Enabling Plug-n-Play in Named Data Networking

Tianyuan Yu
UCLA, Computer Science
tianyuan@cs.ucla.edu

Philipp Moll
UCLA, Computer Science
phmoll@cs.ucla.edu

Zhiyi Zhang
UCLA, Computer Science
zhiyi@cs.ucla.edu

Alexander Afanasyev
Florida International University
aa@cs.fiu.edu

Lixia Zhang
UCLA, Computer Science
lixia@cs.ucla.edu

***Abstract*—"Plug-and-play" is a highly desired property in networking, which enables new entities to be *plugged* into a networked system following a systematic, and automated if possible, process to start *playing*, i.e. sending and receiving packets. In IP networks, DHCP services provide the plug function to enable an IP host to play. In this paper we discuss the plug step in an NDN network, articulate the fundamental differences in NDN's plug step as compared to that of IP, and describe our initial designs for plugging new entities into an NDN network in three different use case scenarios. These design exercises show that NDN's plug process requires mutual authentication between the *configurer* and the *configuree* (the entity to be plugged in), which is context specific and represents a major challenge in the plug process. We addressed this challenge by making use of existing authentication systems.**

***Index Terms*—Named Data Networking, Security, Configuration, Security Bootstrapping**

I. Introduction

Rapid computing and communication technology advances have fueled networked applications, especially those running on mobile devices communicating over wireless connectivities. Communicating entities at wireless edge require strong security support, including data integrity and authenticity, when interacting with ad hoc encounters. They also require resilient data delivery over dynamic, intermittent connectivities.

Many people view Named Data Networking (NDN) [1] as representing a promising direction to meet the above requirements by its direct use of application data names in communication, thus eliminating the need of translating DNS names to IP addresses; by its built-in security support that secures named data objects directly, thus removing the dependency on intermediate channels' security; and by its data-centric design that supports in-network caching and delay/disruption tolerant communications. All the above desired NDN functions are direct results of NDN's network model: a networked system is made of named entities with various trust relations among each other [2], where these named entities can be devices, servers/services, app instances, or anything that produce and/or consume named packets. Since the names of these entities are decoupled from their specific attachment points to the network in general, they can explore any available connectivities to communicate.

This work is partially supported by the National Science Foundation under award 1719403.

The above picture differs fundamentally from IP's view that a network is made of interconnected nodes identified by IP addresses, with no security relations among IP nodes at the IP layer. However NDN's view on networking also raises two questions that must be answered before an entity can be deployed in an NDN network: *where* an entity obtains its name(s) and security credentials, and *how* the initial trust relations are configured into the entity. An NDN entity can be "plugged" into the network once these two questions are answered, enabling it to start playing. Unfortunately, although previous works [3], [4] identified the necessary trust credentials and policies each entity should possess in order to play in an NDN network, they do not address the question of *how* those parameters can be securely installed into new entities. Furthermore, all authentication means during trust relationship establishments rely on existing authentication systems and/or trust relations. We need to deepen the understanding of how to make best use of such existing systems in NDN deployments.

In this work, we filled in that void by making the following contributions. First, we clarify the differences between the goal and process in plugging an IP node into IP networks and plugging an NDN entity into NDN networks. Second, we developed initial solutions in plugging new entities into an NDN network under three different scenarios, which serve as a learning exercise to deepen our understanding of the design space for trust relationship establishments.

In the rest of this report, Section II explains the plug step in TCP/IP networks, and how NDN's plug step is a fundamentally different process; Section III describes the steps of NDN entity configuration (*i.e.* the "plug") to enable secure communication (*i.e.* the "play"); Section IV proposes entity configuration designs that leverage trust relations in three different scenarios, which is followed by the discussions on limitations on establishing trust relations among networked entities in Section V. We conclude our work and discuss future work in Section VI.

II. Plug-n-Play in TCP/IP Networks

According to the Oxford English Dictionary [5], plug-n-play characterizes the ease of installation or use of something. In this section, we first describe the meaning of "plug-n-play" in today's IP networks. Thereafter, we give a brief introduction on the basic concepts, and pieces of solutions, that have been developed over the years with regard to adding new entities into an NDN network.

