ABSTRACT

Smart homes made up of Internet of Things (IoT) devices have seen wide deployment in recent years, with most, if not all, of them controlled by remote servers in the cloud. Such designs raise security and privacy concerns for end users. We believe that the current situation has largely resulted from lacking a systematic home IoT framework to support localized end user control.

To let end users take back the control of smart homes, we propose Sovereign, an IoT system framework that allows users to securely control home IoT systems without depending on a cloud backend. Unlike existing solutions, Sovereign networks home IoT devices and applications using named data with application-level semantics; the names are then used to construct security mechanisms. Users define security policies and these policies are then executed by the localized security modules. We implement Sovereign as a pub/sub based development platform together with a prototype local IoT controller. Our preliminary evaluation shows that Sovereign provides an easy-to-use systematic solution to secure smart homes under user control without imposing noticeable overhead.

1 INTRODUCTION

Home Internet-of-Things (IoT) is coming of age [19]. At the moment, tech giants have provided Home IoT platforms [1–3, 16, 23, 33] utilizing their mature cloud computing infrastructures. Relying on cloud backends to varying degrees, these platforms network home devices and applications with different functionalities to offer a good experience to end users.

Intuitively, a user should have full control of smart devices and applications, in a way similar to her full control of conventional home appliances, without reliance on external parties. Unfortunately, this is not the case: most, if not all, of today’s deployed smart home systems are controlled by remote servers in the cloud, raising various user privacy and security issues [10, 22, 30, 36, 37]. For example, end users’ private data becomes cloud-accessible. Also, any failure in the cloud may threaten the safety and privacy of millions of users, e.g., unavailability [11] caused by the cloud outage or unauthorized remote control [14] due to erroneously designed cloud applications.

Why are there a lack of products to allow direct user control in the age of smart homes? We believe there are two main reasons. First, as data is the new currency, data generated at smart homes is a highly desirable resource [34] for service providers. Second, there seems a lack of highly secure and highly usable solution to enable user-controlled smart homes.

In this work, we aim to solve the second problem by exploring a new direction – designing a home IoT system to be controlled by end user, without external dependencies. Specifically, we want to enable end users to directly manage their home devices, applications, resources (i.e., content and executables) and their privileges locally, so we aim to develop a systematic home IoT framework to realize it. Such a framework should provide application developers with easy-to-use APIs and built-in localized security support and not expose new burden on end users from a system design perspective. We believe that once viable technical solutions are developed that meet users’ needs, new market can be opened as an answer to the first issue.

In this paper we present Sovereign, a home IoT system framework to make the home an autonomous system solely controlled by end users. More specifically, Sovereign secures and verifies all communications strictly following user-defined policies (e.g., which device/application can use which service or access which piece of data) locally.

Our solutions have three major parts:

1. Utilizing semantic-rich names across the entire system (§4). We propose a set of naming conventions to name each entity and resource in the local system based on Named Data Networking (NDN) [43]. Each name, carried in network packets, provides application-level information for the system to make security-related decisions. Importantly, with each entity properly named, security policies can be realized by defining the relations between semantically-rich names.
2. **Building localized security modules** (§5). To enable end user control, Sovereign develops three localized security modules – security bootstrapping, trust management, and access control. Moreover, these modules operate on each system participant in a decentralized way and enforce user-defined policies for every data exchange and device action.

3. **Usable framework with security designed-in** (§6). Sovereign seamlessly integrates security modules into the basic pub-sub communication API. Following user-defined policies, security modules automatically add/check security on each outgoing/incoming packet within the procedure of pub and sub. Therefore, developers can easily build devices/applications that support local user control in Sovereign without having to be security experts themselves.

We implement Sovereign as an IoT application development platform (§7). The platform has been developed and maintained for over one year and has been used by several campuses across the world. Sovereign SDK follows a modular design and can be adopted by any IoT platform that supports standard C language. Along with the development platform, we also provide a python-based prototype controller for the user to control their home. Our preliminary evaluation results show that Sovereign does not pose new manual operations from end users, and it is easy for IoT developers to port existing IoT applications onto Sovereign so as to be user locally controlled; in addition, Sovereign realizes user local control without causing noticeable additional overhead to network delay and constrained IoT devices (§8).

Our contribution is to show a new way of building home IoT systems that empower end users. A home should be an autonomous system controlled by the home owner, and Sovereign lets the owner take back the control from large cloud providers. Our work also contributes to recent new trends (e.g., Web 3.0, Solid [8]) which aim to return the control of user data into users’ hands.

2 **MOTIVATION**

In this section, we provide the motivation for our work. We begin by setting up the background of today’s smart home IoT systems (§2.1). We then introduce the security and privacy issues that have been widely identified and why local user control can potentially address these issues (§2.2). We also discuss existing attempts to provide local home control and their limitations (§2.3). Finally, we introduce our insights on realizing local user control (§2.4).

2.1 **Cloud-based Home IoT Systems**

Typical cloud-based home IoT platforms like Samsung’s SmartThings [33], Amazon’s AWS Home IoT [2], and Google IoT core [16], now account for a big share of the market. According to Samsung [41], SmartThings have 52 million active users across the world by Jan 2020. They serve end users with a cloud-based backend.

The cloud backend is the core in these systems because it usually serves as the authority to (i) bootstrap and manage home devices/applications, (ii) authenticate commands (e.g., unlock the door), and to (iii) manage access rights to data (e.g., access to the temperature of the home). Under this model, smart home devices and applications build secured channels like Transport Layer Security (TLS) to the cloud backend and delegate control to the cloud. Here, we use Samsung’s SmartThings IoT as an example.

SmartThings’ ecosystem has three main components: the SmartThings cloud backend, cloud-connected devices, and cloud-hosted applications. To join a SmartThings system, devices and applications must have the SmartThings certificate pre-installed as the trust anchor. Also, their developers need to register their products to SmartThings in advance, and SmartThings will remember unique IDs and public keys of registered devices and applications. The cloud backend is a rendezvous point of devices and applications, and it manages the whole system through OAuth [17]: applications or devices must have OAuth tokens from the cloud backend before they can assess home resources.

Therefore, we can see the cloud backend in SmartThings provides all three control-related functions listed above. Similar results can also be observed in other cloud-based home IoT systems including AWS IoT, Google IoT Core, and Microsoft’s Azure IoT system (see Appendix A).

On today’s home IoT market, Apple’s HomeKit [3] encourages local communication and allows users to directly control their home IoT system without going through the cloud. This is aligned with the same goal of our work. However, HomeKit still relies on the cloud for device authentication and trust management.

2.2 **Security and Privacy Issues**

In cloud-based home IoT platforms, a cloud backend is in the control loop, potentially causing several critical issues to end users.

