12-4-2017

SECURE BOOTSTRAPPING AND ACCESS CONTROL IN NDN-BASED SMART HOME SYSTEMS

Lei Pi

SECURE BOOTSTRAPPING AND ACCESS CONTROL IN NDN-BASED SMART HOME SYSTEMS

by

Lei Pi

A Thesis

Submitted in Partial Fulfillment of the

Requirements for the Degree of

Master of Science

Major: Computer Science

The University of Memphis

December 2017

Abstract

Smart home systems utilize network-enabled sensors to collect environmental data and provide various services to home residents. Such a system must be designed with security mechanisms to protect the safety and privacy of the residents. More specifically, we need to secure the production, dissemination, and consumption of smart home data, as well as prevent any unauthorized access to the services provided by the system. In this work, we study how to build a secure smart home system in the context of Named Data Networking, a future Internet architecture that has unique advantages in securing Internet of Things. We focus on solving two security problems: (a) mutual authentication between a new device and an existing smart home system to bootstrap the device, and (b) controlling access to smart home data. We designed a naming hierarchy for a smart home system and the corresponding trust model. Based on the naming and trust model, we designed bootstrapping protocols which enforce mutual cryptographic challenges, and a programming template which facilitates Name-based Access Control. We have designed and implemented an application that incorporates these solutions. Evaluation result shows: (a) the bootstrapping protocols can defend against replay attacks with a small computation overhead, and (b) Name-Based Access Control can provide accurate time schedules to restrict access to fine-grained data types with a small computation overhead.

TABLE OF CONTENTS

List of Figures vi

1 Introduction 1

2 Background 5
- 2.1 Existing Smart Home Systems 5
- 2.2 Named Data Networking 6
  - 2.2.1 Interest/Data Exchange and NDN Node Model 7
  - 2.2.2 Naming 9
  - 2.2.3 Trust Model 9

3 Related Works 11

4 Design Overview 13
- 4.1 Assumptions 13
- 4.2 Threat Model 14
- 4.3 Life cycle of a Smart Home Device 15
- 4.4 Naming Design 15
- 4.5 Bootstrapping Protocol Requirements and Design 16
- 4.6 Access Control Requirements and Design 18

5 Bootstrapping Trust 19
- 5.1 Introduction 19
- 5.2 Trust Bootstrapping in Smart Homes 20
- 5.3 Bootstrap Protocol Design 21
  - 5.3.1 Trust model 21
  - 5.3.2 Bootstrap namespace 22
  - 5.3.3 Initialize bootstrapping 22
  - 5.3.4 Learning the trust anchor 23
  - 5.3.5 Learning the device 23
  - 5.3.6 Issuing certificate for the device 23
  - 5.3.7 Design options and reasoning 24
  - 5.3.8 Encryption-based Bootstrapping Protocol 26
- 5.4 Evaluation of Trust Bootstrap Protocol 27
  - 5.4.1 Performance Impacts 27
  - 5.4.2 Security Arguments 29
  - 5.4.3 Evaluation Summary 30

6 An Application and Evaluation of the Name-based Access Control 31
- 6.1 Introduction 31
  - 6.1.1 Intuition of Name-based Access Control 31
- 6.2 Applying NAC to smart home applications 32
- 6.3 Implementing Smart Home Data Access Control using NAC 36
- 6.4 Evaluating NAC 37

6.4.1 NAC Performance Overheads 37
6.4.2 NAC Security Analysis 39

7 Conclusion 42

References 43

LIST OF FIGURES

Figure		Page
1.	Protecting the production and consumption of private data	3
2.	A new device has no trust relationship with the smart home system	3
3.	NDN Interest and Data Packets	8
4.	NDN Node Model	8
5.	NDN Example	9
6.	Overview	14
7.	Attacker in Wireless Environment	15
8.	Attacker in wired Environment	15
9.	Lifecycle of a device's service	16
10.	Overall namespace hierarchy example	17
11.	Overall name space deisn	17
12.	Before and After bootstrapping	20
13.	Bootstrapping Protocol Using HMAC	22
14.	Bootstrapping Protocol Using encryption	26
15.	Fake Owner Attack	29
16.	Fake Device Attack	29
17.	Name-based Access Control	32
18.	NAC Naming Rules	34
19.	Smart Home Data Namespace Example	34
20.	Signing Hierarchy for the Trust Model	35
21.	Class Diagram For the Application Library	35

22. Mini Smart Home System 36

23. Mini Smart Home Smart Sensor 37

24. Owner’s Phone 38

25. Owner’s App 38

26. Consumer 38

27. NAC Computation Overheads - Chart 39

Chapter 1

Introduction

Morden smart home applications utilize sensor collected data to provide various services. Smart climate control uses thermostat and electric meter readings to optimize home temperature level and energy usage. Intelligent intrusion detection system distinguishes normal activities and intrusion behaviors based camera recording and Radio Frequency (RF) readings. Smart home data like the camera recorded images and thermostat readings are private to the owner of the house. Its production and consumption must be well secured to ensure smart service operations and protect owner's privacy. The current Internet lacks adequate architectural support to build secure smart home systems composed of thousands of interconnected sensors and Internet of Things devices. In this work, we study how to create secure smart home systems over Named Data Networking (NDN), a futuristic Internet architecture which adopts the data-centric approaches. We focus on securely establishing trust relationships between new devices and the smart home system (Bootstrapping), and enforce access schedules and policies to authorize the production and consumption of data (Access Control). Our contributions include 1) identifying the typical device lifecycles in NDN, 2) designing an organized and expressive name hierarchy to reflect application meaning and a trust model to facilitate automatic trust management, 3) designing and implementing a bootstrap protocol to defend against man-in-the-middle (MitM) attacks, 4) designing and implementing a smart home application adopting Name-based Access Control to protect home data production and consumption, and 5) developing a programming library with code templates to enforce our designs in application on macOS, Linux and Android platforms.

When a new device is brought to the home, there is no trust relationship between the device and the smart home system (Figure 2.). The device has little knowledge about the owner and approved devices, as a result it cannot trust the message received from other devices. And other devices do not know if the new device is approved by the owner and if

its data can be trusted. This trust relationship must be bootstrapped when the new device joins the home and be managed automatically. After the trust bootstrapping, the new device can publish data that is verifiable by other devices. Bootstrapping trust alone is not enough to secure the smart home system. And access to smart home data must also be restricted in a manageable way. This is because only the homeowner owns the devices and the data they collected or generated. On one hand, it means only the devices approved by the owner can produce trusted private data. A climate control system should only trust the temperature readings produced by a trusted thermostat to ensure the correct instructions are sent to the AC or heater. It is hard to prohibit a device from producing and broadcasting data, but by designing an Owner-oriented trust model, we can make sure the unapproved data cannot be trusted in the network. On the other hand, only a device approved by the owner can consume a type of private data. For example, a smart baby care system uses camera in the baby's room to monitor its activities. Only the family member should be able to watch the baby's recordings. A visitor should not unless allowed by the owner. Figure 1. shows the scenario of the examples.

Smart home system relies on a network infrastructure to distribute application data among devices. It is hard to gain adequate architectural support using the current Internet. Existing solutions for smart home systems largely adopts IP-based or host-centric approaches, and are either inefficient or insufficient to support energy friendly smart home applications [1]. When the Internet was originally designed, computing resources were so scarce that the primary use of a network was to connect people to time-shared servers. As such, a core abstraction in the current Internet architecture is a host (a client or a server), and communication is supposed to happen between end hosts. However, todays Internet applications are increasingly data-centric. People on social networking sites create massive amount of content such as video, audio, news, blogs, tweets, and images. Meanwhile, smart homes also generate a huge amount of data that need to be accessed on-demand for various purposes such as monitoring and decision making. A data-centric

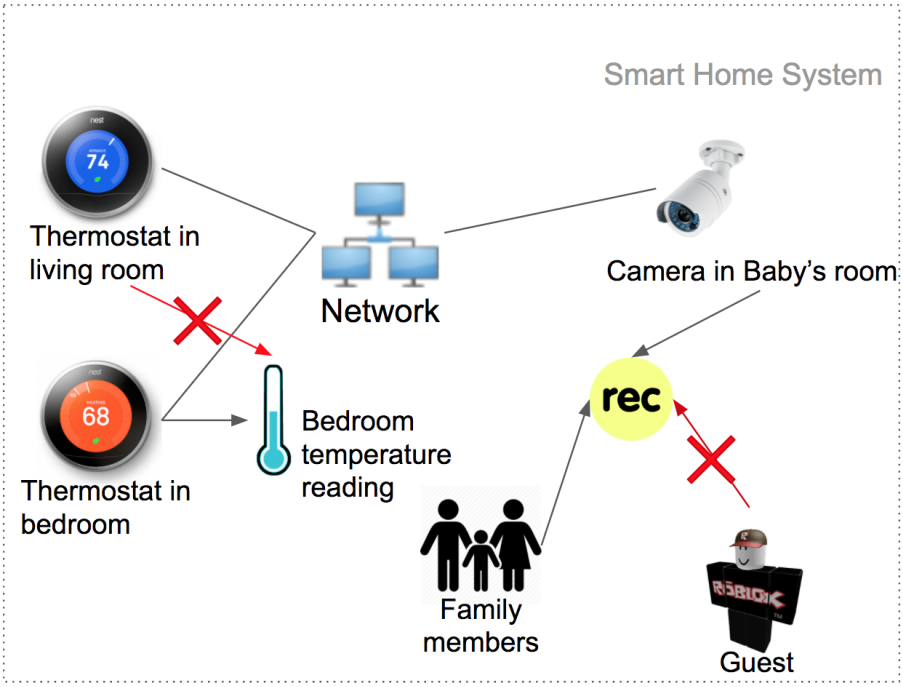

Figure 1. Protecting the production and consumption of private data

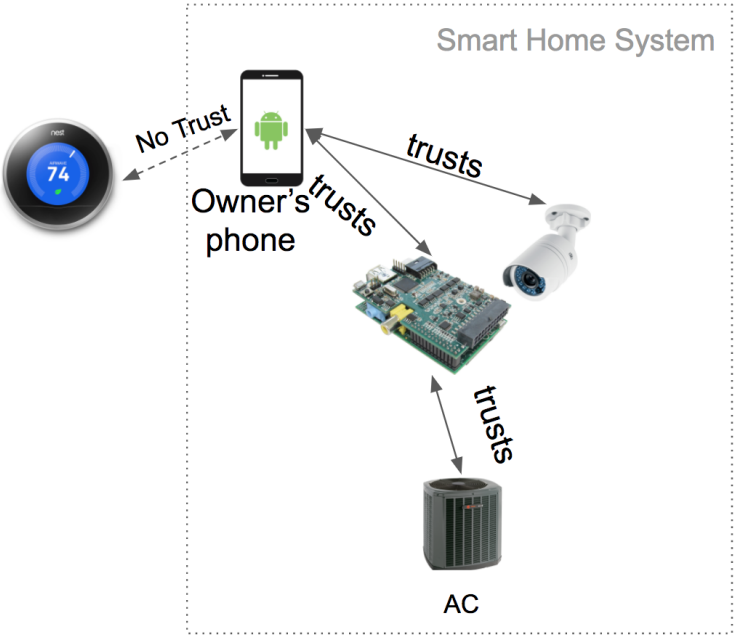

Figure 2. A new device has no trust relationship with the smart home system

or information-centric architecture like the Named Data Networking (NDN) will support more efficient and effective smart-home and other IoT applications. NDN makes ”data”

the primary abstraction of the architecture and it is the most prominient realization of the Information-Centric Networking paradigm.

