Argus: Multi-Level Service Visibility Scoping for Internet-of-Things in Enterprise Environments

Qian Zhou
Electrical and Computer Engineering
Stony Brook University
qian.zhou@stonybrook.edu

Omkant Pandey
Computer Science
Stony Brook University
omkant@cs.stonybrook.edu

Fan Ye
Electrical and Computer Engineering
Stony Brook University
fan.ye@stonybrook.edu

***Abstract*—In IoT, what services from which nearby devices are available, must be discovered by a user's device (e.g., smartphone) before she can issue commands to access them. Service visibility scoping in large scale, heterogeneous enterprise environments has multiple unique features, e.g., proximity based interactions, differentiated visibility according to device natures and user attributes, frequent user churns thus revocation. They render existing solutions completely insufficient. We propose Argus, a distributed algorithm offering three-level, fine-grained visibility scoping in parallel: i) Level 1 public visibility where services are identically visible to everyone; ii) Level 2 differentiated visibility where service visibility depends on users' non-sensitive attributes; iii) Level 3 covert visibility where service visibility depends on users' sensitive attributes that are never explicitly disclosed. Extensive analysis and experiments show that: i) Argus is secure; ii) its Level 2 is 10x as scalable and computationally efficient as work using Attribute-based Encryption, Level 3 is 10x as efficient as work using Paring-based Cryptography; iii) it is fast and agile for satisfactory user experience, costing 0.25 s to discover 20 Level 1 devices, and 0.63 s for Level 2 or Level 3 devices.**

I. Introduction

In IoT, what services from which nearby devices are available, must be discovered by a user's device (e.g., smartphone) before she can issue commands to access them [1]. The visibility of services must be "scoped" to authorized users only, i.e., only the authorized service subset/variant should be "visible" to the user device. Service visibility scoping in large scale, heterogeneous enterprise environments has multiple unique features, which render existing solutions [2], [3], [4], [5], [6], [7] significantly ineffective or completely insufficient.

First, IoT interactions via short-range radios are largely physical proximity based. A user mostly discovers and controls nearby devices, such that desired physical changes would occur in the local environment (e.g., door unlocking, higher ambient brightness, lower room temperature, louder music volume in the current room). A centralized server does not know which devices are around the user device; accurate user location requires more complexity in localization capability. As a result, solutions [2], [3], [4] using centralized servers are unfit for such proximity based discovery.

Second, multiple services of different natures usually exist around a user, and all of them need to be discovered. Imagine a university or corporate campus, thousands of users interact with ten times or more devices/services: 1) public utilities (e.g., thermometers in aisles) may be discovered to everyone, even visitors; 2) yet most devices should be conditionally disclosed according to users' non-sensitive attributes (e.g., expensive multimedia devices, safes in an office should only be visible to employees); 3) still there are services customized for specific populations of sensitive attributes that should be discovered with privacy preservation (e.g., a vending machine dispenses counseling/psychological support flyers, hidden within regular magazines to students of needs). Existing distributed schemes [5], [6], [7] are restricted to one type, and unfit for enterprise IoT with numerous, heterogeneous services.

In this paper, we propose *Argus*, a distributed, proximity-based IoT service discovery algorithm that offers three levels (**public**, **differentiated**, **covert**) of visibility in parallel. Each level is built on top of its prior level with minimum extension, and together they achieve efficient, harmonious, concurrent discoveries covering IoT services of different natures. Also, it minimizes the huge overhead resulting from authorization updating (e.g., user addition/revocation), which is the scalability bottleneck in this context, and fits well with an enterprise scale. We claim our contributions as follows:

- We identify unique requirements of visibility scoping in enterprise IoT, and design a distributed 3-in-1 algorithm for scalable, concurrent, fast service discovery at three levels: **Level 1: public visibility** where services are identically visible to everyone; **Level 2: differentiated visibility** where service visibility depends on users' *non-sensitive attributes*; and **Level 3: covert visibility** where visibility depends on users' *sensitive attributes* that are never explicitly disclosed for the sake of privacy.
- We identify Level 2 as the scalability bottleneck in IoT discovery. It is worth noting that, our Level 2 using conventional crypto (e.g., ECDSA) is found **10x** as **scalable and computationally efficient** as more recent, fancier crypto like Attribute-based Encryption (ABE) [8].
- Argus Level 3 goes beyond existing privacy preservation discovery work, and realizes **indistinguishability** that attackers cannot even know Level 3 is happening, let alone prying into user privacy in Level 3. Besides, our Level 3 is found **10x** as **computationally efficient** as work based on Pairing-based Cryptography (PBC) [9].
- We implement Argus's all three levels, and ABE, PBC for comparison. Extensive analysis and real experiments

on a 20-node testbed are conducted. Besides of the high scalability and computational efficiency mentioned above, it is found secure and fast, costing **0.25 s** to discover 20 Level 1 devices, and **0.63 s** for Level 2 or Level 3 devices.

II. Models and Assumptions

A. IoT Nodes in Enterprise Environments

Node Types. There are three types of nodes: the backend, subject devices and objects. Subjects (i.e. users) use subject devices (e.g., smartphones) to operate objects (i.e. IoT devices). As is widely used in practice, the backend is *not* a single server, but a hierarchy of servers run by the admin to manage registered subjects/objects; it realizes a chain of trust, and resists collapse under the load and a single point of failure. Each subject/object has multiple attributes (e.g., a user's position, department; a device's type, functions), and must register at the backend (e.g., manually out of band), which is a common requirement in real enterprises. It obtains a private key, public key certificate (CERT) and attribute profile (PROF) signed by the backend's private key.

Resource & Power. Objects may have different resources. Resource-constrained objects (e.g., temperature sensors) have poor computing performance and are usually battery-powered. Resource-rich ones (e.g., door locks, HVAC, surveillance cameras) have fairly good computing performance and wall power; they can run public-key algorithms at reasonable speed. We assume objects with higher security requirements (Level 2 and 3) naturally come with richer resources.

Network Connectivity. Objects may have different communication interfaces, e.g., WiFi, Bluetooth, ZigBee. 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) in proximity. The network formed by subject devices and objects is called *ground network*.

B. Service Visibility

Access Control Policy. The backend stores and manages access control policies about what services a subject can access on an object. Given the large numbers of subjects/objects, the policies are frequently defined on categories using attribute predicates, not just individual identities, to avoid inefficient enumeration. E.g., *[subject: position=='manager'; object: type=='door lock' && room_type=='conference'; rights: {'open'; 'close'}]* denotes that all managers have open/close access to the door locks on conference rooms.

There are two types of attributes: ***non-sensitive attributes*** can be safely included in signed credentials (e.g., PROF) and publicly propagated, e.g., a regular employee's position, a corridor light's make/model; ***sensitive attributes*** must be kept on need-to-know basis and secret from the public or unauthorized subjects/objects, e.g., a student/employee's financial (broke), medical status (disabled, depressed).

Service Visibility. Visibility scoping policies are congruent with access control policies such that subjects and their devices should only "see" the services they are authorized to access. We call both of them *policies* for short. To this end, a subject device sends queries in the ground network, and nearby objects return their profiles (PROF), revealing their service information. Depending on the subject's access rights, an object may return differentiated variants of service information.

C. System Scale

We describe the typical scales of several aspects of enterprise IoT, where $10^i, i \in \mathbb{Z}$ denotes order of magnitude. E.g., 10^0: several, 10^2: hundreds. They are intended to give a rough sense, and actual systems may have larger/smaller scales.

