Secure Sign-on Protocol for Smart Homes with Named Data Networking

***Abstract*—We introduce our design of a secure sign-on protocol for smart homes using Named Data Networking (NDN). In NDN, every home has a unique name and a digital certificate to serve as the local trust anchor. In order to sign on to such a home, a device needs to acquire two NDN certificates to secure its communications thereafter: the local trust anchor with which the device can cryptographically distinguish others in this home, and a certificate signed with the trust anchor to certify the device's identity and ensure its authenticity. On the basis of NDN's security framework and its Interest/Data exchange semantic, our protocol automates and secures the process for a device to obtain those two certificates. A piece of pre-shared information is required to establish the initial trust, of which the confidentiality enables the sign-on process to succeed. The protocol is designed to work with constrained devices and our real-world tests shows the process can be executed within seconds. Notably, the protocol offers strong protection to the smart home scenarios even if pre-shared information is leaked later.**

***Index Terms*—Named Data Networking, Sign On, IoT, Security**

I. Introduction

The smart home, a typical application scenario of the Internet of Things (IoT), can improve the quality of our daily lives by connecting intelligent electrical appliances and electronic equipment to form a network and function synergistically. At the same time, a home network often poses higher demands on security and privacy. A fundamental challenge for a new device is to securely sign on to a system and securely communicate with other devices there.

Most of today's smart home ecosystems, such as AWS Greengrass[1] and Apple HomeKit[2], either heavily rely on clouds or other third-party authenticating centers, or require extensive human intervention for device sign-on. However, the need for interaction between local devices and remote clouds not only brings extra latency and unnecessary dependency on external connectivity, but also opens a venue to attacks from adversaries. In addition, cumbersome human operations reduce system efficiency and degrade the user experience.

The work in [1] discusses the opportunities and potential of applying Named Data Networking (NDN)(i.e., a proposed future Internet architecture) to the Internet of Things (IoT), and demonstrates its power in enabling local trust management and rendezvous while leaving the access of cloud optional [2], [3]. In this paper, we design a cloud-independent device sign-on protocol that can enable a secure and automatic sign-on process with minimal human intervention in NDN-enabled smart homes. Besides formally verifying its security via a cryptographic protocol analyzer, we also evaluate the performance of the proposed protocol by implementing it on real IoT devices. Our results indicate that each of the devices can complete the sign-on process with only the need for local processing.

The rest of this paper is organized as follows. Section II provides a brief description of NDN and its security framework. Section III shows our design goals. In Section IV, we introduce the basic sign-on protocol for constrained devices and analyze its security. Section V discusses our protocol extensions from three perspectives. We evaluate the performance with experiments in section VI and conclude the work in Section VII.

II. Named Data Networking

Named Data Networking (NDN) focuses on data themselves rather than where they are located. In NDN, the request-and-response semantic is performed at the granularity of a network packet. Each request is carried in an NDN *Interest* packet along with the name of the requested data to fetch one NDN *Data* packet. An Interest packet is forwarded by its name [1] while the Data packet is forwarded along reverse path of the corresponding Interest packet; neither Interest nor Data packets carry any addresses.

A. Overview of NDN Security Mechanism

NDN builds public-key cryptography into the architecture and requires every Data packet be cryptographically signed [3] and encrypted by its producer if required; we refer readers to our previous paper [4] for more details about NDN security. Here we only review some basic terminologies and semantics for understanding our sign-on protocol. In NDN, any entity possesses one or more names and correspondingly one or more cryptographic public-private key pairs. An NDN *certificate* certifies an entity's ownership of a name and its key(s) by binding the name and key(s) together with a cryptographic signature generated by the certificate authority. Such a certified name is called an *identity*. Notably, an NDN certificate is indeed a Data packet that carries public key information and thus can be retrieved by regular Interests.

The signature of an NDN packet is associated with a "SignatureInfo" field, which records the name of the signing key of this packet. A *certificate chain* can be formed by recursively tracking the signing key of a packet and the signing key of its singing key's corresponding certificate until reaching a root certificate. A *trust anchor* is the root certificate that is trusted within a given namespace. After a trust anchor is installed, an entity can verify other entities' signatures by verifying their certificates along the certificate chain to the trust anchor.

[1] https://aws.amazon.com/greengrass/

[2] https://developer.apple.com/homekit/

[3] An Interest packet can optionally carry a signature as well.