A. *Plug-n-Play in TCP/IP Networks*

In the following, we use the example of connecting a TCP/IP enabled computer H to a wired IP network via Ethernet to show the required step happening without user's awareness.

After one plugs an Ethernet cable into H, the goal of IP configuration is to install into H a few necessary pieces of parameters to enable it to send and receive IP packets. Because IP's network model is collections of subnets interconnected by routers (also called gateways), the three necessary pieces of parameters to be installed include (i) an IP address, out of the address block assigned to the Ethernet H connects to; (ii) a subnet mask that defines the subnet address block size; and (iii) the IP address of default router which connects the Ethernet to the rest of the Internet. Obtaining the above enables H to send and receive IP packets.

The deployment of DHCP services automated the above configurations. However, DHCP server themselves require manual configuration: network operators assign IP address blocks and configure subnet masks for each subnet, DHCP automates only the last step of configuring individual hosts in a long process of IP address space management by network operators, that are hidden from end users who are only concerned with end host plug-in's.

We also note that, IP connectivity between all IP nodes, established during a host's plug step, *does not* directly enable applications running on different hosts to communicate with each other. Applications communicate via DNS names instead of IP addresses, and require authentications of remote parties. To run applications over established IP connectivity require additional services to map DNS names to IP addresses, and to secure the communication channels by establishing trust relations between communicating TCP/IP nodes, which in turn requires a public key infrastructure. These additional infrastructure services, DNS and CA services, bridge the big gap between IP's (unsecured) connectivity and applications' needs for secure communication with named entities. To be able to make use of DNS services, DHCP adds a fourth necessary piece of information into an IP host H's plug process, the IP address of a local DNS resolver, that H can send all DNS queries to.

On the other hand, to be able to make use of the Certificate Authorities (CA) services, the *de facto* PKI of today's Internet, represents a much more difficult problem, because this requires the configuration of a set of trusted certificates (*trust anchors*), which can be used to inform all the applications running on H of which web services to trust. That is, this is about configuring trust relations with *all* web services. At this time, this important decision of selecting the trust anchors is done

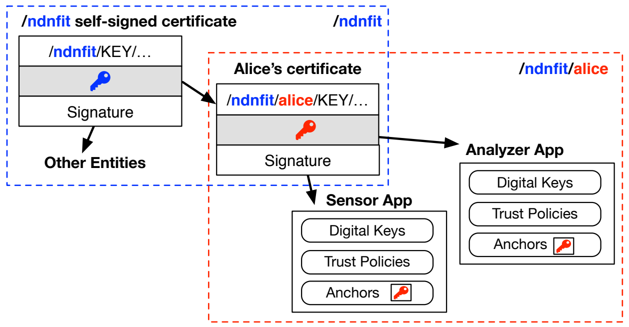


Fig. 1: The relationship between the namespaces /ndnfit and /ndnfit/alice, and between /ndnfit/alice and its sub-namespaces.

in a rather informal way without the users' awareness (more discussions in Section V).

B. *Plug-n-Play in NDN Networks*

By its make, Named Data Networking (NDN) uses names to communicate. NDN does not create any new namespace by itself; instead it directly use the existing application names for networking. NDN entities, as we defined earlier, produce and/or consume semantically named data. To enable these entities to communicate securely across any available connectivity, we need to securely install the name, trust anchor, certificate, as well as trust policies into each entity. Below we explain *where* are the sources of the above information; the next section will discuss *how* to securely install the trust information into an entity. The trust relations lay the foundation for secure networking.

Entity Naming Today's Internet applications use DNS names, so are NDN-based applications. Below we use an example to explain how NDN makes use of application names. Let us assume that an NDN based fitness application (e.g. as the one described in [6]) obtains a DNS name "`ndnfit.org`" (shortened to "`ndnfit`" for brevity), Figure 1 shows that *ndnfit* can then assign names to individual users in this application under its own namespace, e.g. "`/ndnfit/alice`" and "`/ndnfit /bob`" [1]. User Alice can further delegate names to her own devices/applications that play part in *ndnfit*, e.g. "`/ndnfit /alice/sensor`", "`/ndnfit/alice/analyser`".