First, end users’ activities and data are directly exposed to the back-end cloud service, which greatly violates user’s privacy [10]. Ren et. al [28] show that more than half of studied IoT devices share information to third parties or support parties. Recent years have also witnessed a increasing number of smart home data leakage [12, 15]. Second, a home IoT system can suffer from both availability and implementation issues from the cloud. As witnessed in the recent news [11], users cannot even unlock their home when the cloud is down. Fernandes et. al [14] have pointed out severe overprivilege issues in SmartThings that can directly threaten millions of homes around the world.
the home also increases attack surface. Recent research [5] shows that an ISP or network observer can infer private user information by analyzing encrypted traffic sent from a smart home system.

2.3 Existing Attempts to Provide User Control

We are not the first to identify issues in the cloud-based IoT systems. Various approaches have been proposed to enforce user control without relying on the cloud. These approaches have merit and provide innovative techniques to help solve the problem, yet we argue these approaches do not realize user control that is fully independent of the cloud or fail to be fine-grained enough to control home systems.

Approaches [9, 13, 20, 40] try to add run time enforcement by modifying existing IoT application programs or patching extra data flow verification into the system backend. These approaches help to enforce user control on cloud-hosted applications; however, they do not break the cloud-based system model and aforementioned issues are not addressed.

Home automation projects like openHAB [25] and Home Assistant [18] add another layer of control over the existing devices from different IoT systems through a local automation hub. Through the local hub, openHAB and Home Assistant empower end users by allowing them to control IoT devices locally. Nevertheless, to control a cloud-based device, for example, a SmartThings sensor, the local hub usually needs to contact device’s cloud backend directly or indirectly, so the cloud is still in the control loop. Moreover, these approaches do not provide solutions to managing the privileges of system participants.

Another direction is to put user control at the network layer [13, 24, 31, 32, 35, 39]. Their main ideas are similar: given all devices are accessed through the local network, a dedicated manager can provide user control by allowing or blocking network packets based on user-defined policies (e.g., through Software Defined Network). These approaches, however, have a mismatch in their systems, that is, the application’s rich semantics versus the network’s poor ability to understand it. For example, one can obtain little application-related information from an IP header or other network layer headers such as that in Zigbee [46]. Therefore, it leads to coarse granularity of control. For example, SDN can block or allow the traffic to a smart door lock; however, it cannot distinguish whether a packet is to read the battery state or to unlock the door. Although there are increasingly more advanced technologies to analyze application behaviors, the network’s poor understanding is still a roadblock to realizing fine-grained control.

2.4 Our Insights

In this work, our goal is to build a home system where (i) the cloud backend is removed from the control loop totally, and (ii) users manage the whole system, including defining devices and applications’ rights to access data or to command devices.

To realize such a system, we propose a framework where user control is enforced over the network layer. However, unlike existing network-based approaches, we build our system over a network protocol that can bring application-level semantics to the network layer – Named Data Networking (NDN). By naming devices, applications, and resources with predefined conventions, security modules in our system can obtain sufficient information to make security decisions without bothering application logic. These security modules will then enforce user control for each network packet in the local system.

NDN [43] is a proposed future Internet architecture which uses application-level semantically meaningful names at the network layer. This is in sharp contrast to TCP/IP networks where communication is based on IP addresses and port numbers. With the name of data, applications can fetch this piece of data from the network without considering where the data is being host. To be more specific, the named data is carried by a Data packet while a request is called an Interest packet (Figure 1).

To secure communication, NDN supports data-centric security [45] as a new model compared with existing channel-based security (e.g., TLS). In this model, NDN packets can be signed using public key cryptography and verified regardless of where packets are coming from, thus ensuring the authenticity of the packet name and its payload. Importantly, names can be used to reason about security. For example, Yu et al. has proposed to use name relations to represent trust [42]. In our work, we also utilize the power of names and name relations to realize user’s control.

3 SYSTEM OVERVIEW

We turn to the design of Sovereign by illustrating how it supports a typical application scenario.

3.1 A Glance of Sovereign

A user Alice has a Sovereign home IoT system with a number of IoT devices and applications (which are running over

Figure 1: Interest packet and Data packet in NDN
a home hub). Unlike the situation in a cloud-based home system, Alice owns the trust anchor (i.e., root credential) of the home and can directly manage the privileges of each system participant. Alice controls the home through a local IoT controller, which is an app on her phone or a dedicated home hub. In Sovereign, all participants and resources can be uniquely identified by a name. Therefore the controller will translate Alice’s control decisions to security policies which are based on names and distribute them to the whole system. By enforcing these name-based policies, only parties authorized by Alice can access certain private data or execute certain services in the local system.

Taking a look at lower layers, the security of the system is reflected on every single network packet sent and received in the system. To be more specific, each packet can be authenticated so that no command or access can be spoofed by attackers. At the same time, private payloads carried by these packets are encrypted and only authorized parties can decrypt them.

From an IoT developer’s perspective, Sovereign provides usable tools for them to build devices or applications. Though developed in a similar style as in many existing IoT platforms, devices and applications in Sovereign are entirely owned and controlled by end users. With local user control, the chance of unauthorized remote control to the device/application is minimal.

3.2 Main Challenges

As described above, Sovereign works in a very different way from existing home IoT systems. However, to realize it, three technical questions should be taken into consideration.

• **How to name things?** In Sovereign, all devices, applications, and resources have a semantically meaningful name. These names are supposed to compose user-defined policies and be used in every NDN packet to provide sufficient information for security mechanisms to enforce user control. Therefore, how to design a set of naming conventions to satisfy these needs is a problem.

• **How to localize security modules while keeping the reliability?** In cloud-based IoT systems, security modules (e.g., bootstrapping, access control, trust management), by large, are realized on the rendezvous point – the cloud backend. When security modules function locally, following the same centralized design brings serious single point of failure problem. For example, since most home IoT systems do not have a backup rendezvous point (e.g., a backup home hub), when this device is down, end users cannot even unlock the door. In Sovereign, security modules are expected to work reliably at local level and provide security without a rendezvous point.

3.3 Sovereign’s Approach

The whole system starts from a local trust anchor – a public key certificate that contains the unique name of the home and the public key bounded with the name. To handle the three challenges in §3.2, there are three main technical components, making our system a fully user-controlled autonomous IoT system for smart home (Figure 2).

1. **Semantically meaningful names of things (§4).** We design Sovereign naming conventions to name devices, applications, and resources under the home namespace in a structured way. These names bear application level semantics and are directly used at the network layer (i.e., NDN) for content and command delivery. Furthermore, end users define security policies leveraging these conventions, e.g., whether devices under a namespace can access data under another namespace.