The organization of this thesis is as follows. In Chapter 2, we briefly introduce the security threats towards smart home data, the limitations to build secure smart home applications over the traditional network, and Named Data Networking. Chapter 3 introduces related works to protect data security. Our solutions include Chapter 4 to 6. Chapter 4 describes the high-level analysis, requirements and design. In Chapter 5, we design a protocol over NDN to mutually authenticate a new device and the homeowner and establishing a trust relationship between the two to prepare the device for normal operations. In Chapter 6, we use Name-based Access Control (NAC) to restrict data consumption to designated recipients and evaluate the practicality and effectiveness of NAC. And Chapter 7 summarizes and ends the thesis.

Chapter 2

Background

2.1 Existing Smart Home Systems

In recent years, many companies developed programming frameworks to build secure, energy-efficient and user-friendly smart home applications that connects and automates home products. For example, Nest Weave[2],ZigBee Home Automation[3], Samsung SmartThings[4] and Google Thread[5]. They support different underlying networking protocols and topologies and implements security mechanisms accordingly to protect the homeowner and the integrity of the smart home. Table 1. compares these frameworks on the style of networking, trust bootstrapping and access control.

ZigBee Home Automation is an industry standard for smart homes which utilizes ZigBee meshed network stack[3]. Its built-in security features support bootstrapping a new device with the device installation code and establishing trust by distributing a networking key to the device for onboarding. The device can then negotiate application-layer keys to further enforce integrity and develop access control policies. The networking is based on an address-based communication model, and security policies are channel-oriented which focuses on securing the communication channels and the identities of endpoints. The whole home network shares a single network key to enforce the perimeter-style security. Once the network key is disclosed [6], all plaintext communications over the network is exposed. Before the device joins the network, ZigBee

Platform	Networking Style	Trust Bootstrap	Access Control
ZigBee Home	Mesh	Install Code Network Key	Left for Application
SmartThings	Cloud	OCF Security	role-based/topic-based
Goolge Thread	Mesh, 6LoWPAN	Password + DLTS	Left for Application
Nest Weave	Cloud	Pairing Code	OAuth2.0

Table 1. Smart Home Applcation Frameworks

uses plaintext to exchange the device's information for device discovery, which also has security and privacy implications.

Samsung SmartThings is an application framework that eases the development of smart home applications to connect and automate home appliances. It is built upon IP-based network (CoAP over UDP) and enforces Open Connectivity Foundation (OCF) Security Framework to manage trust and provide application-layer privilege separation via subject-based or role-based access control. The current implementation relies on a cloud service. Recent research [7] shows it has the over-privilege risk where an application can easily gain privilege it does not need according to its function description.

Google Thread[5] is a networking protocol which adopts 6LoWPAN [8] to securely establish decentralized networking between IoT devices and to connect the home network to the Internet. It uses EC-JPAKE [9], a password-based key negotiation/agreement protocol to securely establish communication with a new device, and uses DTLS to onboard the device by establishing a secure communication session. As a networking layer protocol, it does not provide access control solutions.

Nest Weave is an open communication protocol to develop smart home devices that can be managed and controlled using Nest services. It powers the Nest devices and applications. It requires the device to provide a built-in pairing key to bootstrap trust and establish ownership with the user's Nest account. The user can grant a device permissions to access its sensitive data on Nest cloud through OAuth2.0 protocol.

All these protocols and solutions either depend on cloud services which will not work when there is a disruption in Internet connectivity or does not provide fine-grained access control to protect data production and consumption.

2.2 Named Data Networking

Smart home applications depend on the underlying networking architecture to support information flow among devices. Plenty networking standards can be used to build smart homes. For example, Zigbee, ZWave, Bluetooth, and IPv6 derived protocols

(such as 6LoWPAN). Despite the differences in connectivity establishment, the mainstream security mechanism is channel-based security, where data is transmitted over an encrypted channel from the sender to its recipient. For example, ZigBee uses a shared network secret-key to secure the network layer broadcasts and application keys for application sub-layer point-to-point channels. Such security models usually result in complex security implementations such as the key management and delivery systems in ZigBee and increase the risk of exposing vulnerabilities to attackers and introduces computation and storage overheads. The complexity and the vulnerabilities along with it are amplified when a middlebox such as a proxy or a cache node is envolved in the loop. An alternative approach to build low-cost, secure and energy efficient applications in smart homes is needed for the IoT devices. To avoid the limitations in channel-based security, Named-data Networking (NDN) implements the *data-centric security* which coincides with the *object-oriented security* model [10] and secures the application data directly instead of the channel in which they are transmitted. The data-centric approach adopted by NDN also brings other benefits such as the decoupling of producer and consumer and in-network caching, which is good to develop energy efficient applications on IoT devices. In this section, we will introduce how NDN works and how trust models in NDN can help to secure the data.

2.2.1 Interest/Data Exchange and NDN Node Model

NDN enables users and applications to fetch data identified by a given name directly. Communication in NDN is driven by receivers, i.e., data consumers, through the exchange of two types of packets: Interest and Data (Figure 3.). A consumer puts the name of a desired piece of data into an Interest packet and sends it to the network. Routers use this name to forward the Interest toward the data producer(s). Once the Interest reaches a node that has the requested data, the node will return a Data packet which contains a cryptographic signature. This Data packet follows in reverse the path taken by the Interest to get back to the requesting consumer. When receiving an Interest packet, an

NDN node (Figure 4.) remembers the interface from which the Interest comes in, forward the Interest according to the Forwarding Information Base (FIB) maintained according to a name-based routing protocol, and puts the Interest in the Pending Interest Table (PIT). If the Interest reaches a node that can serve the data, the nodes send back the Data packet along the reverse route of the interest, removes the Interest from PIT and cache it in the Content Store (CS). The name and signature of a Data packet enable packet-level security. First, the cryptographic signature binds the name and data and removes the need for channel security or container security. Second, every data is named explicitly, and the information contained in the hierarchical name provides context for trust management.

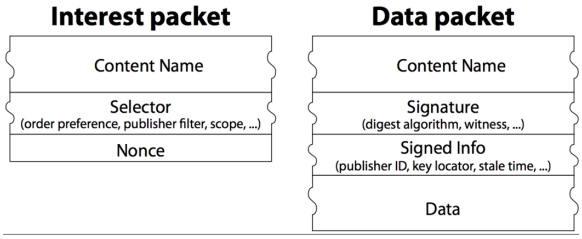

Figure 3. NDN Interest and Data Packets

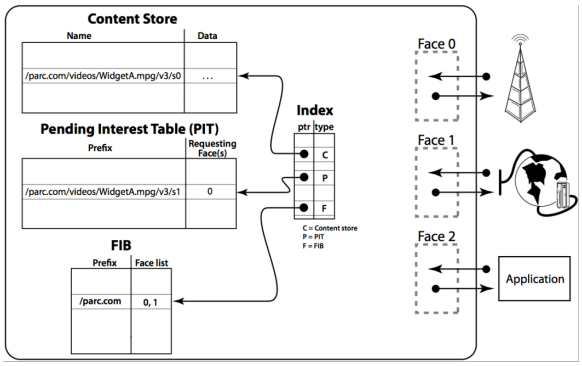

Figure 4. NDN Node Model

For example (Figure 5.), a consumer wants to read the temperature in the bedroom, it sends out an interest with name "`/bedroom/temperature`" using a local Face, the local Face talks with a connected network forwarder (NFD) to forward the interest over NDN according to a name-based routing protocol. Once the producer receives the Interest packet, it generates the data and puts back the Data packet to NDN. NDN sends the data

back along the path of the Interest and caches the data automatically. Finally, the consumer gets the data and consumes it.

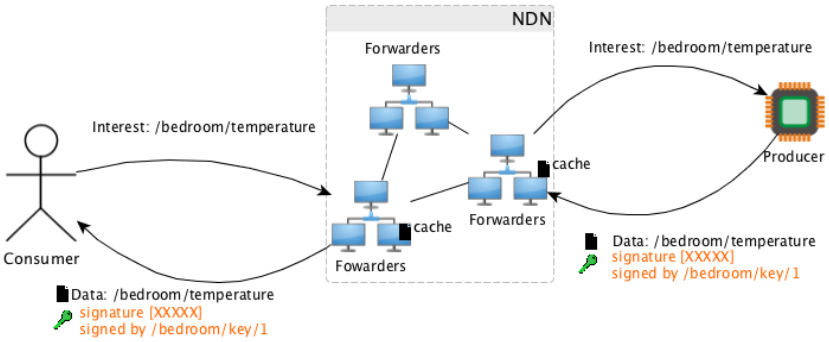

Figure 5. NDN Example

2.2.2 Naming

Distributing and manipulating named information or data is a major application in the modern Internet [11]. NDN recommends hierarchically structured names to represent application data relations and enables scalable routing. For example, the name of a video produced by the camera in the baby's room may be "`/local-home/Recording/Tom/sleeping-101.mp4`". The naming conventions, the application semantics of the names, and the namespace management are opaque to NDN and left to the application's responsibility.

Name is the first-class identifier on an NDN network. But it does not always need to be globally unique. If the application only requires data from a local scope, the name could be just locally unique. The application must be able to generate the name of the prefix of the name for a specific piece of data. For example, one can use "`/local-home/Recording/Tom/sleeping-101.mp4`" and receive the first piece of the video with name "`/local-home/Recording/Tom/sleeping-101.mp4/1`".

2.2.3 Trust Model

NDN requires every data packet digitally signed. The digital signature coupled with data publisher information allows a consumer to verify if the data is authentic

without caring from where the data is fetched. When the data content is a public key, the Data packet with its name is effectively a certificate for the key. Then both key distribution and trust between producers and consumers are simplified. The hierarchical namespace and data provenance through the binding of name and signature leave flexible design options for choosing/designing trust models.

