

On the Security Bootstrapping in Named Data Networking

Tianyuan Yu
UCLA
Los Angeles, USA
tianyuan@cs.ucla.edu

Xinyu Ma
UCLA
Los Angeles, USA
xinyu.ma@cs.ucla.edu

Hongcheng Xie
City University of Hong Kong
Hong Kong, China
hongcheng.xie@my.cityu.edu.hk

Xiaohua Jia
City University of Hong Kong
Hong Kong, China
csjia@cityu.edu.hk

Lixia Zhang
UCLA
Los Angeles, USA
lixia@cs.ucla.edu

ABSTRACT

By requiring all data packets be cryptographically authenticatable, the Named Data Networking (NDN) architecture design provides a basic building block for secured networking. This basic NDN function requires that all entities in an NDN network go through a security bootstrapping process to obtain the initial security credentials. Recent years have witnessed a number of proposed solutions for NDN security bootstrapping protocols. Built upon the existing results, in this paper we take the next step to develop a systematic model of security bootstrapping: Trust-domain Entity Bootstrapping (TEB). This model is based on the emerging concept of *trust domain* and describes the steps and their dependencies in the bootstrapping process. We evaluate the expressiveness and sufficiency of this model by using it to describe several current bootstrapping protocols.

1 INTRODUCTION

Named Data Networking (NDN) [4, 22] architecture provides semantically named and signed data to enable secured data communications. However, the architecture design needs useful tools to realize the security design. There are two fundamental requirements to fulfill NDN security design. First, NDN entities¹ require *Security Bootstrapping* to obtain necessary security components. Then, NDN entities also require *Security Support* to manage trust relations and certificates. Specifically, NDN entities need to know the appropriate certificate to sign data, and the appropriate verification chain to validate signed data. For example, a data producer may have multiple signing certificates, with different certificates used to sign data under different name prefixes, as indicated by the trust schema [17]. This demands that the Security Support not merely store cryptographic tools, but rather manage cryptographic tools following trust policies.

¹NDN entities are applications and all other network communication participants in an NDN network [26]

Recently, a number of security bootstrapping solutions [6, 8, 12] have been proposed. As the understanding of security bootstrapping evolves, in this work we take the next step to extract commonalities from different designs. By doing so, we intend to obtain a generic model describing the procedures of bootstrapping to understand NDN security better and support software development. In this work, we make the following contributions. First, we clarify the concepts of *NDN trust domain* [8] and the necessary steps to set up an NDN trust domain. Second, we develop a systematic understanding of security bootstrapping within a trust domain and propose a generic function model, *Trust-domain Entity Bootstrapping (TEB)*, to describe the procedures. Our TEB model is general enough to effectively model the existing protocols.

In the rest of this paper, Section 2, revisits the NDN network model, articulates the concept of NDN trust domains, and reviews related works. We then describe the TEB model and its individual procedures (Section 3), our evaluation based on protocol analysis (Section 4), and lessons learned from the evaluation (Section 5). Finally, we summarize our contribution, and mention the remaining questions and future work in Section 6.

2 BACKGROUND AND RELATED WORK

This section briefly reviews the NDN networking model and related work to lay the groundwork for introducing the Trust-domain Entity Bootstrapping TED design.

2.1 NDN Networking Model

An NDN is made of connected named entities, with various trust relations among them. Entities utilize all available connectivities to exchange named and secured data, and the defined trust schema to authenticate all received data. Since the trust schema expresses security policies by defining the relations between the names of data and the names of crypto keys used to sign and encrypt data, an NDN entity E must

have a semantically meaningful name to enable schematized trust relations [17].

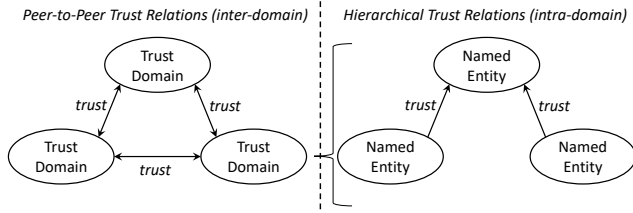


Figure 1: Examples of Trust Relations Among Named Entities

Trust relations can take different models. Two typical ones are the hierarchical trust relations and the peer-to-peer trust relations, as illustrated in Figure 1. Named entities that belong to the same administrative domain generally follow a hierarchical trust model [23], where all the entities in the domain share one single trust root. Independent entities, on the other hand, may take a Web-of-Trust model to establish their trust relations [3, 18], where they endorse each other’s crypto keys in a peer-to-peer fashion. These two models complement, rather than conflict, each other, and may coexist. For example, a smart home made of networked IoT devices can use the hierarchical trust model for all IoT devices in a user’s home, while neighboring homes may establish peer-to-peer trust relations, if they desire to communicate with each other (e.g. when home-A’s ISP has an outage, neighbor home-B may instructs its WiFi router to help forward traffic for home-A, and prohibit all its other devices from interacting with home-A’s traffic).

Sine the trust relations in a hierarchical trust model end at one single trust root, realizing a hierarchical trust model requires that each entity E under the same administrative control establishes a *trust anchor* that cryptographically identifies the trusted root, and installs the trust anchor into E . In order to produce authenticatable data inside its domain, E must also have its name(s) certified. The certified name(s) uniquely identify E in the system and each name’s authentication chain terminates at the same trust anchor. The trust schema, defined by the trust root, limits the signing power of each certificate to a specific data namespace, enabling applications to enforce finer-grained security policies for authentication, authorization, and access control.

2.2 Related Works

Our TEB model is built upon, and further extend, a few pieces of previous work in NDN bootstrapping. Zhang *et al.* [20, 26] identified the necessary security components that must be obtained from NDN bootstrapping process. Later, DCT [9] defines an NDN *trust domain* as a zero-trust network

governed by a single trust anchor and trust schema. The concept of trust domains helps precisely define the scope of security bootstrapping, i.e. configuring an NDN entity into an NDN trust domain. Following [8], Yu *et al.* [16] further introduced the concept of a *trust domain controller* as a trust domain’s governing entity, and articulated the steps of the security bootstrapping process for three different networking scenarios, where the steps of authentication and naming vary based on the application scenarios.

