Sovereign: Self-Contained Smart Home With Data-Centric Network and Security

Zhiyi Zhang, *Member, IEEE*, Tianyuan Yu, *Graduate Student Member, IEEE*, Xinyu Ma, Yu Guan, Philipp Moll, and Lixia Zhang, *Life Fellow, IEEE*

***Abstract*—Recent years have witnessed the rapid deployment of smart homes; most of them are controlled by remote servers in the cloud. Such designs raise security and privacy concerns for end users. In this article, we describe the design of Sovereign, a home Internet of Things (IoT) system framework that provides end users complete control of their home IoT systems. Sovereign lets home IoT devices and applications communicate via application-named data and secures data directly. This approach enables direct, secure, one-to-one, and one-to-many Device-to-Device communication over wireless broadcast media. Sovereign utilizes semantic names to construct usable security solutions. We implement Sovereign as a publish–subscribe-based development platform together with a prototype home IoT controller. Our preliminary evaluation shows that Sovereign provides a systematic, easy-to-use solution to user-controlled, self-contained smart homes running on existing IoT hardware without imposing noticeable overhead.**

***Index Terms*—Internet of Things (IoT), named data networking (NDN), network security, smart home.**

I. Introduction

TECHNOLOGY advances lead to improvements in home living. For example, appliances such as dishwashers and refrigerators, which are directly controlled by home users, improve the quality of life at home. Over the last decade, Internet of Things (IoT) has come of age [1], and networked smart devices at home lead to home reinvention. However, different from conventional home appliances, in most of today's popular smart home solutions [2]–[7], the smart home devices are controlled by cloud-based servers running by the smart home service providers.

Given home devices are remotely controlled by the cloud, a fundamental concern with today's smart home deployment practice is the exposure of users' daily home life operations to these service providers in the cloud [8]–[12]. To control one's home, a user contacts the cloud backend, which authenticates the user, and then authorizes her to issue commands through the cloud to her home appliances. This effectively makes a user an authorized client, rather than the controller, of her own home.

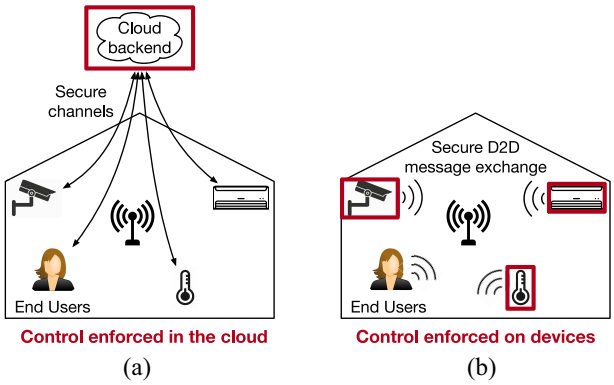

Fig. 1. Current practice versus Sovereign. (a) Cloud-based smart home. (b) Sovereign.

Many previous efforts have been devoted to provide better user privacy protection from the cloud service providers, and stronger user control of home devices and data [13]–[22]. These approaches usually apply cryptographic schemes like identity-based encryption [13] over the existing data flow, or introduce new control layers such as software-defined network (SDN) [17], or add auditing services [14].

With the full recognition of today's successful cloud-based practice and ongoing efforts to enhance the security and privacy of cloud-based services, we propose to meet the end users' privacy requirements by rethinking smart home realization: let the control of smart homes stay at home. More specifically, we propose to build smart homes as independent and self-contained systems solely owned and controlled by end users, whose operations do not rely on cloud or other external services. We do not prohibit the use of cloud services; rather, we propose that all usages of external resources (e.g., remote backup, intensive computation, and data retrieval) should be authorized and controlled by the local smart home systems.

In this article, we describe the design of *Sovereign*—a framework for self-contained smart home systems. Fig. 1 provides a conceptual level comparison between the current practice and Sovereign. Sovereign lets each home operate as an autonomous system, where all the entities belonging to

Manuscript received 16 July 2021; revised 12 October 2021 and 30 November 2021; accepted 7 January 2022. Date of publication 21 January 2022; date of current version 25 July 2022. This work was supported in part by the National Science Foundation under Award CNS-1629922 and Award CNS-1719403. *(Corresponding author: Zhiyi Zhang.)*

This work involved human subjects or animals in its research. The authors confirm that all human/animal subject research procedures and protocols are exempt from review board approval.

Zhiyi Zhang, Tianyuan Yu, Xinyu Ma, Philipp Moll, and Lixia Zhang are with the Computer Science Department, University of California at Los Angeles, Los Angeles, CA 90095 USA (e-mail: zhiyi@cs.ucla.edu; tianyuan@cs.ucla.edu; xinyu.ma@cs.ucla.edu; phmoll@cs.ucla.edu; lixia@cs.ucla.edu).

Yu Guan is with the School of Electronics Engineering and Computer Science, Peking University, Beijing 100871, China (e-mail: shanxigy@pku.edu.cn).

Digital Object Identifier 10.1109/JIOT.2022.3144980

the smart home system (i.e., devices and local or remote applications) are managed locally. To enable reliable and secure communication in the home environment, Sovereign takes the approach of data-centric networking and security. First, every home entity and all available resources, i.e., content, services, keying material, and security policies, are identified by hierarchical and semantically meaningful names. Second, the entities in a home exchange messages with each other directly over the broadcast wireless network, without going through any centralized message broker. At the same time, security policies that regulate named entities' access to named resources are enforced by individual devices instead of by a remote cloud server.

Sovereign is implemented as a software development kit (SDK) [23] that provides developer-friendly application programming interfaces (APIs) to smart home device manufactures and application developers. Programming a device or an application with the Sovereign framework enables it to connect to the localized smart home system and to be controlled by home users without external dependencies. Users control their homes through user interfaces (UIs) provided by the smart home applications programmed over Sovereign. The application developments, e.g., human–computer interaction (HCI) or specific home security policy definitions, are beyond the system framework, thus not covered in this article.

Contributions: Our contributions are twofold. First, we design and prototype Sovereign, demonstrating an alternative approach to building smart home systems that can fully protect user privacy. Our prototype offers a proof of evidence that a smart home can indeed be built as an independent and self-contained system, with end-to-end security to enhance system security and direct device-to-device (D2D) message exchanges to increase resilience and to overcome single points of failure.

Second, we implement the open-source and cross-platform Sovereign SDK. As part of the SDK, we provide a lightweight named data networking (NDN) implementation that works smoothly with constrained IoT devices. Along with the SDK, we also provide a python-based prototype controller application to let end users control their homes. Our evaluation shows that Sovereign is developer-friendly and exhibits low network latency, low computation overhead, and a small resource footprint that enables Sovereign to run on constrained devices.

Outline: The remainder of this article starts with our motivation and Sovereign's design principles in Section II. We then describe the Sovereign framework in Section III and outline its implementation, including Sovereign's SDK with usable APIs, in Section IV. We provide the evaluation results in Section V, talk about related works in Section VI, and give some discussions in Section VII. Finally, we conclude our work in Section VIII.

II. Motivation

A. Cloud-Based Smart Home: Your Privacy Is in the Cloud

Today's dominating IoT frameworks center around the cloud. Popular cloud-based smart home platforms, such as Samsung's SmartThings [2] and Amazon's AWS Home IoT [4], account for a large market share. For example, according to Samsung, SmartThings has 120 million active users worldwide by the end of 2020 [24].

To understand the role played by the cloud in a smart home system, we analyze how Samsung SmartThings works as an example. The SmartThings ecosystem [25] has three main components: 1) the SmartThings cloud; 2) cloud-connected devices; and 3) cloud-hosted applications. SmartThings requires that device vendors and application developers preinstall the SmartThings cloud certificate into their products and register their products to the SmartThings cloud in advance, so that SmartThings can recognize their unique IDs and public keys. To add a new device to a smart home system, the homeowner performs a simple operation, e.g., scanning a QR code or pressing a button on the device. This operation pairs the device with the user's account in the cloud backend, authenticating the device to the homeowner's smart home system. As shown in Fig. 1, all devices and applications in a home system establish secure connections, either directly, e.g., through transport layer security (TLS), or indirectly, e.g., via a WiFi-connected home hub, to the SmartThings cloud, where they take control commands and communicate with each other. Importantly, the cloud manages access to all the home entities using open authorization (OAuth) [26]. An application or device must first obtain OAuth tokens from the cloud before it can access other home resources.

To summarize, the SmartThings cloud backend 1) holds the trust anchor (i.e., cloud's certificate) and maintains the identities of all the devices and applications in a smart home; 2) serves as a rendezvous point for data exchange among devices and applications; and 3) executes security policies (also called control policies) by controlling access to home resources. In addition, the cloud is also used as storage for smart home application data in general. Similar designs are observed in other cloud-based home systems, including AWS Home IoT [4], Google Assistant Smart Home [27], and Microsoft's Azure IoT system [6] (see details in Appendix A).

These cloud-based systems expose home users' privacy to the cloud backend: the cloud can view all user commands and data exchange from the home system.[1] When home application data, such as video camera footage, is stored in the cloud, it adds further privacy concerns [11], [28], [29]. In addition to user privacy concerns, the "control by cloud" design increases the overall system complexity when multiple cloud backends get involved. For example, to automate SmartThings devices with IFTTT [30], users need to grant IFTTT access right to the SmartThings cloud backend, which introduces additional cross-cloud data exchange and security configuration (e.g., through OAuth). Furthermore, the home systems' dependency on the cloud backends provided by tech giants further intensifies Internet consolidation [31]. Finally, although today's cloud services seem reliable in general, the last few years did witness

[1]On today's home IoT market, Apple's HomeKit [3] supports local communication among devices and allows users to directly control their home IoT system, reflecting a shared goal with our work. However, HomeKit is not cloud independent, as it still relies on the cloud for device authentication and identity management.

several large-scale cloud service outages, each time disrupting large numbers of users [32], [33].

We note that there do exist smart home products that are locally controlled and operated. However, these systems lack features, such as secure communication between devices, or exhibit a single point of failure, as we discuss in Section VI.