Since a cryptographic key is carried in and retrieved as an NDN certificate, we use the term *signed with a certificate* as a shortened form of "signed with the private key corresponding to the public key carried in that certificate". Finally, on basis of the structured naming convention, *trust polices* are defined to determine whether an identity is trustworthy and what its privileges are.

With its powerful security mechanism, NDN secures network communications at packet granularity. By verifying the digital signature and checking one's trust policies, the receiver of a signed packet is able to ensure 1) data integrity, 2) the authenticity of received packet, and 3) whether an identity is trustworthy and is granted certain privileges. The first two aspects require signer's certificate and a trusted trust anchor, while the last one requires proper setup of trust policies. Since trust policies can be retrieved separately afterwards, the sign-on protocol is required to help a device to obtain the trust anchor and its own certificate.

B. Related Work

OnboardICNg [5] proposed a protocol for on-boarding a new device in a wireless network over ICN. It requires a Authentication and Authorization Manager (AAM) as a remote trust center, and replies on a pre-shared symmetric key to setup initial trust. Industrial solutions like AWS Greengrass and Apple HomeKit are also utilizing their cloud service to support the security setup in local network systems. Different from the existing practice, our proposed sign-on protocol is independent from the remote services like cloud servers, because NDN enables localized trust anchor by letting each individual system has their own namespace and trust system. Besides, we argue that it's dangerous to rely on a pre-shared symmetric key, and fullfil the potential security hole caused by the exposure of pre-shared information with a series of approaches.

III. System Model and Design Goals

We abstract a smart home as a heterogeneous network composed of resource constrained devices, such as sensors and actuators, and resource unconstrained devices, such as computers and phones. Built on an NDN framework, such a home possesses a global unique name that defines a namespace and a root certificate defined as the trust anchor of the home. Every device belonging to this home is assigned a name in its namespace, and is issued a certificate directly or indirectly signed with the trust anchor. A device with enough computation and storage capabilities can be made a *controller* of the home, which is empowered to sign other devices' certificates with the trust anchor (i.e., the root certificate). The owner of the home, or anyone granted the privilege, can manage the whole system through the controller.

Fig. 1 shows a simple example of an NDN-enabled smart home, where a mobile phone serves as the controller and installs the signing key corresponding to the trust anchor. Every device, including the phone itself, has a certificate directly signed with the trust anchor and can authenticate one another. Four entities are involved in a sign-on process: a home network, a controller, the human who operates this controller, and a device trying to sign on. They are referred to as the *system*, the *controller*, the *operator* and the *device* respectively.

The sign-on protocol is proposed to assist a device to acquire a copy of the trust anchor as well as its certificate signed with the anchor. Device capability is an important factor to consider in our design of the sign-on protocol. We consider three capabilities: the support for human interaction, the availability of permanent storage, and the ability to generate key-pairs. The controller must have all of them. An overview of the sign-on protocol for relatively constrained devices possessing none of aforementioned capabilities is shown in Fig. 1, and we will introduce the detailed steps of the protocol in Section IV. In Section V, we will demonstrate how these capabilities are utilized to simplify the sign-on process.

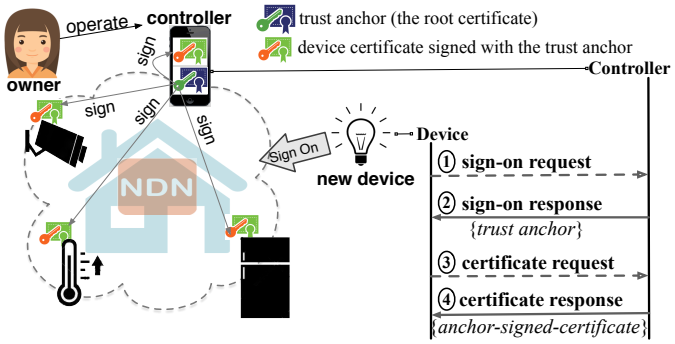

Fig. 1. An NDN based smart home and a new device trying to sign on.

A. Adversary Model

Suppose there exists a powerful adversary that is able to sniff and store all packets transmitted in the network and has sufficient computing power and resources, we consider the following attacks against the sign-on protocol:

1) *fraudulently sign on to the system*, whereafter the adversary may be able to break the whole system.
2) *impersonate the controller*, whereafter the device signs on with this "controller" may function unexpectedly.
3) *flood fake or completed requests*, whereby legitimate requests may be impeded due to resource limitation.
4) *replay completed responses*, whereby the device may be fooled into installing "out-dated" keying materials.

