.

A bulk operation command 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 targets objects are spread over large areas (e.g., remote access of objects in another building), hop-by-hop routing may be slow or even unavailable. Thus the command can be sent via the backend directly to the destination or its vicinity, 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 an attribute-based one has smaller message overhead when the number of targets is large. ID-based TKTs/CMDs are easy to implement and TKTs are short when small numbers of objects are included. However, it cannot handle large numbers of objects efficiently. Since the size grows linearly as more object IDs are enumerated, the TKT may become too large, incurring large overhead and long latency in operation. 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 (see 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 larger (xx give some figures, how large?) because the access right may have been requested by many subjects in a category, thus enumerating ID_{TKT}s is inefficient. In this case the ID of the access right (ID_{AR}) is referenced to revoke all TKTs carrying it. (xx as commented before, how ID_{AR} works is not clear)

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

Fig. 5. Revocation message (the 2nd form)

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 among nodes are as follows:

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 chooses one or multiple as its pre-leader(s) (e.g., based on RSSI) and starts to establish a shared secret key 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 (CMD) 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 this BIND's identity, leader identity, member identity, this 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 object, then sent to and signed by the leader. This nested double signing prevents a leader or member from unilaterally publicizing a forged bilateral relationship.

$M \rightarrow L : N_M$

$L \rightarrow M : CERT_L; EXCH_L$

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

Fig. 6. Member and Leader 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 (xx cite). ECC-based TLS supports multiple key exchange algorithms, with many parameters configurable, including elliptic curves, point formats, etc. By fixing the key exchange algorithm at ephemeral ECDH (ECDHE) and other parameters (e.g., signature algorithm at ECDSA on curve *secp224r1*), we are able to 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 for computing the shared secret). Then the member sends an EXCH with the format $[N_L, KM_M, ID_{BIND}]SIG_M$, delivering the member's key material. Both EXCHs are signed for protecting authenticity and 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. (xx you didnt explain how BIND is created in 3rd step)

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

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

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

Fig. 7. Leader sends an adapted CMD to the member and gets a RES.

Execute. When a leader receives a 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 protected by a message authentication code (MAC) generated from their session key. (xx what's the relation bw the first shared secret and session key? how're session keys generated and updated? cite existing work if same) This CMD includes the same ID_{CMD}, F, P, T. The MAC ensures authenticity and integrity, and the freshness check is done similarly. (xx we assume members have clocks as well?) 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.

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 availability (e.g., jamming) attacks, but complete solutions are out of the scope. Attacks targeting routing or confidentiality/privacy are also 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 and public key certificate (CERT) assigned by the administrator, and will not be able to 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, thus 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 valid private key, thus signatures. A replayed request (REQ) will also fail because of timestamp's protection (xx and backend remember past REQs in e?). A malicious internal subject device can launch attacks targeting authenticity, integrity, freshness. (xx exactly what? need to be more concrete) We may use extra mechanisms (e.g., operation behavior analysis) on the backend to detect a compromised subject device. 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. Even so, 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 (i.e., the leader objects forwarding CMDs to the target, originally such checks are conducted by the target only). Thus an invalid CMD will be dropped before it travels far, effectively reducing the attacking scope.

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 availability attacks, the malicious leader object may send a large number 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. An external member object will fail in making its forged/altered/replayed responses (RES) accepted by a leader object for the same reason. Note that usually a leader object has sufficient energy from wired power supply and does not have the dead battery problem as a member one, 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. Of course the attacker may try requesting more TKTs, but we discussed the countermeasures that hamper it. (xx where? maybe repeat briefly)

If an leader object gets compromised, all of its member objects 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.

(xx need to say in general about resource exhaustion attacks: dumping many message to waster resources, thus DOS legitimate ones. also physical level jamming we dont consider.)