1) huge subject/object amounts. Google search shows that the numbers of employees are: Google 98K, Amazon 613K; so subject amount is $10^4 \sim 10^5$. According to the field study in [10], a typical lab/office may have $\sim$30 objects; even a 2-story building may have $\sim$2K objects. A subject usually has access to N objects, which is around 100 (if she can access a few rooms) or up to 1K (if she can access objects in multiple buildings, e.g., on campus); thus N is usually $10^2 \sim 10^3$.

2) subject/object categories. A subject category (according to a predicate on subjects' *non-sensitive* attributes, e.g., "all students in CS Department, University X") has α (usually $10^0 \sim 10^2$, and possibly $\geq 10^3$) subjects (e.g., group: 10^0; class: 10^1; department: $\geq 10^3$; college: $\geq 10^4$). An object category (according to objects' *non-sensitive* attributes) has β objects, and β has a similar range as α (e.g., "all devices in Room Y": 10^1; "all lights in Building Z": $\geq 10^2$).

3) secret groups. If a policy allows subjects with certain *sensitive* attributes to discover objects with certain *sensitive* attributes, then they belong to one secret group. In reality, the persons with sensitive attributes (e.g., disability) in an enterprise and the objects serving them should not be too many, and a secret group has size γ ($10^0 \sim 10^1$ or 10^2).

4) frequent subject/object churns. In enterprises, employee entry/exit or promotion/demotion/rotation, and device installation/decommissioning happen all the time. All these may affect the access rights for individual/category subjects, and should be effectuated quickly and efficiently. E.g., when a subject leaves, all the N ($10^2 \sim 10^3$) objects she could access should be notified to reject her future discovery attempts.

III. Design Goals

Secrecy. We consider three subitems: ***1) service information secrecy for services behind walls (required by Level 2 and 3).*** Argus should keep subjects from easily gaining the information (e.g., existence, functions) of objects in rooms they cannot enter, otherwise the presence of indoor belongings is exposed to outsiders. Though such objects behind walls are physically invisible, they can be discovered by devices from outside if no security countermeasure is taken (because many radios penetrate walls). Note that protecting physical security and accessibility is not an IT problem, thus out of the scope.

2) sensitive attribute secrecy (required by Level 3). Sensitive attributes of a subject/object should only be disclosed to an authorized object/subject during service discovery.

3) indistinguishability (required by Level 3). Whether a subject/object has a sensitive attribute, or whether Level 3 discovery is happening, should be unknown to attackers.

Authenticity & Integrity & Freshness. Authenticity and integrity ensure that an entity is indeed the claimed one and a message is not forged or altered. Freshness means that a message is recently generated and not a replayed one.

Efficiency & Scalability. The system should be efficient in three aspects to fit for an enterprise scale: ***1) updating overhead.*** Upon any change in the backend database (e.g., policy addition, subject removal), the overhead of propagating and effectuating the changes to affected subjects/objects should be acceptably small. ***2) computation cost.*** The number of compute intensive operations (e.g., public-key) should be minimized. ***3) discovery overhead.*** Discovery should be done with as few rounds and short messages as possible.

Non-Goals. Attackers may know the existence of physically visible devices (e.g., they see door locks). They may physically steal objects' service information or phish subjects' sensitive attributes; subjects may inadvertently leak service information they discovered or sensitive attributes. Besides, attackers may launch DoS attacks. We do no address those issues here.

IV. Backend Operations and Overview

A. Backend Operations

Bootstrapping. A subject or object X must first register at the backend out-of-band. This is a common requirement in real enterprises. The backend adds its information to the database, and issues a private key K_X^{pri}, public key certificate (CERT) and possibly multiple attribute profiles (PROF) to X. The admin's public key K_{admin}^{pub} is also loaded onto the subject device or object. CERT and PROF are *signed by the admin* and cannot be forged/altered. A subject PROF lists the subject's *non-sensitive* attributes and can be publicly disclosed; an object PROF lists provided functions (thus service information) besides the object's *non-sensitive* attributes.

Secret Groups & Fellows. We call subjects and objects in the same secret group *fellows*. The backend issues fellows with one symmetric group key (denoted as K_i^{grp} for secret group i). A subject or object may be in multiple secret groups.

Levels. An object gets its secrecy level defined (1, 2, or 3) and must keep that to itself. The level usually depends on the device type but can be configured by the admin.

1) Level 1. (no secrecy) These objects are usually resource-poor, and offer publicly accessible services identical to everyone (e.g., thermometers in aisles), or are behind walls but not worth hiding (e.g., lights in offices). Their service information needs no encryption, but is signed by the admin for integrity.

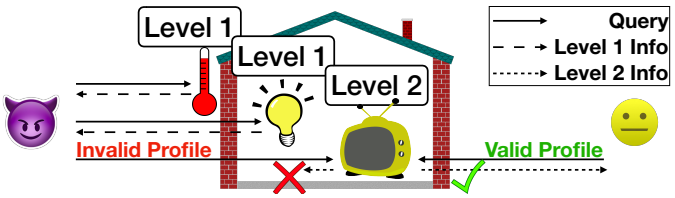


Fig. 1: Discovery in Level 1 and Level 2

2) Level 2. (service information secrecy) These objects have sufficient resource and power for public-key operations. They are behind walls and must resist discovery by outsiders: objects return encrypted, differentiated service information according to subjects' non-sensitive attributes (carried by subjects' PROFs). In Fig. 1, a multimedia device in an office checks query senders' PROFs, and returns encrypted information only to the office employees, not to outsiders.

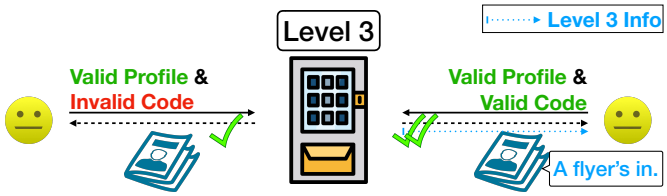


Fig. 2: Discovery in Level 3

3) Level 3. (service information secrecy+sensitive attribute secrecy+indistinguishability) Level 3 objects *secretly* offers customized services to special populations (i.e. subjects with sensitive attributes) while posing as Level 2 objects to others. Imagine student $\mathcal{S}$ with learning disability shows his diagnosis to the university, and is put in the corresponding secret group. When $\mathcal{S}$ uses a campus magazine machine $\mathcal{O}$, if $\mathcal{O}$ happens to have a sensitive attribute "machine serving students with learning disability", they will secretly confirm that they are fellows using a "code", then $\mathcal{O}$ dispenses medical/counseling/university-policy support flyers to $\mathcal{S}$, hidden within regular magazines so others will not know Level 3 discovery is happening. This is one way to offer special populations useful information when they are using daily IoT services (e.g., buy magazines/newspapers). Others will get "clean" magazines when using $\mathcal{O}$, and think $\mathcal{O}$ is Level 2. As a result, others cannot identify $\mathcal{S}$ as a sensitive attribute owner, or $\mathcal{O}$ as a Level 3 machine serving special populations.

Profile. A Level 2 object providing m different services gets m PROF variants from the backend: $\{pred_i, PROF_{O,i}\}, 1 \leq i \leq m$, where $pred_i$ is a predicate on subjects' non-sensitive attributes (e.g., *[position=='manager' && department=='X']*). If the object finds a subject's PROF matches $pred_i$, it will send encrypted $PROF_{O,i}$ to her.

A Level 3 object in m' secret groups gets m' PROF variants from the backend: $\{K_i^{grp}, PROF_{O,i}\}, 1 \leq i \leq m'$, where K_i^{grp} is the symmetric group key of group i. After the object confirms that a subject possesses K_i^{grp}, it will return encrypted $PROF_{O,i}$. A subject in n' secret groups gets n' group keys. Note that even a subject with no sensitive attribute still gets a (fake) group key, called *cover-up key* (details in Section VI).