Regarding the *man-in-the-middle* attack, an adversary in the middle can only create one additional damage other than the aforementioned four. He may be able to isolate either the controller or the controller (or even both of them) by blocking the packets from or for the victim(s). Once the controller is isolated, its operator will immediately recognize this exception and fix it with human intervention. While the isolation of a new device would never affect the system although it is prevented from signing on to the system.

B. Basic Assumptions

We make following assumptions as the minimal requirements to ensure that a device can sign on to the system.

1) not compromised participants: As a fundmental requirement, we assume neither the controller nor the device trying to sign on is compromised. In another word, any confidential data on them is uncompromised.

2) broadcast reachability: We assume that an NDN-layer broadcast from the device can always reach the controller directly or indirectly with the help of its neighbours to forward messages. In another word, we assume the adversary is unable to isolate neither the controller nor the device.

3) shared secret: Before running the protocol, we assume the device has a piece of confidential information encoded somewhere (such as its QR code or bar code) and has shared it with the controller via out-of-band operations. This information is used to establish the initial trust, so the method of sharing should be secure.

C. Design Goals

Our design goals are twofold. First, once all aforementioned assumptions hold, the sign-on must succeed in a reasonable time frame.This is so that the device can obtain a copy of the trust anchor and have its certificate signed with the trust anchor. Second, the sign-on protocol should never put the system in danger in the case that the pre-shared information gets revealed.

More specifically, if all assumptions hold, the device must be able to obtain required keying materials after the sign-on process is completed. Before that, mutual authentication must be achieved, ensuring no impersonation. Fake or replayed requests must be stopped at the earliest stage possible, and at a reasonably low cost.

What if not all assumptions hold true? First of all, there would be no guarantee of successful sign-on. What's worse, the system might be threatened as well. If the adversary has hacked into either the controller or the device, not only will the hacked device function unexpectedly, but also malicious devices will be able to sign on to the system regardless of how the sign-on protocol is designed. In case that the second assumption does not hold, the only new potential attack would be that the adversary isolates either the controller or the device to break their reachability.

This prevents the device from signing on. However, ,as discussed in section III-A, the system would not be affected too much in this case. Therefore, we always consider that the first two assumptions are met, but focus on the breakage of the last assumption.

To meet the second design goal, the following conditions should be met. First, the protocol must prevent any malicious devices from obtaining an anchor-signed certificate. Meanwhile, the risk for a legitimate device to install a fake trust anchor should be reasonably controlled. More importantly, once a device is deceived into trusting the adversary, there must be a way for the controller to detect this case and to react to it quickly.

IV. Sign-on Protocol for Constrained Devices

In this section, we introduce the basic sign-on protocol, named *sign-on-basic*, for relatively constrained devices without any of the aforementioned capabilities (section III). We give the design overview in Section IV-A, and then we introduce a series of design issues along with our considerations and solutions in sections IV-B~IV-G. We show the complete protocol with message exchange details in section IV-H, and analyze its security in section IV-I.

A. Design Overview

The sign-on protocol should assist the device in obtaining the trust anchor and an anchor-signed certificate. Our design uses two rounds of request-response exchanges, as will be discussed in sections IV-B and IV-C. We name these messages the *sign-on request*, the *sign-on response*, the *certificate request* and the *certificate response* respectively, in the order they are transmitted. The next few sections will detail specific protocol design issues and our solutions.

B. Initiation of Sign-On

Theoretically, either the device or the controller can initiate the sign-on process; we argue that the device should initiate the sign-on process.

First, it is likely that the controller has already been started and is ready to respond by the time the device starts. For example, in a typical scenario of a device signing on to a home system, the user has likely already installed other devices in the past and has a controller already prepared.

Second, a device without permanent storage has to sign on again when it restarts; without human intervention or probing, the controller will not know to reinitiate the sign-on.

C. Mutual Authentication

Mutual authentication is mandatory to prevent both ends from being fooled by an adversary. Usually, a pre-shared symmetric key is used to achieve mutual authentication, as mentioned in [5]. However, the revelation of the key will put both ends in danger.

We propose the use of a set of keys to minimize the threats caused by the revelation of the shared secret. One part of this set of keys is an asymmetric key pair generated for the device, the private key of which is installed during its manufacturing.

The other part of this set of keys is a *sign-on code*, a shared secret from which a symmetric key can be derived. The asymmetric key pair is used to authenticate the device, while the *sign-on code* is used to authenticate the controller. In this way, the authentication of the device can survive the revelation of the shared secret.