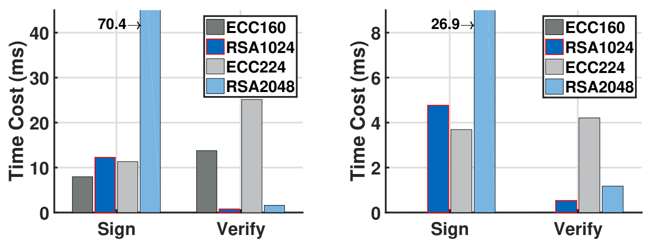
IX. Experimental Evaluation

We have implemented a prototype including three components of Heracles: the backend, subject devices, leader objects. (xx no member?) The backend program runs in a server machine. We use Google Nexus 6 (2.7 GHz quad-core CPU, 3 GB RAM) as subject devices. Three leader objects are deployed in a large room, each emulated by a Raspberry Pi 2 (900 MHz quad-core CPU, 1 GB RAM). (xx onlly 3? too small - can we say more? also need to overview how large scale aspects evaluated)

Different radios can be used, as long as network connectivity and routing exist. In the testbed we choose WiFi and connect every node to the same access point (AP). 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 the following aspects: 1) the time cost of signature signing/verification in different platforms. 2) the impact of ID-based strategy and attribute-based strategy on the efficiency in terms of message overhead in different usage scenarios. We present two scenarios based on a real building on our campus. 3) the impact of the two strategies on the latency from a CMD issuing to the RES reception.

A. *Signature Operation Time Cost*



(a) Crypto++ on Raspberry Pi (b) JCA on Android Phone

Fig. 8. Signature signing/verification time cost on two platforms

We compare the time cost of RSA/ECDSA signature signing/verification on both subject devices and leader objects. Compared with RSA, ECC offers similar security with smaller key sizes. E.g, ECC160 is close to RSA1024, and ECC224 is close to RSA2048. First, we use Crypto++ [7] on Pi (Fig. 8 (a)), and each message to be signed is 1KB, a common CMD length. For both ECC cases, verification consumes about twice as long as signing. RSA1024 has signing time similar to ECC, while RSA2048 costs too long. Also, we notice RSA verifies a signature extremely fast (0.7ms for 1024, 1.5 ms for 2048). Second, Java Cryptography Architecture (JCA) is used on Nexus 6 (Fig. 8 (b)). The result of ECC160 is not shown because the default Android Studio lib (AndroidOpenSSL) supports elliptic curves at least 224-bit long. We have tried other libs but they are inefficient, e.g, Spongy Castle (Bouncy Castle for Android) needs 50.7ms for ECC160 verification. Again, we find RSA1024 has comparable performance to ECC224 in signing but an advantage in verification (0.5ms).

Exactly which signature algorithm 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 check, where a CMD is signed once but verified for multiple times. Though RSA1024 leads to a 88-byte longer signature than ECC160, that overhead is not remarkable in 1KB-long messages.