2. **Localized security modules functions on each system participant (§5).** In Sovereign, basic security modules, including security bootstrapping, trust management and access control, enforce user control locally. Specifically, these modules operate in a decentralized way: by delivering user defined policies to each system participant, the user control is enforced for every network packet on every sender and receiver. Even in the case when the local controller is down, the security of the system stays the same.

3. **Usability by Design-in Security (§6).** Sovereign seamlessly integrates all local security modules into a pub/sub messaging module. By calling pub/sub APIs, each device and application can automatically add/verify the security of each outgoing/incoming network packet against user-defined policies. In this way, Sovereign provides developers with easy-to-use development tools.
3.4 Assumptions

Sovereign assumes the local IoT system to be a networked system based on wireless broadcast media, e.g., WiFi, IEEE 802.15.4 [38]. Heterogeneous broadcast media can be bridged together through hardware and software, e.g., a home hub. NDN naturally supports multi-homing because the network identifiers are not bound with NICs.

Sovereign assumes the existence of one or more devices in the local system serving the role of a home controller. This can be, for example, a dedicated home controller hub or a device with sufficient storage and computation power, like a smart TV or a smart phone. There could exist multiple controllers at the same time.

We realize Sovereign as an IoT application development platform and thus we also assume all the IoT devices at home are programmed over Sovereign SDK. Later in this paper (§8.2), we show that existing pub/sub based applications can be easily migrated to Sovereign SDK and support local user control.

4 NAMING

Sovereign establishes control mechanisms over structured and application-level semantics exposed by a set of names following naming conventions. Therefore, it is important that we properly describe Sovereign’s naming conventions with explanations.

Following conventions, names in Sovereign have two properties. First, names convey application information; each name explicitly conveys what data is being fetched and what service is being called. Second, names are strictly verified; packets with erroneous names will be rejected by receiving side or intermediate devices that support Sovereign. To ensure the correctness of name verification, Sovereign follows NDN’s security principles [45] and each name is protected by packet digital signature.

Therefore, in this section, we start by describing Sovereign’s naming conventions (§4.1), and then explain how names can be authenticated (§4.2). We end this section by shedding some light on how names can be leveraged through security policies to control trustworthiness and access rights in our system (§4.3). We leave a more detailed description of control mechanisms in later sections.

4.1 Naming Conventions

Home Name: Sovereign considers each home IoT system an autonomous system with a unique namespace. To be specific, the namespace is represented in a format of “/str1_str2”. The first string is customized by end users and the second part is a randomness to ensure the uniqueness. For example, Alice’s home name can be “alice-home_eQM9QyByp” (hereafter we omit the randomness part for simplicity).

![Figure 3: Naming conventions in Sovereign.](image-url)

Importantly, there exists a public key certificate, called the trust anchor certificate, which binds the home root digital key and the home namespace together. Such a certificate is self-signed and will be securely installed by every system participant during the security bootstrapping (§5.1). The corresponding private key must be kept only by a home owner.

Under the home namespace, all devices, applications, and resources (e.g., content, command, digital keys) are named as shown in Figure 3.

Identity Name: In Sovereign, devices applications (hereafter called devices) and applications are all represented by the identity name:

```
/<home-prefix>/<service-id>/<location>/<device-or-app-id>
```

For example, a temperature sensor in Alice’s bedroom has a name “/alice-home/TEMP/bedroom/sensor-1”, and a hub-host application can be named as “/alice-home/APP/home-hub/light-control-1”. When a device provides more than one service, it has multiple identity names accordingly, where each name has a different “service-id”.

Associated with each identity name, there is also an identity key, which is a public key pair named as:

```
/<home-prefix>/<service-id>/<identity-name>/KEY/<key-id>
```

A certificate issued by the local controller binds the key name with the public key bits through a digital signature. Such a certificate is an NDN Data packet; it is fetchable by the key name and can be verified using the trust anchor certificate.

Command Name: A command to trigger certain actions on designated device(s) also has a unique name:

```
/<home-prefix>/<service-id>/CMD/<locator>/<command-id><timestamp>
```

The “<locator>” defines the effect scope of the command. It can be empty, one component, or two components; accordingly, it means the command is to command service providers in the whole system, at certain location, or a specific service provider. For example, three different commands to turn on air conditioners with different locators are shown in Table 1.

| Locator | Command ID | Effect Scope |
|---------|------------|--------------|
| /cool1  | 123456    | Whole System |
| /bedroom | 123456   | Certain Location |
| /cool1        | 123456 | Specific Service Provider |

\(^1\)A pair of < > in NDN names represents one or more name components.
A "<command-id>" is an application-defined component to identify the type of a command, e.g., "poweroff"; it will be encrypted with a per-service key in real Sovereign systems.

| turn on all air conditioners | /alice-home/AIRCON/CMD/on |
| turn on bedroom air conditioner | /alice-home/AIRCON/CMD/bedroom/on |
| turn on specific air conditioner | /alice-home/AIRCON/CMD/bedroom/ac-1/on |

Table 1: Command Names with Different Locators

Content Name: Similarly, each piece of content has a content name:

/ <home-prefix> / <service-id> / CONTENT / <producer-locator> / <content-id> / <timestamp>

The home prefix, service ID, and the producer locator can recover content producer’s identity name. For example, the temperature content "/alice-home/TEMP/CONTENT/bedroom/sensor1/temp-fahrenheit/<timestamp>" is generated by the device "/alice-home/TEMP/bedroom/sensor1". Similarly, the "content-id" is to identify the content type and will be encrypted to protect user privacy.

Access Key Names: Besides identity keys, each device/application also owns a number of access keys, which are used by the Sovereign access control module to protect sensitive data. Access keys are named as:

/ <home-prefix> / ACCESS / EKEY / <service-id> / <key-id>
/ <home-prefix> / ACCESS / DKEY / <service-id> / <key-id>

"EKEY" identifies encryption keys while "DKEY" is only used in decryption key names. "<service-id>" indicates data under which service should be protected or accessed with the key.

4.2 Verifiable Names

Sovereign builds security mechanisms over names and name relations and thus the authenticity of names is of vital importance: one must verify the name carried by a network packet is genuine.

In Sovereign, content, commands, and digital keys are kept in NDN Data packets with names introduced above and these packets are cryptographically signed when published. Importantly, the signing key name is also embedded in each Data packet with its signature. Therefore, the authenticity of the packet (including its name and signing key name) can be verified by verifying the certificate chain: A receiver of a Data packet can first fetch the public key certificate of the signing key to verify the packet signature. To further ensure the signer is truly a system member, the receiver can then verify the signature of the certificate using the trust anchor certificate, which is installed by every participant during the security bootstrapping (5.1).

4.3 Names and User Control

Based on rich-semantic and verifiable names, user-defined security policies can be represented by names and realized by restricting relations among named entities. In Sovereign, there are two types of security policies.