In this paper, we adopt the concept of trust domain controller and introduce the new concept of *elemental entities*. We show that TEB as a generalized bootstrapping model can cover all three different networking scenarios described in [16].

3 TEB MODEL

In this section, we formally define the concept of trust domains, and then model the security bootstrapping, by first describing the model in an overview of TEB then introducing each procedure in the model.

3.1 Trust Domain

The introduction of the NDN *trust domain* concept by [8] simplifies the description of trust relation organization. A *trust domain* is made of a collection of *authorized named entities* under the same administrator’s control. The *trust schema* for an entity E is the set of rules that defines E ’s trust relations with the others in the same domain. The entity who controls the trust relations of the domain is the trust domain *controller*. Specifically, a domain controller can control the security within its domain by (i) authenticating and authorizing each new entity E_{new} as a domain member; (ii) installing the trust anchor T and the trust schema into E_{new} ; (iii) naming E_{new} ; (iv) issuing a certificate to E_{new} . We refer to this set of operations as *security bootstrapping*. To set up an NDN trust domain, one needs to first decide the name of the trust domain, and set up a domain controller which will generate a self-signed certificate under that domain name; this certificate is the domain’s trust anchor T . Second, one needs to design the trust domain namespace, and define the trust schema of the domain and individual entities. Finally, one bootstraps entities into the domain as they become available. Among all bootstrapped entities, there may exist *elemental entities* to whom the domain controller partially delegates the control function (e.g. certificate issuance). In this case, the domain controller coordinates the elemental entities to manage the security within its domain. Since this work focuses on intra-domain bootstrapping, below we discuss how the domain controller bootstraps E_{new} .

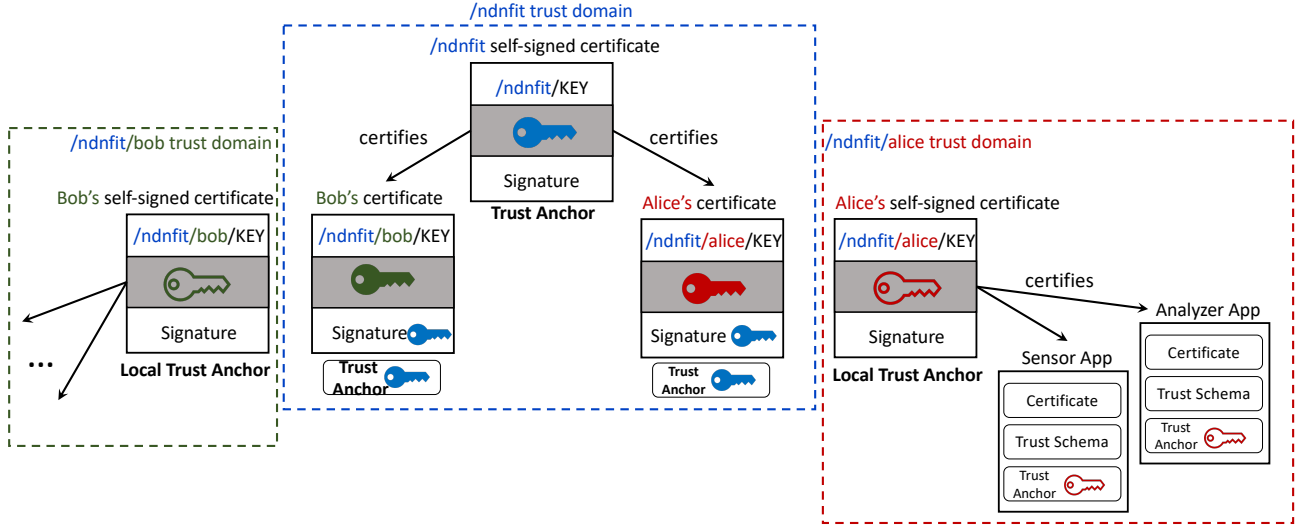


Figure 2: The relationships among the trust domains $/ndnfit$, $/ndnfit/alice$, and $/ndnfit/bob^2$

In Figure 2 (which is adopted from [26]), we illustrate the trust domain relationships using a prototype application NDNFit described in [21], which tracks and shares personal fitness activities. The NDNFit developers start the trust domain $/ndnfit$ by setting up its trust domain controller, which authenticates and authorizes Alice and Bob as its trust domain members. It installs the $/ndnfit$ trust anchor and the trust schema into Alice’s and Bob’s NDNFit application instances, then issues them the certificates $/ndnfit/alice/KEY/123/controller/v=1$ and $/ndnfit/bob/KEY/223/controller/v=1$.

Alice can set up her own trust domain $/ndnfit/alice$, by generating a self-signs $/ndnfit/alice$ certificate as her domain’s trust anchor. She runs a “Sensor” app on her smartphone to collect daily time-location information, and runs an “Analyzer” app on her laptop to produce analytics and visualizations from the data produced by “Sensor”. The trust domain controller of $/ndnfit/alice$ installs into “Sensor” and “Analyzer” their trust anchor, certificates, and trust schema. After security bootstrapping, the *intra-domain data communications* between the “Sensor” app and the “Analyzer” app can be secured.

On the left side of the figure, Bob bootstraps his applications in the same way by creating his own trust domain $/ndnfit/bob$. If Alice wants to share her sensor data with Bob, she can enable *inter-domain data communications* by defining a proper trust schema for authentication between two trust domains and data access control.

3.2 TEB Overview

TEB is designed as a security bootstrapping model for entities in a trust domain. TEB consists of four steps (Figure 3). First, the entity to be bootstrapped, E_{new} , and the trust domain controller need to perform mutual authentication in order to securely communicate with each other. Second, E_{new} needs to establish the trust relation with the trust domain by installing the trust anchor and the trust schema obtained from the controller. Third, the trust domain controller assigns E_{new} a semantic meaningful NDN name. Lastly, the trust domain controller issues E_{new} a certificate under its assigned name. After the above four steps are finished, E_{new} is ready for secure intra-domain data communications.