An NDN trust model defines which identities act as the trust anchors, and the rules to determine which identity can be used to prove the authenticity of another one. An identity is the binding of a namespace and a public key for a specific period. It is implemented in the form of NDN *Certificate* and can be used to authenticate data produced by the real world entity. To prove the identity is authentic and authorized, its certificate must be signed by a trusted and authorized identity. Trust anchors are self-signed identities that are trusted by the network by default.

Chapter 3

Related Works

The need for stronger security assurance in IoT implementations, including Smart Homes, has been trending. There has been plenty research and design work on this topic over NDN from various levels and focusing on different scopes.

Trust schemas [12] are sets of rules that automate trust management by matching content names and key names. By designing and specifying trust schemas, we can initialize and enforce the trust model throughout the application, which includes identifying a trust anchor and the trust relationship between identities and keys.

From a high level, the work of [13] discussed how to use NDN to realize the IoT vision. It identifies the challenges as being designing naming models, bootstrapping trust and discovering services, managing/schematizing trust, access control and other application specific requirements. It introduced fundamental ideas of sharing a separate secret to bootstrap trust, and using name-based access control to authorize data access. These ideas served as a starting point for our security design for a smart home.

In [14], a gateway-based architecture was introduced to build a secure integrated home network over NDN. The gateway serves as both a configuration server and the management center for the network, where trust between new devices and homeowner are bootstrapped by inputting a pairing code on the gateway node. It requires a synchronized clock between devices before bootstrapping. It does not provide access control. As it is published in NDN Technical Reports, we will use the phrase "ndnTR0035" to refer to this work.

Building management systems share common challenges with smart home systems, such as sharing sensors data and access controls. The work of [15] describs a design and implementation to solve these problems by using gateway nodes as proxies between sensors over IP-based networks and the NDN, and provide access control using encryptions and access privileges lists (APL).

NDN-ACE [16] is another access control design tailored for actuator applications over NDN in constrained environments. It assumes trust bootstrapping is already done and introduces authorization servers between commanders and actuators to trade speed for space by delegating key management to authorization servers from actuators.

NDN-Flow [17] is an automated home entertainment application. It focuses on leveraging NDN to solve two problems in smart IoT systems, namely trust management and service rendezvous. The former is implemented by specifying trust schemas and the latter by publishing application metadata under a dedicated sub-namespace.

mHealth [18] focuses on demonstrating how Name-based Access Control (NAC) can be applied to securely share health data only to designated recipients. In NAC, the owner generates and distributes separate production and consumption credentials to producers and consumers over time. A producer generates a content key to encrypt the data using a symmetric algorithm, encrypts the content key using the production credentials and publish the encrypted content key. Only designated recipients who hold the consumption credentials can decrypt the content key to decrypt and consume the content.

Chapter 4

Design Overview

Our goal is to design and implement a smart home system which is capable of bootstrapping new devices and restricting access to data (temperature readings in our demo) produced by each device for each consumer. This chapter gives an overall introduction to the assumptions, architectural overview, and intuitions of the design. We take a top-down approach when analyzing the problem and designing the solution. First, the assumptions and scopes of the problem must be defined. We list the pre-conditions, analyze the lifecycle of an IoT device in the smart home, and identify the problems we need to solve in each phase. Specifically, during the bootstrapping phase, trust must be securely established between the smart home system and the new device. During the configuration phase, the owner must be able to restrict data access to each device. The overall architecture must support multiple devices. Second, the requirements and solutions for each problem are summarized to serve as an outline of the design. The following chapters elaborate on detailed designs.

4.1 Assumptions

When an NDN Network Forwarder Daemon (NFD) receives an Interest, it must know how to forward it according to the Interest packet's name. The forwarding strategies and entries of each NFD can be configured manually or automatically. In our design, we assume there exists a well-known namespace "`/local-home`". Each NFD in the local home knows how to forward packets with this name prefix.

When the manufacturer produces a device, it assigns a universally unique identifier (UUID) to the device. Each device can share a secret code with the owner out-of-band. For example, the device body may have a printed barcode, and the owner can learn the secret code by scanning it. Or the more capable devices can generate and display a code on demand.

In the smart home system (Figure 6.), multiple IoT devices are providing various

services. The owner of the home uses a dedicated controller to manage, configure, and monitor the devices. For example, the controller can an Android phone app. In NDN, identities are represented in the form of certificates. We assume the owner's certificate can be managed and used directly by the controller.

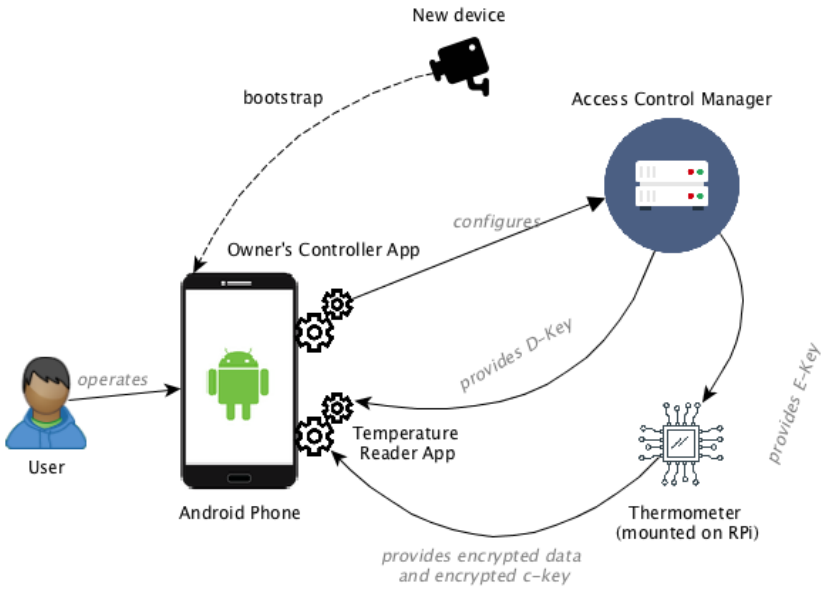

Figure 6. Overview

4.2 Threat Model

We assume an attacker has powerful capabilities to sniff, modify and send NDN packets, but it cannot break cryptographic primitives (Figure 7. and 8.). For example, it can launch the man-in-the-middle (MitM) attacks by recording, modifying and resending a packet, but the attacker cannot break the encryption method in use or fake a keyed-hash message authentication code without knowing the key. We will not discuss side-channel, OS or hardware specific attacks and consider them out of scope. For bootstrapping, we consider two kinds of attacks to evaluate the effectiveness of the protocol to defend against fraudulent bootstrapping attacks. First, an attacker Trudy tries to pretend to be an authentic device and onboard the home system. She uses recorded bootstrapping messages to do so. Second, an attacker Trudy tries to pretend to be a controller to trick the

device into believing it joined an authentic home system. For access control, we consider an attacker Trudy managed to compromise one device and see if she can gain access to data not granted.

4.3 Life cycle of a Smart Home Device

A typical lifecycle of an IoT device in a smart home over NDN has 7 phases (Figure 9.).

Upon power-on, the device performs necessary hardware checks, load the OS and start running; then it starts network configurations. Upon successfully configuring the network, the device gains physical connectivity and reachability with other devices on the same network. It initiates bootstrapping trust with the homeowner to learn the trust anchor of the home and establish an identity of itself by retrieving a certificate signed by the owner. It may need a follow-up phase for application specific configurations, then starts operating normally for daily life. Optionally, the device provides observability for the user or a supervisor application to monitor its service quality and tweak configurations accordingly. At last, it should be able to get unloaded from the smart home system for a reset or decommission.

4.4 Naming Design

We design a hierarchical namespace for a local smart home scope compatible to schematized trusting to build the trust models later. All names start with a "`local-home`" component denoting the local-home scope. The "`Key`" branch represents the name of the

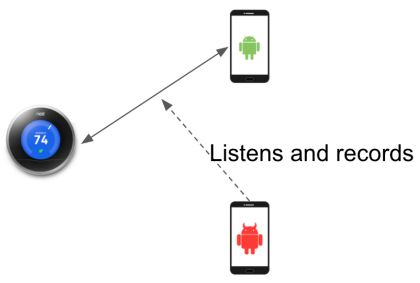

Figure 7. Attacker in Wireless Environment

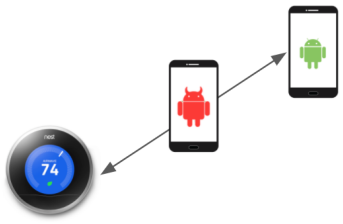

Figure 8. Attacker in wired Environment

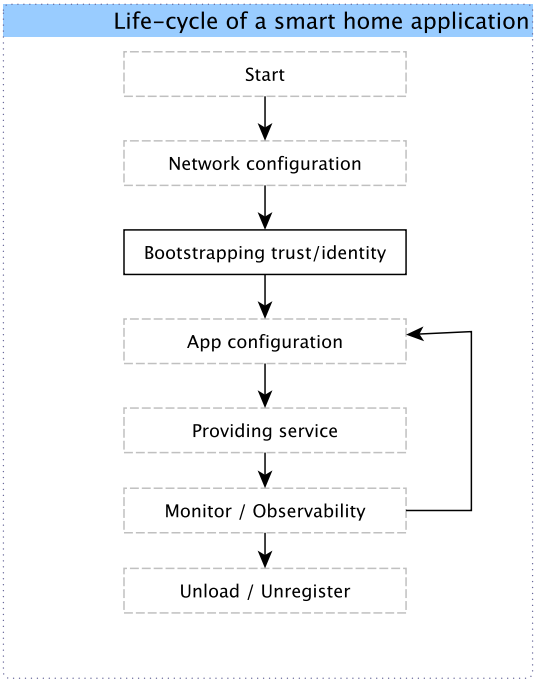

Figure 9. Lifecycle of a device's service

owner's keys. The "`bootstrap`" branch is for naming data packets used in the bootstrapping process. The "`samples`" and "`read`" branches are required by NAC, where the former is for naming produced data, and the latter is for naming production/consumption credentials. Figure 10. shows an example for a smart home with a remote temperature sensor. Figure 11. list the rules for the naming hierarchy following semantics in schematized trusting[12].

