A Secure Sign-On Protocol for Smart Homes over Named Data Networking

Yanbiao Li, Zhiyi Zhang, Xin Wang, Edward Lu, Dafang Zhang, and Lixia Zhang

The authors introduce the design of a secure sign-on protocol, SSP, for smart homes built on named data networking (NDN). Instead of depending on cloud services, NDN supports a new smart home model where each home IoT system is identified by a unique name and has a local trust anchor.

Abstract

This article introduces the design of a secure sign-on protocol, SSP, for smart homes built on named data networking (NDN). Instead of depending on cloud services, NDN supports a new smart home model where each home IoT system is identified by a unique name and has a local trust anchor. To securely sign into such a home, a new device must acquire two certificates to secure its communications thereafter: the local trust anchor's certificate, whereby the device can cryptographically authenticate others in the same home, and its own certificate signed by the trust anchor to certify the device's identity and authenticity. SSP is designed for resource-constrained devices and built on NDN's security framework and Interest/Data exchange communication semantics, and is able to automate the process for a device to obtain those two certificates based on a piece of pre-shared information between the anchor and the device. Our security analysis and prototype implementation show that SSP offers strong protection against attacks even if the pre-shared secret is leaked later. We also discuss how SSP can be simplified and further enhanced for more resourceful devices.

Introduction

Smart homes, 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 enable these devices to function synergistically. At the same time, a home network poses a high demand on security, privacy, as well as usability of any security solutions. This article addresses the specific challenge of the security bootstrapping of a new device into smart homes to enable it to securely communicate with other devices.

Most of today's smart home ecosystems and existing research results either heavily rely on clouds [1, 2] or other remote entities [3], or require extensive human intervention [2] for device sign-on. The need for interaction between local devices and remote entities not only introduces extra latency and unnecessary dependence on external connectivity, but also opens a venue for attacks by adversaries. In addition, cumbersome manual operations create a barrier for users, reducing service usability.

The work in [4] explores the potential of applying named data networking (NDN) to IoT, and demonstrates its power in enabling local trust management and rendezvous, which breaks cloud dependency and enables fully localized control [5]. NDN enables every home to establish a local trust anchor. For such an NDN-based smart home, we design a secure sign-on protocol (SSP) to enable secure and automated new device sign-on with necessary pre-shared information. SSP is not only resilient against entity impersonation, man-in-the-middle attack, denial-of-service attacks, as well as replay attacks, but can also ensure system security in case the pre-shared secret is revealed. In addition, SSP requires only one simple manual operation (e.g., scanning the QR code of the device) for obtaining pre-shared information, making it usable for a wider range of users.

The main contributions of this article are threefold:

1. We develop a concrete device sign-on protocol for NDN-enabled smart homes, paving the way to localized trust and security.
2. We formally verify the security of the proposed protocol, and evaluate its performance over an implementation on real IoT devices. Our results shows that a device can complete the sign-on process at a reasonable cost.
3. In addition to the basic protocol, we explore simpler and more secure alternate designs for more capable devices.

Background and Related Work

Basic NDN Security Mechanisms

In NDN, each request is carried in an *Interest* packet, which contains the name of the requested data, and fetches one *Data* packet back. NDN builds public-key cryptographic protection into the architecture by requiring every Data packet to carry a digital signature to bind its name to the content. We refer interested readers to [6] for more details on the overall NDN security development. Below we introduce a few basic terminologies closely related to the sign-on protocol design.

A *trust anchor* is an entity trusted by all others within a given system. It is represented by a self-signed certificate, often called the root certificate. An NDN *certificate* certifies an entity's

Digital Object Identifier:
10.1109/MCOM.2019.1800789

Yanbiao Li and Dafang Zhang are with Hunan University; Zhiyi Zhang, Edward Lu, and Lixia Zhang are with UCLA; Xin Wang is with the State University of New York at Stony Brook.

ownership of a name and its key(s) by binding the name and key(s) together with a digital signature generated by the certificate authority. The signature component in an NDN packet contains the name of the signing key. One can observe a *certificate chain* by recursively tracking the signing key *K* of a packet *P* and the signing key of the packet carrying *K* until reaching the trust anchor. 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."