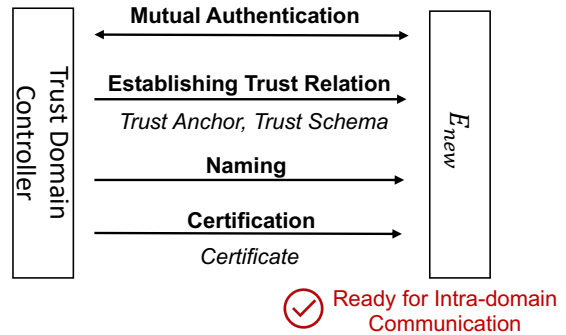


Figure 3: An Overview of the Security Bootstrapping

3.3 Controller Authentication

From E_{new} ’s perspective, the first step of security bootstrapping is to authenticate the trust domain controller and accept

it as its authority. We call this procedure *Controller Authentication*. Because *authentication relies on pre-established trust relations*, E_{new} and controller need some prior existing trust relation. We call this trust relation as *Controller Authentication Context (CAC)* on E_{new} side, and *E_{new} Authentication Context (EAC)* on the controller side. We further define each party’s unique identifier in bootstrapping process as *Controller Identifier (ContID)* and *E_{new} Identifier ($E_{new}ID$)*, respectively. As a result of controller authentication, E_{new} generates an *$E_{new}Approval$* to represent an entity identified by *ContID* as its trust domain controller.

We can define the binding between *$E_{new}Approval$* and *ContID* as *Proof of Authority (POA)* and the controller authentication procedure as *ContAuth*:

$$POA = (ContID, E_{new}Approval)$$

$$ContAuth : CAC \rightarrow POA$$

Authentication Context: Both E_{new} and the trust domain controller need the initial out-of-band trust relation with the other. For example, E_{new} can securely obtain *CAC* out-of-band from either the installed application (which embeds some configured trust) or human input at runtime (e.g. pre-shared keys or passcode). In today’s TCP/IP network practice, Certificate Authorities (CA) are implicitly authenticated by end-users’ trust on the OS and browser vendors, as well as the correct operation of their software. Whereas, IoT device manufacturers burn the initial key materials into devices as factory settings for *CAC*, to be used for authentication by smart home controllers. An NDN trust domain can also make use of these existing trust relations to authenticate its trust domain controller to E_{new} . For example, the trust domain controller can host its trust anchor and trust schema at an HTTPS-enabled website and share the URL with E_{new} out-of-band. In this case, the pre-shared URL, website X.509 certificate [2], and the corresponding CA root certificate together serve as *CAC*. We discuss this approach in detail in Section 4.

3.4 Entity Authentication

In order to join the trust domain, E_{new} can be authenticated and approved by the trust domain controller through the following two steps. First, the controller verifies E_{new} ’s trust domain membership. It checks whether the authentication factor in *EAC* is acceptable to the current trust domain. Then, the controller verifies E_{new} ’s identity with the authentication factor. After this two-step process, the controller approves E_{new} ’s to be a trust domain member with the temporary identifier $E_{new}ID$. We refer to this binding between $E_{new}ID$ and *ContApproval* as *Proof of Membership (POM)*. We can define

POM and the E_{new} authentication procedure *$E_{new}Auth$* as:

$$POM = (E_{new}ID, ContApproval)$$

$$E_{new}Auth : EAC \rightarrow POM$$

Authentication Context: The controller must securely obtain *EAC* before the security bootstrapping. Similar to *CAC*, obtaining *EAC* also relies on some trusted existing authentication system, e.g. it can be obtained from either the installed application or from human input at runtime. Today’s applications over the existing TCP/IP architecture take similar approaches to achieve end-user authentication. For example, today’s practice for registering new accounts to websites typically uses email authentication, which derives the user authenticity from the existing (and trusted) email systems.

3.5 Entity Trust Relations

After executing *ContAuth*, E_{new} can use *POA* to establish the initial trust relation via obtaining and installing the trust anchor and trust schema from the controller. Obtaining the trust anchor and trust schema enables E_{new} to validate Data packets received within the trust domain, including the certificate issued to it later. We name this procedure *$E_{new}Trust$* :

$$E_{new}Trust : POA \rightarrow (TrustAnchor, TrustSchema)$$

Initial Trust Schema: Since the trust schema may need to be updated over time, *$E_{new}Trust$* only installs *initial trust schema* into E_{new} and considers deploying application-specific trust schemas as a post-bootstrapping task for the controller. The initial trust schema includes all necessary rules to validate E_{new} certificate and future trust schema updates. As designed in [24], the trust domain controller can distribute application-specific trust schemas as Data packets after E_{new} bootstrapping.

The initial trust schema can be *implicit*, which means E_{new} by default trust every data produced by the controller until receiving a later trust schema that *explicitly* specifies the data signing relationships. We discuss the usage of implicit trust schema with details in Section 4.

3.6 Entity Naming

After E_{new} ’s authentication, the controller assigns E_{new} a name under the trust domain’s namespace. In the E_{new} naming procedure, the controller uses the trust domain’s *Naming Convention (NameConv)* [19] to determine E_{new} ’s name. The naming convention defines a set of naming rules to facilitate data publication and retrieval. The controller formally approves the binding of the assigned *Name* to $E_{new}ID$.³ This binding indicates E_{new} ’s legitimately possessing *Name*. We

³The format of this approval is defined by specific bootstrapping implementations.

denote this binding as *Proof of Possession (POP)*, and define E_{new} naming procedure as:

$$POP = (Name, E_{new}ID, ContApproval)$$

$$E_{new}Naming : (POM, NameConv) \rightarrow POP$$

At the end of this section, we briefly explain the reason for defining *POP*, rather than the certificate as the $E_{new}Naming$ output. We also discuss it with more details in Section 3.8.

Naming Convention: An entity name must be *unique* and *semantically meaningful*. Naming convention remains an active research topic, so far we have identified two commonly observed cases, and one can choose based on what identifiers are used in the authentication step.