Access Control Policy Update. The admin may need to update the backend database at any time, and possible operations include adding, removing, changing a subject/object individual or category and the access rights. Changes on the backend may need to be immediately propagated to the ground network and effectuated on the affected subjects/objects, such that newly authorized subjects can discover services, or de-authorized subjects stop seeing previously visible services.

B. Overview of Three-Level Discovery

Argus is a 3-in-1 algorithm which discovers services in three levels. We briefly describe the discovery process in each level.

Level 1 Discovery. The discovery of Level 1 objects is a typical 2-way data discovery/retrieval [11] process in the ground network. In Fig. 5, subject $\mathcal{S}$ broadcasts query QUE1 to objects around, and a Level 1 object $\mathcal{O}$ sends back its profile (PROF) in plaintext in response RES1. QUE1 carries random R_S for objects to detect duplicate queries. Because PROFs are signed by the admin, their integrity is ensured.

Level 2 & Level 3 Discoveries. They have two similar phases (Fig. 5): **1) Phase-1.** $\mathcal{S}$ broadcasts query QUE1, a Level 2 or Level 3 object $\mathcal{O}$ sends response RES1 and waits for further interaction. **2) Phase-2.** $\mathcal{S}$ sends query QUE2 individually to each of the Level 2 or Level 3 objects found in Phase-1. A Level 2 object checks the subject's non-sensitive attributes; a Level 3 object verifies that the subject is its fellow. Then the suitable PROF variant is found out and sent back securely in response RES2. We present the design details of Level 2 and 3 discoveries in the following two sections.

V. Level 2: Differentiated Discovery

This discovery targets Level 2 objects which provide differentiated services according to subjects' *non-sensitive* attributes. Argus uses conventional cryptography (e.g., ECDSA, ECDH) for mutual authentication and service information secrecy. We present its design as follows, and will show in Section VIII and IX that it outperforms alternatives using more recent cryptography (e.g., 10x high scalability and computation efficiency), and suits our service discovery context best.

A. Argus's Level 2 Scheme

Main Idea. Subject $\mathcal{S}$ reveals her non-sensitive attributes to object $\mathcal{O}$ by presenting her profile $PROF_S$; $\mathcal{O}$ checks her attributes and chooses the suitable $PROF_O$ variant to return. The messages between $\mathcal{S}$ and $\mathcal{O}$ must be authenticated, and $PROF_O$ must be encrypted for service information secrecy.

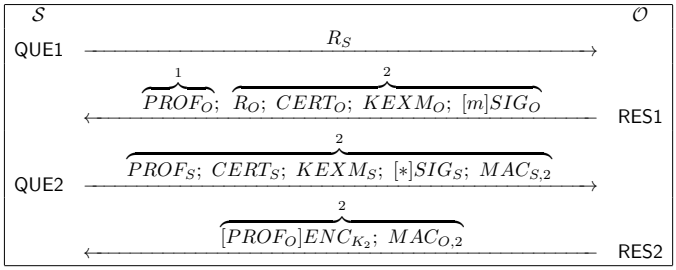


Fig. 3: v1.0, concurrent 2-in-1 algorithm for Level 1 and 2 discoveries. The content in a brace with number i is only sent by subjects performing Level i discovery, or by objects in Level i. $PROF_X, CERT_X, KEXM_X$: attribute profile, public key certificate, key exchange material of entity X; $[...]SIG_X$: signature of content in [ ] signed by X; MAC_X: hash-based message authentication code generated by X; $[...]ENC_K$: ciphertext of content in [ ] encrypted by key K; $||$: concatenation; $*$: all the content sent and received so far.

Algorithm. As shown in Fig. 3, $\mathcal{S}$ first broadcasts query QUE1 carrying random R_S. $\mathcal{O}$ in Level 2 sends back response RES1. Besides random R_O, RES1 carries $\mathcal{O}$'s public key certificate $CERT_O$ and key exchange material $KEXM_O$ which is an ECDH public value. In Fig. 3, $m = R_S||R_O||KEXM_O$ is signed by $\mathcal{O}$ for integrity.

Second, on receiving RES1, $\mathcal{S}$ verifies the signature, and based on $KEXM_O$ she establishes the pre-master secret $preK$. Level 2's session key $K_2 = HMAC(preK, label_K||R_S||R_O)$, $HMAC(secret, seed)$ is an HMAC-based pseudorandom function, $label_K$ is ASCII string "session key". Then $\mathcal{S}$ sends query QUE2 to $\mathcal{O}$, which contains her profile $PROF_S$, public key certificate $CERT_S$ and key exchange material $KEXM_S$. All the content sent and received so far (denoted as $*$, i.e., QUE1, QUE2 and $PROF_S$, $CERT_S$, $KEXM_S$) is signed for integrity. Also, Level 2's $MAC_{S,2} = HMAC(K_2, label_S||Hash(*))$, $label_S$ = "subject finished", is sent for $\mathcal{O}$ to verify the success of handshake.

Third, on receiving QUE2, $\mathcal{O}$ verifies the signature, and based on $KEXM_S$ it establishes the same $preK$ and K_2. Based on K_2 it verifies $MAC_{S,2}$: if valid, $\mathcal{O}$ checks which subject category ($pred_i$) matches the subject's non-sensitive attributes in $PROF_S$, and encrypts the corresponding $PROF_O$ using K_2. $\mathcal{O}$ sends back response RES2 containing the $PROF_O$ ciphertext and $MAC_{O,2}$, where $MAC_{O,2} = HMAC(K_2, label_O||Hash(*))$, $label_O$ = "object finished". $\mathcal{S}$ verifies $MAC_{O,2}$ and decrypts the ciphertext using K_2, getting the service information for her securely.

Our design shares some similarities with TLS handshake [12] in realizing mutual authentication and symmetric key establishment. However, Argus efficiently embeds profile exchange in, accomplishing the whole discovery in a 4-way interaction; besides, it has minimum message overhead by eliminating any unrelated component and fixing the key exchange algorithm at ephemeral ECDH, and authentication at ECDSA, which are significantly more efficient than other algorithms like RSA (experiments in Section IX-B).

VI. Level 3: Covert Discovery

This discovery targets Level 3 objects which provide covert services according to subjects' *sensitive* attributes. Like Level 2, object $\mathcal{O}$ in Level 3 ensures that subject $\mathcal{S}$ has qualifying attributes before returning $PROF_O$. However, those attributes are sensitive (e.g., learning disability) and $\mathcal{S}$ does not want to disclose them to $\mathcal{O}$ before seeing $\mathcal{O}$'s sensitive qualifying attributes (e.g., machine serving special populations). Note that using K_2 established by $\mathcal{S}$ and $\mathcal{O}$ in Section V to encrypt $\mathcal{S}$'s sensitive attributes does not work: it only protects $\mathcal{S}$'s privacy against third parties, but here $\mathcal{S}$ guards against not only third parties, but also $\mathcal{O}$, the one she is interacting with.

Neither $\mathcal{S}$ nor $\mathcal{O}$ is willing to make the first move, thus a chicken-or-egg problem arises. We apply a symmetric-key mechanism similar to [13] for private service discovery, such that sensitive attribute secrecy on both sides (i.e. mutual privacy) is achieved. Besides, we beef mutual privacy with a novel scheme which hides Level 3 discovery in Level 2, making them indistinguishable. As a result, attackers are even unaware that Level 3 discovery is happening, let alone peeking

at entities' sensitive attributes. In Section IX we show that it outperforms alternatives using more recent cryptography (e.g., 10x high computation efficiency), and suits our context best.