Related Work

OnboardICNg [3] is the first secure protocol for authenticating and authorizing IoT devices in mesh networks over information-centric networking. It suggests desirable efficiency and security, but still requires a remote trust center. In contrast, NDN-FLOW [5] explores a new direction enabled by NDN of bootstrapping security upon local trust anchor(s) by local means. Our proposed design follows this direction, but is more resilient to attacks from network adversaries than either OnboardICNg or NDN-FLOW's bootstrapping protocol.

System Model and Design Goals

We abstract a smart home as a heterogeneous network composed of resource-constrained devices (e.g., sensors and actuators) and more capable devices (e.g., laptops and phones). NDN enables every home to establish a local trust anchor, which defines the namespace as well as the root certificate of the home network. A capable device can be a *controller* of the home, which is empowered to sign other devices' certificates with the trust anchor certificate. The home owner, or anyone granted the privilege, can manage the whole system through the controller. Every device in a home is supposed to obtain a name under the home namespace, and a certificate directly or indirectly signed with the trust anchor certificate. Figure 1 shows a simple example of an NDN-enabled smart home: the mobile phone serves as the controller and owns the trust anchor's signing key. Every device, including the phone, has a certificate directly signed with the trust anchor certificate. 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 designed to assist a device in acquiring a copy of the trust anchor certificate and an anchor-signed certificate certifying the device itself. Device capability is an important factor to consider in designing such a protocol. We take three capabilities into consideration:

- The ability to generate key-pairs with high entropy
- The availability of permanent storage
- Support for human interaction

The controller must have all of them. We first design a basic protocol for devices that have none of the aforementioned capabilities, and then simplify the procedures and strengthen the security for devices with some or all of those capabilities.

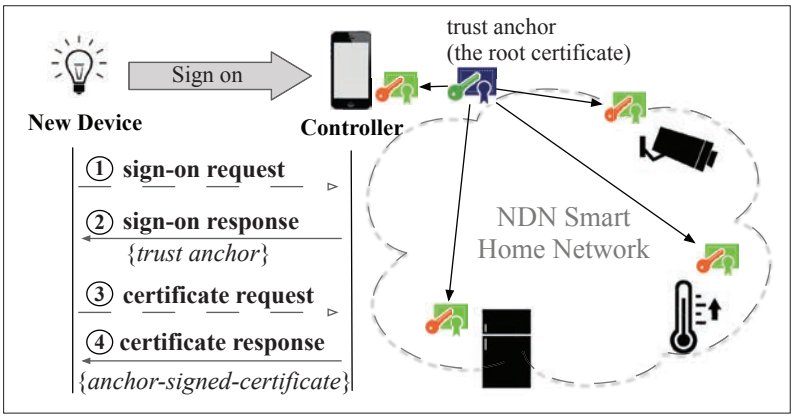

Figure 1. An NDN based smart home and a new device trying to sign on.

Adversary Model

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

1. *Fraudulently signing on to the system*, after which the adversary-controlled device may break the whole system
2. *Impersonating the controller*, where the device will be bootstrapped by an adversary-controlled "controller"
3. *Flooding fake or completed requests*, whereby legitimate requests may be impeded due to resource limitation
4. *Replaying completed responses*, whereby the device may be fooled into installing "outdated" keying materials

In the case of the *man-in-the-middle* attack, an adversary can only do one more attack other than the above four. It may be able to isolate either the device or the controller (or even both) by blocking the packets from or to the victim(s). Once the controller is isolated, its operator will immediately recognize the problem and fix it with human intervention. Although the isolation of a device will prevent it from signing on to the system, it would never affect the functionality of the whole system.

Basic Assumptions

1. The controller and the device allowed to sign on are not compromised before the sign-on process begins.
2. The communication between the controller and the device uses a wireless broadcast channel, which cannot be blocked by malicious parties.
3. The device carries some encoded confidential information that is accessible to the controller via out-of-band operations. This creates a secure way to establish a shared secret.

Design Goals

Our design goals are twofold. First, once all of the aforementioned assumptions hold, the sign-on must succeed in a reasonable timeframe. Second, the sign-on protocol should never endanger the system when the pre-shared secret is revealed.