4.5 Bootstrapping Protocol Requirements and Design

The purpose of bootstrapping protocol is to mutually authenticate the device and the owner, enable the device to learn the owner's identity and establish an identity for the device. Only the authentic device and the real owner can pass the protocol. The bootstrap process must satisfy the following security requirements:

1. Any critical packet that will trigger a protocol status change must be checked for integrity and authenticity;

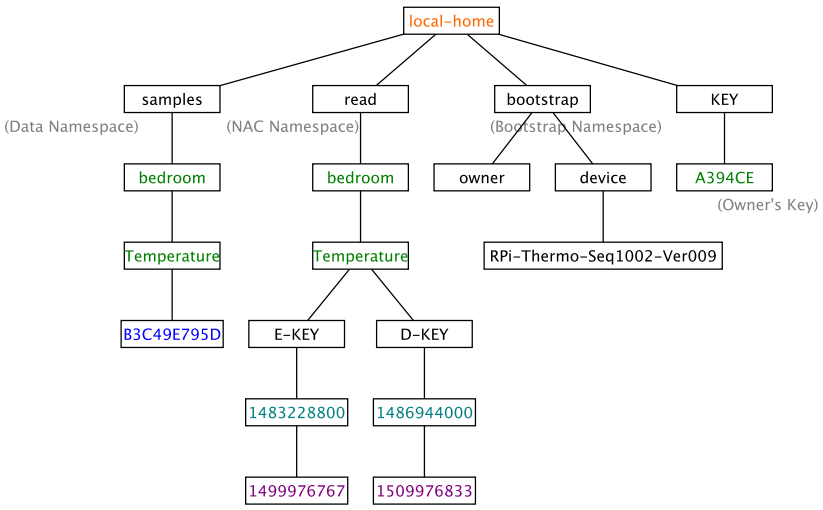

Figure 10. Overall namespace hierarchy example

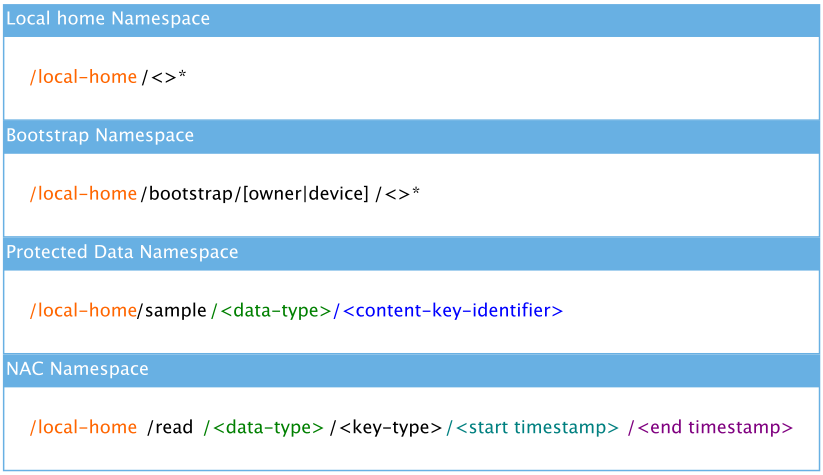

Figure 11. Overall name space deisn

2. The protocol should be secure against replay and MitM attacks;

3. The memory, CPU and network overhead must be reasonable to resist resource exhaustion attacks.

We design the bootstrapping protocol by using secret keys shared out-of-band and performing mutual cryptographic challenges embedded in two synchronized Interest/Data message rounds. In each round, the initiator poses a random number as the challenge, and

the responder must answer by applying a required cryptographic function to the random number with the shared secret between the device and the owner.

4.6 Access Control Requirements and Design

The purpose of implementing access control is to evaluate the effectiveness of NAC and provide fine-grained access control in the smart home system. The implementation must satisfy the following security requirements:

1. An authorized identity must be able to access the content as long as the content is still valid.

2. An unauthorized identity must not be able to access the content

We apply NAC by designing and implementing helper libraries over NAC codebase and develop a mini smart home system which enables a user to read remote temperature data published on a Raspberry Pi on an Android phone and control access to it via an owner's app.

Chapter 5

Bootstrapping Trust

5.1 Introduction

Given the trust model in the NDN context, a new device knows nothing about the trust anchor and vice versa. The device also does not know under which namespace should it publish its data, because this information should be configurable according to the needs of the owner, the higher level application, the specific scenarios, etc. During bootstrapping, the device needs to learn the trust anchor and its namespace for data publishment. The trust anchor should learn the device's public key and generate a certificate for the device which contains both the device's public key, namespace, and the trust anchor's signature. Upon a successful trust bootstrapping, the device should gain confidence it joined a trustable smart home environment under the permission of the homeowner, and its published data will be trusted over the network. Then it may start its operating process to fetch runtime configurations or begins publishing data. The owner's app should gain confidence that the new device is joined by the approval of the user and it is what it claims to be. Then the owner's app can update status according, for example, show the device on a list of all bootstrapped ones. It is vital that during this process each party can authenticate the other party to make sure the other is present and its message is authentic. Otherwise, the attacker can trick one side of the two to transfer it trust status when the other party is not even present, and further trigger the device to perform actions not wanted by the owner. Take a hypothetical smart voice recorder as an example. The recorder automatically starts recording when it detects sound in a specific room (such as a baby's room). If the attacker tricks the recorder to believe it has bootstrapped successfully, it may start recording without the owner's knowledge and use up its storage space slowly. The trust bootstrap protocol must be able to defend against such kind of attacks. By doing so, it also decouples the trust establishment from the other lifecycle phases.

5.2 Trust Bootstrapping in Smart Homes

A homeowner O owns a set of smart devices $\mathbb{D}$ in a home. O configures devices using a controller C. Before working correctly on the network in the house, a new device $D_i \in \mathbb{D}$ must gain mutual trust with the controller C (which represents the owner O) and fetch secure and application configurations approved by O for subsequent operations. After bootstrapping, D_i must have learned how to identify and authenticate data generated by O, and vice versa. In NDN, this means:

- D_i learns the certificate of O so that it knows who is the authentic owner of the home
- D_i retrieves a certificate signed O so that it can publish data under the same name prefix with the newly issued certificate.

Figure 12. shows an example of the trust status before and after bootstrapping. Before bootstrapping, there is no way for the AC to verify if the data produced by the thermometer can be trusted or not. After the bootstrapping, the thermostat produces data under the namespace "`/local-home/bdr1/temperature`", and signs the data using its private key. The AC can verify the authenticity of the data by checking the data signature against the certificate and gain the confidence of trusting the data.

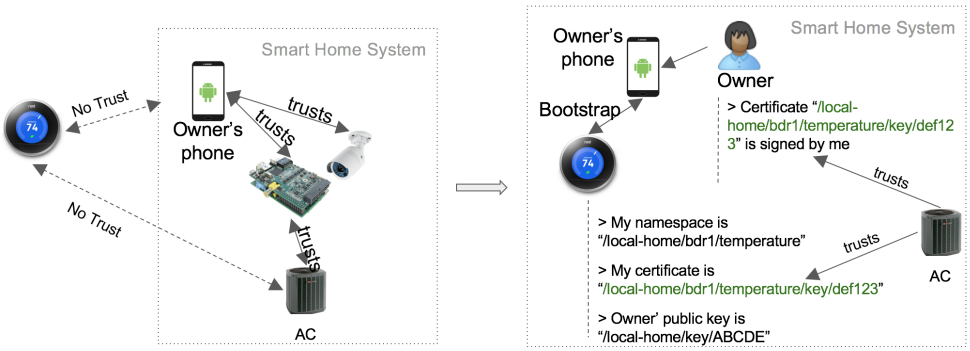

Figure 12. Before and After bootstrapping

5.3 Bootstrap Protocol Design

In this section, we present a protocol design that focuses more on the security goals than communication properties.

To gain trust between two parties without the help of a third one, the two sides must share some secret knowledge between them and be able to use the secret to verify the other's message. In our design, D_i shares its unique identifier "`<device-id>`" and a secret pairing key K with C out-of-band before bootstrapping. Only D_i and C have access to K. K is securely protected from disclosure before bootstrapping.

5.3.1 Trust model

An NDN trust model defines which identity acts as the trust anchor, and which identity can be used to prove the authenticity of another one. An identity is the binding of a namespace and a public key for a specific period. It is implemented in form of *certificates* in NDN. An NDN certificate is a data packet. Its name contains the identity's namespace, and its contents the public key and other parameters. To prove the claim is authentic and authorized, the certificate must be signed by a trusted and authorized identity. We establish a center point trust model in the local home by defining the owner O's self-signed certificate $Cert_o$ as the trust anchor of the network. All other trusted certificates must be signed by O. This design can be easily extended to schematized trusting[12], to devise rules defining which identity is authorized to sign certificates for which namespaces and to automate the distribution and verification of the certificates.

By specifying the trust model, the goal of Bootstrapping becomes 1) mutually authenticate a device and the trust anchor, 2) let the device retrieve the NDN certificate of the trust anchor and 3) trust anchor issues an NDN certificate for the device. The NDN certificate of the trust anchor contains the namespace and public key information of the trust anchor. The NDN certificate for the device is signed by the trust anchor and defines the namespace for the device. After this process, the device can verify the authenticity of other messages over the network, and any consumer of the device's data can do the same.

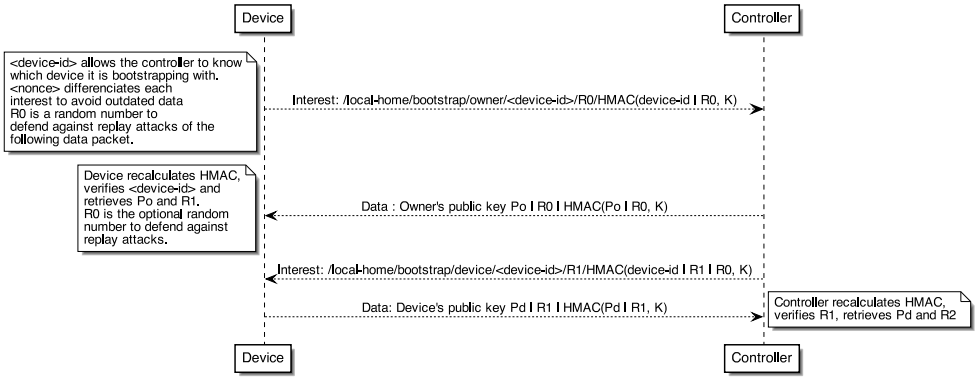

Figure 13. Bootstrapping Protocol Using HMAC

5.3.2 Bootstrap namespace

Before starting the bootstrapping, D_i and C must agree on a well known namespace, for example, "`/local-home/bootstrap`", where both of them are reachable. C and D_i publish data under this well-known namespaces for bootstrapping. For example, C uses "`/local-home/bootstrap/owner`" , and D_i uses "`/local-home/bootstrap/device/<device-id>`". The owner's name can be learned from O's certificate $Cert_o$. And the namespace for D_i to publish its sensor data can be learned after the bootstrapping under another well known namespace such as "`/local-home/configuration/namespace/<device-id>`".