- Trust policies define who can issue what commands. Users grant devices and applications the privileges to command others by trust policies. Trust policies regulate relations between signing key names and command names – only identity keys with certain names can sign and issue commands of certain command names.

- Access policy defines who can access what content. They grant access rights to devices and applications by allowing certain relations between signing key names and access key names – only identity keys with certain names can sign and send requests to fetch keys of certain access key names.

A more detailed description can be found in the next section.

5 DECENTRALIZED LOCAL SECURITY MODULES

To allow users' control on content and executables in a home system, three security modules must be provided.

- Security bootstrapping: New devices/applications install the local trust anchor and obtain identity names and digital keys.

- Trust management: End users manage which devices/applications can command which services (e.g., only the home owner can unlock the door).

- Access control: End users decide which devices/applications can access which set of content (e.g., Air conditioners can access temperature content in the bedroom).

In existing IoT systems, security modules work in a centralized way: all devices connect to a rendezvous point (e.g., a cloud backend as in [2, 33]) for authentication and access control (Figure 4(a)). However, in a local IoT system, this centralized design suffers from serious single point of failure problem because of the lack of redundancy in the home environment. For example, consider a home system where all devices and applications connect to the local home hub for security verification, since most home systems do not have a backup hub, when the hub is down, all security modules are disabled and a user may not even be able to unlock the door.

Thanks to advances in hardware technologies, current commodity IoT platforms can be equipped with usable computation, storage, and dedicated crypto chips (e.g., elliptic curve chip ATECC508A), so it is possible to realize security modules at each system participant. For example, §8.4 shows that a constrained device with 64MHz CPU and 256K memory can effectively operate all security modules in Sovereign.
Therefore, in Sovereign, localized security modules function in a decentralized way. First, users bootstrap devices and applications through the local controller (5.1). In this process, each device and application installs the trust anchor and obtains the necessary keys to execute security modules later. After that, based on user-defined policies, each device/application runs the trust management module (§5.2) and access control module (§5.3) locally to secure every network packet in the system. When some devices (even the local controller) are down, the other working devices still perform well.

5.1 Bootstrapping

Bootstrapping is the prerequisite of trust management and access control since a device or an application obtains necessary names and keys in this process.

In Sovereign, a user uses the controller to bootstrap a new device through device-to-device communication (e.g., Bluetooth). The mutual trust between the device and the controller is based on certain out-of-band means such as QR code scanning. To be more specific, the following keying material is exchanged.

1. The device will install the trust anchor certificate so that it can authenticate messages published by the controller and other system members.
2. The controller certifies the device and issues certificates for the device’s identity keys. With a certified identity key, content and commands published by the device can be authenticated by other system members.
3. The device obtains encryption keys to encrypt data generated by itself and decryption keys to access data published under other services.

Protocol [21] shows a concrete construction of device security bootstrapping in Sovereign. Bootstrapping an application is similar to the aforementioned steps, whereas the out-of-band verification is through the user’s operations (e.g., installing the app from a trustworthy source).

After security bootstrapping, each device and application in the system have sufficient keys to secure every Data packet generated by itself and authenticate other packets published in the system. Authorized parties can also access to payloads of Data packets.

5.2 Trust Management

Sovereign’s trust management is based on names and public key cryptography. In a nutshell, users define trust policies by allowing certain name relations, so network packets that violate these name relations will be rejected by the receiving end. In this process, since packets are protected by their public key signatures, packet names and signer names cannot be spoofed.

Operating locally: Each device and application executes its local trust management module to sign every outgoing packet and verify all the incoming packets. This is feasible because the device/application has the trust anchor and their own identity keys have been certified by the controller. To be more specific, each content and command published under a service $S$ is signed by its producer’s identity key whose key name contains $S$. Once the receiver’s trust module receives content or a command published by other devices/applications, it, following user-defined trust policies, first verifies the signature to ensure it is generated by a valid system member. Then it checks data name and signing identity key name; if such a key is not allowed to sign this piece of named data, the packet will be dropped. Security policies, signed and published by the local controller, are distributed to devices/applications through pub/sub (§6.1).

Fine-grained Trust Policy: The trust management in Sovereign is of fine granularity thanks to the use of names. To be more specific, the user’s trust policies are represented by allowed name relations: a set of named keys are allowed to sign a set of named content or commands. All the content and commands with invalid signatures or signed by improper named keys will be rejected.

For example, we can use the name trees shown in Figure 5 to represent several different trust policies. The left tree represents two identities – a lock device “/alice-home/LOCK/livingroom/front” in the living room and an application “/alice-home/APP/hub/app-1” hosted by the home hub. Notice that each of them have a signing key with their
identity name as the prefix. The right tree represents names of related commands and content. For example, the name "/alice-home/LOCK/CMD/livingroom/front/unlock" represents the command to unlock this lock. In the figure, each bold arrow represents a policy: a key under an identity name in the left tree can sign packets with the names in the right tree.

- Policy ➊ defines that a controller can sign and issue commands with prefix "/alice-home/LOCK/CMD", which means the controller can control all the lock devices, including those in the living room and all the other rooms.
- Policies ➋, ➌, and ➍ are of three levels of granularity. They define the hub application "app-1" can control all locks in the bedroom (room wide), the "front" lock (device wide), and issue only the lock command to the "front" lock (command wide), respectively.
- Policy ➎ requires that only the lock "front" can produce the state of itself.

Notice that the name used in a policy is not necessarily a name prefix. Sovereign supports the use of regular expression in trust policies. For example, a policy that allows a key to sign named packets with name

```
/alice-home/<>/CMD/livingroom/<>*
```

means the key can be used to command any devices under any service in the living room.

### 5.3 Access Control

Similarly, the access control module in Sovereign is also based on names. In Sovereign, the payload carried by each network packet is encrypted in a data-centric way [44]: the ciphertext is not bound with a channel (e.g., TLS; any party who has the decryption key can access it.

**Operating locally:** Each system participant executes the access control model to encrypt the payload of every packet it publishes and decrypt payloads of those packets that it can access. All the access control keys have been installed after the bootstrapping process. As shown in Figure 6, depending on the user-defined access policies, the controller will reply with the requested keys or reject the request (in which case it may need end users’ decisions).

**Access Policy:** The name relations in access control policy are in the format: a set of named keys are allowed to sign the request for named encryption/decryption keys. The controller will not reply with the access keys if the request is not signed by designated identity keys. Access policies also support fine-grained control. For example, in the name trees shown in Figure 6, three bold arrows represent three access rights with different granularity.

- Policy ➊ defines that all air conditioning devices in all the rooms can sign the request for the temperature decryption key (i.e., get access to the temperature data).
- Policy ➋ represents that all air conditioning devices can access to the temperature data.
- Policy ➌ defines the device "ac-1" can access to the temperature data.