A. Argus's Level 3 Scheme for Sensitive Attribute Secrecy

Main Idea. Recall that at bootstrapping subjects and objects are put into the same secret group if they have sensitive attributes which allow them to recognize each other. They are called "fellows" and share one symmetric group key (K_i^{grp} for group i). Confirming one's sensitive attributes is indirectly realized by verifying its group membership, by verifying its possession of the group key. $\mathcal{S}$ and $\mathcal{O}$ send to each other an HMAC generated from the group key to prove her/its possession. $\mathcal{O}$ verifies $\mathcal{S}$ first and then vice versa. $\mathcal{O}$ will send the suitable $PROF_O$ confidentially only if $\mathcal{S}$ is its fellow.

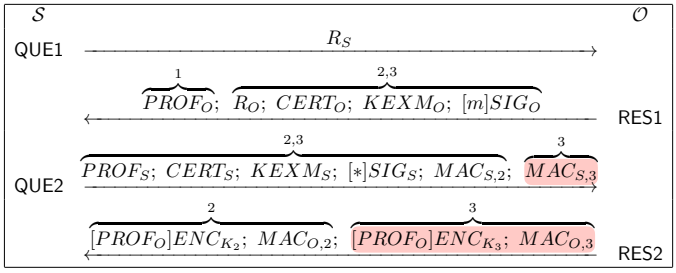


Fig. 4: v2.0, 3-in-1 algorithm for Level 1, 2, 3 discoveries. It supports sensitive attribute secrecy in Level 3. The content in a brace with number i is only sent by subjects performing Level i discovery, or by objects in Level i. The content in red boxes are newly added on top of v1.0 in Fig. 3.

Algorithm. Fig. 4 shows Level 3 discovery which is built on top of Level 1 and 2. Level 3's session key $K_3 = HMAC(K_2||K_i^{grp}, label_K||R_S||R_O)$ (assume $\mathcal{S}$ is in group i), is computed after K_2. When $\mathcal{S}$ performs Level 3 discovery, QUE2 carries Level 3's $MAC_{S,3} = HMAC(K_3, label_S||Hash(*))$. The definitions of $label_K$, $label_S$, $label_O$ and $*$ can be found in Section V.

$\mathcal{O}$ in Level 3 can establish the same K_3 only if it possesses K_i^{grp}, i.e., it is a fellow of $\mathcal{S}$. On receiving QUE2, it does all a Level 2 object does, and additionally verifies $MAC_{S,3}$: if valid, it recognizes $\mathcal{S}$ as a fellow. Then it generates $MAC_{O,3} = HMAC(K_3, label_O||Hash(*))$, encrypts the suitable $PROF_O$ variant with K_3, and adds them to RES2.

Depending on $\mathcal{O}$'s level, the HMAC of RES2 may be $MAC_{O,2}$ or $MAC_{O,3}$. $\mathcal{S}$ first tries to verify it with K_2 to see if it is a $MAC_{O,2}$: if valid, $\mathcal{S}$ knows $\mathcal{O}$ is in Level 2 and she decrypts the ciphertext using K_2. Otherwise she uses K_3 to verify if the HMAC is a $MAC_{O,3}$: if valid, $\mathcal{S}$ knows $\mathcal{O}$ is in Level 3 and she uses K_3 for decryption.

Sensitive Attribute Secrecy. Sensitive attribute secrecy is realized on both $\mathcal{S}$ and $\mathcal{O}$ due to HMAC's one-way feature: if $\mathcal{S}$ and $\mathcal{O}$ are not in the same group, $\mathcal{O}$ will find $MAC_{S,3}$ invalid. However, all it knows is $\mathcal{S}$ is not its fellow, but which secret group $\mathcal{S}$ is in stays unrevealed because the group key is not explicitly exchanged. Vice versa, $\mathcal{S}$ will not know a non-fellow $\mathcal{O}$'s group membership.

Overhead of Extensions. We see v2.0 brings in little extra discovery overhead to v1.0. Most components in RES1 and QUE2 are reused. In QUE2, one HMAC (only 32 bytes if using SHA-256) is added when performing Level 3 discovery. The length of RES2 is unchanged, because either Level 2 or Level 3 service information is sent back, not both. As for computation cost, $\mathcal{S}$ and $\mathcal{O}$ need one more HMAC generation and verification, together costing less than 1 ms.

B. Argus's Level 3 Scheme for Indistinguishability

So far, Level 3 discovery prevents a non-fellow from peeking at an entity's sensitive attributes (aka privacy), but performing Level 3 discovery itself implies that the entity has at least one sensitive attribute. E.g., attackers find a subject is seeking for Level 3 objects, then they guess she is a member of special crowds, though which crowd (e.g., depression or addiction) is unknown. Level 2 and 3 can be easily distinguished due to their message composition differences: i) QUE2 has one more component ($MAC_{S,3}$) when seeking for a Level 3 object; ii) RES2 from a Level 3 object carries $MAC_{O,3}$ other than $MAC_{O,2}$. Attacks in detail are presented in Section VII.

Main Idea. We make Level 2 and 3 indistinguishable, realizing covert visibility. QUE2s from all subjects have identical structures whenever, so are RES2s from all objects. Attackers are even unaware that Level 3 discovery is happening.

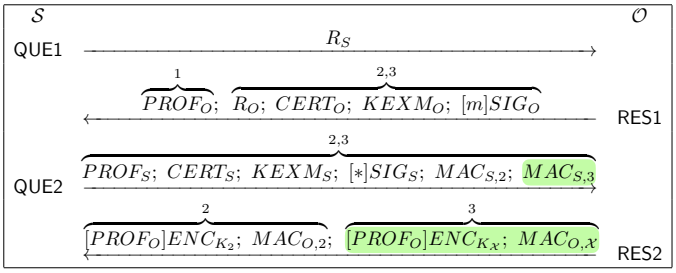


Fig. 5: v3.0, concurrent 3-in-1 algorithm for Level 1, 2, 3 discoveries. It supports both sensitive attribute secrecy and indistinguishability in Level 3. The content in green boxes are modified on top of v2.0 in Fig. 4.

Algorithm. Fig. 5 shows that Level 2 and 3 discoveries now use identical QUE2 which always carries $MAC_{O,2}$ and $MAC_{O,3}$, and are performed concurrently. Besides, a Level 3 object no longer sends $MAC_{O,3}$ constantly in RES2. Instead, its RES2 has $MAC_{O,\mathcal{X}}$, where $\mathcal{X}$ can be 2 or 3: $MAC_{O,3}$ to fellows, with $PROF_O$ encrypted by K_3; $MAC_{O,2}$ to non-fellows, with $PROF_O$ encrypted by K_2.

Indistinguishable Subjects. ***1) concurrent discoveries.*** A subject uses the same QUE2 to discover both Level 2 and 3 objects, so attackers cannot tell Level 3 discovery is happening based on QUE2's composition difference. ***2) cover-up key.*** All subjects perform concurrent Level 2 and 3 discoveries, even if some of them have no sensitive attributes and will not succeed in finding any Level 3 service. Recall that at bootstrapping even a subject $\mathcal{S}$ with no sensitive attribute still gets a (fake) secret group key (called *cover-up key*) from the

backend. A cover-up key is a unique random number and there is no second entity owning it, thus the $MAC_{S,3}$ generated from it will not result in successful handshakes. But using it, $\mathcal{S}$ can send $MAC_{S,3}$ like a sensitive attribute owner; now in attackers' eyes, every subject belongs to special populations with sensitive attributes, and the real ones are concealed.