More specifically, the device must be able to obtain required keying materials after the sign-on process. During this process, mutual authentica-

We let the device initiate the sign-on process for two reasons. First, it is likely that the controller has already been started and is ready to respond by the time the device starts. Second, a device without permanent storage has to sign on again when it restarts. Without human intervention or probing, the controller will not know there is a need to re-initiate the sign-on process.

tion must be achieved to eliminate impersonation. Fake or replayed requests must be detected and ignored at the earliest stage and at a reasonably low cost.

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. Second, the risk of a legitimate device installing a fake trust anchor should be reasonably controlled. Last, if a device is deceived into trusting the adversary, there must be a way for the controller to detect and react to it quickly. Otherwise, a compromised device will break the security and privacy of the living space.

Sign-On Protocol for Constrained Devices

In this section, we introduce the basic SSP, named *ssp-basic*, for those devices that have none of the aforementioned capabilities. As shown in Fig. 1, our design uses two rounds of request-response exchanges initiated by the device. We name these messages the *sign-on request*, the *sign-on response*, the *certificate request*, and the *certificate response*, respectively, in the order in which they are transmitted. The next few subsections detail the specific design issues and our solutions, as well as the security analysis.

Initiation of Sign-On

We let the device initiate the sign-on process for two reasons. First, it is likely that the controller has already been started and is ready to respond by the time the device starts. Second, a device without permanent storage has to sign on again when it restarts. Without human intervention or probing, the controller will not know there is a need to re-initiate the sign-on process.

Mutual Authentication

Generally, a pre-shared symmetric key is used to achieve mutual authentication [3]. However, the revelation of the key will endanger both ends. We propose the use of a set of keys to minimize this kind of threat. They are a symmetric key used to authenticate the controller and an asymmetric key pair used to authenticate the device. The private key of this asymmetric key pair is installed along with the symmetric key during device manufacture, while its public key and this symmetric key are shared with the controller before sign-on. By this means, the revelation of the pre-shared secret (the keys) will not break device authentication.

To mitigate the damage caused when a fake controller knows the pre-shared secret, we propose two tactics. First, the device trusts the first "controller" that replies to its sign-on request with proof of knowledge of the pre-shared secret. A legitimate controller likely has a higher chance than a fake one to reply first, because it is likely physically closer to the device in the case where the adversary is outside the home.

However, there is a chance that the legitimate controller's response arrives too late. In this case, the device may be fooled into installing a fake trust anchor. In order to enable the legitimate controller to detect this exception, we require that the device insert a digest of the trust anchor, which is received from the sign-on response, into the certificate request and broadcast this request. When hearing this broadcast, the legitimate controller can tell whether there is such an exception. This detection procedure works only if the protocol runs with at least two round-trips.

Freshness Verification

Two fresh challenges are used to stop replay attacks. Each end injects a random number into the message to send, and expects it to be in the subsequent message from the other end. More specifically, the device generates and encodes the first challenge into its sign-on request as an Interest parameter. As it contributes to the last name component (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 the 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 decodes this challenge from the sign-on response and then encodes it as a parameter in its certificate request. This not only allows the controller to verify the freshness of the certificate request, but also ensures the freshness of its response, which is verified by the device via name match as well.

Issuance of Anchor-Signed Certificate

To make an NDN certificate of the device, its name and a public key are required. We create a device name by appending a unique device identifier to the home prefix learned from and certified by the trust anchor.

As per the least privilege rule, a new key-pair of the device is generated for use after sign-on. Since the resource-constrained device lacks the ability to generate high-entropy keys for long-term use, the controller is responsible for generating this key-pair, creating the device certificate and signing it with the trust anchor. 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 when to make new ones according to a renewal protocol (which is beyond the scope of sign-on).

Cryptographic Keys

Here, we detail what keys are used in all the cryptographic operations of the ssp-basic protocol.

Between the two ends, the pre-shared information establishes consensus on an asymmetric key pair and a symmetric key, which are used for mutual authentication. The device signs its requests with the private key of this asymmetric key pair, and the controller uses the public key to verify the signature of these requests. The controller signs its responses to these requests with this symmetric key, and the device verifies their signatures via this key as well.