Trust Anchor and Certificates As we mentioned earlier, NDN views a networked system as made of named entities with various trust relations among each other. In our example, *ndnfit* establishes the trust relations with the entities under its namespace[2]. This can be accomplished as follows. First, *ndnfit* creates a self-signed certificate as the trust anchor for this application. Second, *ndnfit* installs its trust anchor into all its users (e.g. */ndnfit/alice*, */ndnfit/bob*, etc.), and generates a certificate for each of them. In this way, Alice and Bob obtain their names and certificates, as well as the application-wide trust anchor, which enable them to produce authenticatable

[1]As a convention, we use typewriter font "`ndnfit`" to indicate the name, and italic font *ndnfit* to mean the application.

[2]Broadly speaking, *ndnfit* may also establishes trust relations with entities external to *ndnfit*, which is beyond the scope of this illustrative example.

data and verify received data that are produced by other entities running the same application.

The above process can be repeated. For example, in order to enable secure communication among her own apps */ndnfit/alice/sensor* and */ndnfit/alice/analyser*, user Alice installs her own trust anchor into her apps and issues a certificate to each of them. This enables Alice, not the *ndnfit* app, to be the whole controller of her own apps.

Formally, [4] defines that each entity in an NDN network is identified by its identity, which is made of a tuple $\langle$name, certificate$\rangle$. Furthermore, any entity *can* create a trust anchor T and install it into all other entities under its direct control. T is called a *local trust anchor*. In our example, Alice creates a local trust anchor T_{Alice}, and can tell all the entities which have T_{Alice} as their trust anchor how to communicate securely. That is, Alice defines the rules which inform each of her entities which keys, for a given data name or name prefix, should be used for signature generation and verification.

Trust Policies NDN trust policies are specified by using *trust schemas* [7], [8], which limit the power of each signing key to Data packets with specific names, supporting data authenticity with fine granularity. To help the reader gain an intuitive understanding of how NDN security policies work, we borrow a simple example from [4] to illustrate below.

Alice configures the trust policies for *Sensor* and *Analyzer* and installs the policies during their security bootstrapping process. Let us assume that Alice defines Policy-1 for her *Sensor* that it can only accept Data packets by an entity whose name contains the name prefix "`/ndnfit/Alice`", and the signing key's certificate chain must end with the trust anchor T_{Alice}. Accordingly, data produced by another *ndnfit* user Bob will not be accepted by *Sensor*. If Alice defines Policy-2 for her *Analyzer* that it can accept data produced by any *ndnfit* users (each should have a name starting with the prefix "`/ndnfit`"), and the signing key's certificate chain ends with the trust anchor T_{ndnfit}, this will allow *Analyzer* to accept Bob's data, assuming that Bob (or Bob controlled entities) signs all the data to be shared with other users using a certificate chain that ends with T_{ndnfit}, and not the trust anchor T_{Bob}.

As one can see from the above example, the trust policy definitions directly use NDN's semantic names to define the data authentication rules. Therefore the ease of these policy definitions can be impacted by the structure of data names and key names. As we will explain in Section III, Section III-B, NDN applications can define naming conventions to be used to simplify the policy definitions, as well as to facilitate discoveries.

C. Establishing Connectivity

Once an entity obtains its identity (name with certificate), trust anchor, and trust policies, it will be able communicate with other entities (that it is allowed to) through any available connectivities. Decades of IP deployment experiences with IP's plug-n-play influenced people's thinking about bootstrapping, earlier efforts in adding new entities to an NDN network largely focused on how to establishing connectivities for new entities. As a result, multiple solutions have already been developed to assist a newly plugged entity E_{new} to establish connectivity with other existing NDN entities. We enumerate a few of the solutions below.

To discover local NDN neighbors, E_{new} may simply broadcast Interests to all its available network interfaces to see whether the requested data can be retrieved from any nearby NDN neighbors. E_{new} may also use Self Learning [9], which will securely establish the forwarding path between E_{new} and the producer of E_{new}'s desired data.