5.3.3 Initialize bootstrapping

The device D_i initiates the process by sending out an interest "`/local-home/bootstrap/owner/<device-id>/<`R_0`>/HMAC(<device-id> | <`R_0`>, <K>)`". "`K`" is the shared secret between the device D_i and owner's controller C. "`HMAC`" is a keyed-hash message authentication code function. "`<`R_0`>`" is a random number. It serves two purposes. 1) It is part of a cryptographic challenge for the C to solve later to prove the authenticity of the controller. 2) It differentiates different bootstrapping interests from the same device with the same key. The network will forward this interest to C. Upon receiving this interest, C recalculates HMAC to verify 1) if the

interest name is altered and 2) if the producer of this interest owns the shared key K. C only proceeds when the verifications pass.

5.3.4 Learning the trust anchor

Once C verifies D_i's bootstrapping interest, it sends back the trust anchor's self-signed certificate $Cert_o$, concatenated with ***HMAC(***$Cert_o$ — ¡R_0¿, ***K***. D_i verifies the message by comparing R0 and recalculating the HMAC. If the verification is passed, D_i stores $Cert_o$, but does not accept it as the trust anchor at this time.

5.3.5 Learning the device

C sends out an interest "`/local-home/bootstrap/device/<device-id>/<`R_0`>/<`R_1`>/HMAC(<device-id> | <`R_0`> | <`R_1`>, K)`". R_1 is a random number generated by C. It serves the same purposes as R_0 in D_i's interest. R_0 in this interest identifies which bootstrapping interest from D_i was received and processed by C. Upon receiving and verifying this interest, D_i sends back its public key $PubKey_{D_i}$, concatenated with ***HMAC(***$PubKey_{D_i}$ *— R1, K)*. Now, D_i considers the challenge of R_0 is fulfilled by the C, and will accept the $Cert_o$ as the trust anchor.

5.3.6 Issuing certificate for the device

C submit $PubKey_{D_i}$ to the owner, and the owner issues a certificate $Cert_{D_i}$ by signing $PubKey_{D_i}$ using O's private key. D_i fetches its certificate by sending interest "`/local-home/bootstrap/cert/<device-id>`". The certificate is encapsulated as the payload of the responding data packet. It is named in format "`/local-home/device/<device-id>/<key-id>/KEY/<version>`", binding the namespace "`/local-home/device/<device-id>`" with the device's public key. The data packet is signed with the C's key, which is signed by the O. And the device D_i can verify the authenticity of the packet by applying schematized trusting.

This step is required for the device to work. But as our designs simplifies the process for naming the device, real work application should adapt to requirements to

negotiate a name for the device before issueing the certificate. On one hand, the device and the owner has mutually authenticated and learned each other's public key to authenticate further messages. On the other hand, though issuing certificate and binding namespaces are necessary to perform meaningful operations in NDN, it is also customizable to each application. The namespace to bind to the device could be different and how the owner's keys are securely managed is left an open question. One simple solution is letting the C to store and update the O, making it the only controller in the home. In a sophisticated, systematic design, the storage of the O should be decoupled with the implementation of C and should only be accessible to the user who proves itself as an owner.

5.3.7 Design options and reasoning

In this part, we will discuss other design options and the reasons why we didn't choose them. First, to initiate the bootstrapping process, we had two options:

1. D_i initiates a bootstrapping session by expressing an interest periodically until it receives a verifiable data published by C.

2. C initiates a bootstrapping session by probing the presence of D_i by expressing an interest periodically.

The two options differ in which principle (C or D_i) will do the polling while waiting for the other principle to be ready. Hypothetically, C is more likely to be ready and present on the network than D_i. For example, the C is a smartphone app which is always running in the background or can be launched very fast on demand, and a new device such as a newly bought thermometer needs to power up and boot before onboarding. When the device intiates the process, the owner's app won't need to know the device information ahead of time. Because the device can securely embed the information in the first Interest to the owner. Then owner only needs to input the device's secret to continue the bootstrapping. Thus in our design, the device D_i sends out the first interest to start the process when it is

ready, rather than the C polling a namespace until D_i is ready. Second, to ensure packet integrity and authenticity, we can either

1. use an authenticated encryption, such as AES-EAX, AES-OCB[19], etc, on each packet.

2. use keyed-hash message authentication code (HMAC).

We consider two questions. 1) is it necessary to keep confidential the details of the bootstrapping packets? We use random numbers to defend against replay attacks. Leaving those numbers in plaintext may expose an attack surface making it vulnerable to table attacks. We will discuss later in this article that, given benign devices all behaving well, encryption is not necessary to resist table attacks. 2) what's the performance and message overhead? Hash functions are in general faster than encryption and decryption. In the HMAC design, there are four messages required. In the encryption-based design, there are 6. We prefer using HMAC than encryption, especially for its performance benefits. However, we also present an encryption-based bootstrapping protocol as a supplement. Third, to differentiate each packet from the same principle for the same purpose and resist replay attacks, we could use sequence numbers, timestamps, or random numbers. Sequence numbers trade extra state management on each principle for the ordinal property. It introduces more storage overhead if the sequence numbers need to be remembered across power on and off, and it is easier to record and reply messages with sequence numbers if they were reset for each bootstrapping. After all, knowing the order of repeated messages is not necessary, though it helps in deciding which message is the latest sent or expected one. Timestamps provide not only ordinal property but also a precise timing of each event. It brings in another dependence on a synchronized clock between the device and the C. The random numbers require a random number generator in the system. The correctness and resistance to attacks of the random-number-based design rely on the implementation quality of the pseudo-random number generator. Many

platforms come with a built-in pseudo-random number generator, and we consider the state-of-the-art implementations are good enough for domestic cryptographic use cases. However, the lack of ordinal information may require extra state management to ensure correctness in imperfect scenarios, trading performance for robustness.

5.3.8 Encryption-based Bootstrapping Protocol

As a supplement, we now present an encryption-based bootstrapping protocol. The two protocols both achieve the same security goals. When benign devices' behavior cannot be predicted and may misbehave, the encryption-based method is more reliable than the HMAC based method in defending against replay attacks, especially table attacks.

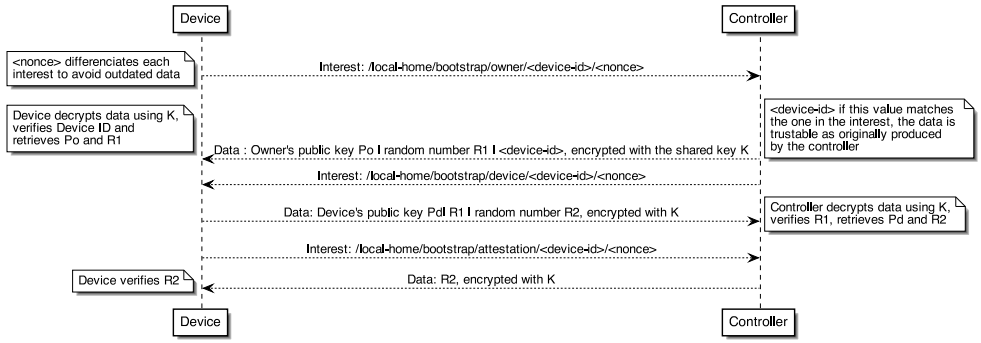

Figure 14. Bootstrapping Protocol Using encryption

A device shares an identifier D and a secret key K with the homeowner via side channels. The bootstrapping process is illustrated in Figure 14. and is described as follows:

1. Homeowner publishes $M_1 = encr(P_o|D|r_1, K)$, where P_o is the owner's public key, r_1 is a random number, and $encr$ is a symmetric encryption function, under the name $/localhome/bootstrap/device/ownerkey/for/D$;

2. Device fetches M_1, decrypts it, verifies D, and obtains r_1. It then obtains P_o and validates the signature in M_1;

3. Device publishes $M_2 = encr(P_d|D|r_1|r_2, K)$, where P_d is the device's public key

and r_2 is a different random number, under the name $/localhome/bootstrap/device/D/pubkey$;

4. Homeowner fetches and decrypts M_2, then verifies D and r_1, obtains P_d and r_2, and validates the signature in M_2;

5. Homeowner publishes $M_3 = encr(r_2|nonce, K)$, where $nonce$ is random number, under the name $/localhome/boostrap/device/auth/for/D$;

6. Device fetches M_3, decrypts it, and verifies r_2.

5.4 Evaluation of Trust Bootstrap Protocol

In this section, we evaluate our trust bootstrap protocol regarding performance and security. Based on the cryptographic tools and parameters used in one scheme, we estimate the runtime computation overheads introduced by the two design proposal of this protocol compared to the method proposed in ndnTR0035[14]. Then we give an informal argument on how the protocol is secure from potential attacks.

5.4.1 Performance Impacts

The major difference between the trust bootstrapping schemes is the cryptographic tools used to prove the authenticity of the other party. The two schemes introduced in this chapter use HMAC [20] and AES [21], and are named as *homesec-hmac* and *homesec-aes* respectively. The ndnTR0035 uses HMAC. To measure the computation overheads, we use the parameters listed in table 2. and simulate for 10000 runs for each scheme. Then we record the total time cost for each scheme and compute the average computation overhead in milliseconds.

The results are listed in table 3. Unsurprisingly, the HMAC-based approaches are more computationally efficient than the AES-based ones. The bootstrapping process doesn't frequently happen in daily use, and any less-than-50-milliseconds overhead per bootstrapping will not cause a performance problem to the smart home system.

Scheme	Parameter	Value
ndnTR0035-hmac	cipher	sha256
ndnTR0035-hmac	keybits	128
homesec-hmac	cipher	sha256
homesec-hmac	keybits	128
homesec-aes-128	cipher	aes
homesec-aes-128	keybits	128
homesec-aes-128	cipher	aes
homesec-aes-256	keybits	256
all	cryptolib	openssl
all	CPU Speed	2.20 GHz
all	Name & Data Size	$\leq$ 500 bytes

Table 2. Trust Bootstrap Evaluation Parameters

Scheme	Msg Rounds	Runtime Overhead
ndnTR0035	1	9.38ms
homesec-aes-128	3	37.46ms
homesec-aes-256	3	39.32ms
homesec-hmac	2	16.36ms

Table 3. Trust Bootstrap Evaluation Results

5.4.2 Security Arguments

We consider two kinds of attacks to evaluate the effectiveness of the protocol to defend against fraudulent bootstrapping attacks.