The exposure of a device's private key endangers the whole system, so it is a must to encrypt the private key that the controller generates for and transmits to the device. Neither of the aforementioned two sets of keys is a good option for this purpose. On one side, asymmetric encryption and decryption are always costly, especial-

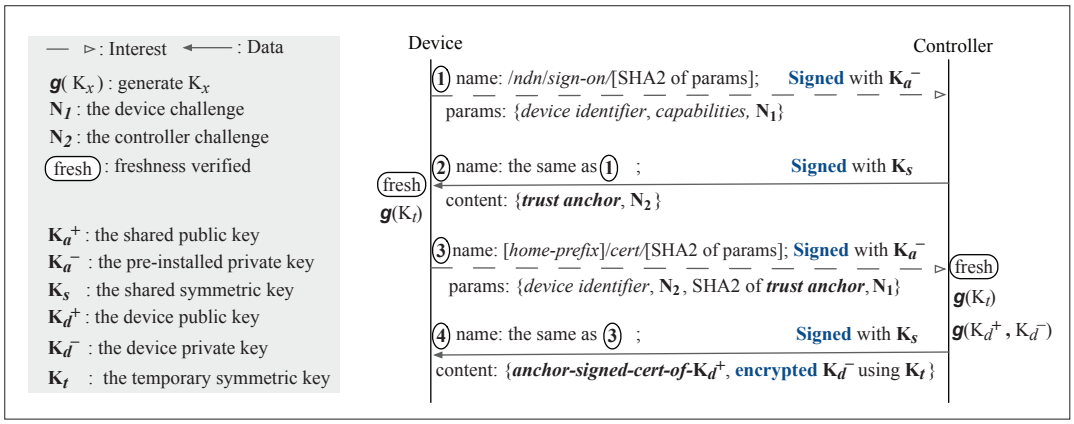

Figure 2. Message exchange details of the ssp-basic protocol.

ly in constrained devices. On the other side, the encryption using the pre-shared symmetric key can easily be cracked in the case of its revelation. As such, we propose the use of a separate temporary symmetric key, which is generated via a Diffie-Hellman key exchange between the two ends. The two fresh challenges mentioned earlier are also used as the keys to exchange in this scenario.

Preventing Denial-of-Service Attacks

We use two approaches to mitigate potential denial-of-service attacks made by an adversary of flooding fake or completed requests to the controller.

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

Second, for any device, only one sign-on instance is maintained at the controller. Such an instance is created after a sign-on request of a device is validated, and is destroyed once a certificate request of the same device is validated and processed. Additionally, a timer is set at its creation to prevent it from existing for too long. Before the sign-on instance for a device finishes or expires, all validated sign-on requests of this device will share this instance and get the same fresh challenge in their replies.

Message Exchange Details and Cryptographic Primitives

The message exchange details of the ssp-basic protocol are shown in Fig. 2, where a set of cryptographic keys are involved. In the pre-shared secret, there is a symmetric key and a public key whose corresponding private key is pre-installed at the device. Another asymmetric key-pair will be generated for the device to use after sign-on, which is referred to as a "device key-pair." The two fresh challenges generated at the device and the controller are referred to as "device challenge" and "controller challenge," respectively. They are also used as public keys to establish a temporary symmetric key via Diffie-Hellman key exchange.

Our reference implementation of the protocol targets a 128-bit security level [7] and adopts elliptic-curve cryptography, where all asymmetric key-pairs are on a 256-bit elliptic curve, and all symmetric keys are 128 bits in length. ECDSA and HMAC with SHA256 are used to sign messages with asymmetric and symmetric keys, respectively; AES128 is adopted for data encryption. The temporary symmetric key is generated with ECDH. Due to the lack of device capabilities, the two fresh challenges as well as the temporary symmetric key are generated with low entropy. This is acceptable because they are all for short-term use within the lifespan of a sign-on instance (a couple of seconds per our evaluations).