B. Letting Smart Home Stay Home

Recent years witnessed great successes of cloud computing which takes advantage of economy of scale. Therefore, it is natural that smart home developments latched to the readily available cloud infrastructure for quick development and deployment. However, when examining the details, controlling home IoT systems from the cloud does not seem to share the same benefits as resource-intensive applications. Although a home system may need to run computation-intensive apps, e.g., face recognition, which can benefit from cloud assistance, the control logic of a smart home does not need heavy computation and could be done locally, especially as smart home devices become cheaper and more powerful over time. Controlling smart homes from the cloud seems mostly driven by business incentives rather than a technical necessity.

From a technical point of view, bringing the smart home control from the cloud to the home provides the most effective solution to enhance user privacy. To figure out the needed functional support, Section II-A already identified the cloud's role in smart homes: 1) the cloud hosts a smart home controller; 2) individual home entities communicate with each other by using the cloud as a message broker; and 3) security is provided by the secure connections from individual devices to the cloud using TLS.

Correspondingly, the first requirement for building a local-home-based smart home system is a local controller that takes the user's configuration and manages the home accordingly. The second requirement is enabling communication between smart devices and applications (hereafter *entities*). Finally, after moving the control out of the cloud, smart homes must be protected from local surrounding adversaries, i.e., one must add strong security and privacy protection to local communications in the smart home system.

Addressing the above-mentioned requirements brings up the following identified challenges. Setting up a controller at home differs from running a cloud-based controller because the local controller can be a single point of failure. Communication between individual home entities, on the other hand, can be trivially achieved by supporting direct D2D communication, assuming each home is connected by a WiFi wireless network that is broadcast in nature, but applying fine-grained security to D2D communication is a challenge.

C. Design Choices

We identified mitigating the single point of failure and providing secure D2D communication as the challenges in realizing a self-contained home system. The first one includes two cases: 1) the controller fails or 2) gets compromised. One can mitigate the failure by providing home entities the ability for secure message exchange without involving the controller; mitigating the compromise can also be achieved, which we discuss in Section VII. To achieve secure D2D communication, we see the following two possibilities: 1) a channel-based solution, e.g., utilizing TLS, or 2) a data-centric solution, as provided by information-centric networking (ICN).

Channel-based communication and its security model have been the default choice in network system design for many years. However, it is not necessarily the best fit for all use cases. Deploying channel-based security in a smart home scenario requires all home entities to set up bidirectional TCP/IP connections, secured using TLS, which introduces limitations and complexity in various aspects. First, TLS does not utilize the broadcast/multicast capabilities of wireless home networks. For example, to turn on all the lights at home, a simple way would be to issue a verifiable command through multicast. However, with TLS, the command must be sent through each TLS channel to all the lights. TCP/IP connections also require a mapping between application-level identities and network identifiers, i.e., IP addresses, and maintaining such mappings requires additional synchronization services. Moreover, executing control policies (e.g., the air conditioner (AC) can access temperature data) requires application semantics, which must be either installed at all the entities or otherwise has to be performed by the controller.

In contrast, data-centric networking and its security model provide a new option: each piece of data is identified by a semantically meaningful name, and security is carried in the data instead of on the channel. This data-centric networking model is realized by the NDN architecture. We introduce the key function of NDN below to prepare readers ready with data-centric security in the next section.

In NDN, applications pull data from the network by data names. A request for data, called *Interest* packet, carries the name of the desired Data. Packets carrying the fetched data are called *Data* packets. When produced, each Data packet is secured by the producer application by adding a digital signature generated by the producer's private key. The signature allows data consumers to verify the authenticity of every received Data packet, so that a Data packet can be retrieved not only from the producer but also from storage components or in-network caches. NDN directly uses application layer names for data fetching, where the names are semantically meaningful and follow a hierarchic structure in general, similar to uniform resource locators (URLs) used in today's Internet. As Fig. 2 shows, both NDN packet types carry data names; Data packets also carry the requested payload and a cryptographic signature. To facilitate data retrieval, applications make use of *naming conventions*, i.e., well-established naming patterns, to help consumers construct names of desired data. As an example naming convention, the temperature produced in the bedroom of Alice's home can be named as "`/alice/temp/bedroom`." This allows a potential requester to fetch data without having to discover the name first.

The properties of NDN (and ICN in general) enable secure D2D communications by securing data directly. In a home network, an entity can directly fetch desired data by sending Interest packets to the local broadcast network, eliminating the

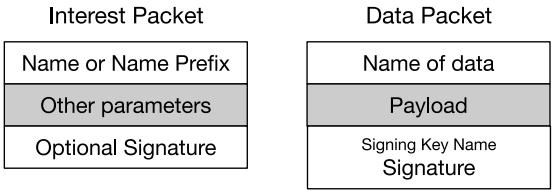

Fig. 2. Interest and data packets in NDN.

additional layer of indirection between IP address and application layer data identifiers in a TCP/IP network. Security is realized in a data-centric way by letting the data producers sign and encrypt the data using their keys. In this way, a piece of secured data can be fetched and verified by multiple entities while only those who have sufficient keys can access the payload. The semantically meaningful name carried in each message can be directly used to construct and enforce security policies (see Section III-B for an example).

III. Design of Sovereign

As stated, today's cloud-based smart home design puts trust, control, and security to the cloud. In Sovereign, this is fundamentally changed by bringing the trust anchor to the local network. After being bootstrapped by the local controller, individual entities enforce security policies by directly securing the D2D communication. Specifically, producers will sign and encrypt each piece of data being produced and consumers will authenticate and decrypt, if authorized, every piece of data being consumed. As such, Sovereign removes the single rendezvous point (i.e., the local controller) since the data-centric communication takes place directly on the local broadcast network among home entities.

Importantly, both networking and security in Sovereign center around names.

1) Content fetching is implemented by sending requests carrying the content name and service invocation is realized by issuing commands carrying the service name.
2) Security is implemented by expressing constraints on different names. For example, service access control is to constrain which named entities can issue commands to which named services.

Our new framework impacts the way of networking and security but is not supposed to affect the application logic, and hence, does not change end users' experiences of using smart home compared with today's cloud-based systems: a home user Alice only does minimal operations to set up the whole system and adding new entities, e.g., by scanning a QR code mounted on a device with the controller. Importantly, Alice controls her home by deciding the rules of the system through UIs provided by the local controller.[2]

In this section, we first use an example to provide an overview of Sovereign's system design and then introduce individual components from an entity's perspective.

[2]We acknowledge that correctly configuring security policies possibly goes beyond the capabilities of most homeowners. However, the support of cleverly designed UIs (not in the scope of this work) can ease the configuration process, e.g., by providing default policies with best practices.

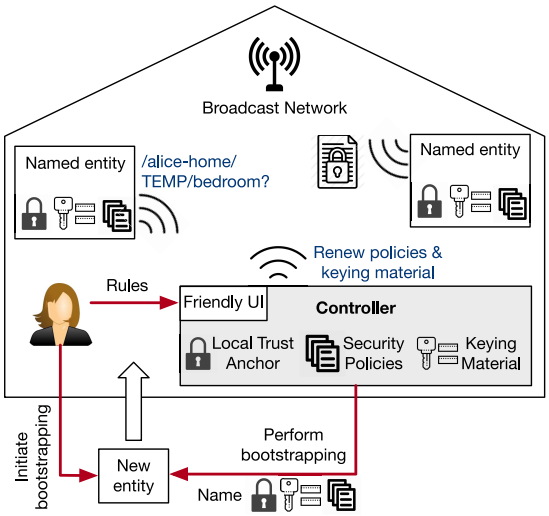

Fig. 3. High-level overview of the Sovereign home system.

A. Overview

Fig. 3 visualizes the high-level overview of Sovereign. Let us assume the user Alice installs a new AC in her home and wants to connect it to Sovereign. Therefore, Alice uses the controller to scan a QR code mounted on the AC, which initiates the entity bootstrapping process (see Section III-C). In this process, the device registers itself to the controller and the controller 1) installs the system trust anchor to the device; 2) sends keying material and security policies needed by the device; as well as 3) assigns a name to the device according to the naming conventions (see Section III-B). In Sovereign, the security policies are based on names and are converted from user-decided rules by the controller. In addition, the entity's name is associated with a public-key pair whose public key will be certified by the controller, and the private key is kept secret by the AC.

After joining the system, the AC starts running by fetching the temperature data and setting the target temperature accordingly. In addition, the AC will also close bedroom windows when it is running. In Sovereign, the temperature data fetching and window command sending all take place directly between the involved devices over the broadcast network in the form of NDN Interest–Data exchange. To get temperature data, the entity directly broadcasts the desired data name so that any other entities who have the data can reply. After fetching the Data packet carrying the temperature, the AC will first verify the packet is signed by another home entity, whose certificate is endorsed by the trust anchor. Then, to access the payload, the AC needs to decrypt it using keys obtained from the controller. When issuing a command to the window, the AC will also name this command according to the naming convention, encrypt the payload, and sign it with its own private key. Therefore, windows in the system can fetch the command by the name. If the AC is allowed by the security policies to generate such a command, home windows will successfully verify, decrypt, and execute the command. A detailed description of these workflows is provided in Section III-D.

TABLE I
NAMING CONVENTIONS OF ENTITIES AND DATA (I.E., SERVICES, CONTENT, AND KEYING MATERIAL)

Type	Naming Convention
Home entity (*i.e.,* device or application)	/<home-prefix>/<service>/<location>/<entity-id>* *e.g.,*/alice-home/AirCon/bedroom/north-ac-1
Command to executable services	/<home-prefix>/<service>/<**scope**>/CMD/<cmd-id>** *e.g.,*/alice-home/AirCon/**bedroom/north-ac-1**/CMD/set-temp – device-level command *e.g.,*/alice-home/AirCon/**bedroom**/CMD/set-temp – room-level command *e.g.,*/alice-home/AirCon/CMD/set-temp – home-level command
Data produced by services	/<home-prefix>/<service>/CONTENT/<location>/<entity-id>/<content-id>** *e.g.,*/alice-home/TEMP/CONTENT/bedroom/senor-1/temp
Encryption/Decryption Key	/<home-prefix>/<scope>/EKEY /<home-prefix>/<scope>/DKEY
Security Policy	/<home-prefix>/RULE/<location>/<entity-id>**

Notation: A component with <> represent a variable. A component without <> represent a constant string component.
∗: Actual service command, content, and policy NDN Data packets will have a timestamp suffix to achieve the data uniqueness.
∗∗: The corresponding identity key name is the identity name with a "KEY/<key-id>" suffix.