Same as trust policies, Sovereign allows the use of regular expressions in access policy as well.

### 6 USABILITY BY DESIGN-IN SECURITY

Since localized security modules need to be supported by each system participant, we need to provide an easy way for devices and applications to use them. Sovereign does not require developers to be security experts who have to dive into the security and crypto details.

To provide ease of use, Sovereign adopts pub/sub as the way of communicating in the local home system (§6.1). Importantly, since the security modules function locally and solely takes input (i.e., user-defined policies) from the local controller, all security modules can be integrated into pub/sub seamlessly (§6.2). Therefore, without any extra efforts, developers can make devices and applications that support users’ local control by simply calling pub/sub APIs (§6.3).

#### 6.1 Pub/Sub based Communication

Pub/sub is a messaging pattern where data is categorized into topics to which message producers (called publishers) publish their messages without knowing the set of message consumers (called subscribers). Similarly, subscribers receive messages under topics that are of interest. At present pub/sub has been widely used in IoT application development (e.g., SmartThings, Amazon AWS IoT). Being data-centric and name-based, NDN supports pub/sub naturally.

Based on naming conventions (§4), Sovereign treats topic identifiers as NDN name prefixes. For example, "/alice-home/TEMP/CONTENT/bedroom" can refer to the temperature in the bedroom. Following this direction, publishing content or command to a topic is to generate Data packets named under the topic prefix while subscribing to a topic is to fetch Data packets from the name prefix with Interest packets.

In this way, communication among the controller, devices, and applications can be effectively realized by pub/sub.
• **User-defined Policy Distribution**: Users configure security policies through the local controller and these policies will be published by the controller. After users configure policies, the local controller publishes policies to a topic (e.g., `/alice-home/POLICY`), to which is subscribed by all system participants by default. Published policies, signed by the private key corresponding to the trust anchor, can be verified by devices and applications in Sovereign.

• **Content Delivery**: Content is distributed among devices and applications through pub/sub. To be more specific, devices/applications wrap content into Data packets named under the corresponding topic name prefix following naming conventions (§4). These Data packets can be fetched with Interest packets carrying the topic name prefix by subscribers either in a periodic manner or immediately after hearing a notification.

• **Command Delivery**: A command is also generated by pub in a similar way as content delivery. Since commands are usually in real time, a notification will be broadcast to the system to trigger an immediate fetch.

Importantly, utilizing NDN and the local broadcast media, pub/sub in Sovereign does not require a centralized broker. This is because there is no need for the mapping between each topic and its subscribers (e.g., in MQTT [7], subscribers are identified by their IP addresses and port number). This reduces the system’s dependency on a single point: even when the controller is down, the system is still by and large secured because unauthorized operations will still be dropped at command receivers.

### 6.2 Integrate Security Modules in Pub/Sub

In Sovereign, we integrate all security modules under pub/sub (Figure 7), thus application developers only need to call our pub/sub APIs to enjoy Sovereign’s user control, without operating with our security modules.

| Application Space | Pub API | Logic in Pub | Data packet | Available to |
|-------------------|---------|--------------|-------------|-------------|
|                   | Encrypt |              | Sign        |             |
| Logic in Sub      | Decrypt | Trust Mgmt   | Verify      | Fetch from  |
|                   | cmd or content |          | Data packet | Local NDN Network |

Figure 7: Security processing flow in pub/sub

To be more specific, after security bootstrapping and with user-defined policies, pub/sub in Sovereign works as follows.

• **Pub**: When a device/application is publishing content or command, Sovereign will (i) name the content/command following naming conventions, (ii) invoke the access control module to use a matched encryption key to encrypt the content or the command, and (iii) invoke the trust management module to use a proper signing key according to trust policies to sign the packet. After that, the published data are available to the local network.

• **Sub**: After a device/application subscribes to a topic, when new updates arrive, Sovereign will (i) invoke the trust management module to verify the signature and then check the data name and signing key name against the trust policies, and (ii) invoke the access control module to decrypt the content if the subscriber has the right to access it. Eventually, only verified and decrypted payloads will be passed to application logic.

As shown above, Sovereign will automatically apply security protection to each packet generated and verify the security of each packet received.

### 6.3 An Example of Pub/Sub

We use an air conditioner control pub/sub example to illustrate the process of pub/sub along with the security modules. In our example, we have an application (`/alice-home/APP/hub/app-1`) to control the air conditioner (`/alice-home/AIRCON/bedroom/ac-1`) on certain conditions. We assume there is a policy: `/alice-home/APP/hub/app-1/KEY` can sign commands under `/alice-home/AIRCON/CMD`, and both of them have obtained the necessary signing keys and access rights during the security bootstrapping.

**Applications: Publishing Command.** The application publishes a command to set the temperature to be “70F” on air conditioners in the bedroom. With the function called, Sovereign will generate an NDN Data packet accordingly. First, following naming conventions, the packet will be named as `/alice-home/AIRCON/CMD/bedroom/set-temp/<timestamp>`.

Then the command payload, which is “70F”, will be encrypted with a encryption key with name prefix

```
/alice-home/ACCESS/EKEY/AIRCON
```

After that, according to the policy, the packet will be signed by the application’s signing key

```
/alice-home/APP/hub/app-1/KEY
```

Finally, this encrypted and signed packet will become available to the local network with a broadcast Interest as a notification.

**Device: Subscribing Command.** The air conditioner subscribes to commands on air conditioning service. Once a notification Interest arrives, the air conditioner checks the name of newly published data; since the device’s own identity name is within the command’s scope, `/bedroom`, the air conditioner will send out an Interest packet to fetch the command Data packet immediately.
After getting the command packet, the device first verifies its signature and checks the packet name and signing key name against user-defined trust policies. If the signing key does not have a valid certificate issued by the trust anchor or is not allowed to issue the command, this command will be dropped. After that, the device decrypts the content using the corresponding decryption key and finally executes the decrypted command.

7 IMPLEMENTATION

In this section, we describe how our Sovereign is realized for developers and end users. For device and application developers, Sovereign has been implemented as an open source cross-platform software development kit (SDK)\(^2\) in standard C11. Its core library contains more than 51K lines of code and has been developed and maintained for more than one and a half years. In addition, we also implement a prototype of the local controller with GUI in python for end users to bootstrap devices/applications and control the system.

Sovereign SDK (Figure 8) includes a core library and adaptations to different software and hardware platforms. The core library realizes the pub/sub module, security modules, and other supporting functionalities, while the adaptation layer makes the core library cross-platform and work with dedicated crypto chips. So far, we have successfully adapted our SDK to various platforms, including Linux/Unix, Raspbian, RIOT OS \(^6\), and Nordic NRF boards, with support of different layer protocols, including Bluetooth, IEEE 802.15.4, and legacy IP\(^3\).