The first case is that the controller converts a semantically meaningful and authenticated identifier to an NDN name. In general, we may be able to satisfy both requirements by making use of the already existing identifiers used in authentication. In some scenarios, the new entity’s authentication identifier (e.g. an email address, or a DNS name) semantically encodes its existing trust relations. For example, Alice is identified by her email address “alice@example.com”, which consists of a semantically meaningful domain name and a user name. The trust domain controller of “/ndnfit” can interpret this email address with the structure “user@sld.tld”. If the trust domain defines its naming convention as “/ndnfit/<tld>/<sld>/<user>”⁴, then the controller can assign Alice the name “/ndnfit/com/example/alice”, with the confidence that this name should be unique because email addresses are globally unique.

Another case is to manually assign names. For example, if Alice is authenticated by her SSH [15] public key without meaningful semantics⁵, The controller may request human input (such as via out-of-band operations) to fill in a name.

Name Possession: As stated in the *POP* definition, *POP* only indicates the controller’s approval on binding a specific name to an $E_{new}ID$, thereby being different from a certificate. Note that $E_{new}ID$ is not a cryptographic identifier for E_{new} , but rather an identifier exclusively used during bootstrapping before E_{new} can be uniquely identified by its certified name. *POP* conceptually decouples the naming procedure from certification, so that the domain certificate issuer can be agnostic to the entity naming convention and takes *POP* as input to certify E_{new} .

⁴<> indicates a wildcard name component.

⁵A SSH key pair is uniquely identified by its public key (or fingerprint). The default naming convention “user@hostname” cannot uniquely identify a key.

3.7 Certification

After E_{new} obtains its name assignment from $E_{new}Naming$, it needs to obtain the certificate from *POP*. The certification procedure requires *POP* for name input, and *Certification Context (CertC)* as certificate issuer context. Moreover, the procedure needs *TrustAnchor* and *TrustSchema* to validate the issued certificate conforming to the trust schema of the domain. In our model, we describe certification as the $E_{new}Cert$ procedure:

$$E_{new}Cert : (POP, CertC, TrustAnchor, TrustSchema)$$

$$\rightarrow Certificate$$

Certificate Issuer: Inside a trust domain, either the domain controller itself, or another delegated entity, needs to be responsible for certificate issuance and revocation, as well as making them available by publishing to some repository [13]. That is, a certificate issuer is not necessarily the trust domain controller itself. During certificate issuance, the issuer validates *POP* and binds the contained name assignment to E_{new} ’s public key. Today’s CAs, such as Let’s Encrypt [1], issue domain-validated certificates (DV) following a similar logic. The domain validation process validates the requester’s proof of DNS domain possession (e.g. publishing a DNS TXT Record) by trusting the global routing system for delivering all validation requests to correct destinations. NDN certificate issuers rely on *POP* to provide authenticated name–entity binding. Later in Section 4, we show individual security bootstrapping protocols have their own *POP* realizations, such as session encryption keys and temporary certificates.

3.8 TEB Dataflow Graph

The previous subsections explained individual procedures in security bootstrapping. Now we show how the procedures of TEB work together to form a framework for the security bootstrapping process.

Figure 4 shows the dataflow graph of security bootstrapping, where *CAC* and *EAC* represent the pre-existing trust relation between the domain controller and E_{new} . When the security bootstrapping starts, *ContAuth* ❶ and $E_{new}Auth$ ❷ generate *POA* and *POM*, respectively. Then, $E_{new}Trust$ ❸ obtains *TrustAnchor* and *TrustSchema*. In parallel, $E_{new}Naming$ ❹ takes *POM* and *NameConv* as input to produce *POP*. As the final step, $E_{new}Cert$ ❺ utilizes *POP* and *CertC* to obtain a certificate, and validates it with *TrustAnchor* and *TrustSchema*.

Modularized Bootstrapping Design: Since the dataflow graph reveals the procedure dependencies in bootstrapping, it informs a new possibility in the bootstrapping protocol design. That is, the two-way authentication can be decoupled, so that one can authenticate and name E_{new} before E_{new}

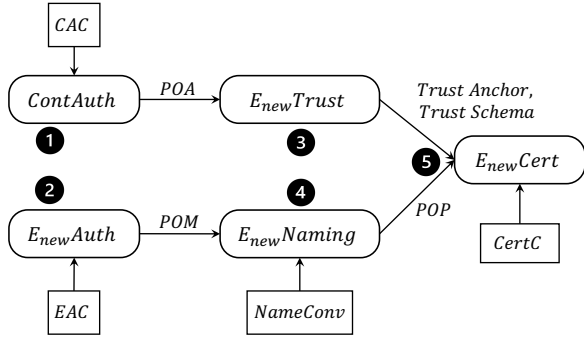


Figure 4: Dataflow Graph of the Bootstrapping Model

accepts the trust anchor and the initial trust schema. The certification can be simplified to be a procedure that binds E_{new} name assignment to a public-private key pair, which requires accomplishing authentication and naming a prior, rather than in real-time.

The dataflow graph also enables modularity in the protocol design and implementation. Because a valid security bootstrapping process is a set of procedures that follow the dataflow graph, developers can realize each procedure separately with any execution order that satisfies the TEB dataflow graph. Then a TEB implementation bootstraps E_{new} by executing individual procedures in the specified order. One can also summarize commonly used procedure implementations and plug them into various TEB-based bootstrapping designs. For example, an $E_{new}Auth$ implementation that authenticates E_{new} based on email addresses can be shared among multiple trust domain bootstrapping solutions to ease the protocol development.

4 EVALUATION

We evaluate our model by analyzing several security bootstrapping solutions.

SSP: SSP [6] is a security bootstrapping protocol aiming at smart home device. In SSP, smart home devices (*i.e.* E_{new}) pre-share a QR code to the home controller (*i.e.* trust domain controller). The QR code includes a public key, a symmetric key and the device identifier. In this case, EAC is the shared public key, symmetric key and device identifier, and CAC is the corresponding private key and symmetric key.

Firstly, the device initiates security bootstrapping by broadcasting a sign-on Interest packet (❶). The sign-on Interest carries the device identifier, device capability, and a nonce $N1$, as $E_{new}ID$, signed with the device’s private key. Before broadcasting the Interest, the device hashes the all the four parameters as a whole and appends it as the last name component. After receiving the sign-on Interest, the controller verifies the signature with the pre-shared public key (❷).