First, an attacker Trudy tries to pretend to be an authentic device and onboard the home system (Figure 15.). She uses recorded bootstrapping messages to do so. For the homesec-hmac scheme, the second interest from the Owner's controller includes a random number as a challenge. Only the authentic device can encrypt the random number correctly. The randomness ensures the attacker can hardly record a correct message for the same number. Thus the attacker cannot achieve success in her attempt. A similarly effective protection mechanism can be found in the homesec-aes scheme. The ndnTR0035 uses a timestamp embedded in the Interest name from the device as an implicit challenge to the device. The implicit assumptions for it to work is 1) an authentic device has a synchronized clock with the controller, and 2) the communication cost for the interest to reach the controller is ignorable. In real life scenarios, these two assumptions cannot always hold. On the one hand, for a low-cost smart home device, it's hard to ensure the clock is synchronized securely before it joins the home system. On the other hand, if the device has a slow network interface card or suffering a slow connection, the timestamp it sends out may always miss the permitted threshold of the controller and can hardly bootstrap successfully. The result is high false positives. The ndnTR0035 scheme only

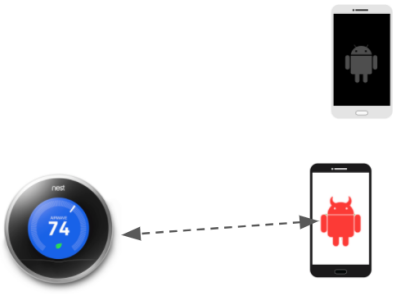

Figure 15. Fake Owner Attack

Figure 16. Fake Device Attack

works under optimistic scenarios. But the homesec schemes are more reliable under different situations for such an attack.

Second, an attacker Trudy tries to pretend to be a controller to trick the device into believing it joined an authentic home system (Figure 16.). In homesec schemes, Trudy cannot answer correctly the random challenge posed by the device in the first round interest exchange. Thus her attempt will not achieve success. For ndnTR0035's scheme, the timestamp in the Interest name can serve as a challenge and Trudy cannot encrypt or sign the message including the same timestamp without the correct shared secret between the device and the owner. In this sense, it is also effective in protecting the device from trusting a fake home network. However, as we previously mentioned, the device must have a globally synchronized clock that never resets across power on and offs to ensure an attacker cannot record already used timestamps and its corresponding replies.

5.4.3 Evaluation Summary

As a summary, all the three schemes can defend against replay attacks effectively. The homesec schemes are slower than the ndnTR0035 scheme when it comes to both message overheads or computation overheads acceptably. And they are more reliable in protecting the owner's home network and device with fewer assumptions and restrictions.

Chapter 6

An Application and Evaluation of the Name-based Access Control

6.1 Introduction

Fine-grained access control to smart home data helps to defend against privacy leaks and user behavior surveillance which would furtherly lead to many severe damages such as physical intrusion such as burgery [7]. Name-based Access Control is a content-based access control model [22] which do the work by requiring encrypting the data upon production and decrypting the data upon consumption. In this section, our goals are 1) use Name-based Access Control (NAC) in a smart home application to provide access control to data, and 2) evaluate the usability and effectiveness of the NAC. In the rest of this section, we will explain the intuition of the NAC design, identify the difficulties and potential problems needs to be solved to apply the NAC implementation in a real smart home application, present our design and implementation of the home app, and show the evaluation metrics and results for NAC.

6.1.1 Intuition of Name-based Access Control

The NAC design models the data production and consumption in three parts: the producer, the owner, and the consumer. The producer produces data, and the consumer consumes the data, all under the permission of the owner. The owner controls the production and consumption permissions by distributing production and consumption credentials. The production and consumption credentials are pairs of public and private keys. To give the permit to a data production process, the owner must grant the producer the production credential. A data producer must generate a content key (C-Key) before producing a piece of data, and use the key to encrypt the content. It then encrypts the content key using the production credential. To give access to a piece of data, the owner distributes the consumption credential to the consumer. The consumer uses the credential to decrypt the content key and uses the content key to decrypt the data before consuming the content. Given the actual purpose of the credentials, we call the production credential

the key encryption key (E-Key) and the consumption one the key decryption key (D-Key). C-Key, D-Key, and E-Keys are generated over a configurable time interval, and the problem of securely naming and distributing all the keys are solved by defining a regular naming scheme and applying schematized trusting over NDN. Figure 17. shows a high-level view of the relations between each party and keys.

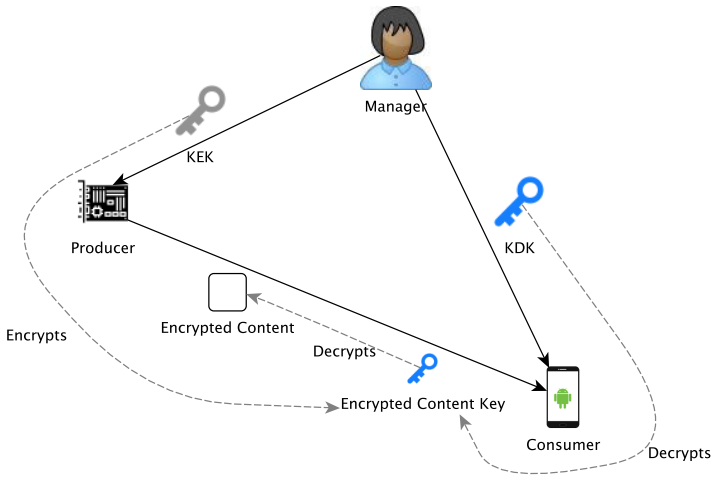

Figure 17. Name-based Access Control

6.2 Applying NAC to smart home applications

The Name-based Access Control (NAC) design provides a fine-grained scheme to control the access to data over time. On the one hand, it provides a powerful and expressive API. One can define access schedules for datatypes, and enforce identities to produce or consume data packets at a schedule. But on the other hand, the implementation of fine-grained control exposes too many parameters and implementation details to an application developer, whose major concern is developing functionality instead of digging into details of how the NAC works under the hood. Additionally, the concept differences between the NAC API and the NAC design, the lack of the implementation of a higher level access control manager, the complexity of those API, the inconsistency between API document and the actual function, the lack of proper exception handling, the under-maintained status with respect to the NDN primitives implementation (ndn-cxx),

the implicitly defined API behaviors and uncertain outcomes of the same API calls all dramatically increases the difficulty to directly apply NAC in an application. To overcome these difficulties, we analysed the code of NAC implementation (the name-based-access-control project), suggested documentation and code changes [23] [24] [25], implemented adapters [26] to properly catch and handle exceptions, and designed and implemented a higher level access control manager to simplify APIs exposed for applications and to achieve more explicit and deterministic outputs of event-driven callbacks.

We identify the problem of applying NAC design as devising time-based schedules for accesses to data under the same namespace which represents a particular data type and solve this problem in four steps.

First, NAC defines a set of basic naming rules, and we must incorporate them when designing the namespace for shared data. Figuire 18. lists NAC naming rules for the three genres of data. In the figure, "`<prefix>`" refers to the scoped namespace for the smart home application. In our application the prefix is "`local-home`". "`<data-type>`" refers to the sub-namespace of the data that needs to be protected using NAC. For example, if the data is about the temperature of the owner's bedroom, its value is "`bedroom/temperature`". "`<content-key-identifier>`" is a string that is used to identify the key used to encrypt the content. "`<start timestamp>`" and "`<end timestamp>`" are the start and end of intervals to locate the E-Key and D-Key for data production and consumption. Figure 19. shows an example of a valid namespace hierarchy for smart home data.

Second, apply schematized trusting [12] to enforce the same trust model throughout the smart home system. In our smart home system, the homeowner is the trust anchor. The owner publishes a self-signed NDN certificate for itself under namespace "`/local-home`". All other nodes in the smart home retrieve this certificate through the bootstrapping process and accept it as the root of trust. As introduced in previous sections,

Protected Data Name Format

/<prefix> /sample/<data-type> /<content-key-identifier>

Production Credential Name Format

/<prefix> /read /<data-type> /E-KEY /<start timestamp> /<end timestamp>

Consumption Credential Name Format

/<prefix> /read /<data-type> /D-KEY /<start timestamp> /<end timestamp>

Figure 18. NAC Naming Rules

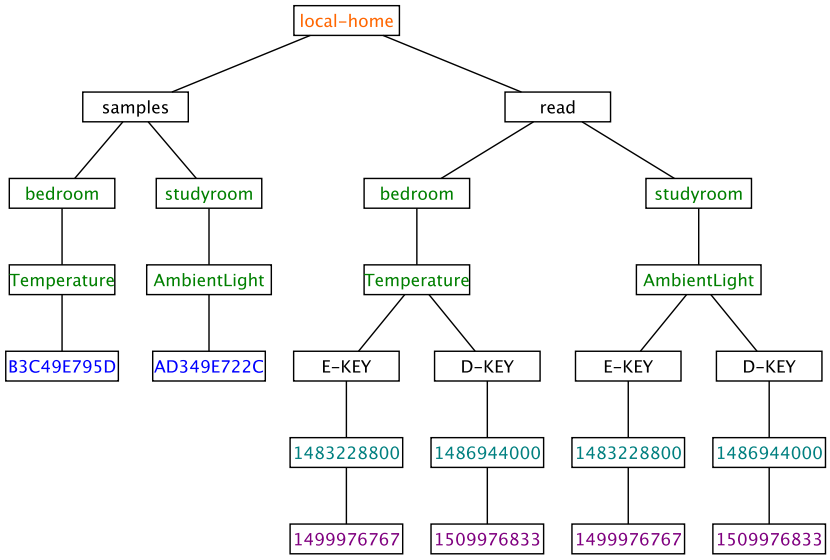

Figure 19. Smart Home Data Namespace Example

it is necessary for all other data producers in the smart home system to have a data-signing key signed by the owner's private key. A data producer signs its data using its key. Figure 20. shows an example snippet of the signing hierarchy that enforces the trust model.