If E_{new} has no physically connected NDN neighbors, it may use NDN Neighbor Discovery protocol (NDND) [10]. NDND sets up a server as the rendezvous point for all the NDN entities who are configured to look up the same NDND server for neighbor discovery, e.g. all NDN devices on the same campus, or different instances of the same app. These entities can then establish NDN connectivity among each other over TCP or UDP tunnels. E_{new} may also use ndn-autoconfig [11] to discover an NDN testbed router R, enabling it to use R as the default router to forward all the Interests, assuming the NDN testbed should know how to reach all the producers. If E_{new} wants others to fetch data it produces, it must announce its data's prefix to R, which will verify E_{new}'s security credential it obtains from the bootstrapping process. R may then announce E_{new}'s data prefix to the rest of the testbed through the testbed NDN routing protocol NLSR [12].

In general, NDN connectivity establishment is not a "once and for all" step. Instead, it can be a continuous process, as the existing connectivity may fail. Whenever that happens, each entity can utilize a combination of the above mentioned solutions to re-establish connectivity with others. They can take the simple broadcast approach to discover new encounters, and securely communicate with them by using the configured security credentials and policies.

III. Plugging entities into NDN networks

As we described in the last section, each entity in an NDN network is identified by its identity, which is made of a tuple $\langle$name, certificate$\rangle$; furthermore, any entity *can* create a trust anchor T and install it into all other entities under its direct control. We call T a local trust anchor. Together with semantic naming and trust policies, they make the three pillars that NDN security is built on.

In this work, we further define that all the entities under the same trust anchor T make a *trust zone*[3], and we call the owner of the trust anchor T the *controller* of this trust zone[4]. Plugging a new entity E_{new} into an NDN network translates to configuring E_{new} into a trust zone. More specifically, the

[3]We adopt the term *trust zone* from [8] and give it a simple and precise definition as stated above.

[4]Note that a trust zone is defined by the collection of entities sharing the same trust anchor. By definition, these entities do not necessarily share the same name prefixes, as in our illustrative example. As we explain later, NDN makes heavy use of trust policies which operate on the names, and policies are defined by the zone controller. As such, the names with different prefixes in the same trust zone may add complexity to the policy definitions.

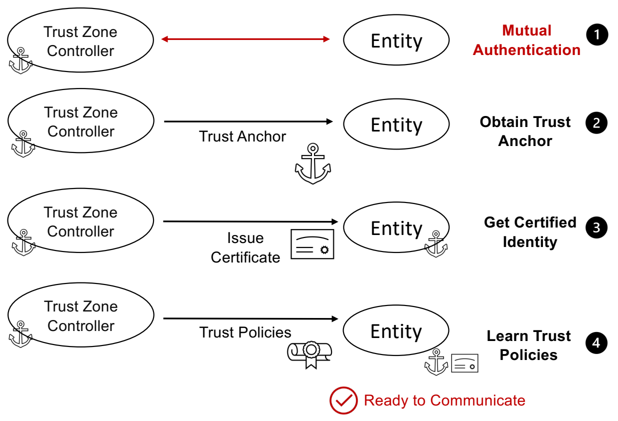


Fig. 2: Logical steps of security bootstrapping

zone's controller installs the trust anchor into E_{new}. E_{new} must have a name, and is further installed with its certificate and the trust policies it should follow, as we explained earlier.

A. Security Bootstrapping

In this section, we explain the logical steps of security bootstrapping in our design. As shown in Figure 2, the *trust zone controller* configures a new entity E_{new} (represented by "Entity" in the figure) under its namespace, while E_{new} accepts security components from it.

The first step of security bootstrapping is to achieve mutual authentication (①) between the trust zone controller and E_{new}. The trust zone controller must authenticate E_{new} to confirm that it truly represents the entity it wishes to bootstrap. At the same time, E_{new} needs to authenticate the trust zone controller to be its authority. As we show in Section IV, such mutual authentications rely on pre-existing trust relations.

Once the mutual authentication is accomplished, trust anchor is installed into E_{new} (②), establishing the trust relation between E_{new} and the controller. The installed trust anchor enables E_{new} to validate the receive identity certificate issued by the trust zone controller (③).

As the last step in the security bootstrapping, E_{new} fetches the trust policies that the controller has defined for it (④). The trust zone controller may change these policies from time to time. Since the policies can be encoded into NDN data packets, named by defined naming conventions and secured by the controller's key, the entity can easily fetch new policies securely in the same way as fetching any other types of data.