B. Name Design

One key difference of Sovereign compared to IP-based systems is the use of semantic names in the network. Since endpoint addresses are used for decades, the advantage of name-based communication might not be obvious. This is why we first focus on answering the question: *Why bothering with designing semantic naming schemes?* First, semantic metadata is needed to represent identities and data when realizing application logic. In Sovereign, naming conventions allow entities to follow established rules to infer the name of the desired data. Second, it is possible to design security policies based on names. This means that security policies can directly define which *named* entities are allowed to produce or access which *named* resources. Hence, in Sovereign, name-based security policies allow inferring which entities are granted access to certain resources by checking their names.

Naming Conventions: Prior studies [34], [35] suggest that smart home access control and authentication systems should be flexible enough to support a wide variety of use cases and types of relationships that exist in homes. Therefore, Sovereign's naming conventions embed the attributes held by entities and data into semantic names. These attributes include an identifier of the home system, a service type, the location of the corresponding entity or data, and a resource type. Table I summarizes the structure of names (i.e., naming conventions) used in Sovereign. Note that the home prefix is defined during the system setup and is better to be globally unique considering potential communication among homes in a neighborhood. One approach is by attaching a random string to a user-specified name.

Sovereign allows flexible service invocation and content fetching by constructing different names based on the application's need. For indicating the capabilities of Sovereign's naming scheme, we provide a small example of an AC in the bedroom of Alice's home. The AC listens to requests under three prefixes that allow other entities to send control commands on a device-level, room-level, and house-level.

/alice-home/AirCon/**bedroom/north-ac-1**/CMD/set-temp
/alice-home/AirCon/**bedroom**/CMD/set-temp
/alice-home/AirCon/CMD/set-temp

Thus, the controller or other authorized entities can flexibly control the home temperature in the desired granularity. Importantly, semantic naming combined with broadcast media allows sending one command to control multiple devices. This is a sharp contrast to IP-based systems, where commands address single devices only.

Name-Based Security Policies: Sovereign's naming conventions facilitate the use of name-based security policies. As elaborated earlier, security policies generally specify entities that are authorized to perform certain operations. In Sovereign, flexible control can be achieved by specifying the names of entities and resources in the policies. Given the name P representing one or multiple entities, and the name R representing one or multiple resources. One can limit P's authorized actions to R by defining a security policy. Such a policy can be written as a triple based on P and R, which can be either be specific names, name prefixes, or regular expressions of names.

$$<P\text{'s name},\ \ verb,\ \ R\text{'s name}>.$$

For example, the verbalized policy *the controller can command all door locks* can be written as the name-based policy *<controller name, produce, door lock command prefix>*. As another example, the verbalized policy "*temperature sensors can produce temperature data*" can be represented by the policy *<prefix of temperature sensors, produce, temperature content prefix>*.

We want to note that using names in security policies is assumed to be secure since names are assigned and certified with the entity's certificate. This certificate is derived from the trust anchor in the entity bootstrapping phase (see Section III-C). Thereby it is ensured that entities cannot spoof the names of other entities.

C. Security Design

In Sovereign, the authentication and access control are managed by the controller but enforced distributedly in D2D communication. In general, after converting user-defined rules into name-based security policies, the controller distributes these policies among all entities. During runtime, all Data packets are signed and encrypted by producers, and verified and decrypted by consumers. In the verification step,

consumers consult the available security policies and verify whether data producers are authorized to sign the piece of data before consuming its content or executing the command.

System Setup: In the system setup phase, the controller generates an asymmetric key pair. The public key is bound to the home prefix and published as a self-signed certificate. This certificate represents the home's trust anchor. The trust anchor will be installed on each entity during the entity bootstrapping. The private key is kept secret by the controller and is used to sign security policies, cryptographic keying material, and certificates for new entities.

Entity Bootstrapping: In the entity bootstrapping phase, a new entity joins the system and learns keying material and security policies required for later communication.

The process takes place in the default broadcast media and starts with mutual authentication between the controller and the entity, in which out-of-band operations may be needed (e.g., QR code scanning). Then, the following information is secretly exchanged.

1) The trust anchor certificate is installed on the new entity.
2) The controller assigns the new entity an appropriate name according to the naming convention (supported by additional input from the homeowner, e.g., entity location or customized device name).
3) The controller issues a public-key certificate binding the entity's public key and name together. The certificate is signed with the private key of the trust anchor.
4) The entity will obtain security policies and symmetric cryptographic keys that the entity is authorized to have. These keys will be used in data encryption and decryption for access control and privacy protection purposes. In addition, the new entity and the controller will negotiate another symmetric to encrypt the future cryptographic material issued by the controller to this entity.

An implementation of Sovereign's entity bootstrapping process is discussed in [36].

Bootstrapping hardware-independent applications follows similar steps as given above. Note that applications can run either locally (e.g., software executed on the home hub) or remotely (e.g., software hosted in the cloud).

Enforcing Security Policies: While name-based security policies are maintained by the controller, individual participants enforce these security policies for the D2D communication. To elaborate on how these policies are enforced, we differentiate between two types of security policies: 1) *produce*-policies define which entities are allowed to produce data of a specific name pattern (e.g., sending an actuating command to a door lock) and 2) *decrypt*-policies define the entities that are allowed to access data of a specific name pattern (e.g., data from specific sensors).

Produce-Policies Are Enforced by Data Receivers: To be more specific, after authenticating a Data packet, the receiving entity extracts the data name and producer's name from NDN Data's name and signature fields. These names allow checking whether the data were generated by an authorized entity defined in security policies. For example, the following policy defines that temperature content can only be signed by a temperature sensor, where "`/alice-home/TEMP`" is the shared prefix of all home temperature sensors, and "`/alice-home/TEMP/CONTENT`" is the common prefix of temperature content.

$$<\!/alice-home/TEMP,\ produce,\ /alice-home/TEMP/CONTENT\!>$$

Combining the Data's name, the producer's identity, and the available security policies allow receiving entities to reject temperature content produced by unauthorized parties.

The same procedure is applied for restricting entities to issue commands. For example, a *produce*-policy as follows indicates that all the automation applications running on the home hub named "hub-1" can invoke executables whose names match the specified regular expression.[3]

$$<\!/alice-home/AUTO/hub{-}1,\ produce,\ /alice-home/LOCK/<>*/CMD\!>$$

Before executing the issued command, the receiving entities first check the verified producer name against the available security policies and reject the command when issued by unauthorized entities.

Decrypt-Policies Are Enforced by Utilizing Data Encryption: To be more specific, every entity can fetch data by emitting an Interest packet carrying the desired content name. The encrypted data, however, can only be accessed when having access to the correct decryption key. In Sovereign, the controller is providing all encryption and decryption keys and allows authorized entities to obtain certain keys.[4] This reduces access control to maintaining the access of corresponding decryption keys. For example, the *decrypt*-policy "bedroom AC can read the temperature" is written as follows:

$$<\!/alice-home/AirCon/bedroom,\ decrypt,\ /alice-home/TEMP/DKEY\!>$$

The subject name "`/alice-home/TEMP/DKEY`" represents the decryption key to the content produced under the temperature service.

Privacy Protection: Sovereign protects the privacy carried in network packets with three methods.

1) As stated, the payload of all Data packets is only sent in the encrypted form.
2) Since semantic names may also leak sensitive information, Sovereign applies name obfuscation. That is, instead of using plaintext name components, Sovereign can choose to use pseudonyms derived from a keyed hash (e.g., HMAC of the name component with a random key generated by the controller). Besides, service-specific name components, such as "`content-id`" and "`command-id`" are encrypted with the payload's encryption key.

[3]The regular expression <>* matches zero or more name components.

[4]The Sovereign controller makes encryption and decryption keys available for authorized entities. After the distribution of the keying material, the controller is not involved in the D2D communication, allowing entities to communicate securely without requiring a central message broker.

3) In Sovereign, the data exchange happening in the local network is invisible to the outside. When homeowners allow services provided by trusted remote entities, the latter can only access the data allowed by the security policies.

Key Distribution: As stated, encryption and decryption keys are first distributed during the entity bootstrapping process. In the system runtime, since each key has a lifetime and security policies can change (e.g., end users add or revoke access rights), keys need to be periodically renewed. Therefore, entities will need to retrieve the updated keys from the controller. In Sovereign, decryption keys will be retrieved by their names (Table I) and are encrypted for each authorized entity using the shared symmetric key between the entity and the controller. Importantly, this process can be asynchronous: the controller can preprovision Data packets carrying the encrypted renewed keys into a storage component, allowing the key renewal process to operate when the controller is temporarily down.

D. Putting Everything Together

The previous sections explained the individual components of Sovereign. In this section, we show how everything works together to build a local, user-controlled smart home system.

Reviewing current smart home systems shows that the individual entities can be classified into two groups. The first group represents devices that provide executables. These devices can be as simple as a light bulb that can be switched on or off, but also more security-critical, such as the door lock of the front door. The second group represents devices that produce content, e.g., sensors that monitor the smart-home environment, providing temperature information or motion detection. Also, some devices combine both groups.

When invoking an executable, or when requesting content, entities are typically not interested in communication with a specific device or application, but in a specific service on a certain location (e.g., switching on the kitchen light and getting the ambient temperature of the living room). Without knowing the actual entity required for the action, the name can be inferred by joining the service name with the intended location scope (e.g., "`/alice-home/Light/kitchen/CMD/switch-on`" and `/alice-home/TEMP/CONTENT/living room`"). When it comes to networking, we differentiate between actuating messages used for invoking executables, and content consumption messages for reading data. To fetch a piece of content, the desired content name is encoded into an Interest packet for broadcast. In contrast, when issuing a command, the command is encoded in a Data packet to carry the signature and sufficient parameters in the payload. The command sender will emit a notification Interest to the broadcast network so that relevant other entities can fetch the command. Note that command fetching only needs to take place once because other entities can also hear the command via broadcast.

To fulfill security policies, the produced content and commands are encrypted and signed by its producer. Hence, when receiving a Data packet, the receiver can first verify whether the producer is authorized to produce the data by checking the signature against security policies. In case the producer is authorized, only authorized receivers can access the decryption key and successfully decrypt the Data packets content. Hence, security policies are enforced.