To mitigate the damage caused when a fake controller knows the sign-on code, we propose two tactics. First, the device trusts the first "controller" that replies to its sign-on request with proof of knowledge of its sign-on code. A real controller likely has a higher chance than a fake one to send back the reply first, because it is only one hop away and may be physically closer to the device in case that the adversary is outside the home.

However, there is a chance that the real controller's response arrives too late. To allow the legitimate controller to detect that a fake controller has hijacked a device, our protocol requires

that the device insert a digest of the trust anchor received in its certificate request. If the real controller receives a copy of the certificate request broadcasted, it can detect a fake controller by comparing its own trust anchor digest to the one present in the certificate request.

D. Freshness Verification

Freshness verification is important to prevent the controller from performing unnecessary but costly operations, and to prevent the device from installing "out-dated" keying materials obtained from replayed messages. Two fresh challenges are used to stop replay attacks. Each end generates a random number, encodes it into messages to send, and expects it to be in the subsequent message from the other end.

More specifically, a device generates and encodes the first challenge into its sign-on request as an Interest parameter. As it contributes to the last name component (i.e., the digest of parameters) of this request's name, its presence in the sign-on response is automatically verified at the NDN layer via the name match between a Data (the response) and its Interest (the request). The controller generates the second fresh challenge, and encodes it as part of the content of the sign-on response. The device then encodes this challenge as a parameter in its certificate request. In this way, the freshness of the on-going sign-on instance can be verified by both ends.

E. Issuance of Anchor-signed Certificate

There are two issues to deal with in issuing the device its anchor-signed certificate: how to create it, and when to make a new one.

An NDN certificate certifies an entity's ownership of its name and a public key. We make a device's name by appending a unique device identifier to the home network prefix learnt from the trust anchor. The device identifier can be directly included as part of the pre-shared information or derived from the pre-shared key (section IV-H). As per the least privilege rule, a new key-pair is generated for use after sign-on. To avoid burdening the resource constrained devices with key generation, the controller is responsible for generating the key-pair as well as creating the corresponding anchor-signed certificate. In the first-time sign-on, this anchor-signed certificate, along with its encrypted private key, is encapsulated in the certificate response. Thereafter, the controller keeps track of these keying materials and determines whether or not to make new ones according to a renewal protocol[4].

F. Cryptographic Primitives

This section will detail what keys are used in all the cryptographic operations of the sign-on protcool.

Between the two ends, the pre-shared information establishes consensus on two sets of keys: 1) the pre-shared asymmetric key and its corresponding private key, and 2) the symmetric key derived from the sign-on code. The pre-shared asymmetric key pair is used to sign / verify the sign-on request and the certificate request from the device. The responses to these requestsare signed / verified by the symmetric key derived from the sign-on code. Either the asymmetric key pair or the symmetric key can be used to encrypt / decrypt the private key of the key-pair a controller generates for a device, but we argue that neither should be. Asymmetric operations are always costly, especially in constrained devices, and although the use of the derived symmetric key is more efficient, this approach opens an attack vector to crack the device's private key if the sign-on code is compromised.

To address the above issues, we propose the use of a symmetric key dynamically negotiated between the two ends to achieve both desirable security and good performance. Following the Diffie-Hellman key exchange semantic, the two fresh challenges mentioned earlier are generated to carry enough randomness and exchanged between the two ends to establish a dynamic shared secret, from which a *temporary symmetric key* is derived. This key is used to encrypt / decrypt the device's private key. As long as the sign-on code is not compromised, this key also proves the presence of the real controller and thus can be used to sign the certificate response. This minimizes the use of the key derived from the static sign-on code, thus lowering the chances of it being cracked.

G. Prevent Denial-of-service Attacks

We use two approaches to mitigate denial-of-service attacks.

First, costly operations (such as key generation and signing) on both ends are postponed, if feasible, until the other end has been authenticated and the freshness of the on-going communication is confirmed.

Second, only one sign-on instance is maintained at the controller for any device Once such an instance is created for a device, no later validated sign-on request from this device can lead to a new instance. This instance is destroyed once a certificate request of the same device is validated and processed. Additionally, a timer is set at the creation of a sign-on instance to prevent it from existing for too long.

H. Message Exchange Details

The complete *sign-on-basic* protocol with message exchange details is shown in fig. 2. To simplify the discussion, we use a set of abbreviations. N_1 and N_2 denote the two fresh challenges generated at the device and the controller respectively. K_s and K_d denote two key-pairs, the one corresponding to the pre-shared asymmetric key and the one generated for device to use after sign-on. K_c and K_t represent symmetric keys: the one derived from the sign-on code, and the one negotiated based on a Diffie-Hellman key-exchange. For any key-pair K_x, the public and private keys are referred to as K_x^+ and K_x^- respectively.