Third, design and implement a standard code base for similar application development. For example, implement the adapters as mentioned earlier and integrated access manager to smoothe the developing practice, and implementing base classes and

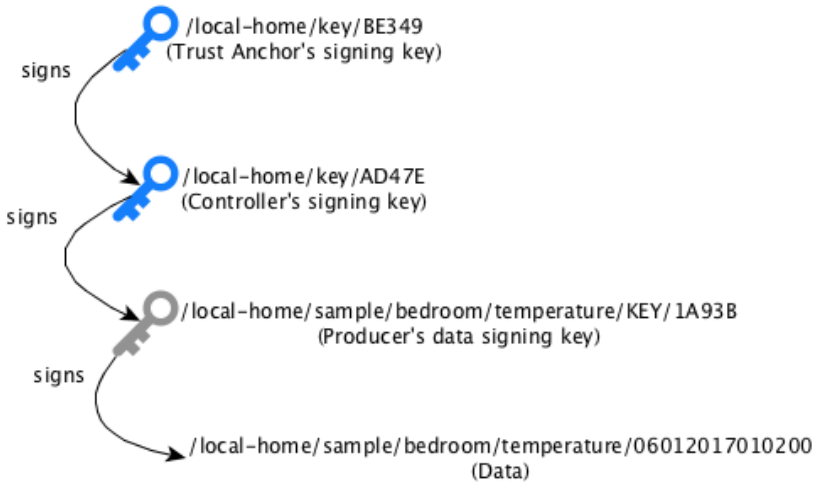

Figure 20. Signing Hierarchy for the Trust Model

code templates to enforce the same trust model and serialization formats. Figure 21. shows the design of the code base regarding class diagrams.

Forth, build applications upon the established code base. For demo NAC, our mini smart home system has four parts: the owner's app on an Android phone, a thermometer mounted on a Raspberry Pi, a NAC manager gateway on a laptop, and a temperature reporter on the owner's phone. Figure 22. shows the overall relations between each parts.

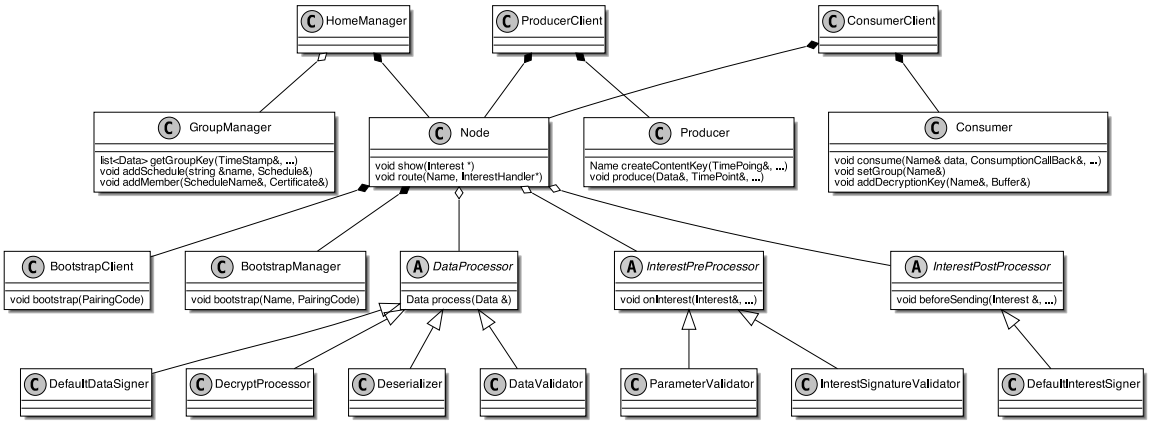

Figure 21. Class Diagram For the Application Library

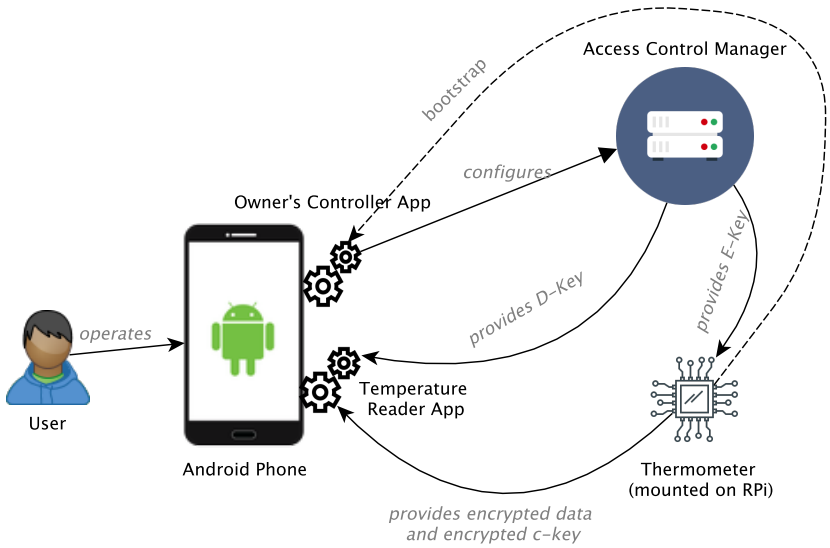

Figure 22. Mini Smart Home System

6.3 Implementing Smart Home Data Access Control using NAC

In this section, we present how we implement the smart home system across x64 and ARMv8 platforms by developing cross-platform libraries and dedicated apps for macOS, Linux, and Android systems. Our implementation consists of two parts: the common smart home node API library (Node API)and applications. The Node API implements functions described in Section 6.2 and encapsulate reusable code templates. The applications include owner's controller and data consumer apps on Android, NAC manager daemon on macOS, and data producers on Raspberry Pi Ubuntu system. The macOS and Linux implementations relies on *ndn-cxx* [27] and *lib-name-based-access-controll* [28]. The former implements the NDN primitives using C++, and the latter provides the NAC API. On Android, we use *jndn* [29], which is one NDN Common Client Library [30] for Java. Figure 23. shows the hardware settings of Raspberry Pi and a connected digital thermometer (MCP9808). Figure 24. to Figure 26. shows the owner's Android phone, required and implemented apps.

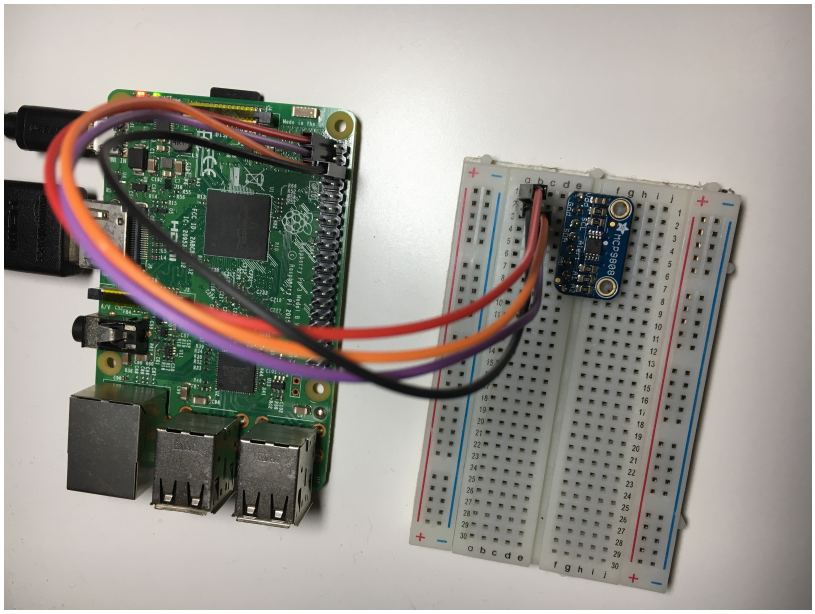

Figure 23. Mini Smart Home Smart Sensor

6.4 Evaluating NAC

The evaluation of NAC has two parts. First, we estimate the computation and message overheads in the NAC implementation. The NAC implementation's code uses AES128 for content encryption in a hardcoded way. But we also consider the case of using AES256 as this currently recommended way to perform data encryption on the Internet of things devices[31]. Second, we analyze the usage scenarios that the NAC design can cover.

6.4.1 NAC Performance Overheads

By examining the NAC code and observing Interest and Data packet logs via ndndump [32], we count the additional messages sent and received while performing one data packet production and one consumption comparing to directly sending an Interest and receiving a Data packet. We assume the Interest and Data packets are sent for the first time so that a middle-way NFD couldn't have cached it and there is no packet resending occurs in the process. The measurements are done in three parts. First, the group manager doesn't actively send out packets. But it publishes 2 data packets as production and consumption credentials and sends them upon Interests is received. The data producer should request

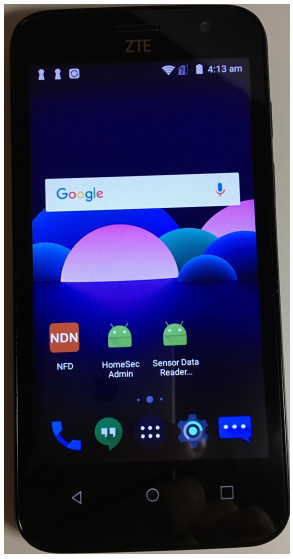

Figure 24. Owner's Phone

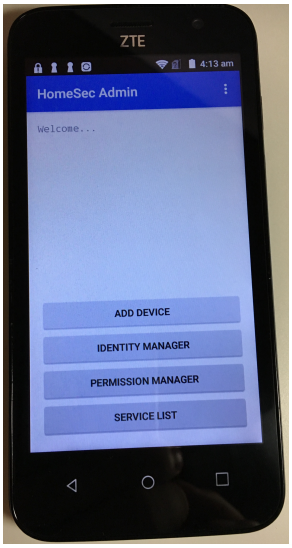

Figure 25. Owner's App

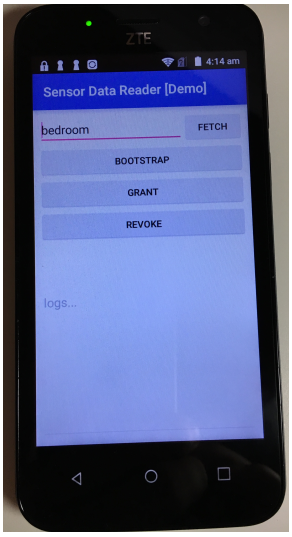

Figure 26. Consumer

production credentials with an Interest. But in the implementation, if the data's name has N components, there will be $N-1$ interests sent out for requesting the production credential though only one Data packet is expected to return. This is because the producer does not know the correct name for the credential and thus it must guess by iterating all possible names. Then it publishes M encrypted content key packets for M schedules that has access to the content. The consumer sends one Interest for the consumption credential and one Interest for the encrypted content key. The total number of packets in transmitting is 8. The total number of message rounds is 3. Table 4. lists all the count side by side.

For computation overheads, we measure the runtime overheads introduced by crypto functions. A combination of parameters for each crypto method is a profile. And for each profile, the estimation is run 1000 times and take the average time cost in

Role	Packets	Message Rounds
Manager	2	1
Producr	4	1
Consumer	2	1

Table 4. NAC Packet and Message Overheads

milliseconds as a result. The profile parameters are list in Table 5.. The result is presented in Figure 27.. As we can see, though the NAC implementation chooses AES 128 over AES 256, the latter is not introducing more performance impacts than the former. Overall, the performance impact excluding the communication costs is under 100ms per process for all profiles.