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, which includes a public key, a symmetric key, and the device identifier. In addition to making an NDN name, the controller also uses the device identifier to distinguish sign-on requests of different devices. We encode this information into a QR code. Before the sign-on starts, the controller scans this QR code to obtain this information.
2. *The sign-on request:* The device initiates the sign-on process by broadcasting a sign-on request, using an NDN Interest packet named under the "/ndn/sign-on" prefix and signed with the pre-installed private key. Three parameters are encoded with this Interest: the device identifier, device capabilities, and device challenge. The digest of all the parameters as a whole forms the last name component. The capabilities parameter is a 1-byte bitmap that encodes the availability of every capability leading to sign-on procedure simplifications. After receiving a sign-on request, the controller verifies its signature using the shared public key of the corresponding device. Upon successful verification, and if there is no existing one for this device, a sign-on instance is created with a timer activated. Additionally, the controller challenge is generated and associated with this instance.
3. *The sign-on response:* In replying to a sign-on request, a sign-on response, as an NDN Data packet, is made with the same name, whose content encapsulates the trust anchor and the controller challenge. To validate this response, the match of its name to that of the pending sign-on request is verified at NDN's data plane, which guarantees the controller's knowledge of the device challenge and thus ensures the freshness of this

The temporary symmetric key is generated with ECDH. Due to the lack of device capabilities, the two fresh challenges as well as the temporary symmetric key are generated with low entropy. This is acceptable because they are all for short-term use within the lifespan of a sign-on instance (a couple of seconds per our evaluations).

If a device has an interactive interface in addition to permanent storage, such as an operable input interface or a visible display, there is another option for its initial sign-on. In this case, instead of letting the controller to obtain static pre-shared information, a dynamic secret can be generated and shared between them for both mutual authentication and freshness verification.

response. Then its signature is verified using the shared symmetric key. On a validated response, the devices will trust the other end to be a legitimate controller. At this point in time, it is safe to generate the temporary symmetric key and to install the trust anchor at the device.

4. *The certificate request:* After validating the sign-on response, the device broadcasts a certificate request signed with the pre-installed private key to acquire the anchor-signed certificate. Its name is made by concatenating the home prefix learned from the trust anchor, the verb "cert," and the digest of its parameters. After receiving a certificate request, the controller decodes its parameters, uses the device identifier to locate a sign-on instance, and then compares the controller challenge associated with this instance against that carried in the request to confirm its freshness. Thereafter, the controller verifies the request's signature using the shared public key and examines the digest of the trust anchor installed at the device. A bad signature or an inconsistent controller challenge leads to the request being dropped. However, the detection of an incorrect trust anchor in a validated request triggers a special alert. If everything is correct, the controller generates the temporary symmetric key based on the device challenge carried in this request. Depending on whether they exist and the renewal policies, the device key-pair and the anchor-signed certificate are either retrieved or generated.
5. *The certificate response:* As the final step, the controller replies to the validated certificate request with a response signed with the shared symmetric key, whose content encapsulates the anchor-signed certificate and the encryption of the device private key using the temporary symmetric key. Then, after receiving and validating this response, the device decrypts the encrypted private key, and installs it and the anchor-signed certificate to complete the sign-on process. If required, the controller can also distribute other parameters of NDN security to devices via this message.

Security Analysis

We analyze the ssp-basic protocol with the Cryptographic Protocol Shapes Analyzer,[1] which outputs the shapes of all possible protocol executions. From the shapes, we summarize meaningful attack instances categorized as follows, and confirm that our protocol survives them.

1. *Fraudulently signing on to the system:* Without the pre-installed private key, the adversary will fail either authentication or freshness verification, and thus be unable to fool the controller. Besides, he/she is unable to decrypt the private key issued to a legitimate device, as the temporary symmetric key will never be exposed to a third party.
2. *Impersonating the controller:* The adversary can only impersonate the controller when he/she knows the shared symmetric key, but in this case our protocol enables the controller to be notified of the problem by checking the certificate request.
3. *Flooding fake or completed requests:* Our protocol stops all malicious messages immediately except replayed sign-on requests, which can only be recognized via validating the certificate request. However, the cost of unnecessary replies to them is acceptable, because only one fresh challenge is generated within a sign-on instance, and all other costly operations are postponed until both authentication and freshness 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.

Protocol Modifications with Capable Devices

SSP can be simplified or enhanced for more resourceful devices.

Generating Device Key-Pairs at the Device

If the device is able to generate key-pairs with high enough entropy itself, there is no need to transmit the encrypted private key or to negotiate the temporary symmetric key. The cost is that the device has to generate a key-pair every time it restarts unless it has permanent storage. And the device's newly generated public key must be provided in the certificate request, to be converted into an anchor-signed certificate.