8 EVALUATION

In this section, we first show that Sovereign can realize user control purely locally, addressing the security and private issues caused by relying on cloud backends (§8.1). Second, our empirical evaluations show that Sovereign provides usable tools for developers (§8.2) and does not require extra operations from end users (§8.3). Third, we evaluate Sovereign’s performance (§8.4). Compared with existing cloud-based IoT systems, Sovereign does not introduce significant system overhead. We also show that, using only a small amount of memory and flash, Sovereign can function normally on constrained devices.

8.1 Privacy and Security Assessment: Case Study and Comparison

We assess Sovereign’s privacy and security by analyzing packet flows in two real programs in Sovereign and SmartThings. Specifically, we select a typical open-source device program and an application program from official SmartThings GitHub repositories and port them into Sovereign. The pseudo codes of original programs and their Sovereign based versions are shown in Figure 9. The selected programs are simple but representative: the device program (code block 1 & 2) changes its own state to “on” when it gets a turn-on command from an authorized party; the application program (code block 3 & 4) subscribes to the state of a contact sensor and turn on the switch when the contact sensor is touched.

Bootstrapping. In the SmartThings device program (code block 1), the first line of the code connects the device to the home’s cloud backend. After that, the cloud recognizes this new device, learns its profile, and registers it to the cloud database. In contrast, the first line of the Sovereign program code (code block 2) bootstraps the device to a local controller with no traffic out of the home system. In addition, all the sensitive information transmitted in the bootstrapping protocol is protected by encryption.

Pub/Sub. The first lines in both the SmartThings application (code block 3) and the Sovereign application (code block 4) subscribe to certain topics in the home system. However, the underlying operations are very different: the SmartThings application notifies the cloud backend its interest in certain services while the Sovereign application simply starts to listen to data published under a new name prefix in the local network. Similarly, the SmartThings application program turns on the switch with “pub_content”, after which, the program sends a message to the cloud backend and the cloud backend then sends another command back to turn on the home switch. In contrast, publishing a command in Sovereign application is nothing more than making a new Data packet available in the local network.

The comparison shows that, Sovereign does not expose any control information to remote parties while at the same time achieves the same goals as in cloud-based systems. Since Sovereign encrypts and signs each packet in the local network, it is difficult for a remote or a local (e.g., attackers in neighborhood) attacker to forge commands or to learn private information by eavesdropping the traffic.

\(^{2}\)https://github.com/sovereign-home/sovereign-core

\(^{3}\)NDN considers IP/TCP/UDP as link layer protocols
8.2 Usability: Software Development

Sovereign provides developers with pub/sub APIs, allowing developers to easily port IoT device and application programs from existing pub/sub based platforms to Sovereign. Figure 9 shows the line-by-line comparison between the original SmartThings programs and their Sovereign based versions. The comparison and our experience show that, though some trivial modifications (e.g., language change, pub/sub function names and parameters change) are needed, the migration does not require developers to change the main logic of a program.

8.3 Usability: User Experience

Moving control from the cloud backend to the local network, Sovereign does not necessarily need to degrade user experiences. To illustrate how end users potentially use our system, we perform an integration test using a Raspberry Pi 3 mode B+ (RPI) as an IoT device and a laptop as the controller, and we count the major operations required by end users to add a new device or to issue a command to a device.

As shown in Table 2, though UIs are not optimized to be as user-friendly as a real product, Sovereign does not require more manual operations from end users compared with current industrial smart home IoT solutions.

8.4 Performance

We show that Sovereign does not pose extra latency overhead and storage/memory overhead to the system even though all security operations are local.

Latency of Common Operations. We provide the overall time consumption to finish common operations in Sovereign and AWS IoT, including device bootstrapping, content delivery (by subscribing to a service), and command delivery (by publishing a command). We first compare the overall latency of communicating with a RPI (ARM cortex A53 @1.4GHz) device in AWS IoT and in Sovereign. As shown in the left two bars in Figure 10, though security operations are done by the local system, Sovereign does not need the long round trip time to the cloud and thus has a similar or even lower latency: Sovereign is 6% slower, 62% faster, and 42% faster than AWS IoT in device bootstrapping, content delivery, and command delivery, respectively. We also test Sovereign over a more constrained device, NRF52840 (ARM Cortex M4).
cortex M4 @64MHz), over the IEEE 802.15.4 link layer protocol. As shown in the right bar in the figure, because of the low computation power, it takes longer time than using a RPI to finish measured operations. Nevertheless, the latency of bootstrapping (which only needs to be done for one time) is still less than 1 second and commonly-used pub/sub only takes less than 0.5 seconds, which is practical enough to be deployed in real systems.

Execution Time Breakdown. We provide the breakdown of execution time of applications and devices developed with Sovereign SDK using different levels of hardware: a core i7 laptop (as the home hub), a RPI, and an NRF52840. The micro operations being measured include digital signature signing and verification (e.g., ECDSA), content encryption and decryption (e.g., AES CBC), policy checking, NDN packet encoding/decoding, and other cryptographic operations (e.g., ECDH, KDF). Figure 11 shows the results obtained from bootstrapping process and pub/sub. Our results show that the most time consuming operations are asymmetric cryptographic operations, especially on a constrained device. However, the execution time can be much reduced when dedicated crypto chips are used.

ROM and RAM Use. We also evaluated the ROM and RAM use of IoT devices programmed over Sovereign. The hardware platform we used is NRF52840, which only has 1MB ROM, 0.25MB RAM, and a 32-bit Cortex M4@64MHz CPU. Table 3 shows the RAM and ROM use of a subscriber and a publisher programmed in Sovereign over RIOT OS [6]. A detailed breakdown of memory and flash use is also shown in the table. We show that Sovereign can successfully operate with all main modules mounted and only uses <70KB memory and flash on a constrained device.

Table 3: ROM and RAM Consumption

| Program/Modules       | ROM Use  | RAM Use  |
|-----------------------|----------|----------|
| Subscriber in total   | 62KB     | 47.3KB   |
| Publisher in total    | 52.4KB   | 38.2KB   |
| Application           | 1.8%     | 7.3%     |
| High-level Modules    | 20.7%    | 34.2%    |
| Utilities             | 3.3%     | 14.4%    |
| Crypto Tools          | 25.1%    | 0.2%     |
| Network Forwarder     | 24.1%    | 25.0%    |
| OS and Adaptation     | 25.1%    | 18.9%    |

Figure 11: Breakdown of the execution time of computational phases in Sovereign Device/Application

9 FREQUENTLY ASKED QUESTIONS

Does Sovereign eliminate the use of cloud?