Upon successful verification, the controller replies an sign-on Data packet with the same name. The sign-on Data encapsulates $TrustAnchor$ (controller’s self-signed certificate), $TrustSchema$ (implicit), and nonce $N2$ as $ContID$, signed with the shared symmetric key. Because the sign-on Data name carries the same digest of parameters with the sign-on Interest, the data signature ensures the device membership in the smart home trust domain. Meanwhile the symmetrically signed data signature is verifiable to the device, thereby the sign-on Data packet is both POM and POA .

Secondly, the device receives and validates the sign-on Data. It installs $TrustAnchor$ and $TrustSchema$ from the sign-on Data content (❸) and performs ECDH between its private key with the controller’s public key. The temporary symmetric key obtained from ECDH is the $CertC$ used later.

Thirdly, the device broadcasts a certificate request Interest (❹). The certificate request Interest carries the device identifier, $TrustAnchor$ digest, $N1$, $N2$, and a signature from the device. Similar to the sign-on Interest, the device appends the digest of parameters to the name. Upon successful verification, the controller assigns a name to the device according to $NameConv$, generates a new key pair for the device, and certifies the device’s new public key under the name assignment. It also performs ECDH between the controller private key and the device pre-shared public key to negotiate a temporary symmetric key for private key encryption. Then the controller replies with a certificate Data carrying the same name. The certificate Data contains the device’s certificate and the encrypted device’s new private key and is signed by the shared symmetric key. As POP , its data signature authenticates the binding between $N1$ and the device name assignment.

Finally, the device validates the received certificate Data, decrypts the new private key with $CertC$, and installs the certificate (❺) to complete the security bootstrapping process. We summarize the SSP bootstrapping model in Table 1.

NDN Testbed: NDN Testbed distributes the trust anchor through NDN Website [7] and NDN CERT codebase [14] that implements the certificate issuance protocol of Testbed. Testbed users authenticate, obtain and install $TrustAnchor$ and $TrustSchema$ out-of-band (❶ ❸), then follows the NDN CERT [23, 27] protocol to request a certificate on Testbed. NDN Testbed bootstraps users via authenticating users’ email addresses and issuing certificates to users. In the bootstrapping process, a user runs the NDN CERT protocol and acts as a certificate requester.

Firstly, the certificate requester takes EAC to initiate user authentication (❷). EAC contains the user’s email address and the NDN Testbed Certificate Authority (CA) prefix. The requester sends a NEW Interest that carries an ECDH public key, the public key to be certified, and a signature generated

| Operations | SSP Devices (D) | SSP Controller (C) |
|---|--|--|
| Out-of-band (OOB) | Obtaining <i>CAC</i> | Obtaining <i>EAC</i> , <i>NameConv</i> |
| D \Rightarrow C: sign-on Interest | E_{newID} Generation | $E_{newAuth}(EAC) \rightarrow POM$ |
| D \Leftarrow C: sign-on Data | $E_{newTrust}(ContAuth(CAC)) \rightarrow$ <i>TrustAnchor</i> , <i>TrustSchema</i> , <i>CertC</i> Generation | <i>ContID</i> Generation |
| D \Rightarrow C: certificate request Interest | | $E_{newNaming}(POM, NameConv) \rightarrow POP$ |
| D \Leftarrow C: certificate Data | $E_{newCert}(POP, CertC, TrustAnchor, TrustSchema) \rightarrow$ <i>Certificate</i> | |

Table 1: SSP Security Bootstrapping Model

| Operations | NDN Testbed Users (U) | NDN Testbed CA (C) |
|--|---|--|
| OOB | <i>TrustAnchor</i> , <i>TrustSchema</i> installation | Obtaining <i>EAC</i> , <i>NameConv</i> |
| U \Leftarrow C: NEW Interest-Data | Obtaining E_{newID} | E_{newID} Generation |
| U \Leftarrow C: CHALLENGE Interest-Data (Round-trip 1) | Obtaining <i>POM</i> | $E_{newAuth}(EAC) \rightarrow POM$ |
| U \Leftarrow C: CHALLENGE Interest-Data (Round-trip 2) | Obtaining <i>POP</i> , <i>CertC</i> | $E_{newNaming}(POM, NameConv) \rightarrow POP$ |
| U \Leftarrow C: Certificate Interest-Data | $E_{newCert}(POP, CertC, TrustAnchor, TrustSchema) \rightarrow$ <i>Certificate</i> | |

Table 2: NDN Testbed Security Bootstrapping Model

by the corresponding private key. In reply, the CA sends a Data packet carrying *RequestID*, as E_{newID} , which is randomly picked to identify the certificate request instance. Note that both Interest and Data contain the public information used for the ECDH key agreement and are signed with the timestamp and nonce to prevent replay attack.

Secondly, the requester sends a CHALLENGE Interest 1 carrying *RequestID* and the user’s email address which is encrypted using the key negotiated from the NEW Interest-Data exchange. The CA obtains the email address from the CHALLENGE Interest 1 and decides whether it accepts this email address in Testbed. Afterward, it sends a randomly generated PIN to the email address and replies CHALLENGE Data 1. Similar to the NEW step, CHALLENGE Interest and Data packets are signed by the sender and verified by the receiver. As *POM*, the CHALLENGE Data 1 binds the *RequestID* with the CA signature. The Testbed user obtains the PIN out-of-band and inputs it to the requester.

Thirdly, the requester sends CHALLENGE Interest 2 carrying *RequestID* and the encrypted PIN. Upon successful PIN verification, the CA assigns a name, certifies it under the requester public key, and replies CHALLENGE Data 2 (4). As *POP*, CHALLENGE Data 2 contains *RequestID*, the encrypted certificate name, and the CA signature. Thereafter, the requester obtains and decrypts the *CertC* from *POP*. *CertC* consists of a certificate name and a forwarding hint to facilitate certificate retrieval.

As the final step, the requester expresses Interest with the certificate name and a forwarding hint to retrieve the issued

certificate to be installed locally (5). We summarize the NDN Testbed bootstrapping model in Table 2.