IV. Implementation of Usable Framework

The Sovereign framework is implemented as a smart home SDK. Our SDK supports developers and device manufacturers in building Sovereign-compatible devices and applications. The design of our API design aims to provide a high developer-friendliness and is outlined in Section IV-A. Our SDK's main components are discussed in Section IV-B.

We also provide a proof-of-concept controller with preliminary UIs. This controller allows home users to bootstrap devices and applications and to define security policies. More sophisticated UIs can be designed to allow end users to use Sovereign the same way as current smart home systems. Sovereign's main contribution—enabling a self-contained, local smart home—is beneath the applications and should be transparent to end users.

A. Encapsulation of Framework Details

The core implementation idea of Sovereign is to use a publish–subscribe (pub/sub) communication module that encapsulates naming, security, and networking primitives in one API. Pub/sub is a messaging pattern that categorizes messages into semantically meaningful topics and is often seen in the context of IoT. Message producers (called publishers) publish messages to topics without knowing the set of message consumers (called subscribers). Subscribers choose to receive messages under predefined topics without the need to know the actual message producers. Pub/sub is used for two main reasons. First, pub/sub is data-centric which matches the data-centric networking and security design of Sovereign. Second, pub/sub has been widely adopted in existing IoT frameworks, and hence, it provides convenience for developers.

Pub/sub API is directly built over Sovereign's D2D communication as presented in the previous section by handling name prefixes as topic identifiers. That is, subscribers use name prefixes to decide whether a message is under a certain topic or not. For example, a message carrying temperature data of the bedroom is mapped to the name prefix "`/alice-home/TEMP/CONTENT/bedroom.`" This name prefix is further used as the pub/sub topic identifier. In this way, producers that publish content or commands under a topic is to generate Data packets named under the corresponding prefix. Subscribing to a topic is implemented by issuing Interests containing the topic's name prefix to fetch relevant data.

As indicated in Fig. 4, in Sovereign implementation, the pub/sub API embeds naming, security, and networking considerations to make them transparent to developers. We illustrate the underlying workflow with an example, where an application issues a command to set the bedroom temperature to 70 °F. After calling the publish API, Sovereign defines

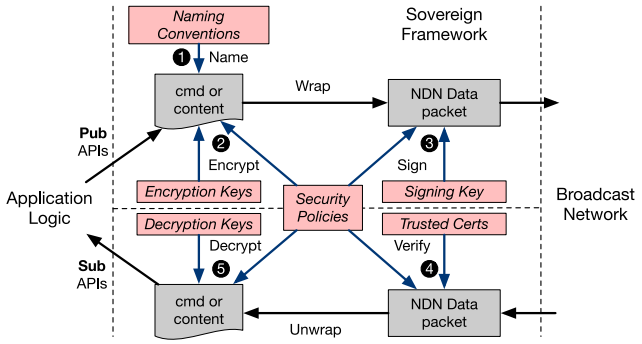

Fig. 4. Workflow beneath the pub/sub API.

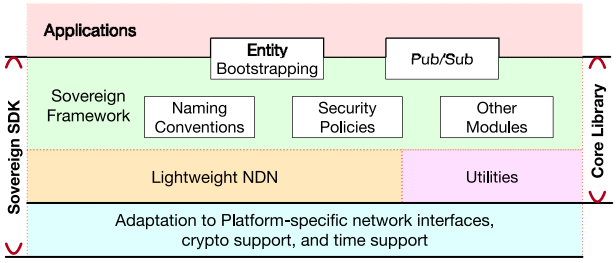

Fig. 5. Structure of Sovereign's smart home SDK.

the command name based on application parameters following naming conventions (❶). Sovereign identifies an encryption key according to the topic and encrypts the payload (❷). Further, the API wraps the command name and the encrypted payload into a Data packet. Finally, the API uses the application's private identity key to sign the Data (❸) and makes it available on the local network. On the receiving end, after subscribing to a topic, the API automatically fetches the relevant Data packet. Once a Data packet is received, Sovereign verifies its signature and checks against security policies (❹). In the next step, the Data are decrypted with the corresponding decryption key (❺). Once verified and decrypted, the command is delivered to the application. For clarity, we omit low-level protocol details of Sovereign's pub/sub transport in this publication. We provide a discussion of the pub/sub design, protocol details, and implementation in a supplemental technical report [37].

B. Sovereign Software Development Kit

Sovereign is made available as an open-source, cross-platform SDK [23] in C language for device and application developers. The SDK's structure is visualized in Fig. 5. The core library implements the Sovereign framework as detailed in Section III. As indicated, the only components exposed to developers are the *Entity Bootstrapping* and the *Pub/Sub* APIs. Other components are transparent for developers. Moreover, the SDK includes an adaptation layer making the core library work across different platforms and communication media. So far, the adaptation layer is tested for platforms, including Linux/Unix, RIOT OS [38], and Nordic NRF boards [39]. Regarding connectivity, the adaptation layer allows using Bluetooth, IEEE 802.15.4, and the legacy TCP/UDP used as link layer protocols.

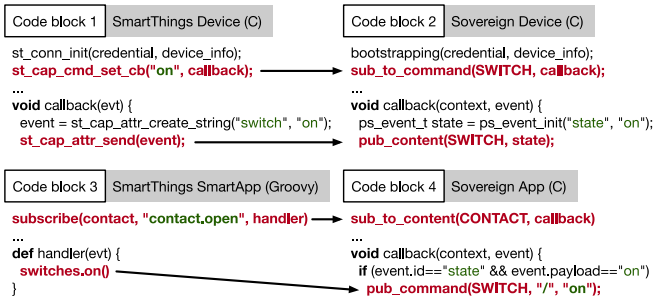

Fig. 6. Code snippets in SmartThings and Sovereign.

Importantly, the Sovereign SDK includes a standalone NDN stack that follows the official NDN specification [40] and is lightweight enough for constrained devices. NDN-LITE is required since the official NDN library and forwarder—ndn-cxx [41] and NFD [42]—are not designed for being used on constrained IoT devices.

C. Case Study on Real-World Smart Home Programs

In this section, we conduct a case study with two real-world applications and compare our work with SmartThings, a leading smart home ecosystem on the smart home market. Through the study, we show real examples of the use of the Sovereign SDK and at the same time assess the privacy and security by analyzing packet flows behind the API calls.

We select two applications [43], [44] from the official open-source codebase [45] of SmartThings and implement the same functionality in Sovereign. The first application is designed for a smart switch device and the second is an automation applet turning on a remote switch when a contact sensor is touched. The first application changes its own state to "on" when it gets a turn-on command, and the second application subscribes to the state of a contact sensor and turns on a remote switch when the subscribed state is changed to be on. The code snippets of the original programs (code blocks 1 and 3) and their Sovereign-based equivalents (code blocks 2 and 4) are compared in Fig. 6.

As shown, the programs in Sovereign are similar to their original version. This is because by adopting the pub/sub, Sovereign provides a similar development pattern as the existing SDKs, allowing developers to port their applications to Sovereign with a minimum amount of code changes.

Despite the similarities, the underlying traffic flow is fundamentally different: Sovereign provides the same functionality as cloud-based smart homes via local secure communication. The home activities and data are not accessible to the cloud unless users explicitly grant access rights to a cloud service.

First, in the SmartThings device application (code block 1), the first line securely connects the device to the home's cloud-backend. After that, the cloud recognizes the new device, learns its profile, and registers it to the device database for the home on the cloud. In contrast, the first line of the Sovereign device application (code block 2) bootstraps the device to a local controller. All sensitive information that is transmitted in the bootstrapping phase stays in the local network and is protected by encryption.

Second, the first line of the code blocks 3 and 4 subscribes to a given topic. However, the underlying operations differ: the SmartThings application notifies the cloud backend about its interest in the given service, while the Sovereign application simply starts listening to a local name prefix under which the desired data is published. Similarly, when the SmartThings application turns on the switch, the command is sent to the cloud backend, where the command is verified and sent back to the switch at home. In contrast, publishing a command in Sovereign means producing a new Data packet and making it accessible in the local home network.

V. Evaluation

We evaluate the Sovereign prototype and answer the following questions.

1) Can Sovereign enhance smart home security and privacy compared with existing cloud-based smart home systems?
2) Is Sovereign resilient enough to operate when the controller is temporarily down?
3) Does the use of cryptographic operations on smart home entities impair the usability of Sovereign in a real deployment, especially on low-power devices?
4) Are Sovereign SDK's programming interfaces friendly to IoT developers?

A. Security and Privacy

The security and privacy in Sovereign is provided through a collection of factors. First, from a design perspective, Sovereign removes the external dependency from the content fetching and service invocation at home, which minimizes the exposure of local sensitive information. Second, by heavily relying on the local communication, link layer security like Wi-Fi protected access (WPA) can provide more security as both ends of the communication in Sovereign are local. Third, identity authentication and traffic encryption implementation follow the mature symmetric and public-key cryptography. We also provide an analysis of Sovereign against various attacks in smart home and network systems in Appendix C.

Note that a smart home system is complicated in the sense that applications and devices are provided by different parties and each of them can have their own external dependencies and vulnerabilities. Sovereign, as a system framework, is just one part of the overall smart home security.

B. Resilience, Performance, and Overhead Evaluation

In this section, we first test the Sovereign's resilience by taking down the controller intentionally at the system runtime. We then evaluate the performance of Sovereign by measuring the latency in common system operations. We also compare our results with cloud-based home IoT solutions; since the latency bottleneck is in the network round trip time and cloud-based approaches share the similar communication model as discussed in Section II-A, we selected AWS IoT as a representative for comparison and the comparison results also apply to other cloud-based home IoT systems. Finally, we measure the resource footprint required by Sovereign on IoT devices.

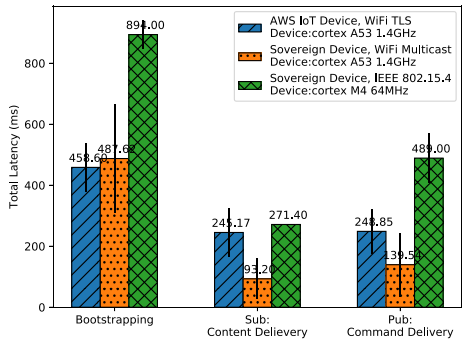

Fig. 7. Latency in Sovereign and cloud-based home IoT.