No. Sovereign gets rid of the cloud backend of a home system to let end users fully control the trust anchor and define policies. However, this does not mean eliminating the use of cloud services. Rather, Sovereign can make use of cloud resources to better serve end users. First, cloud services can and should be used as backup storage for user data. Sovereign has the data encrypted at production, using cloud storage resources while keeping data confidentiality. Second, a cloud-hosted application can also be used as a Sovereign application. By establishing secure connectivity from cloud to the home network through the home hub (which is an NDN forwarder), a remote application acts like a hub-hosted application and its rights to access resources or devices can only be granted by the local controller. Third, a cloud service should also be used when the user is not at home. In this scenario, to send a message to a home without a public static IP address, the cloud service can be used as a bridge to tunnel through Network Address Translation (NAT). Notice that a cloud server in this context only serves as a forwarder for already secured NDN packets, without getting confidential information from the packets.

What major challenges remain in realizing a secure and user-controlled home system?

Building secure and user-controlled home systems requires efforts from different aspects. First, applications must be correctly designed and implemented. Currently, this is done by the IoT service providers, e.g., Apple’s MFI [4]. However, supporting an open market would desire third party trusted organizations to take the responsibility. Second, today’s stove pipe solutions have occupied a big market, where IoT ecosystems are usually incompatible with each other and so are devices. A home user may own several different IoT systems at the same time. Recent trend [27] to increase compatibility among smart home products shows the right direction to go.

Does Sovereign require NDN deployment outside the local home network?

No. Sovereign deploys NDN at the local home network and does not assume the wide deployment outside the local network. In fact, even the deployment of NDN at the local home...
network is transparent to developers and end users because it is automatically configured by Sovereign SDK.

10 CONCLUSION

Our research aims to show that user control in home IoT system can be realized without relying on a cloud backend. Fundamentally different from today’s dominating cloud-based system model, in Sovereign, control of the home is in the hands of end users. Through our work, we show that to really control the home, end users must own the system trust anchor. We also identify the advantages of using semantic names to realize user control: these names help the system to understand application behaviors and thus to better enforce user control.

REFERENCES

[1] Alibaba Inc. 2020. Alibaba Cloud: Internet of Things. (2020). https://www.alibabacloud.com/solutions/IoT
[2] Amazon Inc. 2020. AWS IoT - Connected Home. (2020). https://aws.amazon.com/iot/solutions/connected-home/
[3] Apple Inc. 2020. HomeKit. (2020). https://www.apple.com/ios/home/
[4] Apple Inc. 2020. MFi Program. (2020). https://developer.apple.com/programs/mfi/
[5] Noah Apthorpe, Dillon Reisman, Srikanth Sundaresan, Arvind Narayanan, and Nick Feamster. 2017. Spying on the Smart Home: Privacy Attacks and Defenses on Encrypted IoT Traffic. (2017). arXiv:cs.CR/1708.05044
[6] Emmanuel Baccelli, Oliver Hahn, Mesut Gunes, Matthias Wählisch, and Thomas C Schmidt. 2013. RIOT OS: Towards an OS for the Internet of Things. In 2013 IEEE conference on computer communications workshops (INFOCOM WKSHPS). IEEE, 79–80.
[7] Andrew Banks, Ed Briggs, Ken Borgendale, and Rahul Gupta. 2019. MQTT Version 5.0. OASIS Standard. https://docs.oasis-open.org/mqtt/mqtt/v5.0-os/mqtt-v5.0-os.html (March 2019).
[8] Tim Berners-Lee et al. 2020. Solid. (2020). https://solid.mit.edu/
[9] Z Berkay Celik, Gang Tan, and Patrick D McDaniel. 2019. IoTguard: Dynamic Enforcement of Security and Safety Policy in Commodity IoT. In NDSS.
[10] Eun Kyoungh Choi, Sunny Consolvo, Jaeyeon Jung, Beverly Harrison, and Julie A Kientz. 2011. Living in a glass house: a survey of private moments in the home. In Proceedings of the 13th international conference on Ubiquitous computing. 41–44.
[11] Computer Business Review. 2020. Google Cloud in Major Global Outage: Numerous Services Fail. (2020). https://www.chronline.com/news/google-cloud-down
[12] DailyMail. 2020. Data leak by smart home device company Wyze exposes personal details of 2.4 million users including email addresses and health data. (2020). https://www.dailymail.co.uk/sciencetech/article-7835713/Data-leak-smart-home-device-company-Wyze-exposes-personal-details-2-4-million-users.html
[13] Soteris Demetriou, Nan Zhang, Yeonjoon Lee, XiaoFeng Wang, Carl A Gunter, Xiaoyong Zhou, and Michael Grace. 2017. HanGuard: SDN-driven protection of smart home WiFi devices from malicious mobile apps. In Proceedings of the 10th ACM Conference on Security and Privacy in Wireless and Mobile Networks. 122–133.
[14] Earlene Fernandes, Jaeyeon Jung, and Atul Prakash. 2016. Security analysis of emerging smart home applications. In 2016 IEEE Symposium on Security and Privacy (SP). IEEE, 636–654.
[15] Forbes. 2020. Confirmed: 2 Billion Records Exposed In Massive Smart Home Device Breach. (2020). https://www.forbes.com/sites/daveywinder/2019/07/02/confirmed-2-billion-records-exposed-in-massive-smart-home-device-breach
[16] Google Inc. 2020. Google Cloud IoT. (2020). https://nest.com/
[17] D. Hardt. 2012. The OAuth 2.0 Authorization Framework. RFC 6749. RFC Editor. http://www.rfc-editor.org/rfc/rfc6749.txt http://www.rfc-editor.org/rfc/rfc6749.txt
[18] Home Assistant Project. 2020. Home Assistant. (2020). https://www.home-assistant.io/
[19] iPropertyManagement.com. 2020. Smart Home Statistics. (2020). https://ipropertymanagement.com/research/IoT-statistics
[20] Yunhan Jack Jia, Qi Alfred Chen, ShiQi Wang, Amir Rahmati, Earlene Fernandes, Zhiqiuong Morley Mao, Atul Prakash, and Shanghai Jiao-Tong University. 2017. ContextIoT: Towards Providing Contextual Integrity to Appified IoT Platforms... In NDSS.
[21] Y. Li, Z. Zhang, X. Wang, E. Lu, D. Zhang, and L. Zhang. 2019. A Secure Sign-On Protocol for Smart Homes over Named Data Networking. IEEE Communications Magazine 57, 7 (July 2019), 62–68. https://doi.org/10.1109/MCOM.2019.1800789
[22] Huichen Lin and Neil W Bergmann. 2016. IoT privacy and security challenges for smart home environments. Information 7, 3 (2016), 44.
[23] Microsoft Inc. 2020. Azure IoT. (2020). https://azure.microsoft.com/en-us/overview/IoT/
[24] Mehdi Nobakht, Vijay Sivaraman, and Roksana Boreli. 2016. A host-based intrusion detection and mitigation framework for smart home IoT using OpenFlow. In 2016 11th International conference on availability, reliability and security (ARES). IEEE, 147–156.
[25] openHAB Community and Foundation. 2020. openHAB. (2020). https://www.openhab.org/
[26] Philips Hue. 2020. How do I connect my Hue lights to Alexa? (2020). https://www2.meethue.com/en-us/support/friends-of-hue
[27] Project Connected Home over IP Working Group. 2020. Project Connected Home over IP. (2020). https://www.connectedhomeip.com/
[28] Jingjing Ren, Daniel J Dubois, David Choftines, Anna Maria Mandalari, Roman Kolcum, and Hamed Haddadi. 2019. Information exposure from consumer iot devices: A multidimensional, network-informed measurement approach. In Proceedings of the Internet Measurement Conference. 267–279.
[29] Samsung. 2020. SmartThings Smart Bulb. (2020). https://www.samsung.com/us/smart-home/smartthings/bulbs/smartthings-smart-bulb-gp-lbu019bbawu/
[30] Wentao Shang, Zhehao Wang, Alexander Afanasiev, Jeff Burke, and Lexia Zhang. 2017. Breaking out of the cloud: Local trust management and rendezvous in Named Data Networking of Things. In Proceedings of the Second International Conference on Internet-of-Things Design and Implementation. 3–13.
[31] Anna Kornfeld Simpson, Franziska Roesner, and Tadayoshi Kohno. 2017. Securing vulnerable home IoT devices with an in-hub security manager. In 2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops). IEEE, 551–556.
[32] Vijay Sivaraman, Hassan Habibi Gharkheili, Arun Vishwanath, Roksana Boreli, and Olivier Mehani. 2015. Network-level security and privacy control for smart-home IoT devices. In 2015 IEEE 11th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob). IEEE, 163–167.
[33] SmartThings Inc. 2020. SmartThings. (2020). https://www.smartthings.com/
[34] The Economist. 2020. The world’s most valuable resource is no longer oil, but data. (2020). https://www.economist.com/leaders/2017/05/06/the-worlds-most-valuable-resource-is-no-longer-oil-but-data
[35] Yuan Tian, Nan Zhang, Yueh-Hsun Lin, XiaoFeng Wang, Blase Ur, Xianzheng Guo, and Patrick Tague. 2017. Smartauth: User-centered authorization for the internet of things. In 26th USENIX Security Symposium (USENIX Security 17). 361–378.