1) pre-shared information: Via some out-of-band operation, a piece of information is shared between the two ends before the sign-on process starts to establish consensus on an asymmetric key-pair K_s, a symmetric key K_c, and the device identifier. Our implementation targets a 128-bit security level [6], where K_s is on a 256-bit elliptic curve, K_c[5] is of 128

[4]it is beyond the scope of sign-on and thus is not discussed in this paper.

[5]It is derived from a 128-bit sign-code by PBKDF2.

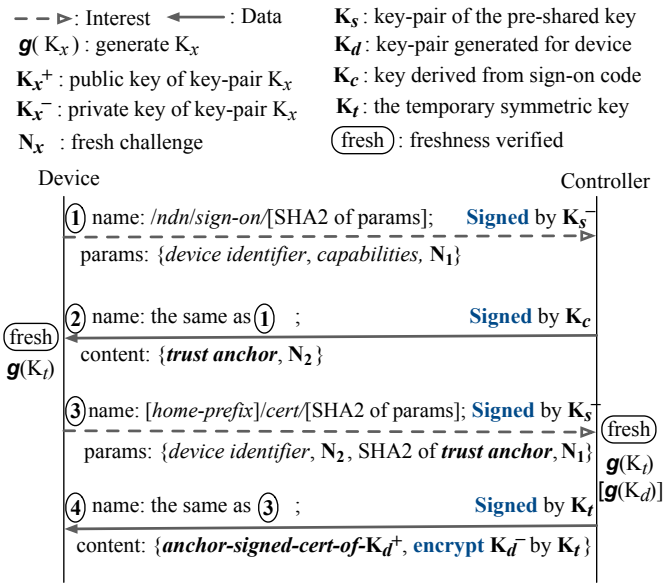

Fig. 2. message exchange details of the *sign-on-basic* protocol.

bits in length, and the device identifier is a 96-bit electronic product code [7]. Following an encoding header describing the type and the length of K_s, the pre-shared information (K_d^+, the sign-on code, as well as the device identifier) form a 512-bit data block, which is encoded as a QR code.

2) the sign-on request: The device initiates the sign-on process with a sign-on request, using an NDN Interest packet named under the "`/ndn/sign-on`" prefix and signed by K_s^-. Three parameters are encoded into this interest: the device identifier, the device capabilities, and a fresh challenge N_1. The digest of all the parameters as a whole forms the last name component. The *capabilities* parameter is a one byte bitmap, which will be introduced in more details in Section V. To achieve the targeted security level, we derive a 128-bit symmetric key, i.e., K_t, from the shared secret established by performing ECDH over a 256-bit elliptic curve, where N_1 and N_2 are public keys.

After receiving a sign-on request, the controller verifies its signature using the K_s^+ corresponding to the device identifier. On a successful verification, a sign-on instance is created if there is no existing one. This new instance is associated with a fresh challenge N_2. In addition, a timer (e.g. 3 seconds) is set and the instance is removed upon its expiration.

3) the sign-on response: Upon validating a sign-on request, a response is made with the same name, whose content encapsulates the trust anchor and the fresh challenge N_2. This response is signed using K_c.

Upon the arrival of the response, the device verifies the controller's knowledge of N_1 to ensure freshness of message, and checks if the sign-on response matches the pending sign-on request. Further, if the response's signature is verified by K_c, the device will trust the controller. At this point of time, it is safe to calculate K_t based on N_2 and the private key paired with N_1. The device also installs the trust anchor, and learns the home prefix from it.

4) the certificate request: With the sign-on response validated, the device sends a certificate request to acquire thean anchor-signed certificate. Forming the name by concatenating the home prefix with the verb "cert", this request carries four parameters. Besides the device identifier, N_2 is included to confirm the freshness of this request, and the digest of the trust anchor is inserted to allow the legitimate controller to determine if the device has installed a fake trust anchor. A controller may receive multiple sign-on requests of the same device through replay attacks, and cannot tell which one is fresh at the time of processing them. Accordingly, the controller is unable to pick the right N_1, which is supposed to be carried in the fresh sign-on request, for the existing sign-on instance. So, N_1 is also provided in the certificate request. And this request is signed by K_s^- to ensure its authenticity.