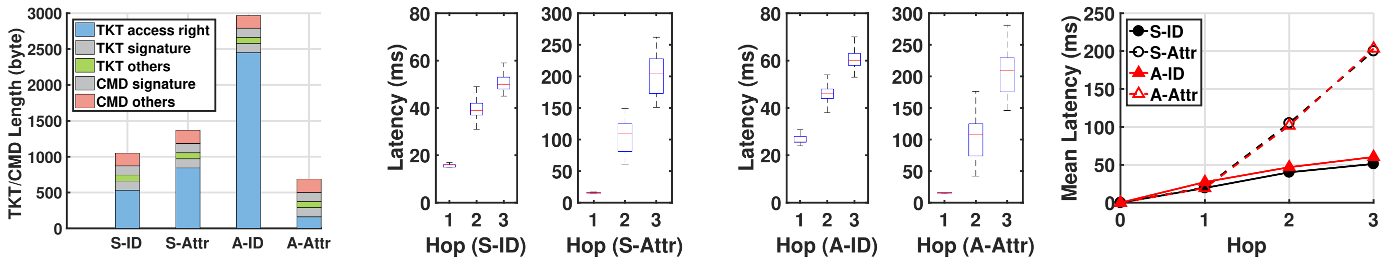
B. *Ticket/Command Message Overhead*

By comparing the length of TKTs/CMDs which are either ID- or attribute-based in two real cases (Student Case, Admin Case), 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 types, while attribute-based ones work better for bulk operations that target large amounts of objects in a few types. The types and amounts of all objects below are from a field study on an engineering building on our campus. The building has two floors, with 32 offices/labs in the first, and 36 in the second. A medium office/lab is used as a representative which has 6 lights, 8 lamps, 5 computers, 1 to 2 doors, 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. (xx is it one TKT or one each object? In design the ID based TKT is for one object only. you may say he requests one TKT per object) A graduate student requests a TKT in the morning which covers his access rights to certain objects installed in his lab, for only the functions he knows he will probably use this day. There are 8 objects included: 2 lights, 2 lamps, 1 door, window, coffee maker, air conditioner. This TKT is a representative covering a small amount of objects which however are in quite different types, and later the subject will usually use it to operate a single object at a time.

Admin Case. A building/campus administrator requests a TKT carrying his access rights to all 408 lights and 68 alarms in this building. This TKT is a representative covering a great amount but very limited types of objects and will be used to invoke bulk operations. For example, the administrator uses it to trigger all alarms in the building when an emergency occurs, and also to turn off all lights in rooms to force people to put their work aside and leave this building for safety.

Fig. 9 (a) (xx see if you can use different patterns in place of colors so it's visible when printed b/w) shows the length of ID- and attribute-based TKT/CMD for Student Case and Admin Case respectively. (xx the first points should be relative lengths of overall message, and student for ID, and admin using attr. Then you go to details like AR parts) Except A-Attr (Admin Case, attribute-based), the access right (AR) part is always the largest component in a TKT/CMD and it affects the overhead. In Student Case, an ID-based TKT/CMD has a shorter AR: $len(AR_{Attr})/len(AR_{ID}) = 158\%$ because the objects do not share characteristics much, thus using attribute predicates to describe them is less efficient than simply enumerating their IDs (4 bytes for each object in our implementation). In Admin Case, however, an attribute-based TKT/CMD is more efficient due to its much shorter AR: $len(AR_{Attr})/len(AR_{ID}) = 6\%$. It is because the administrator attempts to operate two object



(a) Message Overhead Comparison (b) Execution Latency, Student Case (c) Execution Latency, Admin Case (d) Mean Latency Comparsion

Fig. 9. **S-ID**: Student Case, ID-based; **S-Attr**: Student Case, attribute-based; **A-ID**: Admin Case, ID-based; **A-Attr**: Admin Case, attribute-based.

types (lights, alarms) only, and each can be specified succinctly with attribute predicates. An ID-based TKT has to enumerate the IDs of 476 objects.

C. Command Execution Latency

We test the time difference between a CMD's sending and its RES's receiving, with both strategies on both cases involved. In our lab environment, the latency mainly results from the transmission time of CMD/RES in each hop and the target leader's signing RES (done only once, 12.2ms). Other time cost like encoding/decoding a message is negligibly short.

Fig. 9 (b) (d) show the impact of hop counts on the execution latency of Student Case. The 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 for attribute-based CMD propagation among objects, and it is slower than unicast, which is 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 (around 100 ms) than an ID-based one, which is because a broadcast packet will be hold by the AP till the current Beacon Interval runs out and then forwarded. The latency can be reduced by setting the Beacon Interval smaller. Furthermore, Fig. 9 (c) (d) show Admin Case and give similar results. Its ID-based CMD costs slightly longer because the message is almost 3 times as large as the one in Student Case. (xx c shows admin ID is much faster - need to argue why still attr is prefered)

In Student Case, the access to a target 3-hop away using an ID-based CMD can be accomplished within 51ms, and 3-hop covers the objects a subject is likely to operate in most cases. In Admin Case, an attribute-based bulk operation costs about 200 ms (mainly due to the Beacon Interval) to control 3-hop targets, which is still reasonable.