Reusing Existing Keying Material at Re-Sign-On

For a device that has 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. Right after it restarts, the device loads these credentials and uses the private key to sign a sign-on request.

After validating this request, the controller can move on with the sign-on process in the following ways. If all related keys and certificates are still valid, it ends the sign-on process by responding with a confirmation of their validity. If only the anchor-signed certificate needs to be renewed, a new one is made and carried in the sign-on response. In the worst case, when the key-pair needs renewing or the trust anchor has been upgraded, the sign-on protocol is essentially the same as *ssp-basic*, but with two simplifications. First, the same trust anchor will not be transmitted again. Second, if the key-pair stays the same, there is no need to transmit the encrypted private key or negotiate the temporary symmetric key.

Sharing Dynamic Secrets via Interactive Interfaces

If a device has an interactive interface in addition to permanent storage, such as an operable input interface or a visible display, there is another option for its initial sign-on. In this case, instead of letting the controller obtain static pre-shared information, a dynamic secret can be generated and shared between them for both mutual authentication and freshness verification. Similar to Bluetooth pairing, such interactive sharing allows one end to manually input the

[1] The cryptographic protocol shapes analyzer (CPSA); http:// hackage.haskell.org/package/cpsa

secret dynamically generated on the other end. In this case, only one round of request/response exchange is required, where a symmetric key derived from the shared secret is used to sign both messages and encrypt confidential information.

Performance Evaluation

With two Android phones (as controllers) and a couple of ESP32 boards (as devices), we evaluate the performance of ssp-basic and all the versions for less resource-constrained devices, which are able to generate high-entropy key-pairs (*ssp-hk*), have sufficient permanent storage (*ssp-ps*), or can share a dynamic secret with the controller (*ssp-ds*). For the second one, two sub-versions are evaluated: *ssp-ps-1* and *ssp-ps-2*. In *ssp-ps-1*, only the anchor-signed certificate needs renewal, and the whole process completes in one round-trip. In *ssp-ps-2*, only the trust anchor is updated, so two round-trips are required but without transmitting the encrypted private key.

For every protocol, we evaluate the computation and communication costs of a sign-on process following this protocol in terms of the number of cryptographic operations performed on two ends and the number of bytes transmitted between them, respectively. Two time metrics are measured: the time taken to sign on a device, and the lifespan of the sign-on instance on the controller. 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.

Impact of Different Security Strengths

We implement ssp-basic with different security strengths: 80-bit security, 128-bit security, and a hybrid of the above two. The cryptographic primitives adopted in every implementation are presented in Table 1, which are selected according to [7] and the availability on the device. In the hybrid implementation, all cryptographic primitives are guaranteed to have 128-bit security except the ECDH for negotiating the temporary symmetric key, where 80-bit security is used. The reasons for lowering this process's security strength are twofold: it is relatively costly, and the lifespan of the temporary symmetric key is short.

As shown in Fig. 3, the lower the security strength, the faster the sign-on process. In all cases reported, the sign-on completes within 2 s. The computation time spent on cryptographic operations accounts for only 17~31 percent of the total sign-on time. The time taken by communications, including wireless transmission and NDN stack processing, is the bottleneck. We also evaluated an implementation using RSA with 1024-bit keys (80-bit security), suggesting a slower sign-on (~3 s) and a higher proportion of computation (~50 percent).

Performance of Different Versions

To give insight into computation and communication costs, we measure different cryptographic operations separately, and divide packet contents into three categories, the bytes of which are counted separately. The sign-on category contains entities acquired by the device for 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 (names, Interest parameters, etc.).

As shown in Table 2, 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). NDN packet encoding accounts for the majority (45~60 percent) of the transmission cost excluding those needed for sign-on.

Security strength	Key length (in bits)			Curves or algorithms			
	Ka, Kd	Ks	Kt	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

Table 1. Cryptographic primitives.