The controller verifies the signature of the request using K_s^+, and also examines N_2 and the digest of the trust anchor. A bad signature or an inconsistent N_2 leads to the dropping of the request. However, the detection of an incorrect trust anchor in a validated request will trigger the reactions introduced in Section IV-C. By contrast, if everything is correct, the controller will calculate K_t based on N_1 and the private key corresponding to N_2 (the two challenges are actually public keys for performing the Diffle-Hellman key exchange). Depending on if they exist and the renewal policies, K_d and the anchor-signed certificate are either retrieved or generated. Lastly, K_d^- is encrypted by K_t.

5) the certificate response: As the final step, the controller replies to the validated certificate request with a response of the same name signed by K_t, the content of which encapsulates the anchor-signed certificate and the encryption of K_d^-. The device completes the sign-on process by verifying the certificate response's signature, decrypting the encrypted item by K_t, and installing the keying materials obtained from the certificate response.

I. Security Analysis

To formally verify its security properties, we analyze the *sign-on-basic* protocol with the Cryptographic Protocol Shapes Analyzer (CPSA) [8], which outputs the shapes of all possible protocol executions from the perspective ofevery party. Based on the shapes, we summarize in table I several meaningful attack instances, along with where, when, and how they are stopped. The following subsections analyze how our protocol can survive from the attacks categorized in Section III-A.

1) fraudulently sign on to the system: To sign on to the system, the adversary needs at least an anchor-signed certificate and the corresponding private key. The adversary may obtain these two by either fooling the controller or cracking the messages issued to a legitimate device. Without K_s^-, the adversary can only pass the authentication of requests by replaying completed ones, but this will fail the freshness verification as demonstrated in attack instance-4 in table I. Even if an adversary captures its encryption, the adversary is unable to decrypt the private key issued to a legitimate device, as the decrypting key K_t is negotiated via a Diffie-Hellman key exchange and will never be exposed to a third party.

2) impersonating the controller: As long as the sign-on code is kept secret, there is no way for an adversary to impersonate the controller. If an impersonation attempt does not

TABLE I
ATTACK INSTANCES AND HOW THE *sign-on-basic* PROTOCOL SURVIVES

	attack execution sequence	protocol executions at where the exception is detected	
		sign-on code is kept secret	sign-on code is exposed
1	snd(F1)-rcv(2)-snd(F3)-rcv(4)	controller: rcv(F1); *bad signature*	
2	snd(F1)-rcv(2)-snd(R3)-rcv(4)	controller: rcv(F1); *bad signature*	
3	snd(R1)-rcv(2)-snd(F3)-rcv(4)	controller: rcv(R1)-snd(2)-rcv(F3); *bad signature*	
4	snd(R1)-rcv(2)-snd(R3)-rcv(4)	controller: rcv(R1)-snd(2)-rcv(R3); *inconsistent* N_2	
5	rcv(1)-rcv(2)-snd(F3)-rcv(4)	controller: rcv(1)-snd(2)-rcv(F3); *bad signature*	
6	rcv(1)-rcv(2)-snd(R3)-rcv(4)	controller: rcv(1)-snd(2)-rcv(R3); *inconsistent* N_2	
7	rcv(1)-rcv(2)-rcv(3)-snd(F4)	device: snd(1)-rcv(2)-snd(3)-rcv(F4); *bad signature*	
8	rcv(1)-snd(F2)-rcv(3)-snd(F4)	device: snd(1)-rcv(F2); *bad signature*	controller: rcv(1)-snd(2)-rcv(3); *fake trust anchor*
9	rcv(1)-snd(F2)-rcv(2)-snd(F3)-rcv(3)-snd(F4)	device: snd(1)-rcv(F2); *bad signature*	controller: rcv(1)-snd(2)-rcv(F3)-rcv(3); *fake trust anchor*

the numbers 1, 2, 3, 4 inside brackets denote the four messages defined in the protocol. the characters F, R represent the words "fake" and "replayed".

have K_t, the malicious entity will fail either in authentication or freshness verification. However, once an adversary knows the sign-on code, the device can never tell the adversary and the controller apart. Fortunately, our protocol enables the controller to be notified of the problem by checking the certificate request, as shown in attack instances 8 and 9.

3) flooding fake or completed requests: As demonstrated in attack instances $1 \sim 6$, our protocol stops all fake or replayed requests, regardless of the sign-on code's confidentiality. All malicious messages but replayed sign-on requests are immediately stopped. Before receiving the certificate request, replayed sign-on requests cannot be recognized. The cost of unnecessary replies to replayed sign-on requests is acceptable, because only one fresh challenge is generated within a sign-on instance. Additionally, all other costly operations are postponed until both authentication and freshness of a sign-on instance are verified.