(xx need to say a few things: the networking part is very dependent on which routing connectivity mechanism we use. thus latency numbers are to be interpreted as magnitude or range, not exact value; there're many further network optimization we can do, e.g., we're building a peer routing protocol using dual wifi direct and AP connection, and it will reduce the latency) We can extend our 3-hop experimental results to estimate the latency of operating objects further, e.g., 9-hop should cover a normal building and it needs about 800ms.

(xx also need to say large scale: we only have 3 nodes and claim enterprise scale, this is odd. can we add some simulation/calculation numbers?)

X. Related Work

ACL and capability are two common forms of access control matrix [8], with their differences in computer systems analyzed in [][]. Access control policies include discretionary, mandatory, role based ones. Attribute based access on encrypted data in cloud [9] is explored using attribute based encryption [10].

Exiting smart home products have mostly coarse grained all-or-nothing access [11], [3]. Recent work provides access control based on subject-object pairs using hierarchical data names [12], or extensions on time by abstracting smart objects as peripherals to a computer [13]. They intend for traditional computer systems/cloud, targets small scale homes, or provides coarse grained, basic ACL based access control.

Many approaches [3], [14], [15] use centralized execution strategies to secure access, and all access must go through the cloud server for enforcing authorization policies, at the expense of weaker availability and responsiveness. Kerberos [16], which has been widely adopted by industry, realizes distributed authentication by granting parties tickets that prove their identities. It does not cover distributed execution.

Existing capability-based access control solutions [5], [4] lack deep justification proving capability's advantage over ACL in enterprise-scale IoT. Also, they support no efficient bulk operations. Ye Ning et al. [17] propose an approach that a command will be accepted iff its subject and target share enough attributes, which is quite limited compared with those allowing a subject to specify the target with a compound attribute predicate. Besides, none of these work offers thorough design, implementation or evaluation.

XI. Discussion

A bulk operation CMD is usually propagated with a scope control mechanism than blind flooding, and one solution is to use filters based on object locations. E.g, a CMD with attribute predicate $\{type = light \wedge floor = 2\}$ targets the objects in Floor 2 only, and an object should not forward the CMD to objects out of the scope (e.g., Floor 1, 3). This is easy to realize if an object maintains the location-based hierarchical names of its neighbors, which is the case in data centric networks.

Our design does not in particular ensure TKT/CMD confidentiality. Though authenticity, integrity, freshness are protected, the content itself is not encrypted, thus adversaries may find out one's access rights, history operations, which may be sensitive. Given that each subject/object has a public-private key pair, establishing symmetric keys to encrypt conversations are 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 the subjects not of the appropriate level. In the future we will make PROFs customized such that subjects discover different versions for the same object and gain only the knowledge allowed.

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.

Leader objects can conduct en-route checking to find and drop invalid CMDs. This alleviates DoS attacks that flood large numbers of 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 leaders in vicinity to switch on en-route checking, with possible hints on what to check (e.g., TKT/CMD integrity/freshness).

REFERENCES

[1] C. M. Vigorito, D. Ganesan, and A. G. Barto, "Adaptive control of duty cycling in energy-harvesting wireless sensor networks," in *Sensor, Mesh and Ad Hoc Communications and Networks, 2007. SECON'07. 4th Annual IEEE Communications Society Conference on*. IEEE, 2007, pp. 21–30.

[2] 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.

[3] Smartthings, "SmartThings Developer Documentation," https://media.readthedocs.org/pdf/smartthings/latest/smartthings.pdf.

[4] 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.

[5] 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.

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

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

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

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] Amazon, "AWS IoT Developer Guide," http://docs.aws.amazon.com/iot/latest/developerguide/iot-dg.pdf.

[15] IBM, "Meet Watson: the platform for cognitive business," http://www.ibm.com/watson/ .

[16] 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.

[17] 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.