NDNViber: NDNViber[12] is an automated bootstrapping protocol for IoT devices. It uses the vibration channel in physical proximity to realize the two-way authentication. The vibration channel information (e.g. coding scheme) is the *CAC* on E_{new} side. NDNViber controller initiates the bootstrapping process by expressing a TRIGGER Interest in the vibration channel. The TRIGGER Interest includes the trust domain name and a temporary encryption key. Because of the secrecy of the vibration channel, the NDNViber device considers the TRIGGER Interest as *POA* (1) and replies with the pre-installed device identifier as *EAC*. When the controller receives the corresponding TRIGGER Data, it obtains the *EAC* for device authentication.

After the TRIGGER Interest-Data exchange, the device expresses ANCHOR Interest over the traditional channels and uses the temporary encryption key from *POA* to obtain *TrustAnchor* and *TrustSchema* (implicit) (3). Finally, the device runs the NDNCERT protocol to obtain *POP* and *Certificate*. The controller authenticates (2) and names the device based on *EAC* and *NameConv*, and issues certificates (5) to the device via the NDNCERT CHALLENGE Interest-Data exchanges. We omit the detailed analysis because of its similarity to the NDN Testbed $E_{newAuth}$, $E_{newNaming}$ and $E_{newCert}$. The main difference is that $E_{newAuth}$ is over the vibration channel. We model the protocol in Table 3.

| Operations | NDNViber Devices (D) | NDNViber Controller (C) |
|---|--|---|
| OOB | Obtaining EAC, CAC | Obtaining $NameConv$ |
| (Vibration Channel) D \leftrightarrow C: TRIGGER Interest-Data | $ContAuth(CAC) \rightarrow POA$ | Obtaining EAC |
| D \leftrightarrow C: ANCHOR Interest-Data | $E_{new}Trust(ContAuth(CAC)) \rightarrow$ $TrustAnchor, TrustSchema$ | |
| D \leftrightarrow C: NDN CERT Interest-Data (multiple rounds) | Obtaining $POM, POP, CertC$ | $E_{new}Auth(EAC) \rightarrow POM,$ $E_{new}Naming(POM, NameConv) \rightarrow POP$ |
| D \leftrightarrow C: Certificate Interest-Data | $E_{new}Cert(POP, CertC, TrustAnchor, TrustSchema) \rightarrow$ $Certificate$ | |

Table 3: NDNViber Security Bootstrapping Model

| Operations | PION Devices (D) | PION Authenticator (A) | Controller (C) |
|--|---|--|---------------------------------------|
| OOB | Obtaining CAC | Obtaining $EAC, NameConv$ | |
| A \leftrightarrow D: PAKE Interest-Data | | $E_{new}Auth(EAC) \rightarrow POM$ | |
| A \leftrightarrow D: CONFIRM Interest-Data | $ContAuth(CAC) \rightarrow POA,$ $E_{new}Trust(POA) \rightarrow TrustAnchor, TrustSchema,$ Obtaining $POM, CertC$ | | |
| A \leftrightarrow D: CREDENTIAL Interest-Data | | $E_{new}Naming(POM, NameConv) \rightarrow POP$ | |
| A \leftrightarrow D: Certificate Interest-Data | Obtaining POP | | |
| D \leftrightarrow C: NDN CERT Interest-Data (multiple rounds) | Obtaining $POP', CertC$ | | $E_{new}Cert_1(POP) \rightarrow POP'$ |
| D \leftrightarrow C: Certificate Interest-Data | $E_{new}Cert_2(POP', CertC, TrustAnchor, TrustSchema) \rightarrow$ $Certificate$ | | |

Table 4: PION Security Bootstrapping Model

PION: PION [10, 11] is a password-based security bootstrapping for IoT devices. Different from the aforementioned protocols, in PION, the controller delegates authentication and naming to an elemental entity called the PION authenticator. The PION authenticator authenticates and names E_{new} with a temporary certificate as POP , and E_{new} further uses the temporary certificate to apply the formal certificate from the controller. It realizes the two-way authentication through the first pre-shared password as both EAC and CAC , then derives a shared secret following the SPAKE2 [5] scheme to secure communications during bootstrapping.

As a trusted elemental entity, the PION authenticator initiates the bootstrapping process by sending PAKE Interest, which includes the authenticator SPAKE2 public share pA . The device receives PAKE Interest and processes pA based on the password, then replies PAKE Data which includes its public share pB and key confirmation message cB . When the authenticator receives this PAKE Data, the authenticator (i) processes pB , verifies cB , generates its confirmation message cA ; (ii) assigns SID as $E_{new}ID$, and names E_{new} according to the $NameConv$ obtained OOB; (iii) derives a temporary encryption key Ke from the previous SPAKE2 exchanges; (iv) sends a CONFIRM Interest as POM (2) that carries SID and Ke encrypted E_{new} name assignment. Upon successful cA verification, the device accepts the received CONFIRM Interest as POM and POA (1). It extracts the $TrustAnchor$ and

the $TrustSchema$ (implicit) from the POA (3), and replies a CONFIRM Data. The CONFIRM Data includes SID and Ke encrypted E_{new} self-signed certificate. The PION authenticator receives this Interest, signs a temporary E_{new} certificate as POP , and sends the Ke encrypted certificate name back to the device through a CREDENTIAL Interest (4). In order to obtain POP , the device uses Ke to decrypt the certificate name from CREDENTIAL Interest with, then fetch and temporary certificate.

Afterwards, the device follows the NDN CERT protocol and uses POP and $CertC$ to obtain the name of the formally issued certificate from the trust domain controller (5). In PION, Since the temporary certificate as POP already binds the controller's approval on E_{new} 's identifier to the name, we model the CHALLENGE Data of the NDN CERT exchanges in PION as POP' . The device first follows the NDN CERT protocol to obtain POP' and update $CertC$ to $CertC'$, then uses $CertC'$ to decrypt the certificate name from POP' and expresses Interest to fetch the formally issued certificate. We use the following two subprocedures to describe the $E_{new}Cert$.

$$\begin{aligned}
E_{new}Cert_1 &:(POP) \rightarrow POP' \\
E_{new}Cert_2 &:(POP', CertC, TrustAnchor, TrustSchema) \\
&\rightarrow Certificate
\end{aligned}$$

Table 4 shows the PION protocol analysis.