Finally, we note that, although Figure 2 shows four distinctive logical steps in bootstrapping, steps 2-4 all install the information from the controller into the entity.

B. Naming Conventions

In Section II-B, we mentioned that NDN applications can design their namespace to have desired structures in order to make the trust policy definitions as simple as possible. Further, in order to request Data, NDN entities also need to know how desired data pieces are named. For these reasons, NDN applications can define a set of naming rules, called *naming conventions* [13], to facilitate data retrieval process, especially at the start-up time.

Identities are named under the system root prefix of trust (*e.g.*"`/ndnfit`") and further distinguished by the assigned mission and entity identifiers.

Identity Name = "`/<system-prefix>/<entity-id>`"

As shown in the Figure 1 from Section II, an NDNFit end-user Alice has the identity "`/ndnfit/alice`", with the system prefix be "ndnfit" and entity-id be "alice".

Certificate names follow the prefix of identities and use the keyword "`KEY`" to separate the certificate information and the certified identity name.

Certificate Name = "`/<system-prefix/<entity-id>/KEY /<key-id>/<issuer-info>/<cert-version>`"

The "`key-id`" identifies the key pair binded to this certificate versioned by "`cert-version`" with "`issuer-info`" revealing the certificate signer. For instance, Alice's certificate is named as "`/ndnfit/alice/KEY/001/ndnfit-agent/v1`". This name indicates the certificate is issued by an ndnfit application agent with the key-id "001" and version "v1".

Trust Policies Data are also named under the identity prefix and have the keyword "`POLICY`" that separates the prefix and the policy version:

Trust Policies Name = "`/<system-prefix>/<entity-id> /POLICY/<policy-version>`"

The trust policy Alice's configuree installs can have the name "`/ndnfit/alice/POLICY/v1`", with "v1" be the policy version.

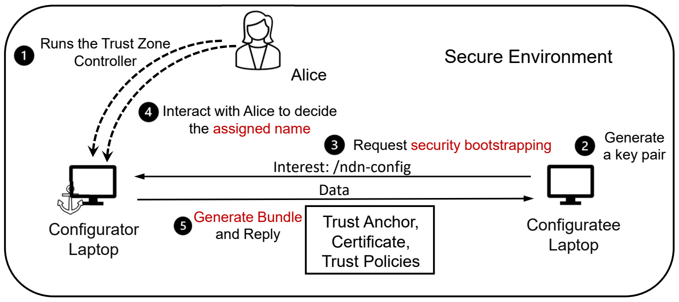
C. Existing Tools in Realizing the Configuration Process

A few pieces of tools have been developed over the years that can be utilized in realizing some of the steps in the configuration process mentioned above. This work can make use of these existing tools.

NDNCERT [14] defines a protocol exchange for certificate issuance, which requires an out-of-band "name ownership challenge" to authenticate E_{new}. Since our plug design starts with performing the mutual authentication, after that step we can make use of NDNCERT to issue certificates.

Another existing NDN security tool, ndnsec [15], installs obtained trust anchors and identity keys into the local system's keychain. Our work also makes use of this tool to save E_{new}'s trust anchor and signing keys securely.

DCT [16] provides a set of tools that enable trust zone controllers defining the trust anchor, certificate signing chain, and trust schemas with a domain-specific language called VerSec. Controller can bundle these security components together into an Identity Bundle, and install into E_{new} out-of-band. Our designs in Section IV can leverage DCT to bootstrap E_{new} with Identity Bundle after mutual authentication is achieved.