Experiment Setup: We conduct our experiments using 1) a laptop with a 2.2-GHz Intel quad-core Core i7 CPU; 2) Raspberry Pi (RPI) 3B with an ARM Cortex A53 @1.4-GHz processor; and 3) an nRF52840 board [39] to mimic smart home entities with different capabilities. Among them, we classify nRF52840 chip with 32-bit Cortex M4@64-MHz CPU, 1-MB ROM, and 0.25-MB RAM[5] as constrained hardware. The experiments with RPI are over WiFi connectivity and with a Sovereign controller running on the laptop. Evaluations involving the nRF52840 are conducted over IEEE 802.15.4 and use a simplified controller installed on another nRF52840. For the AWS IoT, we use the cloud backend provided by the official AWS IoT Core services [51] and an RPI as the local device.

Resilience to Controller Outage: To evaluate Sovereign in the case of a temporary controller outage, we first set up the system and bootstrap new devices with the controller. After some time, we bring the controller offline and test whether the content fetching and service invocation can still be finished and whether the security policies are still executed as normal.

The experiments presented below confirm that the communication happens in a D2D manner and is secured without a controller involved, showing Sovereign's ability to continue operation even when the controller is temporarily down.

Latency Measurement: We then evaluate the latency of common operations in the system, including *entity bootstrapping*, *content delivery*, and *command delivery*, between Sovereign and AWS IoT. The experimental results are visualized in the left two bars (i.e., blue bars for AWS IoT and orange bars for Sovereign) in each bar group in Fig. 7.

As shown, there is no significant difference between the bootstrapping operation, but the advantage of keeping communication local in Sovereign becomes clear when focusing on content and command delivery, where Sovereign is about 62% faster in content delivery and 42% faster in command delivery than the AWS IoT.

Latency on Constrained Devices: In addition, to measure Sovereign's latency on constrained devices, we use the nRF52840 as the device and IEEE 802.15.4 network in our experiments. The result is visualized in the right bars (i.e.,

[5]We acknowledge that a part of current IoT devices shows lower capabilities. However, market studies [46]–[50] have seen the market turning to 32-bit microcontrollers. Hence, we assume the nRF52840 as an appropriate candidate for future smart home systems.

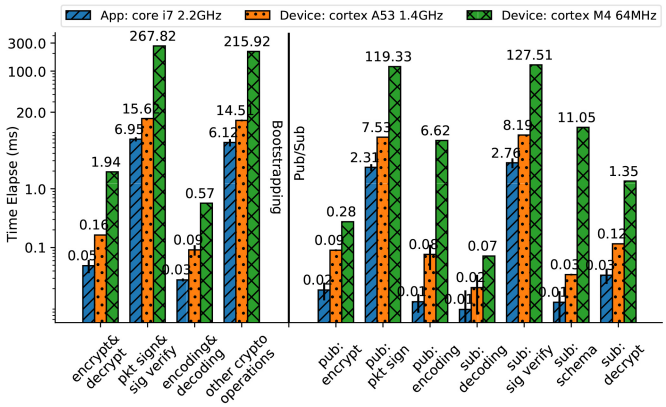

Fig. 8. Breakdown of the execution time.

TABLE II
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%

green bars) of Fig. 7. As shown, it requires about 2×–3× latency compared with the RPI implementation of Sovereign. However, since the bootstrapping is infrequent (e.g., once in each device's lifetime) and the latency of content and command delivery are below 500 ms, Sovereign is practical for the use of constrained devices in smart homes.

Execution Time Breakdown: A breakdown of Sovereign's runtime into individual operations is provided in Fig. 8. The visualized operations are performed when preparing Data before broadcasting to the network and Data processing after receiving. This includes digital signature creation and verification (ECDSA), content encryption and decryption (AES CBC), security policy checking, NDN packet encoding/decoding, and other cryptographic operations (including ECDH and KDF). The results show that asymmetric cryptography consumes most of the computation time. This trend is observable across all evaluated devices.

ROM and RAM Footprint: We programmed an nRF52840 chip using RIOT OS [38] for measuring Sovereign's memory footprint. Table II reports the size taken by individual Sovereign modules. All the main modules together require less than 50 kB of RAM and 70 kB of ROM. This indicates Sovereign's ability to be used on resource-constrained smart home devices.

C. Developer Friendliness Evaluation

As shown by the line-by-line comparison between the Sovereign and SmartThings applications in Fig. 6, it becomes apparent that Sovereign offers a similar experience to developers as SmartThings. Specifically, the highlighted lines reveal a high API similarity and indicate that developers can port existing smart home applications easily to Sovereign without requiring changes to the application's logic.

TABLE III
RESULTS OF THE PROGRAMMING EXPERIMENT

Stages	Descriptions	Avg. Time*	Min Time*	Max Time*
Tutorial	Finish the Sovereign tutorial	68	20	90
Task 1	Write an application that triggers an alarm when detecting smoke	41	23	90
Task 2	Write an application that controls an air conditioner based on temperature values	24	15	40
Task 3	Write an application that used motion sensor and lights to detect if someone is at home	26	17	35

∗: Counted in minutes

Metrics	Avg.	Min	Max
Lines of Code	72	15	114
Preparation Time** [min]	39	10	90
Debugging Time [min]	32	5	60

∗∗: The time spent for reading documentation after finishing the tutorial.

We evaluated the developer friendliness by a programming experiment.[6] We recruited computer science students as participants and asked each participant to write three new sample applications, including related device drivers, and to finish a questionnaire afterward. We give no assistance except for providing a brief introduction, instructions, and the SDK documentation with code examples. Appendix B provides additional details on the programming experiment.

We found ten participants for the final programming experiment, none of which had prior experience in developing smart home software. Among them, two participants experienced issues with the controller's graphical UI, and hence, did not finish the experiment. Table III summarizes the results of the remaining eight participants. As shown, the tutorial was finished by all participants in an average time of 68 min. The average time for completing the first task was 41 min, and the remaining two tasks were solved in less time. We expect the speedup to be caused by a learning effect.

After completing the programming tasks, all participants finished an online questionnaire about Sovereign's ease of use. Two participants expressed concerns regarding the vague specification of function parameters and the long function names that are challenging to remember. Also, four participants suggested adding more illustrations about the system logic in the documentation. Other than that, the participants faced no difficulties in using Sovereign's SDK.

VI. RELATED WORK

Smart Home User Control: Many works aim to enhance user control to today's smart home systems. For example, HomeOS [52] was proposed before the booming of the commercial smart home market. Its main goal is to hide device heterogeneity from application development by treating all IoT devices as directly attached peripherals to a home PC. However, the high dependency on a single centralized PC

[6]Our study has been certified as an exempt study by UCLA Research Administration with protocol ID IRB#20-001611.

TABLE IV
COMPARISON OF REPRESENTATIVE RELATED WORK

System	Cloud Independent Home	Fine-grained Security Policy	Resilience to controller failure
HomeOS [52]	Conditionally*	✓	✗
IoTGuard [18]	N/A	✓	✗
SmartAuth [16]	N/A	✓	✗
HanGuard [17]	N/A	✗	✗
ContextIoT [15]	N/A	✓	✗
EXPAT [19]	N/A	✓	✗
IoT-IDM [21]	N/A	✗	✗
SMP [22]	N/A	✗	✗
OpenHAB [53]	Conditionally**	✗	✗
Home Assistant [54]	Conditionally**	✗	✗
Sovereign	Fully	✓	✓

N/A: The work is not relevant to whether the home relies on the cloud.
*: Cloud independent, when the controller is implemented locally.
**: Cloud independent, when purchased devices do not rely on the cloud.

raised the concern of single point of failures; if one was to move that device to the cloud, it would resemble today's cloud-based solutions. After cloud-based smart home deployment became popular, different solutions have been proposed to add runtime enforcement of security policies for better user control. Some focus on modifying existing IoT application programs, or adding additional data flow verification onto the existing cloud backend [15]–[19]. Other solutions add user control at the network layer [17], [20]–[22], e.g., by assigning dedicated managers sitting at the network gateway to control whether to block individual network packets based on user-defined policies (e.g., through SDN). Nevertheless, these approaches do not move away from the cloud-based system model and thus do not fully address the issues as mentioned in Section II. A comparison between these approaches and Sovereign is shown in Table IV.

Home automation projects that provide user control with a local hub, such as OpenHAB [53] and Home Assistant [54], are seemingly similar to Sovereign's local control. The hub in these projects can directly control those devices that are cloud-independent, thus realizing local user control to some extent. However, these systems have a different goal from ours: they focus more on providing a uniform control interface for users and facilitating the delivery of users' commands. Thus, they do not provide a systematic solution to securing the communication between devices and to enforce interdevice access control. In addition, since all user commands go through the local hub, the hub is a single point of failure.

Use of Names: Proposals that utilize semantically meaningful names closely related to our approach. For example, the intentional naming system (INS) [55] utilized semantic names inspired from earlier works [56]–[58] to build an application overlay for information and resource discovery. Bolt [59] also uses semantic names from the application layer by abstracting data as a stream of time-tag-value records, in which the semantic tags are used for data management. Besides, SemIoTic [60] utilizes a semantically meaningful metamodel based on semantic sensor network (SSN) ontology to describe smart spaces. These works use names to facilitate service discovery and data management at the application layer, which indicates the undesired overhead like the IP-name mapping and the complexity to bridge the gap between the data-centric security and secured TCP/IP channels. IoT wireless protocols, such as Bluetooth low energy (BLE) [61] and ZigBee [62], also define service identifiers rather than using opaque network addresses for communication. Nevertheless, the identifiers in BLE and ZigBee are solely for the purpose of communication. In contrast, the names in Sovereign are not only used for identifying resources but also used to execute security and define security policies.

Pub/Sub: Regarding the related works in pub/sub, conventional realizations, such as MQTT [63], rely on a central message broker that handles message filtering and forwarding. The message broker serves as the rendezvous point between publishers and subscribers, interconnecting both ends at the application layer over a network of channel-based communication models. Sovereign's pub/sub implementation uses NDN's name semantic for defining message topics. This allows publishers and subscribers to rendezvous by namespace over broadcast network media, removing the need for a central broker. Our implementation also differs from existing NDN-based pub/sub designs [64]–[66], which are designed to run over multihop networks and utilize a synchronization protocol between publishers and subscribers. Sovereign leverages the local broadcast network setting to remove the need and overhead for a synchronization protocol.