DCT: DCT [8] is a data-centric toolkit for secure NDN IoT applications. In order to bootstrap entities inside a DCT trust domain, developers specify the trust anchor and the trust schema for a trust domain, and generate *Identity Bundle* for each entity. The Identity Bundle contains the trust anchor, trust schema, certificate chain, and the corresponding private keys. Human operators bootstrap each entity by securely installing its Identity Bundle. Specifically, DCT suggests operators install bundles via command line within the development environment, and it leaves other installation mechanisms to individual trust domain deployment.

Although DCT bootstrapping does not involve explicit data communications, it still follows the TEB model. Securely installing bundles requires the trust domain controller and E_{new} mutually authenticating each other (❶ ❷). Then the parties can establish a secured channel in between and enable the controller securely transfer (e.g. command line) the bundle into E_{new} (❸ ❹ ❺).

Besides the above representative bootstrapping protocols, [16] proposes three cases of security bootstrapping. The first case assumes a secure environment between the trust domain controller and the new entity (E_{new}), which is similar to the DCT deployment scenario (Section 4). While the second case assumes the new entity is within the trust domain controller’s physical vicinity, where our analyses of SSP (Section 4) and NDNViber (Section 4) are applicable. The third case leverages existing authentication solutions in today’s Internet and NDN CERT for both authentication and certification. This case naturally fits into our analysis of the NDN Testbed bootstrapping model (Section 4). Therefore, TEB covers all three different networking scenarios in [16].

5 DISCUSSION

Having described the TEB design and evaluation, in this section we discuss TEB’s design decisions and the lessons learned from the evaluation.

5.1 Authentication and Certification

A certificate represents the trust domain controller’s endorsement of the binding between an entity’s name and its public key. In order to issue a certificate, the controller must (i) authenticate the public key owner and binds it to a name; and (ii) endorse the name–key binding through digital signature. If a domain supports multiple certificate issuers, these issuers can share the same *EAC* and *NameConv*.

TEB breaks the aforementioned two steps into three procedures: $E_{new}Auth$, $E_{new}Naming$, and $E_{new}Cert$. Since each procedure serves one specific purpose, their realizations are relatively simple. In this regard, PION separates the name authenticator from the certificate issuer. The authenticator

issues a temporary certificate that carries the authenticated name assignment. Consequently, the certificate issuer can be *EAC* and *NameConv* agnostic by re-signing a formal certificate to the public key owner.

5.2 Multi-Named Entities

TEB allows an entity to have multiple names within a trust domain, with each name uniquely identifying the entity. A typical scenario that may need multiple names for one entity is Name-based Access Control (NAC) [25]. In the example of Figure 2, the NDNFit controller can assign Alice two additional names “/ndnfit/alice/admin” and “/ndnfit/alice/customer”, each representing a different user role. When the controller specifies access control policies, it can let Alice request decryption keys based on role names.

To support multi-named entities, the controller assigns multiple names in $E_{new}Naming$. The certificate issuer of each name takes actions according to the trust schema and $E_{new}Cert$ realization. A possible approach is to certify “/ndnfit/alice” first, and then to certify the two additional names. We observed the same design pattern in DCT, where the DCT controller certifies devices and the devices further sign their capability certificates (e.g., light). Therefore, each DCT device may have multiple names, depending on the capability it equips with.

5.3 Trust Schema: Implicit vs. Explicit

As we mentioned earlier, the trust schema is a critical security component that must be obtained during bootstrapping. Designing the trust schema for a trust domain requires understanding both application semantics and security requirements.

TEB allows the initial trust schema to be implicit (Section 3.5), which provides a shortcut for the out-of-band configuration of the new entity before bootstrapping. However, since the implicit trust schema only allows data produced by the controlled to be trusted, the new entity cannot securely communicate with any other entity, until it learns how to validate data produced by others. In other words, distributing an explicit and finer-tuned trust schema after security bootstrapping is necessary, which leads to extra overheads.

We believe that the best practice is to explicitly specify an initial trust schema in $E_{new}Trust$. DCT achieves this by putting the certificate chain and the trust schema into the identity bundle. All entities in a trust domain are able to validate others’ data publication once obtaining the bundle.

6 CONCLUSION AND FUTURE WORK

NDN’s network model requires all named entities to establish trust relations with others. As explained in [16], security

bootstrapping fulfills this requirement by answering two basic questions: from where a new entity can obtain its name(s) and security credentials, and how the initial trust relations can be configured into the entity. However, our observations on recent NDN application development suggest that it is non-trivial for developers to understand the necessary procedures to perform security bootstrapping and properly implement entity bootstrapping operations.

To address the above issues, we proposed TEB to formally define the goal and procedures of security bootstrapping based on the concept of NDN trust domain. We evaluated TEB against various existing security bootstrapping solutions and show that those individual protocols all fit into the TEB model. Our experience so far gives us confidence in the model's generality in support security bootstrapping for most, if not all, application scenarios in a trust domain. We believe we have contributed a meaningful step towards a reusable approach in bootstrapping protocol designs: defining abstract variables, realizing logical procedures, and executing procedures based on functional dependencies.

Note that security bootstrapping is only the first step in securing data communications. One remaining research question is where to store these security components and how to automatically execute trust policies. Since TEB focuses on securing intra-domain data communications, another question is how to establish inter-domain trust relations. Communicating with semantically named and signed data empowers one to leverage naming conventions and automate security workflows. However, we need well-integrated and easy-to-use implementations that realize such automation, and hide security primitives from the application developers. As our future work, we plan to develop solutions to the two aforementioned questions, and provide a complete framework to realize the TEB model.