[36] Samuel Tweneboah-Ruduaa, Knud Erik Skouby, and Reza Tadayoni. 2017. Cyber security threats to IoT applications and service domains. Wireless Personal Communications 95, 1 (2017), 169–185.

[37] Blase Ur, Jaeyeon Jung, and Stuart Schechter. 2013. The current state of access control for smart devices in homes. In Workshop on Home Usable Privacy and Security (HUPS), Vol. 29. HUPS 2014, 209–218.

[38] Wireless Personal Area Network (WPAN) Working Group. 2016. IEEE Standard for Low-Rate Wireless Networks. IEEE Std 802.15.4-2015 (Revision of IEEE Std 802.15.4-2011) (April 2016), 1–709. https://doi.org/10.1109/IEEESTD.2016.7460875

[39] Ke Xu, Xiaoliang Wang, Wei Wei, Houbing Song, and Bo Mao. 2016. Toward software defined smart home. IEEE Communications Magazine 54, 5 (2016), 116–122.

[40] Moosa Yahyazadeh, Proyash Podder, Endadul Hoque, and Omar Chowdhury. 2019. Expat: Expectation-based policy analysis and enforcement for appified smart-home platforms. In Proceedings of the 24th ACM Symposium on Access Control Models and Technologies. 61–72.

[41] Yonhap News Agency. 2020. Over 110 mln people have downloaded Samsung’s IoT app. (2020). https://en.yna.co.kr/view/AEN20200108006700320

[42] Yingdi Yu, Alexander Afanasyev, David Clark, KC Claffy, Van Jacobson, and Lixia Zhang. 2015. Schematizing trust in named data networking. In Proceedings of the 2nd ACM Conference on Information-Centric Networking. 177–186.

[43] Lixia Zhang, Alexander Afanasyev, Jeffrey Burke, Van Jacobson, Patrick Crowley, Christos Papadopoulos, Lan Wang, Beichuan Zhang, et al. 2014. Named data networking. ACM SIGCOMM Computer Communication Review 44, 3 (2014), 66–73.

[44] Zhiyi Zhang, Yingdi Yu, Sanjeev Kaushik Ramani, Alex Afanasyev, and Lixia Zhang. 2018. NAC: Automating access control via Named Data. In MILCOM 2018-2018 IEEE Military Communications Conference (MILCOM). IEEE, 626–635.

[45] Zhiyi Zhang, Yingdi Yu, Haitao Zhang, Eric Newberry, Spyridon Mastorakis, Yanhiao Li, Alexander Afanasyev, and Lixia Zhang, 2018. An overview of security support in Named Data Networking. IEEE Communications Magazine 56, 11 (2018), 62–68.

[46] Zigbee Alliance,. 2020. Zigbee Alliance. (2020). https://zigbeealliance.org/

A THE ROLE OF THE CLOUD BACKEND IN EXISTING HOME IOT SYSTEMS

Amazon AWS IoT: AWS IoT consists of three main components: cloud-connected devices, cloud-hosted applications, and the AWS Cloud. In an AWS IoT home system, devices and applications must connect to the AWS Cloud through TLS with mutual authentication. Therefore, each device and application must install two public key certificates in advance: one is AWS’s certificate as the trust anchor and the other one is device/application certificate, which was issued when developers registered their products at AWS. The cloud serves as the message broker and the authority to manage the system – any unauthorized access to home resources will be rejected by the cloud. Though the recent AWS Greengrass framework encourages local communication, the management is still realized at the cloud.

Google IoT Core: Google IoT Core defines a device as a processing unit that is capable of connecting to the Internet and exchanging data with the cloud backend. Each device registry is created in a specific cloud region and belongs to a cloud project. The device configuration is done by a user-defined blob sent from cloud. Similarly to AWS, trust management and access control under this framework go through cloud, leveraging the Identity and Access Management module, using the cloud as a rendezvous point.

Microsoft Azure IoT: Azure IoT uses the cloud to interact with individual devices. When receiving data, analysis will be performed at cloud side which is connected to other Azure cloud services. In D2D scenario, cloud will also act as a message broker between devices. In the middle of cloud back-end and devices, a predefined cloud gateway called Azure IoT Hub is involved. Azure IoT Hub has the capability of identity management for devices. When connecting to the cloud, the device and cloud will be mutually authenticated by a TLS-based handshake.