4) replaying completed responses: Randomness is ensured in every request, therefore replayed responses will be filtered out directly by NDN due to a mismatch of names.

V. Protocol Extensions with Capable Devices

The protocol can be extended to support eless resource constrained devices. We consider devices that are equipped with three capabilities: interactive interfaces, writable permanent storage, and asymmetric key-pair generation.

A. Sharing Dynamic Secrets via Interactive Interfaces

If a device has an interactive interface, such as an operable input interface or a visible display, a dynamic secret can be generated and shared between the two ends for both mutual authentication and freshness verification. Similar to Bluetooth pairing, such an interactive sharing allows one end to manually input the secret dynamically generated on the other end.

The short lifespan of such a secret enhances security. As a result, the sign-on protocol may be simplified to take just one round of request / response exchange; in this case, a symmetric key derived from the shared secret is used to sign both messages and encrypt confidential information.

B. Reusing Existing Keying Material at Re-Sign-on

For a device equipped with permanent storage, sign-on processes after the initial one can be significantly simplified. In this case, the device keeps the trust anchor, the anchor-signed certificate, and the corresponding private key in its permanent storage. When the device needs to restart, it loads these credentials and uses it's certificate's corresponding private key to sign a sign-on request. With copies of keying materials of every device stored, the controller can determine how to move on with the sign-on process. If the signing key of the sign-on request is different from that recorded, the controller directly rejects the request. Otherwise, if the certificate of the legitimate signing key is still valid, the controller ends the sign-on process by responding with a confirmation of the validity of keying materials of the device. However, if the device's current key's certificate needs to be renewed, a new anchor-signed certificate is made and carried in the sign-on response. In the worst case that the key-pair needs renewing or that the trust anchor has been upgraded, the sign-on protocol is essentially the same as *sign-on-basic*, but with two simplifications. First, the same trust anchor will not be transmitted again. Second, in case the key-pair stays the same, there is no need to transmit the encrypted private key, nor to negotiate K_t. In this case, the controller would issue a new certificate signed with the new trust anchor for the device's original key-pair, and sign the response to the certificate request by K_c.

C. Generating Device Key-Pairs at the Device

If the device is capable of generating key-pairs itself, there is no need to transmit the encrypted private key in the certificate response, nor to negotiate K_t. The cost is that the device has to generate a key-pair every time it restarts unless it has permanent storage. Additionally, the device's newly generated public key must be provided in the certificate request, in order to be converted into an anchor-signed-certificate.

VI. Performance Evaluation

We implemented the *sign-on-basic* protocol and its extensions on two android phones (as controllers) and ESP32 boards (as devices). Two metrics were measured: 1) the time taken to sign on a device, and 2) the lifespan of the sign-on instance on the controller, from the time the controller approves the sign-on request to the time it sends out the certificate response. The first metric indicates the efficiency of sign-on, while the second one can be used as a reference value for setting a timer

to keep a sign-on instance on the controller. Both metrics were measured in seconds, and the average over 10 trials is reported.

A. Impact of Different Security Strengths

We implement *sign-on-basic* with different security strengths: 80-bit security, 128-bit security and a hybrid of the above two. Based on elliptic-curve cryptography, the primitives adopted in every implementation are presented in Table II, which are selected according to [6] and the availability on the device. In the hybrid implementation, all cryptographic primitives are guaranteed to have a 128-bit security except the Diffle-Hellman key exchange for negotiating K_t between two ends, where 80-bit security is used. The reasons for lowering down this process's security strength are twofold: 1) it is relatively costly and 2) we only need to keep K_t safe within one lifespan of the sign-on instance.

As shown in Fig. 3, the lower the security strength, the faster the sign-on process. A higher security strength requires more computation in cryptographic operations and longer keys to transmit. In all cases reported, the sign-on completes within 2 seconds. The computation time spent on cryptographic operations accounts for only 17% ~ 31% of the total sign-on time. The time taken by communications, including wireless transmission and NDN stack processing, is the bottleneck.

TABLE II
CRYPTOGRAPHIC PRIMITIVES

security strength	key length (in bits)			curves or algorithms			
	K_s [a]	K_c	K_t	ECDSA	ECDH	HMAC	AES
80-bit	160	128	128	secp160r1	secp160r1	SHA224	AES128
128-bit	256	128	128	secp256r1	secp256r1	SHA256	AES128
hybrid	256	128	128	secp256r1	secp160r1	SHA256	AES128

[a] K_d is of the same length as K_s.