REFERENCES

- [1] Josh Aas, Richard Barnes, Benton Case, Zakir Durumeric, Peter Eckersley, Alan Flores-López, J Alex Halderman, Jacob Hoffman-Andrews, James Kasten, Eric Rescorla, et al. 2019. Let's Encrypt: an automated certificate authority to encrypt the entire web. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 2473–2487.
- [2] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk. 2008. *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*. RFC 5280. RFC Editor.
- [3] Ashlesh Gawande, Jeremy Clark, Damian Coomes, and Lan Wang. 2019. Decentralized and secure multimedia sharing application over named data networking. In *Proceedings of the 6th ACM Conference on Information-Centric Networking*. 19–29.
- [4] Van Jacobson, Diana K. Smetters, James D. Thornton, Michael F. Plass, Nicholas H. Briggs, and Rebecca L. Braynard. 2009. Networking Named Content. In *Proceedings of the 5th International Conference on Emerging Networking Experiments and Technologies (Rome, Italy) (CoNEXT '09)*. Association for Computing Machinery, New York, NY, USA, 1–12.
- [5] Watson Ladd and Benjamin Kaduk. 2022. *SPAKE2, a PAKE*. Internet-Draft draft-irtf-cfrg-spake2-26. Internet Engineering Task Force. Work in Progress.
- [6] Y. Li, Z. Zhang, X. Wang, E. Lu, D. Zhang, and L. Zhang. 2019. A Secure Sign-On Protocol for Smart Homes over Named Data Networking. *IEEE Communications Magazine* 57, 7 (July 2019), 62–68.
- [7] Named Data Networking Project. 2022. NDN Testbed. <https://named-data.net/ndn-testbed/>.
- [8] Kathleen Nichols. 2021. Trust Schemas and ICN: Key to Secure Home IoT (*ICN '21*). Association for Computing Machinery, New York, NY, USA, 12 pages.
- [9] Kathleen Nichols. 2021. Trust schemas and ICN: key to secure home IoT. In *Proceedings of the 8th ACM Conference on Information-Centric Networking*. 95–106.
- [10] Davide Pesavento, Junxiao Shi, Kerry McKay, and Lotfi Benmohamed. 2022. PION: Password-based IoT Onboarding Over Named Data Networking. In *2022 IEEE International Conference on Communications*. IEEE.
- [11] Pesavento, Davide and Shi, Junxiao. 2022. PION Specification. <https://github.com/usnistgov/PION/blob/main/docs/protocol.md>
- [12] Sanjeev Kaushik Ramani, Proyash Podder, and Alex Afanasyev. 2020. NDNViber: Vibration-Assisted Automated Bootstrapping of IoT Devices. In *2020 IEEE International Conference on Communications Workshops (ICC Workshops)*. IEEE, 1–6.
- [13] The NDN Team. 2022. NDN Python Repo. <https://github.com/UCLA-IRL/ndn-python-repo>
- [14] The NDN Team. 2022. NDN CERT Codebase. <https://github.com/named-data/ndncert>
- [15] Tatu Ylonen and Chris Lonvick. 2006. *The secure shell (SSH) transport layer protocol*. Technical Report. RFC 4253, January.
- [16] Tianyuan Yu, Philipp Moll, Zhiyi Zhang, Alexander Afanasyev, and Lixia Zhang. 2021. Enabling Plug-n-Play in Named Data Networking. In *MILCOM 2021-2021 IEEE Military Communications Conference (MILCOM)*. IEEE, 562–569.
- [17] Yingdi Yu, Alexander Afanasyev, David Clark, kc claffy, Van Jacobson, and Lixia Zhang. 2015. Schematizing Trust in Named Data Networking. In *Proceedings of the 2nd ACM Conference on Information-Centric Networking (San Francisco, California, USA) (ACM-ICN '15)*. Association for Computing Machinery, New York, NY, USA, 177–186.
- [18] Yingdi Yu, Alexander Afanasyev, Zhenkai Zhu, and Lixia Zhang. 2014. An endorsement-based key management system for decentralized NDN chat application. *University of California, Los Angeles, Tech. Rep. NDN-0023* (2014).
- [19] Yingdi Yu, A Afanasyev, Z Zhu, and L Zhang. 2014. Ndn technical memo: Naming conventions. *NDN, NDN Memo, Technical Report NDN-0023* (2014).
- [20] Haitao Zhang, Yanbiao Li, Zhiyi Zhang, Alexander Afanasyev, and Lixia Zhang. 2018. NDN Host Model. *SIGCOMM Comput. Commun. Rev.* 48, 3 (Sept. 2018), 35–41.
- [21] Haitao Zhang, Zhehao Wang, Christopher Scherb, Claudio Marxer, Jeff Burke, Lixia Zhang, and Christian Tschudin. 2016. Sharing mhealth data via named data networking. In *Proceedings of the 3rd ACM Conference on Information-Centric Networking*. 142–147.
- [22] Lixia Zhang, Alexander Afanasyev, Jeffrey Burke, Van Jacobson, kc claffy, Patrick Crowley, Christos Papadopoulos, Lan Wang, and Beichuan Zhang. 2014. Named Data Networking. *SIGCOMM Comput. Commun. Rev.* 44, 3 (July 2014), 66–73.
- [23] Zhiyi Zhang, Su Yong Wong, Junxiao Shi, Davide Pesavento, Alexander Afanasyev, and Lixia Zhang. 2020. On Certificate Management in Named Data Networking. *arXiv preprint arXiv:2009.09339* (2020).
- [24] Zhiyi Zhang, Tianyuan Yu, Xinyu Ma, Yu Guan, Philipp Moll, and Lixia Zhang. 2022. Sovereign: Self-contained Smart Home with Data-centric

- Network and Security. *IEEE Internet of Things Journal* (2022).
- [25] Zhiyi Zhang, Yingdi Yu, Sanjeev Kaushik Ramani, Alex Afanasyev, and Lixia Zhang. 2018. NAC: Automating access control via Named Data. In *MILCOM 2018-2018 IEEE Military Communications Conference (MILCOM)*. IEEE, 626–633.
- [26] Zhiyi Zhang, Yingdi Yu, Haitao Zhang, Eric Newberry, Spyridon Mastorakis, Yanbiao Li, Alexander Afanasyev, and Lixia Zhang. 2018. An overview of security support in named data networking. *IEEE Communications Magazine* 56, 11 (2018), 62–68.
- [27] Zhang, Zhiyi and Shi, Junxiao and Pesavento, Davide. 2022. NDN CERT v0.3 Specification. <https://github.com/named-data/ndncert/wiki/NDNCERT-Protocol-0.3>