Related Works in ICN: Recent years have also seen researches [12], [65], [67]–[74] on the direction of ICN-based smart homes. For example, Shang *et al.* [12], [67] analyzed the functions of cloud in today's cloud-based smart home systems and discussed how NDN can be potentially utilized to provide a replacement of cloud to operate the smart home in a local environment. Ascigil *et al.* [68] explored the different strategies of using NDN names for data fetching and computation power placing in edge IoT networks. Other works [73], [74] also investigate how to combine the NDN/ICN with existing protocols like LowPAN or CoAP for IoT networking. Discussing potential benefits and applying NDN/ICN to specific IoT protocols, existing works are in an early stage of the direction. In this article, we present a systematic NDN-based smart home framework with concrete implementation, which we believe is a big step forward in this direction and helps to examine the benefits of using NDN/ICN in IoT scenarios.

Comparing Sovereign's access control with existing NDN-based access control solutions like name-based access control (NAC) [75], the similarity is that they both utilize naming conventions for automated key delivery. However, in order to work with constrained devices, Sovereign's access control approach is more lightweight by directly delivering sealed symmetric decryption keys to authorized entities without computation-intensive asymmetric key encryption schemes.

VII. DISCUSSION

After presenting the design and evaluation of Sovereign in the earlier parts of this article, this section critically discusses our design choices.

Sovereign Controller and System Resiliency: The controller failure does not stop the system's normal operation. Our experiments show that Sovereign can continue operating during failures of the smart home controller.

However, since Sovereign's security design relies on the controller acting as a trust anchor, a compromised controller means hijacking the trust anchor, bringing all vulnerabilities similar to the compromised root certificate of a system. Hence, we highly recommend hardening the access to the controller, and developing means for quick compromise detection. One can add special protection for the controller, such as two-factor authentication, the use of hardware TPM's, and build anomaly detection on top of our framework. At the same time, we recognize that such measures might interfere with the smart home system's usability for end users.

As part of our future work, we plan to relax the single point of trust, for example, with secret sharing schemes [76] or threshold signatures [77]. In this way, the trust is jointly held by multiple parties, namely, multiple controllers can be installed (e.g., a hub device in the home and the homeowner's smartphone). Such schemes allow that all security-critical actions need to be verified by more than a single party, reducing the impact of a single compromised controller.

Resiliency to Packet Losses and Link/Node Failures: Two other commonly encountered failures in a networked system are network packet losses and link or node failures. In Sovereign, data consumers are responsible for reliable data fetching. If an Interest packet times out without receiving the requested data, the consumer retransmits the Interest. In addition, Sovereign also utilizes redundancy mechanisms to improve packet delivery success: 1) fully utilizing broadcast media: if a device misses a notification message, it may hear a related response message issued by another device that received the notification and 2) redundant notification: when sending a notification for a newly available Data packet (i.e., a command), the entity may send redundantly by retransmitting aggressively. Regarding a failed or compromised devices in Sovereign, the damage is limited. A failed node does not affect others' direct communication, and even a compromised node cannot access resources that it is not allowed to. When such a device is identified (e.g., by applying existing intrusion detection mechanisms [18], [78]–[81] above Sovereign), its identity, as well as access rights, can be revoked by updating security policies and not renewing its keying material.

Using the Cloud in Sovereign: Sovereign's main contribution to user privacy protection is putting the *control* of users' data in the users' hands. This does not exclude or discourage the use of cloud services. A smart home system is free to outsource individual services like backup storage, or computationally intensive tasks such as voice recognition, to cloud service providers. Sovereign supports this by handling remote services as normal system entities. That is, same as local entities, cloud services need to obtain permission from the home controller before they can access home data.

Deployment of Sovereign: The deployment of smart home solutions is far more than a technical question and reaches beyond the scope of scientific research. The goal of Sovereign is to demonstrate an alternative to today's practice, and to encourage more discussion and development around this new direction. It is noteworthy that Sovereign does not depend on a global NDN deployment. Even the deployment of NDN in the home network is transparent because the communication specifics are included in, and automatically handled by, the Sovereign core library. When NDN-based designs become gradually adopted on edge systems and applications and bring desired benefits to end users, they can be expected to drive broader adoptions of NDN at a larger scale.

VIII. Conclusion

We make three observations based on our investigation into the Sovereign design and development. First, although cloud-based applications have achieved great successes, home IoT systems should take a different direction; putting the home system control in end users' hands offers a sure way to fully protect user privacy. Second, the biggest challenge in designing a user-controlled home IoT system is security, and today's network security practices do not fit the home problem space. Third, our experience with Sovereign suggests that the new networking direction pointed out by ICN/NDN research, using semantically named and secured data as the basic system building block, shows great promise in enabling user-controlled smart home systems.

The Sovereign design and development are still in an early stage with a few important pieces remain to be done. In addition to improving the reliability of the local controller, another urgent task is the design and integration of a local storage component for home operation logging, which can further be used for auditing and data analysis. The reliability of such a local data storage can be enhanced by saving an encrypted backup in cloud-based storage systems.

We hope that this article can serve as an invitation to all interested parties in joining us in exploring and experimenting with this new way of building smart homes.

Appendix A
Cloud in Existing Smart Home Systems

In this Appendix, we use Amazon AWS IoT [4], Google IoT Core [27], and Microsoft Azure IoT [6] as examples to demonstrate the use of cloud backend in smart home systems.

Amazon AWS IoT: AWS IoT consists of three main components: 1) cloud-connected devices; 2) cloud-hosted applications; and 3) 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 the 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 [82] encourages local communication, the management is still realized at the cloud.

Google Assistant and Google IoT Core: Google Assistant and Google IoT Core in general manage home IoT system with Google's cloud services and OAuth. Specifically, the cloud creates a database for each home containing the information of the structure, rooms, and devices. This database serves as a global view of the home and is queried whenever there is an

intent (e.g., turn on the light). For a device or an application to access some services, they need to obtain OAuth tokens from an OAuth server running in the cloud. Besides, devices are implemented with code that is deployed as a webhook in the cloud. When users send a command to the device, the command is first processed by the webhook and then forwarded back to the device at home.

Microsoft Azure IoT: Azure IoT uses the cloud to interact with individual devices. When receiving data, the analysis will be performed at the cloud side which is connected to other Azure cloud services. In a D2D scenario, the cloud will also act as a message broker between devices. In the middle of cloud backend 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 the cloud will be mutually authenticated by a TLS-based handshake.

APPENDIX B
DEVELOPER FRIENDLINESS STUDY

The programming experiment presented in Section V-C was designed as follows. First, participants were from the computer science department and were recruited via e-mail, and a link to the experiment website was also included in the same e-mail. The website contains 1) the background information; 2) the instructions for the experiment; 3) the documentation for Sovereign SDK; and 4) a tutorial with code templates and an example program. Beyond that, participants received no further assistance. Then, we asked each participant to go through the tutorial, write three programs for three simple smart home use cases, and complete a questionnaire along the experiment. Specifically, these three programs aim to implement the following functionalities: 1) to raise the alarm based on the smoke detector's report; 2) to monitor the ambient temperature and turn on the air conditioner when the received temperature is higher than 80°; and 3) to turn off all lights in the living room when the motion sensor does not detect any motion for more than five minutes. The questionnaire contained four parts, which were presented to the participant at different stages of the experiment. Table V lists all the questions. The first part surveys the participant's background information before the experiment starts. The second part is after the tutorial but before the actual programming starts. The third part is after the programming and the last part is after the whole experiment, focusing on difficulties of using the Sovereign APIs.

In addition, the participants were asked to end the experiment if they have spent 2 h in total. Note that all participants took the experiment remotely so we had no control over the participant's experiment environment; however, the impact can be ignored as the students who agreed to participate should already have an appropriate workspace. Before the real experiment, we also did a pilot test with two volunteers to evaluate the experiment setup; some minor improvements were made after that, mainly in regards to the clarity of API documentation and instructions.

TABLE V
QUESTIONNAIRE ACCOMPANYING THE PROGRAMMING EXPERIMENT

No.	Questions
Part 1	*Background*
1.1	Do you have experience on home automation?
1.2	If you have experience on home automation, what's the advantages/disadvantages of Sovereign comparing to the home automation library you used before
Part 2	*Learning*
2.1	How many minutes did it take for you to finish the quickstart example/tutorial?
2.2	How many minutes did it take to learn the library before you begin coding?
Part 3	*Programming Experiment*
3.1	How many minutes did it take for you to finish your task 1?
3.2	How many minutes did it take for you to finish your task 2?
3.3	How many minutes did it take for you to finish your task 3?
3.4	How many minutes did it take to debug your application?
3.5	How many lines of code did you write for your application?
Part 4	*Summary and Suggestion*
4.1	Do you face any difficulty when using the API?
4.2	Do you have any suggestion to improve the API?

APPENDIX C
SECURITY ANALYSIS AGAINST EXISTING ATTACKS

In this Appendix, we analyze Sovereign's security strength by checking its resistance to various attacks that can exploit common security design flaws in smart home and network system design.

Smart Home Attacks: Recently, researchers have identified various security design flaws at the application layer. For example, Fernandes *et al.* [83] proposed a set of attacks that exploit the coarse binding between authorized SmartThings devices and applications. A typical attack caused by coarse access control is a *Pin Code Snooping Attack*, where attackers leverage an overprivileged battery monitoring application to read the door lock's PIN code. Whereas in Sovereign, semantic meaningful names ensure the fine granularity for access control. Using the same attack scenario, security policies enforce a Sovereign application under "`/alice-home/Battery`" can only obtain a decryption key for decrypting data under the same prefix. If a malicious app disguises as a battery monitoring app, it cannot get the decryption key for door lock-related data and thus cannot access the PIN code.

Security policies not only restrict the read access to certain named data but also regulate the authority of producing data. Attacks, such as *Disabling Vacation Mode Attack* and *Fake Alarm Attack*, spoof the current smart home applications by raising fake physical device events. By the least privilege design of Sovereign's access control, malicious applications that try to produce fake physical device events, like fake smoke detection, will not obtain an acceptable signing key for the data. Thus, the false alarm cannot pass the security policy verification process as the malicious applications' names do not match those names for producing a smoke detection.