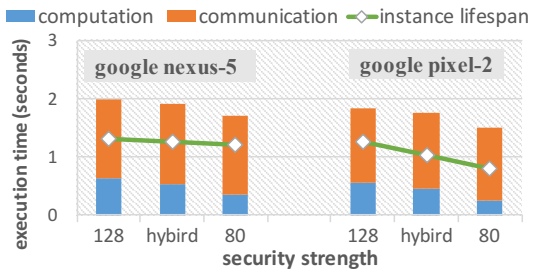

Fig. 3. exectution time of sign-on with different security strengths.

B. Performance of Different Versions

To give an insight of computation and communication costs of sign-on following our protocol, we measured the number of cryptographic operations performed and the number of bytes transmitted. In addition to the *sign-on-basic* protocol, we evaluated the same metrics for four of its extensions introduced in Section V. *sign-on-ds* and *sign-on-dk* represent respectively that a dynamic secret is shared between two ends and the device is able to generate key-pairs itself. In the case that the device has permanent storage, two versions are evaluated: *sign-on-ps-1* and *sign-on-ps-2*. In *sign-on-ps-1*, only the anchor-signed certificate needs renewal, and the whole process completes in one round trip. In *sign-on-ps-2*, the trust anchor is updated but not the key-pair. This means two round trips are required, but the encrypted private key does not need to be sent over the network as it is already in the device.

We implemented these versions based on elliptic-curve cryptography, with a 128-bit security strength. We measured the time taken for four types of cryptographic operations, the ECDH for K_t, the ECDSA for signing / verification using asymmetric keys, the HMAC for symmetric key signing / verification, and AES for symmetric key encryption / decryption. We divide packet contents into three categories and counted the bytes of each separately. The *sign-on* category contains entities acquired by the device as the purpose of sign-on, including the trust anchor, the device's anchor-signed certificate, and the corresponding private key. The *security* category has entities added to secure the sign-on process. The *ndn* category contains entities introduced by NDN packet encoding (such as names, Interest parameters, and so on). As shown in Table III, a more capable device requires fewer cryptographic operations for sign-on. Among all versions, the one for a device with an interactive interface is the fastest, as the sign-on process only involves four operations with a symmetric key (signing, verification, encryption, and decryption). Besides, NDN packet encoding accounts for the majority (45% ~ 60%) of the transmission cost excluding those needed for sign-on.

TABLE III
INSIGHTS OF COMPUTATION AND TRANSMISSION COSTS

protocol version	# of cryptographic operations				# of transmitted bytes		
	ECDH	ECDSA	HMAC	AES	sign-on	security	ndn
sign-on-basic	1	5	4	2	1632	441	376
sign-on-ds	0	0	2	2	1632	85	144
sign-on-ps-1	0	3	0	0	800	117	144
sign-on-ps-2	0	5	4	0	1600	269	308
sign-on-dk	0	5	4	0	1696	237	290

VII. Conclusion

In this paper, we propose a secure sign-on protocol for NDN-enabled smart home, where the local trust anchor facilitates the trust management. The proposed protocol enables a new device to obtain the trust anchor certificate as well as an anchor-signed certificate from the controller of a home. This establishes the foundation for applying NDN to build a secure home network. We describe the basic protocol for constrained devices, and also show that a more capable device can enable simplifications for sign-on through protocol extensions.

References

[1] L. Zhang, A. Afanasyev *et al.*, "Named Data Networking," *ACM SIGCOMM Computer Communication Review*, 2014.

[2] 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.

[3] 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.

[4] Z. Zhang, Y. Yu, H. Zhang, E. Newberry, S. Mastorakis, Y. Li, A. Afanasyev, and L. Zhang, "An overview of security support in named data networking," Technical Report NDN-0057, NDN, Tech. Rep., 2018.

[5] A. Compagno, M. Conti, and R. Droms, "Onboardicng: a secure protocol for on-boarding iot devices in icn," in *Proceedings of the 3rd ACM Conference on Information-Centric Networking*. ACM, 2016, pp. 166–175.
[6] D. Giry, "Bluekrypt-cryptographic key length recommendation," *URL: www. keylength. com*, 2013.
[7] E. W. Schuster, S. J. Allen, and D. L. Brock, *Global RFID: the value of the EPCglobal network for supply chain management*. Springer Science & Business Media, 2007.
[8] J. Ramsdell, "The cryptographic protocol shapes analyzer (cpsa)."