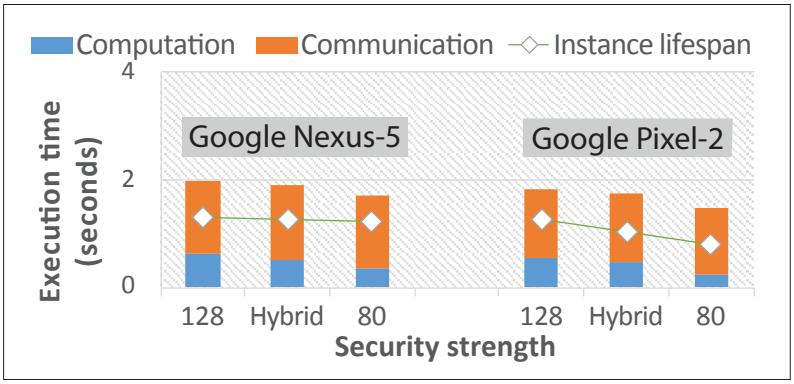

Figure 3. Execution time of sign-on with different security strengths.

Protocol version	Number of cryptographic operations				Number of transmitted bytes		
	ECDH	ECDSA	HMAC	AES	Sign-on	Security	NDN
ssp-basic	1	5	4	2	1632	441	376
ssp-hk	0	5	4	0	1696	237	290
ssp-ps-1	0	5	4	0	1600	269	308
ssp-ps-2	3	0	0	0	800	117	144
ssp-ds	0	0	2	2	1632	85	144

Table 2. Insights of computation and transmission costs.

Conclusion

In this article, we have proposed a secure sign-on protocol for NDN-enabled smart homes, where the local trust anchor facilitates the trust management. The proposed protocol enables a new device to obtain the trust anchor certificate and 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 enables simplifications for sign-on.

Acknowledgment

This work is partially supported by the National Natural Science Foundation of China under award 61702174 and the National Science Foundation under award CNS-1719403.

References

[1] Amazon, AWS Greengrass; https://aws.amazon.com/greengrass/, accessed Oct. 1, 2018.
[2] Apple, HomeKit; https://developer.apple.com/homekit/, accessed Oct. 1, 2018.
[3] A. Compagno *et al.*, "Onboardicng: A Secure Protocol for On-Boarding IoT Devices in ICN," *ICN 2016*, 2016, pp. 166–75.
[4] W. Shang *et al.*, "Named Data Networking of Things," *IoTDI 2016*, 2016, pp. 117–28.
[5] W. Shang *et al.*, "Breaking Out of the Cloud: Local Trust Management and Rendezvous in Named Data Networking of Things," *IoTDI 2017*, 2017, pp. 3–14.
[6] Z. Zhang *et al.*, "An Overview of Security Support in Named Data Networking," *IEEE Commun. Mag.*, vol. 56, no. 11, Nov. 2018, pp. 62–68.
[7] E. Barker *et al.*, "Recommendation for Key Management Part 1: General (Revision 3)," NIST Special Publication, vol. 800, no. 57, 2012, pp. 1–147.

Biographies

Yanbiao Li (lybmath@cs.ucla.edu) received his Ph.D. degree in computer science from Hunan University. He was a postdoctoral scholar at the University of California Los Angeles (UCLA), and his research focus is on networked systems.

Zhiyi Zhang (zhiyi@cs.ucla.edu) is a Ph.D. candidate in the Computer Science Department at UCLA. His research focus is on network security and NDN.

Xin Wang [M'1, ACM'4] (x.wang@stonybrook.edu) received her Ph.D. degree in electrical and computer engineering from Columbia University. She is an associate professor at the State University of New York at Stony Brook. Her research interests include wireless networks and communications, and networked sensing and detection.

Edward Lu (edwardzlu98@gmail.com) is an undergraduate student in the Computer Science Department at UCLA. His research focus is on NDN.

Dafang Zhang (dfzhang@hnu.edu.cn) received his Ph.D. degree in application mathematics from Hunan University. He is a professor at Hunan University, and his research interests include networked systems and big data.

Lixia Zhang [F] (lixia@cs.ucla.edu) is a professor in the Computer Science Department at UCLA. She received her Ph.D. from MIT. She is a Fellow of ACM, the recipient of an IEEE Internet Award, and the holder of the UCLA Postel Chair in Computer Science. Since 2010 she has been leading the effort on the design and development of NDN.