1 Runs the Trust Zone Controller
Alice
Secure Environment
4 Interact with Alice to decide the assigned name
3 Request security bootstrapping
2 Generate a key pair
Interest: /ndn-config
Data
5 Generate Bundle and Reply
Trust Anchor, Certificate, Trust Policies
Configurator Laptop
Configuratee Laptop

Fig. 3: Setting up trust zone controller on the configurator laptop, and bootstrapping the chatroom app on the configuratee laptop

IV. Exploring Design Space in Security Bootstrapping

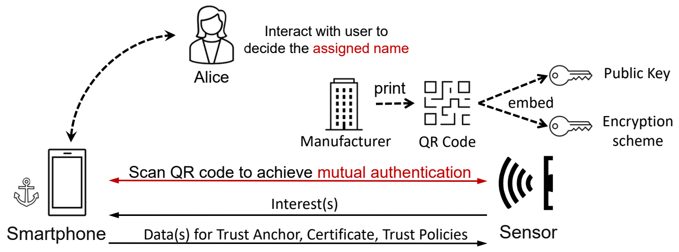
In this section, we develop a specific plug process for each of the following three different use case scenarios: installing and configuring an application in a local secure environment, configuring a device, which has its app installed already, with physical vicinity, and configuring an NDN application running on a remote host. These three different scenarios require different approaches to accomplish the mutual authentication between the trust zone controller and a new entity E_{new} to be bootstrapped.

A. Bootstrapping New Entities in a Secure Environment

In this use case scenario, we assume that a trust zone controller configures E_{new} to be under its control over a direct physical connectivity in an isolated environment, i.e. no third party can communicate with either of them. That is, the network environment is physically secured, which offers the two parties the mutual authentication: when one receives an input, it can only be from the other intended party.

To illustrate with a specific example, let us assume that two laptops, both are owned by user Alice, are connected via an isolated Ethernet switch, with no third party connected to the same Ethernet. We call one laptop *configurator* laptop and the other *configuratee* laptop. Alice installs a chatroom application, referred to as E_{new}, into the configuratee laptop, and needs to perform security bootstrapping for that app. In order to do so, Alice runs a trust zone controller on the configurator laptop (①), and uses it to bootstrap the chatroom app on the configuratee laptop. Since this isolated Ethernet can be considered as a secured environment, the mutual authentication between the trust zone controller and chatroom app is achieved: one party's input must be from, and can only be from, the other party.

As the preparation for bootstrapping, the chatroom app generates a key pair on the configuratee laptop (②). Then it initiates the security bootstrapping by sending an NDN Interest packet with a pre-defined "`/ndn-config`" (naming convention) (③). This Interest also carries the app's public key (it can be carried in the "application parameter" field in an Interest packet [17]). The trust zone controller on the configurator laptop receives this Interest, and asks Alice to assign this new NDN entity a name under the system prefix "`/alice`" (④). With the knowledge of E_{new} name (*e.g.*"`/alice/chatapp`") from Alice, the trust zone controller generates the certificate for the app. Afterwards, the trust zone controller can bundle the trust anchor, the newly issued certificate, and the trust polices defined for this specific app into one data object as the response to the configuration Interest (⑤)[5]. After the chatroom app installs the security components inside the replied data object, it has been successfully bootstrapped.



Alice
Interact with user to decide the assigned name
print
Manufacturer
QR Code
embed
Public Key
Encryption scheme
Scan QR code to achieve mutual authentication
Interest(s)
Smartphone
Data(s) for Trust Anchor, Certificate, Trust Policies
Sensor

Fig. 4: Achieving mutual authentication between sensor and smartphone

B. Bootstrapping Entities within Physical Vicinity

In this second use case scenario, we consider a case where the trust zone controller bootstraps an E_{new} within its physical vicinity. However the configuration process may not be performed in a sealed communication environment as in the first use case. Although user Alice owns both the trust zone controller and E_{new}, if they communicate over wireless channels, potential third parties can also communicate within the physically reachable area.

Therefore, the approach of obtaining mutual authentication by secured physical connectivity does not apply. In this case, the trust zone controller and E_{new} must authenticate each other via other available means.

To aid the reader's comprehension, we use the *ndnfit* app as an illustrative example to describe how our design works. We assume that Alice's smartphone and her sensor are within one-hop wireless communication range of each other. Alice has the ndnfit application installed on her smartphone, and wants to configure the time-location collecting application, which is built into the sensor. Her smartphone is the trust zone controller under the namespace (*e.g.*"`/ndnfit/alice`"), and holds the local trust anchor (*e.g.*"`/ndnfit/alice/KEY/...`") and the corresponding private key. The first task of Alice's smartphone is to authenticate the sensor, so that it can issue the sensor a certificate under "`/ndnfit/alice`". Meanwhile the sensor application also needs to ensure that Alice's smartphone is indeed its authority.