Indistinguishable Objects. ***1) double-faced role.*** Each Level 3 object plays a "double-faced" role: it returns $MAC_{O,3}$ and offers Level 3 special service information to its fellows (special populations), while returns $MAC_{O,2}$ and offers Level 2 services to non-fellows. Non-fellows always receive $MAC_{O,2}$ and they do not know the object's another role (i.e., Level 3). ***2) constant RES2 length.*** The ciphertexts of service information ($PROF_O$) in Level 2 and 3 may have different sizes. To eliminate that, $\mathcal{O}$ appends minimum meaningless bytes to each of its $PROF_O$ variants before transmission to make them identically long. ***3) constant response time.*** In Level 3 discovery $\mathcal{O}$ verifies one more HMAC ($MAC_{S,3}$) than Level 2, so response times are different. It is negligibly small ($<$ 0.1 ms) on resource-rich platforms like Pi, but can be longer on constrained platforms (mostly $<$ 1 ms). $\mathcal{O}$ waits for that time difference before sending Level 2 RES2.

Overhead of Extensions. We see v3.0 brings in little extra overhead to v2.0. In QUE2, now $MAC_{S,3}$ is mandatory, so QUE2 should always have these 32 bytes (if using SHA-256). RES2's length and computation cost are unchanged.

C. Multiple Sensitive Attributes

Level 3 discovers the fellow objects in one secret group at a time, because $MAC_{S,3}$ is generated from one group key. In reality a subject may have multiple (usually no more than a few) sensitive attributes and are members in multiple secret groups. Her device can automatically use her group keys in turns (one at a time) to generate $MAC_{S,3}$ and launch discoveries, till all her authorized covert services are found.

VII. Security Analysis

Threat Model. We assume the backend is trustworthy and well-protected. Also, communication between the backend and subject/object devices is secure. Subject devices and resource-rich objects are reasonably well protected, e.g., by their OS.

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

Like TLS and many other algorithms, the security of ours is on the premise that secret information is kept to its owner, and is computationally infeasible to compromise, and cannot be obtained from sources outside of the channel. However, in reality it can, e.g., attackers have military computing resources, or they leverage malware or social engineering to steal private keys from users/devices. Resisting those attacks is out of the scope. Our analysis below shows that attackers will fail unless they have session key K_2, K_3, or private key K_X^{pri} (X is a subject or object) and/or secret group key K_i^{grp} (for group i).

A. Level 2 Attacks from External Attackers

Service Information Secrecy. In Level 2, attackers may try to view a $PROF_O$ they are unauthorized to. Also, they may spread a false $PROF_O$. Their possible roles and actions are:

Case1: Eavesdropper. Passive attacker $\mathcal{E}$ eavesdrops the conversation between subject $\mathcal{S}$ and object $\mathcal{O}$ to view $PROF_O$. She must compromise K_2 to decrypt the ciphertext of $PROF_O$, which is infeasible. Note that $\mathcal{E}$ cannot obtain K_2 by compromising K_S^{pri} or K_O^{pri} (cracking a long-term key might be easier than a session key), because we use ephemeral ECDH for key exchange which has *forward secrecy*.

Case2: Subject/Object Impostor. Active attacker $\mathcal{E}_S$ poses as $\mathcal{S}$ to interact with $\mathcal{O}$ and request $PROF_O$. Since the interaction is authenticated, she will fail due to the lack of K_S^{pri}. $\mathcal{E}_O$ poses as $\mathcal{O}$ to give $\mathcal{S}$ fake service information, but it will fail without K_O^{pri}. Besides, $PROF_O$ is signed by the admin for integrity, breaking which is considered infeasible.

B. Level 3 Attacks from External Attackers

Service Information Secrecy. First, attackers may launch the same attacks in Level 2 to get $PROF_O$. We assume $\mathcal{S}$ and $\mathcal{O}$ are fellows in secret group i, and they share K_i^{grp}.

Case3: Eavesdropper. This time $\mathcal{E}$ needs K_3 to decrypt the ciphertext of $PROF_O$. She may: i) compromise K_3 directly, which is infeasible; ii) or compromise K_2 and K_i^{grp} because they together generate K_3, but this does not make things easier.

Case4: Subject/Object Impostor. $\mathcal{E}_S$ poses as $\mathcal{S}$ to interact with $\mathcal{O}$ for $PROF_O$. To succeed, she needs K_S^{pri} and K_i^{grp}. Like in Case2, $\mathcal{E}_O$ will fail in giving fake service information.

Sensitive Attribute Secrecy. Second, attackers may try to find out what sensitive attributes $\mathcal{S}$ or $\mathcal{O}$ has.

Case5: Eavesdropper. Attacker $\mathcal{E}$ eavesdrops the conversation of $\mathcal{S}$ and $\mathcal{O}$, but she will know $\mathcal{S}$ or $\mathcal{O}$ is in group i only if she can confirm that $MAC_{S,3}$ or $MAC_{O,3}$ is a valid HMAC generated using K_i^{grp}, which needs K_2 and K_i^{grp}. Besides, knowing that an entity belongs to group i does not mean knowing its sensitive attribute, unless the mapping relationships between group IDs and attributes are also known. However, that knowledge is kept to the admin only.

Case6: Subject/Object Impostor. $\mathcal{E}_S$ poses as $\mathcal{O}$'s subject fellow to interact with $\mathcal{O}$, and tries to find out $\mathcal{O}$'s sensitive attributes. To succeed, she needs a valid subject private key, K_i^{grp} and the mapping relationships. Similarly, $\mathcal{E}_O$ may pose as an object fellow to explore $\mathcal{S}$'s sensitive attributes, and it needs a valid object private key and K_i^{grp}. Neither will work.

Indistinguishability. Third, attackers may try to find out if $\mathcal{S}$ or $\mathcal{O}$ has any sensitive attribute (regardless of what sensitive attribute), and if Level 3 discovery is happening.

Case7: Eavesdropper. i) subject distinguishability. As introduced in Section VI-B, we use cover-up keys on subjects who have no sensitive attributes to make them pose as real sensitive attribute owners. $\mathcal{E}$ have no way to distinguish them. ii) object distinguishability. Since only Level 3 objects's RES2 may carry $MAC_{O,3}$ other than $MAC_{O,2}$, $\mathcal{E}$ may leverage this to recognize Level 3 objects. However, to recognize $MAC_{O,3}$, she needs K_3, which is impractical.

Case8: Subject/Object Impostor. $\mathcal{E}_S$ interacts with $\mathcal{O}$ to see if it is in Level 3. If $\mathcal{E}_S$ recognizes the HMAC in RES2 as $MAC_{O,3}$, then she knows $\mathcal{O}$ is. This needs a valid private key and K_i^{grp}. Alternatively, she may try an elimination method: tell if the HMAC is $MAC_{O,2}$, and if not, it is $MAC_{O,3}$ then. Verifying $MAC_{O,2}$ only needs a valid private key, so the security strength is degraded if this works. However, Level 3 objects play "double-faced" roles: they send $MAC_{O,2}$ to attackers, thus attackers cannot use the elimination trick.

$\mathcal{E}_O$ may impersonate $\mathcal{O}$ to interact with $\mathcal{S}$ to see if she has any sensitive attributes, but cover-up keys impede that.

Case9: Side-Channel Attacks. Beware of side-channel attacks which may compromise indistinguishability. Particularly, due to a Level 3 object's additional computation on base of Level 2, it will take longer to respond than a Level 2 one. An attacker may recognize Level 3 objects through timing measurements and analysis (i.e. *timing attacks*). However, in Argus a Level 3 object only needs to verify one more HMAC (i.e. $MAC_{S,3}$) than a Level 2 one, which costs < 0.1 ms on Raspberry Pi. It cannot be detected when buried under much larger time fluctuations from OS, network, etc.