Network Attacks: Sovereign leverages NDN's built-in security properties to handle common network attacks, such as *Man-in-the-Middle Attack* and *Replay Attack*. Specifically, each Data packet in NDN carries a cryptographic signature

that ensures the data integrity and authenticity. Any manipulation will fail the verification on the consumer end. In addition, each signature comes with a nonce, a sequence number, or a timestamp. This prevents the replay attack if a dead nonce list or a latest sequence number is maintained or the clock is synchronized in the system.

REFERENCES

[1] "Smart Home Statistics." iPropertyManagement.com. 2020. [Online]. Available: https://ipropertymanagement.com/research/iot-statistics (accessed Jul. 15, 2021).

[2] "Samsung Smartthings." SmartThings. 2020. [Online]. Available: https://www.smartthings.com/ (accessed Jul. 15, 2021).

[3] "Nest and Google Home." Google. 2020. [Online]. Available: https://nest.com/ (accessed Jul. 15, 2021).

[4] "AWS IoT, Connected Home." Amazon. 2020. https://aws.amazon.com/iot/solutions/connected-home/ (accessed Jul. 15, 2021).

[5] "Apple Homekit." Apple. 2020. [Online]. Available: https://www.apple.com/ios/home/ (accessed Jul. 15, 2021).

[6] "Azure IoT." Microsoft. 2020. [Online]. Available: https://azure.microsoft.com/en-us/overview/iot/ (accessed Jul. 15, 2021).

[7] "Alibaba Cloud: Internet of Things." Alibaba. 2020. [Online]. Available: https://www.alibabacloud.com/solutions/IoT (accessed Jul. 15, 2021).

[8] S. Tweneboah-Koduah, K. E. Skouby, and R. Tadayoni, "Cyber security threats to IoT applications and service domains," *Wireless Personal Commun.*, vol. 95, no. 1, pp. 169–185, 2017.

[9] H. Lin and N. W. Bergmann, "IoT privacy and security challenges for smart home environments," *Information*, vol. 7, no. 3, p. 44, 2016.

[10] B. Ur, J. Jung, and S. Schechter, "The current state of access control for smart devices in homes," in *Proc. Workshop Home Usable Privacy Security (HUPS)*, vol. 29, 2014, pp. 209–218.

[11] E. K. Choe, S. Consolvo, J. Jung, B. Harrison, and J. A. Kientz, "Living in a glass house: A survey of private moments in the home," in *Proc. 13th Int. Conf. Ubiquitous Comput.*, 2011, pp. 41–44.

[12] W. Shang, Z. Wang, A. Afanasyev, J. Burke, and L. Zhang, "Breaking out of the cloud: Local trust management and rendezvous in named data networking of things," in *Proc. 2nd Int. Conf. Internet Things Design Implement.*, 2017, pp. 3–13.

[13] S. Kumar, Y. Hu, M. P. Andersen, R. A. Popa, and D. E. Culler, "JEDI: Many-to-many end-to-end encryption and key delegation for IoT," in *Proc. 28th USENIX Security Symp. (USENIX Security)*, 2019, pp. 1519–1536.

[14] J. Wilson, R. S. Wahby, H. Corrigan-Gibbs, D. Boneh, P. Levis, and K. Winstein, "Trust but verify: Auditing the secure Internet of Things," in *Proc. 15th Annual Int. Conf. Mobile Syst. Appl. Services*, 2017, pp. 464–474.

[15] Y. J. Jia *et al.*, "ContexIoT: Towards providing contextual integrity to appified IoT platforms," in *Proc. NDSS*, 2017, pp. 1–15.

[16] Y. Tian *et al.*, "SmartAuth: User-centered authorization for the Internet of Things," in *Proc. 26th USENIX Security Symp. (USENIX Security)*, 2017, pp. 361–378.

[17] S. Demetriou *et al.*, "HanGuard: SDN-driven protection of smart home wifi devices from malicious mobile apps," in *Proc. 10th ACM Conf. Security Privacy Wireless Mobile Netw.*, 2017, pp. 122–133.

[18] Z. B. Celik, G. Tan, and P. D. McDaniel, "IoTGuard: Dynamic enforcement of security and safety policy in commodity IoT," in *Proc. NDSS*, 2019, pp. 1–15.

[19] M. Yahyazadeh, P. Podder, E. Hoque, and O. Chowdhury, "Expat: Expectation-based policy analysis and enforcement for appified smart-home platforms," in *Proc. 24th ACM Symp. Access Control Models Technol.*, 2019, pp. 61–72.

[20] A. K. Simpson, F. Roesner, and T. Kohno, "Securing vulnerable home IoT devices with an in-hub security manager," in *Proc. IEEE Int. Conf. Pervasive Comput. Commun. Workshops (PerCom Workshops)*, 2017, pp. 551–556.

[21] M. Nobakht, V. Sivaraman, and R. Boreli, "A host-based intrusion detection and mitigation framework for smart home IoT using openflow," in *Proc. 11th Int. Conf. Availability Rel. Security (ARES)*, 2016, pp. 147–156.

[22] V. Sivaraman, H. H. Gharakheili, A. Vishwanath, R. Boreli, and O. Mehani, "Network-level security and privacy control for smart-home IoT devices," in *Proc. IEEE 11th Int. Conf. Wireless Mobile Comput. Netw. Commun. (WiMob)*, 2015, pp. 163–167.

[23] "NDN-lite GitHub Code Repository." NDN-LITE Contributors. [Online]. Available: https://github.com/named-data-iot/ndn-lite (accessed Jul. 15, 2021).

[24] "Samsung's Smartthings Just Got Smarter, With New App and Ecosystem Updates." Samsung Newsroom. 2020. [Online]. Available: https://news.samsung.com/uk/samsungs-smartthings-just-got-smarter-with-new-app-and-ecosystem-updates (accessed Jul. 15, 2021).

[25] "Smartthings Developer Documentation." SmartThings. 2020. https://smartthings.developer.samsung.com/docs/index.html (accessed Jul. 15, 2021).

[26] D. Hardt, "The OAuth 2.0 authorization framework," Internet Eng. Task Force, Fremont, RFC 6749, Oct. 2012. [Online]. Available: http://www.rfc-editor.org/rfc/rfc6749.txt

[27] "Google Assistant." Google. 2020. [Online]. Available: https://developers.google.com/assistant (accessed Jul. 15, 2021).

[28] "Confirmed: 2 Billion Records Exposed in Massive Smart Home Device Breach." Forbes. 2020. https://www.forbes.com/sites/daveywinder/2019/07/02/confirmed-2-billion-records-exposed-in-massive-smart-home-device-breach (accessed Jul. 15, 2021).

[29] "Data Leak by Smart Home Device Company Wyze Exposes Personal Details of 2.4 Million Users Including Email Addresses and Health Data." DailyMail. 2020. [Online]. Available: https://www.dailymail.co.uk/sciencetech/article-7836713/Data-leak-smart-home-device-company-Wyze-exposes-personal-details-2-4-million-users.html (accessed Jul. 15, 2021).

[30] "IFTTT: Applets for Your Home." IFTTT. 2020. [Online]. Available: https://ifttt.com/explore/home-collection (accessed Jul. 15, 2021).

[31] "Consolidation." Internet Architecture Board. [Online]. Available: https://www.ietf.org/blog/consolidation/ (accessed Jul. 15, 2021).

[32] "6 Cloud Computing Failures that Shocked the World." The RIQ News Desk. 2020. [Online]. Available: https://www.readitquik.com/articles/cloud-3/6-cloud-computing-failures-that-shocked-the-world/ (accessed Jul. 15, 2021).

[33] D. Kaur. "3 of the Biggest Cloud Outages of 2020." 2020. [Online]. Available: https://techhq.com/2020/12/3-biggest-public-cloud-outages-of-2020/ (accessed Jul. 15, 2021).

[34] W. He *et al.*, "Rethinking access control and authentication for the home Internet of Things (IoT)," in *Proc. 27th USENIX Security Symp. (USENIX Security)*, 2018, pp. 255–272.

[35] E. Zeng and F. Roesner, "Understanding and improving security and privacy in multi-user smart homes: A design exploration and in-home user study," in *Proc. 28th USENIX Security Symp. (USENIX Security)*, 2019, pp. 159–176.

[36] Y. Li, Z. Zhang, X. Wang, E. Lu, D. Zhang, and L. Zhang, "A secure sign-on protocol for smart homes over named data networking," *IEEE Commun. Mag.*, vol. 57, no. 7, pp. 62–68, Jul. 2019.

[37] T. Yu, Z. Zhang, X. Ma, P. Moll, and L. Zhang, "A pub/sub API for NDN-lite with built-in security," Named Data Netw., Rep. NDN-0071, Jan. 2021.

[38] E. Baccelli, O. Hahm, M. Gunes, M. Wahlisch, and T. C. Schmidt, "RIOT OS: Towards an OS for the Internet of Things," in *Proc. IEEE Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, 2013, pp. 79–80.

[39] "Nordic NRF52840." Nordic Semiconductor. [Online]. Available: https://www.nordicsemi.com/Products/Low-power-short-range-wireless/nRF52840 (accessed Jul. 15, 2021).

[40] "NDN Packet Format Specification Version 0.3." NDN Specification Contributors. [Online]. Available: https://named-data.net/doc/NDN-packet-spec/current/changelog.html#version-0-3 (accessed Jul. 15, 2021).

[41] "NDN-Cxx: NDN C++ Library with Experimental Extensions." NDN-Cxx Contributors. [Online]. Available: https://github.com/named-data/ndn-cxx (accessed Jul. 15, 2021).

[42] "NFD—Named Data Networking Forwarding Daemon." NFD Contributors. [Online]. Available: https://github.com/named-data/nfd (accessed Jul. 15, 2021).

[43] "Example Code in Smartthings SDK for Direct Connected Devices for C." SmartThingsCommunity Contributors. 2020. [Online]. Available: https://github.com/SmartThingsCommunity/st-device-sdk-c/blob/master/example/example.c (accessed Jul. 15, 2021).