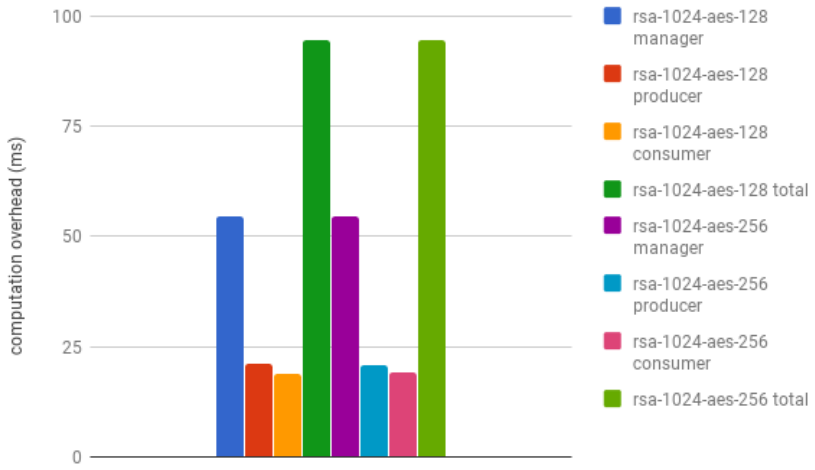

Figure 27. NAC Computation Overheads - Chart

6.4.2 NAC Security Analysis

In this part, we evaluate how well NAC works for different hypothetical scenarios.

In our mini smart home system, the owner controls the permission for the thermometer to produce temperature data. To gain permission to produce, the thermostat must retrieve a certificate from the owner for temperature data's namespace and fetch production credentials. If the owner didn't issue the certificate for the thermometer, then thermometer's data won't be trusted by other nodes. If the owner didn't publish the production credential, then the thermostat won't be able to encrypt the content key, and no one will be able to read the temperature except for itself. To gain access to read temperature, the app must retrieve the correct consumption credential from the owner. Otherwise, the app cannot decrypt the content key, and cannot decrypt the encrypted

Role	Step	crypto method	parameter
Manager	credentials key-gen	rsa	1024
Manager	credentials encryption	rsa	1024
Producer	C-Key key-gen	urandom	128
Producer	C-Key key-gen	urandom	256
Producer	Content Encryption	AES	128
Producer	Content Encryption	AES	256
Producer	C-Key Encryption	RSA	1024
Producer	E-Key Decryption	RSA	1024
Consumer	D-Key Decryption	RSA	1024
Consumer	C-Key Decryption	AES	128
Consumer	C-Key Decryption	AES	256

Table 5. NAC Parameters

Profile	Role	Cmpt. Overhead
rsa-1024-aes-128	manager	54.512419
rsa-1024-aes-128	producer	21.088183
rsa-1024-aes-128	consumer	18.893829
	total	94.494431
rsa-1024-aes-256	manager	54.512419
rsa-1024-aes-256	producer	20.869772
rsa-1024-aes-256	consumer	19.08343
	total	94.465621

Table 6. NAC Computation Overheads - Data

temperature reading data. Thus without getting permission, an app cannot finish the production or consumption of a piece of data by enforcing NAC.

When the owner revokes the permission for a producer to publish data, it stops offering the producer the production credentials. The producer won't be able to continue production when its current production credential expires. When the owner revokes the permission for a consumer to read data, it stops offering the consumer the consumption credentials. This works for future data when the current consumption credentials expire. However, this won't prevent the consumer from decrypting and reading the previous data before the revocation because the consumer may still have access to keys before the revocation. The NAC authors suggested one way to do the renovation work by requiring the app to encapsulate the consumption process in a black box, and all the keys are forgotten upon usage. However, this solution is not strong enough to protect the user's data. To entirely revoke the read access from the app, the app must not remember any previous keys, the network must not have caches of earlier keys, and the owner stops publishing then app's consumption credentials.Even so, if a device is compromised, it won't behave as good as defined by the protocol and there is no way to restrict it. The same issue exists for the producer. A producer app is compromised and is publishing wrong data. The owner revoked its write permission by stopping publishing certificate and the production credential. The compromised producer may use the previous keys to produce and sign the data and cheat the consumer to use an erroneous or adversary history data.

Chapter 7

Conclusion

In this work, we studied current smart home technologies and identified security risks in producing and accessing smart home data. We designed and implemented protocols in the context of NDN to protect the home owner's data by securing the bootstrapping of new devices and enforcing fine-grained access control. We include a mutual authentication process in the bootstrapping protocol to eliminate the strong assumptions made in previous work and enhance the defense against replay attacks. We developed an access control manager with a friendly API to enforce name-based fine-granularity access control to smart home data. Our evaluation shows that NAC does not introduce significant computation overhead and performs well in granting and preventing accesses. Last but not the least, we provide a Smart Home Common Node API incorporating the trust bootstrapping and access control protocols for building future smart home applications over NDN across macOS, Linux and Android platforms.

REFERENCES

[1] W. Shang, Y. Yu, R. Droms, and L. Zhang, "Challenges in iot networking via tcp/ip architecture," NDN Project, Tech. Rep. NDN-0038, Tech. Rep., 2016.

[2] "Weave — nest developers," https://developers.nest.com/weave/, (Accessed on 10/31/2017).

[3] "Zigbee home automation — zigbee alliance," http://www.zigbee.org/zigbee-for-developers/applicationstandards/zigbeehomeautomation/, (Accessed on 11/24/2017).

[4] "Smart home - smartthings — samsung us," https://www.samsung.com/us/smart-home/smartthings/, (Accessed on 10/31/2017).

[5] "About," https://threadgroup.org/About, (Accessed on 11/14/2017).

[6] T. Zillner, "Zigbee exploited: The good the bad and the ugly."

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

[8] G. Montenegro, C. Schumacher, and N. Kushalnagar, "IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals," RFC 4919, Aug. 2007. [Online]. Available: https://rfc-editor.org/rfc/rfc4919.txt

[9] "draft-cragie-tls-ecjpake-01 - elliptic curve j-pake cipher suites for transport layer security (tls)," https://tools.ietf.org/html/draft-cragie-tls-ecjpake-01, (Accessed on 11/24/2017).

[10] "draft-ietf-core-object-security-06 - object security for constrained restful environments (oscore)," https://datatracker.ietf.org/doc/draft-ietf-core-object-security/, (Accessed on 11/14/2017).

[11] "Irtf information-centric networking research group (icnrg)," https://irtf.org/icnrg, (Accessed on 11/02/2017).

[12] Y. Yu, A. Afanasyev, D. Clark, V. Jacobson, L. Zhang, *et al.*, "Schematizing trust in named data networking," in *Proceedings of the 2nd International Conference on Information-Centric Networking*. ACM, 2015, pp. 177–186.

[13] W. Shang, A. Bannis, T. Liang, Z. Wang, Y. Yu, A. Afanasyev, J. Thompson, J. Burke, B. Zhang, and L. Zhang, "Named data networking of things," in *Internet-of-Things Design and Implementation (IoTDI), 2016 IEEE First International Conference on*. IEEE, 2016, pp. 117–128.

[14] A. Bannis and J. A. Burke, "Creating a secure, integrated home network of things with named data networking," 2015.

[15] W. Shang, Q. Ding, A. Marianantoni, J. Burke, and L. Zhang, "Securing building management systems using named data networking," *IEEE Network*, vol. 28, no. 3, pp. 50–56, 2014.

[16] W. Shang, Y. Yu, T. Liang, B. Zhang, and L. Zhang, "Ndn-ace: Access control for constrained environments over named data networking," NDN Project, Tech. Rep. NDN-0036, Revision 1, Tech. Rep., 2015.

[17] 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 *Internet-of-Things Design and Implementation (IoTDI), 2017 IEEE/ACM Second International Conference on*. IEEE, 2017, pp. 3–14.

[18] H. Zhang, Z. Wang, C. Scherb, C. Marxer, J. Burke, L. Zhang, and C. F. Tschudin, "Sharing mhealth data via named data networking," in *ICN*, 2016, pp. 142–147.

[19] P. Rogaway, M. Bellare, and J. Black, "Ocb: A block-cipher mode of operation for efficient authenticated encryption," *ACM Transactions on Information and System Security (TISSEC)*, vol. 6, no. 3, pp. 365–403, 2003.

[20] H. Krawczyk, M. Bellare, and R. Canetti, "Hmac: Keyed-hashing for message authentication," RFC Editor, United States, Tech. Rep., 1997.

[21] J. Daemen and V. Rijmen, *The design of Rijndael: AES-the advanced encryption standard*. Springer Science & Business Media, 2013.

[22] Y. Yu, A. Afanasyev, and L. Zhang, "Name-based access control," *Relatório Técnico TR NDN-0034, University of California, Los Angeles, Los Angeles*, 2015.

[23] "Task #4250: fit coding format with ndn-cxx-code-style - nac - ndn project issue tracking system," https://redmine.named-data.net/issues/4250, (Accessed on 11/05/2017).

[24] "Task #4243: Replace link with forwardinghint - nac - ndn project issue tracking system," https://redmine.named-data.net/issues/4243, (Accessed on 11/05/2017).

[25] "Task #4242: Support security::v2 changes - nac - ndn project issue tracking system," https://redmine.named-data.net/issues/4242, (Accessed on 11/05/2017).

[26] E. Gamma, J. Vlissides, R. Johnson, and R. Helm, *Design patterns: elements of reusable object-oriented software*. Addison-Wesley, 1994.

[27] "named-data/ndn-cxx: ndn-cxx: Ndn c++ library with experimental extensions," https://github.com/named-data/ndn-cxx, (Accessed on 11/07/2017).

[28] "named-data/name-bases-access-control: Ndn named-based access control," https://github.com/named-data/name-bases-access-control, (Accessed on 11/07/2017).

[29] "named-data/jndn: Ndn client library for java," https://github.com/named-data/jndn, (Accessed on 11/07/2017).

[30] "Ndn common client libraries api ndn common client libraries api 0.5.1 documentation," http://named-data.net/doc/ndn-ccl-api/, (Accessed on 11/07/2017).

[31] "Iot security guidance - owasp," https://www.owasp.org/index.php/IoT_Security_Guidance, (Accessed on 11/10/2017).

[32] "ndn-tools/ndndump.rst at master named-data/ndn-tools," https://github.com/named-data/ndn-tools/blob/master/manpages/ndndump.rst, (Accessed on 11/10/2017).