C. Attacks from Internal Attackers

Assume benign entity $\mathcal{I}$ goes rogue. Unlike an external attacker, she already has a valid private key K_I^{pri}. However, note that if she eavesdrops (Case1, 3, 5, 7) or impersonates others (Case2, 4), her own private key is useless and the attacks are the same as external attacks. It becomes easier to crack Case6, 8: since $\mathcal{I}$ already has a valid private key, she just needs to compromise K_i^{grp}, which is still difficult.

D. Consequences of Key Compromise

If a session key is compromised, only that session's content (e.g., service information) will be exposed; if a private key is compromised, only that entity will be impersonated. If a private key and a group key are both compromised, attackers may find out members in that one secret group only, by interacting with them one by one instead of getting the entire member list. Each of these cases has a limited impact, and cannot paralyze the service discovery system.

VIII. Scalability Analysis

The system must be scalable to enterprises, and the most critical metric for scalability in our context is updating overhead instead of discovery overhead. This is because Argus is for discovering services in proximity, the number of which is usually not more than dozens. To the contrary, any change in the backend database (e.g., policy addition, subject removal) related to Level 2 or 3 should be immediately synchronized to affected subjects/objects on the ground, otherwise authorized users may fail to discover and access new services timely, while unauthorized users continue to see services they are no longer eligible for. Such *updating overhead* (defined as the number of affected subjects and objects) can be huge.

Level 1 & 3 Scalability. Level 1 is little relevant to this authorization-related updating because it offers identical information to everyone. When changing a subject/object/policy in Level 3, the worst case (e.g., remove a person from a secret group) is all the other fellows in the group should get new keys, i.e., the overhead is $(\gamma-1)$ (defined in Section II, usually $10^0 \sim 10^1$). It is small due to a secret group's limited size.

Level 2 Scalability. Level 2 has the largest updating overhead. We find Argus is up to **1000x** as efficient as ID-based ACL alternatives in adding a subject, and up to **10x** as efficient as Attribute-based Encryption (ABE) alternatives in removing a subject. To add/remove an object/policy, mostly just that object or the objects mentioned in that policy should be updated, thus the overhead is 1 or β (defined in Section II, usually $10^0 \sim 10^2$). Adding/removing a subject is the bottleneck of scalability, so we show the analysis on it in detail.

A. Detailed Scalability Analysis on Level 2

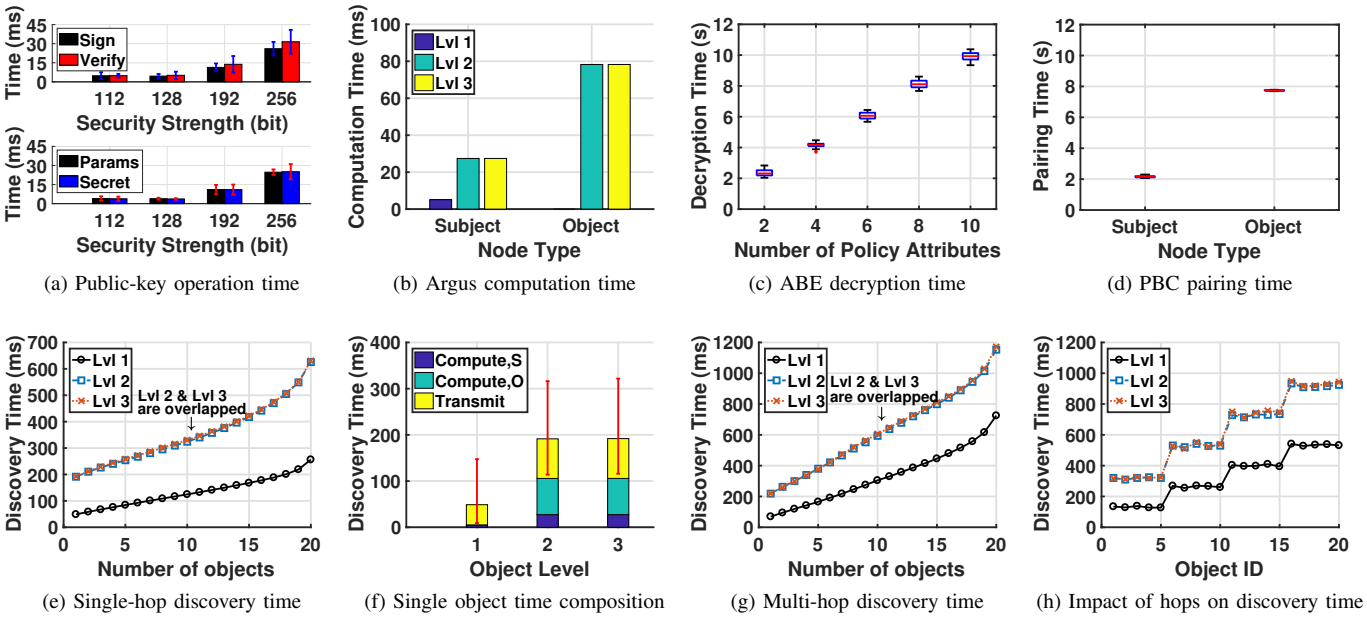
ID-based ACL. In this method, every object locally stores its access control list enumerating the identities of subjects which are allowed to access and discover it. In Section II we show that a subject may access N ($10^2 \sim 10^3$) objects typically. Then when subject $\mathcal{S}$ joins/leaves the system, the N objects she can access should be notified to add/remove her ID (i.e. ID_S) to/from their ACLs.

TABLE I: Updating Overhead Comparison

	Add a subject	Rmv a subject
ID-based ACL	N	N
ABE	1	$(\xi_o N + \xi_s(\alpha-1)) \approx 10N$
Argus	1	N

Argus. In Argus, an object stores an attribute-based ACL, which uses predicates on subject attributes (e.g., *[position=='manager' && department=='X']*) to describe its authorized subjects. To access objects, a newcomer $\mathcal{S}$ just needs to contact the backend once to get her attribute profile (overhead: 1), and present it to objects; objects do not need to update their ACLs. This significantly outperforms ID-based ACL (up to **1000x**). However, when $\mathcal{S}$ leaves, the backend should notify the N objects that she could access, to remove ID_S from their ACLs and refuse her future discovery.

ABE. Ciphertext-Policy Attribute-based Encryption [8] can be used for Level 2 discovery. At bootstrapping, the backend issues $\mathcal{S}$ with a set of keys, each corresponding to her one attribute (e.g., *department:X*); also, the backend issues $\mathcal{O}$ with ABE ciphertexts—$PROF_{O,i}$ encrypted using policy $pred_i$ (e.g., *[position=='manager' && department=='X']*). Based on ABE's principle, the $PROF_{O,i}$ ciphertext can be decrypted only if $\mathcal{S}$ has all the attributes (thus the keys) to meet $pred_i$.



(a) Public-key operation time (b) Argus computation time (c) ABE decryption time (d) PBC pairing time

(e) Single-hop discovery time (f) Single object time composition (g) Multi-hop discovery time (h) Impact of hops on discovery time

Fig. 6: Computation Time Cost and Discovery Time Cost

About updating, a newcomer $\mathcal{S}$ just gets her secret keys from the backend, then she can discover services (overhead: 1). To revoke $\mathcal{S}$, the backend has to *globally* revoke a set of her attributes to make her no longer belong to any subject category. E.g., to revoke her attribute *department:X*, it: i) re-encrypts *all* ciphertexts whose encryption policies contain *department:X*, and delivers them to their objects (overhead: $\xi_o N, \xi_o \geq 1$); ii) re-generates those attributes' secret keys, and delivers them to *all* subjects owning the attributes except $\mathcal{S}$ (overhead: $\xi_s(\alpha - 1), \xi_s \geq 1$). α (defined in Section II) is the number of subjects in a subject category, usually $10^0 \sim 10^3$, possibly $\geq 10^4$. Such attribute-level updating often affects more subjects than $\mathcal{S}$'s category members, and more objects than what $\mathcal{S}$ could access, that is why ξ_s, ξ_o mostly go over 1. The overall overhead is $(\xi_o N + \xi_s(\alpha - 1))$. When $\xi_o, \xi_s > 1$, or α is large (e.g., $10^3 \sim 10^4$, if $\mathcal{S}$ is in a large category like a department or college), the overhead easily goes to $10N$ or more.