[44] "Turn-It-On-When-It-Opens." SmartThingsCommunity Contributors. 2015. [Online]. Available: https://github.com/SmartThingsCommunity/SmartThingsPublic/blob/master/smartapps/smartthings/turn-it-on-when-it-opens.src/turn-it-on-when-it-opens.groovy (accessed Jul. 15, 2021).

[45] "Smartthingscommunity Github." SmartThingsCommunity Contributors. [Online]. Available: https://github.com/SmartThingsCommunity (accessed Jul. 15, 2021).

[46] "IoT Microcontroller Market 2020 Global Industry Size, Opportunities, Key Vendors Analysis, Business Strategy, Future Plans, Competitive Landscape and Outlook 2023." MarketWatch. [Online]. Available: https://www.marketwatch.com/press-release/iot-microcontroller-market-2020-global-industry-size-opportunities-key-vendors-analysis-business-strategy-future-plans-competitive-landscape-and-outlook-2023-2021-01-05?tesla=y (accessed Jul. 15, 2021).

[47] "Microcontroller Sales Regain Momentum After Slump." IC Insights. 2015. [Online]. Available: https://www.icinsights.com/news/bulletins/Microcontroller-Sales-Regain-Momentum-After-Slump/ (accessed Jul. 15, 2021).

[48] "Mcus Sales to Reach Record-High Annual Revenues Through 2022." IC Insights. 2018. [Online]. Available: https://www.icinsights.com/news/bulletins/MCUs-Sales-To-Reach-RecordHigh-Annual-Revenues-Through-2022/ (accessed Jul. 15, 2021).

[49] "Microcontrollers Will Regain Growth After 2019 Slump." IC Insights. 2018. [Online]. Available: https://www.icinsights.com/news/bulletins/Microcontrollers-Will-Regain-Growth-After-2019-Slump/ (accessed Jul. 15, 2021).

[50] "Mcu Market Turns to 32-Bits and Arm." EE Times. 2013. https://www.eetimes.com/mcu-market-turns-to-32-bits-and-arm/ (accessed Jul. 15, 2021).

[51] "AWS IoT Core." Amazon Web Services. [Online]. Available: https://aws.amazon.com/iot-core/ (accessed Jul. 15, 2021).

[52] C. Dixon *et al.*, "An operating system for the home," presented at the 9th USENIX Symp. Networked Syst. Design Implement., 2012, pp. 337–352.

[53] "Openhab." openHAB Community Found., 2020. [Online]. Available: https://www.openhab.org/ (accessed Jul. 15, 2021).

[54] "Home Assistant." Home Assistant Project. 2020. [Online]. Available: https://www.home-assistant.io/ (accessed Jul. 15, 2021).

[55] W. Adjie-Winoto, E. Schwartz, H. Balakrishnan, and J. Lilley, "The design and implementation of an intentional naming system," in *Proc. 17th ACM Symp. Oper. Syst. Principles*, 1999, pp. 186–201.

[56] D. Estrin, R. Govindan, J. Heidemann, and S. Kumar, "Next century challenges: Scalable coordination in sensor networks," in *Proc. 5th Annu. ACM/IEEE Int. Conf. Mobile Comput. Netw.*, 1999, pp. 263–270.

[57] V. Jacobson, "How to kill the Internet," in *Proc. SIGCOMM*, vol. 95, 1995, p. 10.

[58] B. Oki, M. Pfluegl, A. Siegel, and D. Skeen, "The information bus: An architecture for extensible distributed systems," in *Proc. 14th ACM Symp. Oper. Syst. Principles*, 1993, pp. 58–68.

[59] T. Gupta, R. P. Singh, A. Phanishayee, J. Jung, and R. Mahajan, "Bolt: Data management for connected homes," in *Proc. USENIX NSDI*, Apr. 2014, pp. 243–256.

[60] R. Yus, G. Bouloukakis, S. Mehrotra, and N. Venkatasubramanian, "Abstracting interactions with IoT devices towards a semantic vision of smart spaces," in *Proc. 6th ACM Int. Conf. Syst. Energy-Efficient Build. Cities Transport.*, 2019, pp. 91–100.

[61] R. Heydon and N. Hunn, *Bluetooth Core Specification v5.2*, Bluetooth Special Interest Group (SIG), Kirkland, WA, USA, 2019. [Online]. Available: https://www.bluetooth.com/specifications/bluetooth-core-specification/

[62] "ZigBee Alliance." ZigBee Alliance. 2020. [Online]. Available: https://zigbeealliance.org/ (accessed Jul. 15, 2021).

[63] A. Banks, E. Briggs, K. Borgendale, and R. Gupta. "MQTT Version 5.0 OASIS Standard." Mar. 2019. [Online]. Available: https://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html

[64] M. Zhang, V. Lehman, and L. Wang, "Scalable name-based data synchronization for named data networking," in *Proc. IEEE Conf. Comput. Commun.*, 2017, pp. 1–9.

[65] W. Shang *et al.*, "Publish–subscribe communication in building management systems over named data networking," in *Proc. 28th Int. Conf. Comput. Commun. Netw. (ICCCN)*, 2019, pp. 1–10.

[66] K. Nichols, "Lessons learned building a secure network measurement framework using basic NDN," in *Proc. 6th ACM Conf. Inf. Centric Netw.*, 2019, pp. 112–122.

[67] W. Shang *et al.*, "Named data networking of things," in *Proc. IEEE 1st Int. Conf. Internet Things Design Implement. (IoTDI)*, 2016, pp. 117–128.

[68] O. Ascigil, S. Reñé, G. Xylomenos, I. Psaras, and G. Pavlou, "A keyword-based ICN-IoT platform," in *Proc. 4th ACM Conf. Inf. Centric Netw.*, 2017, pp. 22–28.

[69] Z. Zhang *et al.*, "NDNoT: A framework for named data network of things," in *Proc. 5th ACM Conf. Inf. Centric Netw.*, 2018, pp. 200–201.

[70] C. Gündoğran, P. Kietzmann, M. Lenders, H. Petersen, T. C. Schmidt, and M. Wählisch, "NDN, CoAP, and MQTT: A comparative measurement study in the IoT," in *Proc. 5th ACM Conf. Inf. Centric Netw.*, 2018, pp. 159–171.

[71] A. Chakraborti, S. O. Amin, A. Azgin, S. Misra, and R. Ravindran, "Using ICN slicing framework to build an IoT edge network," in *Proc. 5th ACM Conf. Inf. Centric Netw.*, 2018, pp. 214–215.

[72] Z. Zhang *et al.*, "Evolving intelligent devices for the future via named data networking," *Crossroads ACM Mag. Students*, vol. 26, no. 1, p. 36–39, Sep. 2019. [Online]. Available: https://doi.org/10.1145/3351482

[73] C. Gündogan, C. Amsüss, T. C. Schmidt, and M. Wählisch, "Toward a RESTful information-centric Web of Things: A deeper look at data orientation in CoAP," in *Proc. 4th ACM Conf. Inf. Centric Netw.*, 2020, pp. 77–88.

[74] M. S. Lenders, C. Gündogan, T. C. Schmidt, and M. Wählisch, "Connecting the Dots: Selective fragment recovery in ICNLoWPAN," in *Proc. 4th ACM Conf. Inf. Centric Netw.*, 2020, pp. 70–76.

[75] Z. Zhang, Y. Yu, S. K. Ramani, A. Afanasyev, and L. Zhang, "NAC: Automating access control via named data," in *Proc. IEEE Military Commun. Conf. (MILCOM)*, 2018, pp. 626–633.

[76] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, pp. 612–613, Nov 1979. [Online]. Available: https://doi.org/10.1145/359168.359176

[77] L. T. A. N. Brandao, N. Mouha, and A. Vassilev, "Threshold schemes for cryptographic primitives," Nat. Inst. Stand. Technol., Gaithersburg, MD, USA, Rep. NISTIR 8214, Mar. 2019. [Online]. Available: https://doi.org/10.6028/NIST.IR.8214

[78] Z. B. Celik, P. McDaniel, and G. Tan, "Soteria: Automated iot safety and security analysis," in *Proc. USENIX Annu. Tech. Conf. (USENIX ATC)*, 2018, pp. 147–158.

[79] S. Birnbach, S. Eberz, and I. Martinovic, "Peeves: Physical event verification in smart homes," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security*, 2019, pp. 1455–1467. [Online]. Available: https://doi.org/10.1145/3319535.3354254

[80] H. Chi, Q. Zeng, X. Du, and J. Yu, "Cross-app interference threats in smart homes: Categorization, detection and handling," in *Proc. 50th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw. (DSN)*, 2020, pp. 411–423.

[81] C. Fu, Q. Zeng, and X. Du, "HAWatcher: Semantics-aware anomaly detection for appified smart homes," in *Proc. 30th USENIX Security Symp. (USENIX Security)*, Aug. 2021, pp. 4223–4225, [Online]. Available: https://www.usenix.org/conference/usenixsecurity21/presentation/fu

[82] "AWS IoT Greengrass: Bring Local Compute, Messaging, Data Management, Sync, and ML Inference Capabilities to Edge Devices." Amazon Web Services. [Online]. Available: https://aws.amazon.com/greengrass/ (accessed Jul. 15, 2021).

[83] E. Fernandes, J. Jung, and A. Prakash, "Security analysis of emerging smart home applications," in *Proc. IEEE Symp. Security Privacy (SP)*, 2016, pp. 636–654.

[84] N. Carlini *et al.*, "Hidden voice commands," in *Proc. 25th USENIX Security Symp.*, 2016, pp. 513–530.

[85] G. Zhang, C. Yan, X. Ji, T. Zhang, T. Zhang, and W. Xu, "DolphinAttack: Inaudible voice commands," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security*, 2017, pp. 103–117.

[86] T. Sugawara, B. Cyr, S. Rampazzi, D. Genkin, and K. Fu, "Light commands: Laser-based audio injection attacks on voice-controllable systems," in *Proc. 29th USENIX Security Symp. (USENIX Security)*, Aug. 2020, pp. 2631–2648. [Online]. Available: https://www.usenix.org/conference/usenixsecurity20/presentation/sugawara

[87] E. Ronen, A. Shamir, A.-O. Weingarten, and C. O'Flynn, "IoT goes nuclear: Creating a ZigBee chain reaction," in *Proc. IEEE Symp. Security Privacy (SP)*, 2017, pp. 195–212.