IX. Experimental Evaluation

We implement all the three levels of Argus. Besides, two alternatives are implemented: one is based on Attribute-based Encryption (ABE) and used for Level 2 (introduced in Section VIII); the other uses Pairing-based Cryptography (PBC), which has been applied on mobiles' secret handshake [14], and can be easily adapted for Level 3 discovery.

Testbed Rationality. We conduct experiments on a testbed consisting of 1 subject device (Nexus 6) and 20 objects, each emulated by a Raspberry Pi 3. 1) As mentioned in Section II, we assume Level 2 and 3 objects, which have high security requirements, are equipped with sufficient resource and power, and can run public-key algorithms at reasonable speed. 2) Though Level 1 objects in reality have poor computing resource, they simply return profiles after receiving queries, and no computation is needed, thus emulating them using Pis or Arduinos makes no difference. 3) Our design is above the network layer and orthogonal to radios. As a result, we believe Pis communicating with WiFi well emulate objects in all three levels, and networking and power aspects. 4) Argus aims at local discovery for services in proximity, the number of which is usually not more than dozens. The scalability challenge exists in updating (analyzed in Section VIII), not discovery; for discovery tests, our testbed with 20 Pis is sufficient.

We test the computation time cost on the subject device and objects, and find that to achieve 128-bit security strength, Argus needs only 105 ms while ABE and PBC cost at least **10x** long. As for the overall discovery time cost (mainly by computation and transmission), Argus takes 0.25 s to discover 20 Level 1 objects and 0.63 s for 20 Level 2 or 3 objects (each object is 1 hop from the subject). For a multi-hop case where 20 Level 2 or Level 3 objects are a mixture of objects from 1-hop to 4-hop away from the subject, Argus costs 1.15 s for discovery completion and is still well responsive.

A. Message Overhead

When strength is 128-bit, $CERT_X$ is an X.509 ECDSA certificate of 552 B; $KEXM_X$ and SIG_X are 64 B. $PROF_X$ averagely has 200 B. $[PROF_O]ENK_K$ is assumed to use AES in CBC mode with 16-byte IV and 32-byte MAC, thus has 248 B. Besides, R_X is 28 B and MAC_X (SHA-256) is 32 B. Level 1 has 28-byte QUE1 and 200-byte RES1, thus 228 B in total. Level 2 or 3 has 28-byte QUE1, 772-byte RES1, 1008-byte QUE2, 280-byte RES2, thus 2088 B in total.

B. Computation Time Cost

The cryptographic computation time for Argus, AES and PBC are evaluated. We use crypto libraries OpenAndroidSSL

on the subject device and JCA on objects, for their efficient implementation compared with others (e.g., Spongy Castle). ECDSA is preferred to RSA because the latter costs much longer (e.g., 18x for 128-bit strength). The experiments show that Argus is **10x** as computationally efficient as ABE, PBC.

Fig. 6 (a) shows the computation time on the subject side for ECDSA (for signature) and ECDH (for key exchange) operations, under strength 112-bit, 128-bit, 192-bit, 256-bit. As shown, computation time increases with security strength, e.g., 112-bit costs 4.7 ms in signing while 256-bit costs 26.0 ms. For each strength, ECDSA verification/ECDH secret computation costs similar or slightly longer time than signing/parameter generation. Similar results are observed on the object side. Other operations like HMAC and AES cost less than 1 ms on both sides. In the following experiments, we use 128-bit due to its fast speed while sufficient strength.

Argus. Fig. 6 (b) shows the overall computation time on subjects and objects in all levels. In Level 1 discovery, a subject only needs to verify one signature (of $PROF_O$), costing 5.1 ms; an object has no compute intensive operation. In Level 2 and 3, a subject needs 1 signing, 3 verifications (for $CERT_O$, $KEXM_O$, $PROF_O$), 2 ECDH operations (parameter generation, secret computation), costing 27.4 ms; an object needs the same, costing 78.2 ms. Note that the public-key operations in Level 2 and 3 are identical.

ABE. When using ABE, encryption is performed by the backend and decryption by subject devices. Objects do neither. Considering that the backend has superior performance and ciphertexts can be generated beforehand, here we focus more on subject decryption time, tested using ABE library [15]. Fig. 6 (c) shows that ABE's decryption time is well linearly to the number of attributes in the ciphertext policy. Each attribute leads to about 1 second decryption time increase.

PBC. Pairing time is the time cost for computing a pairwise symmetric key using PBC keys. We evaluate JPBC library [16] on the subject device and objects, and pairing costs 2.2 s and 7.7 s respectively, as shown in Fig. 6 (d).

Note that test results depend significantly on the crypto library implementation. The ABE and PBC libraries currently available are probably preliminary, thus the decryption time and pairing time presented should not be interpreted literally, but rather revealing the likely magnitudes. According to our experiments, ABE and PBC cost at least **10x** as long as Argus.

C. Overall Discovery Time Cost

We present in Fig. 6 (e) (g) the overall time cost (mainly by computation and transmission) for Argus to discover 20 objects, in all the three levels, in both single-hop and multi-hop conditions. Even in Level 2 and 3, discovering 20 single-hop objects costs only 0.63 s, while multi-hop objects only 1.15 s. Such short latencies result in positive user experience.

Single-Hop. Fig. 6 (e) shows that in each level the discovery time cost increases with the number of objects to be discovered. The completion of discovering 20 objects is quick, which costs 0.25 s for Level 1 and 0.63 s for Level 2 and 3. Level 1 needs less than half of the time of Level 2/3 because it is 2-way communication while the other two are 4-way. Also, Level 1 has less computation. Fig. 6 (f) shows the time composition for discovering 1 single-hop object: in Level 1 89% of the time is on transmission; in Level 2/3 it is 45%. The variance in (f) mainly comes from changeful wireless transmission time.

Notice that Level 2 and Level 3 have overlapped time curves, thus indistinguishable time cost. This is because Argus Level 3 only has one more HMAC generation than Level 2, which averagely costs 0.08 ms on Pi. In practice, such tiny difference will not give attackers chances to distinguish Level 3 objects from Level 2 ones via timing measurements, because it is buried in timing fluctuations (e.g., from OS, program run, network) of higher orders of magnitude.

Multi-Hop. We also test the discovery time cost in a multi-hop condition. The 20 objects are divided to 4 equal groups: Object 1–5, 6–10, 11–15, 16–20 are 1, 2, 3, 4 hops away from the subject respectively. Fig. 6 (g) shows that the discovery costs longer than the single-hop case: here discovering 20 Level 1 objects needs 0.72 s, and Level 2 or Level 3 objects 1.15 s. But still, the latency is short. Fig. 6 (h) reveals the impact of hops on latency: discovering a 1-hop Level 1 object averagely costs 0.13 s, while 4-hop needs 0.53 s; as for a Level 2 or Level 3 object, 1-hop takes about 0.32 s (0.1 s computation + 0.22 s transmission) while 4-hop 0.92 s (0.1 s computation + 0.82 s transmission). We see transmission time increases roughly linearly with hop counts. In all cases Argus is fast enough to achieve satisfactory user experience.

X. Related Work

A. Centralized vs. Distributed Discovery

Much existing work [2], [3], [4] depends on centralized servers as repositories for efficient, scalable and wide-area discovery. However, they may encounter a single point of failure or long latency, and do not support proximity-based discovery. Distributed solutions like UPnP [5] and Bluetooth SDP [6] are infrastructure-less, and any service may announce itself or reply a query. Multicast DNS [17] and Bonjour [7] (Apple) support both centralized and distributed discoveries. Distributed discovery has the advantage of discovering nearby services robustly (no single point of failure) and quickly. However, it does not have a wide-area discovery scope.

B. Secure Discovery

Authenticated & Encrypted Discovery. Security is limitedly covered in existing work. Some [2] authenticate neither users nor service information, others [4] authenticate services, and a few [3], [5] authenticate both. Besides, UPnP [5] and Bluetooth [6] have messages encrypted for confidentiality.

Private Discovery. In an ad-hoc network, if a user and a service both have privacy concerns, neither wants to expose its sensitive information before the other does, a chicken-or-egg problem arises. In solutions like [3], [18], a trustworthy proxy is assumed and used as a bridge between users and services. Both entities simply send to the proxy encrypted messages which only the proxy can decrypt. This model avoids the chicken-or-egg problem but is infrastructure-dependent.

Multiple cryptographies have been applied to achieve distributed private discovery. Some [19], [20] are public-key based, and they have huge computational cost. Some [13], [21] are symmetric-key based: a user and the service she can discover get a symmetric key at bootstrapping; during discoveries, they test each other's possession of the symmetric key. Besides, paring-based [22] solutions are also proposed, which give users and services pairing-based keys at bootstrapping which will be used to generate pairwise symmetric keys during discoveries. Then the possession of the symmetric key is tested. MASHaBLE [14] combines this with BLE to discover secret community members.

Argus's Features. 1) Argus consciously chooses a distributed, P2P discovery strategy because IoT interactions are largely physical proximity based. 2) It achieves concurrent visibility scoping at three levels, while existing solutions are restricted to one and unfit for enterprise IoT with numerous, heterogeneous services. 3) It minimizes the updating overhead upon the frequent enterprise user churns, making the system scalable to enterprise IoT. 4) It goes beyond mutual privacy in existing work and achieves indistinguishability, such that attackers do not even know Level 3 discovery is happening.

XI. Discussion

Physically Visible Services. Argus prevents user devices from collecting service information of unauthorized objects/services, especially those behind walls, which usually have larger amounts and higher secrecy requirements than those in public areas. Humans may still gain knowledge of those physically visible services, but that belongs to physical access security, which is out of the scope.

Revocation. When a subject loses access rights, Argus efficiently revokes her discovery capabilities. Of course, if she has already gained an object's information, revocation cannot remove the knowledge from her head. However, for proximity-based discovery in a large-scale enterprise environment, it is not uncommon that a subject has not yet discovered all the objects she could, then revocation stops her knowing more.

Unlinkability. Unlinkable discovery [14] is a type of private discovery with extra requirements: an eavesdropper should be unable to determine that the two messages she sniffed are from/to the same entity, thus she cannot identify or track users. Those work usually has a city-scale context, where the visiting to certain places (e.g., hospitals, bars) involves privacy and should be kept secret. Argus does not target unlinkability, because we believe a person's location history within an enterprise/campus scope is less sensitive.

XII. Conclusion

In this paper, we describe the design, implementation and evaluation of Argus, a proximity-based service discovery algorithm which efficiently discovers three levels of visibility (public, differentiated and covert) in parallel, and is scalable to IoT in enterprise environments. It has much less computation cost and updating overhead than alternatives using ABE and PBC. It is very responsive, taking only 0.25 s to discover 20 nearby public services, 0.63 s for 20 differentiated or covert services, agile for satisfactory user experience.

Acknowledgment

This work is supported in part by the National Science Foundation under grant number 1652276.

References

[1] Q. Zhou and F. Ye, "Apex: automatic precondition execution with isolation and atomicity in internet-of-things," in *Proceedings of the International Conference on Internet of Things Design and Implementation*. ACM, 2019, pp. 25–36.

[2] P. V. Mockapetris, "Domain names: Implementation specification," Tech. Rep., 1983.

[3] S. E. Czerwinski, B. Y. Zhao, T. D. Hodes, A. D. Joseph, and R. H. Katz, "An architecture for a secure service discovery service," in *MobiCom*, vol. 99, 1999, pp. 24–35.

[4] E. Guttman, C. Perkins, J. Veizades, and M. Day, "Service location protocol, version 2," Tech. Rep., 1999.

[5] U. Forum, "UPnP Device Architecture 2.0," http://upnp.org/specs/arch/UPnP-arch-DeviceArchitecture-v2.0.pdf.

[6] B. SIG, "Bluetooth SDP," https://www.bluetooth.com/specifications/assigned-numbers/service-discovery.

[7] A. Inc., "Bonjour," https://developer.apple.com/bonjour/.

[8] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-policy attribute-based encryption," in *2007 IEEE symposium on security and privacy (SP'07)*. IEEE, 2007, pp. 321–334.

[9] N. Koblitz and A. Menezes, "Pairing-based cryptography at high security levels," in *IMA International Conference on Cryptography and Coding*. Springer, 2005, pp. 13–36.

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

[11] 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.

[12] T. Dierks and E. Rescorla, "The transport layer security (tls) protocol version 1.2," Tech. Rep., 2008.

[13] J. Lindqvist, T. Aura, G. Danezis, T. Koponen, A. Myllyniemi, J. Mäki, and M. Roe, "Privacy-preserving 802.11 access-point discovery," in *Proceedings of the second ACM conference on Wireless network security*. ACM, 2009, pp. 123–130.

[14] Y. Michalevsky, S. Nath, and J. Liu, "Mashable: mobile applications of secret handshakes over bluetooth le," in *Proceedings of the 22nd Annual International Conference on Mobile Computing and Networking*. ACM, 2016, pp. 387–400.

[15] J. Wang, "Java realization for ciphertext-policy attribute-based encryption," 2012.

[16] A. De Caro and V. Iovino, "jpbc: Java pairing based cryptography," in *2011 IEEE symposium on computers and communications (ISCC)*. IEEE, 2011, pp. 850–855.

[17] S. Cheshire and M. Krochmal, "Multicast dns," Tech. Rep., 2013.

[18] F. Zhu, M. Mutka, and L. Ni, "Splendor: A secure, private, and location-aware service discovery protocol supporting mobile services," in *Proceedings of the First IEEE International Conference on Pervasive Computing and Communications, 2003.(PerCom 2003)*. IEEE, 2003, pp. 235–242.

[19] M. Abadi and C. Fournet, "Private authentication," *Theoretical Computer Science*, vol. 322, no. 3, pp. 427–476, 2004.

[20] D. J. Wu, A. Taly, A. Shankar, and D. Boneh, "Privacy, discovery, and authentication for the internet of things," in *European Symposium on Research in Computer Security*. Springer, 2016, pp. 301–319.

[21] F. Zhu, W. Zhu, M. W. Mutka, and L. M. Ni, "Private and secure service discovery via progressive and probabilistic exposure," *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, no. 11, pp. 1565–1577, 2007.

[22] S. D. Galbraith, K. G. Paterson, and N. P. Smart, "Pairings for cryptographers," *Discrete Applied Mathematics*, vol. 156, no. 16, pp. 3113–3121, 2008.