Alice's smartphone and her sensor can accomplish mutual authentication based on physical vicinity and by using the sensor's built-in QR code. As shown in Figure 4, the manufacturer

[5]If the data object is larger than one NDN Data packet, it will be automatically segmented into multiple Data packets to be delivered to the app via the standard NDN approach.

prints a QR code on the sensor, and the QR code contains the sensor's public key. Assuming Alice holds a trust on the manufacturer, she uses her smartphone to authenticate the sensor by scanning the QR code[6], and the sensor application authenticates Alice's smartphone once the phone exhibits the possession of the information encoded in the sensor's QR code.

Once the sensor's authenticity is confirmed, Alice's smartphone asks her to determine a name for the sensor under the trust zone "`/ndnfit/alice`" [7]. Then it can issue a certificate to the assigned name and specify corresponding trust policies. Afterwards, Alice's smartphone can securely (*e.g.*, with encryption schemes embedded in the QR code [18]) transfer the trust anchor, the newly issued certificate, and the trust policies through encrypted communication channel, in a way as if the communication were in a secured communication environment.

There also exist other authentication means, such as vibration [19], button pressing following a defined pattern, or short-range connectivity like NFC, that can be utilized for mutual authentication of devices within physical vicinity.

C. Bootstrapping Remote Entities via Existing Authentications

In addition to the need of bootstrapping new entities in local environments, another common use case scenario is to bootstrap remote NDN entities, that the trust zone controller can reach only over TCP/IP connectivity. Therefore, the solutions developed for the first two use cases do not apply. In order to achieve mutual authentication between a trust zone controller and an E_{new} connected over the Internet, we look into the direction of leveraging the trust relations and authentication solutions that already exist in today's Internet.

Indeed, a number of authentication systems already exist today. One is the widely used Certificate Authority system (CAs) that we mentioned in Section II-A. Another one is DNS with Security Extensions (DNSSEC [20]).

Any of the above solutions can be used to authenticate a remote entity E_{new}, say an NDN app, which needs to be bootstrapped, once the user who performs the bootstrapping learns the identifier of E_{new}'s host H, *e.g.* H's DNS name. However, we also need a means to have E_{new} authenticate the remote trust zone controller, so that E_{new} can accept the trust zone controller's inputs. Following the current software installation practice, we assume that a user can install a new application E_{new} into host H, with the app package containing a trust anchor for that application. The authenticity of this process is assured by today's web service security support such as git. We argue that the above assumptions accomplish the goal of E_{new} authenticating the app package.

[6]The manufacturer can also encode sensor's hardware profile, which includes sensor's series number, in the QR code. In this situation, Alice's smartphone can query the manufacturer whether the sensor is a valid product made by it.

[7]Otherwise Alice's smartphone can assign a name to the sensor by default, based on the sensor's brand and category. This information can be obtained from the hardware profile.

We apply the above solution direction to develop the bootstrapping solution for a distributed, federated storage system, dubbed *Hydra*, which is under actively development. Designed as a federated system, Hydra expects its user community made of different organizations to contribute storage servers, which can then collectively provide a high volume, distributed data repository.

To join this Hydra system, Hydra participating campuses (*e.g.* UCLA) need to contribute file servers. We assume that each contributed file server has an assigned host identifier, *e.g. bruins.cs.ucla.edu*, under the campus domain name. To authentiate the identifier, the CA of campus generate a SSL/TLS certificate for the identifier *bruins.cs.ucla.edu*. The file server securely obtains the SSL certificate from the operators. The Hydra Network Operating Center (Hydra NOC) will serve as the trust zone controller for "`/hydra`" and perform security bootstrapping for Hydra app on each contributed file server H_f under the Hydra namespace. Hydra NOC operators trusts CAs out-of-band for each contributing campus, so that the Hydra NOC can verify the trustworthiness of the Hydra app by verifying corresponding file server's SSL certificate.

Specifically in this example, we assume user Alice wants to contribute a UCLA server to the Hydra system. Therefore, Alice installs the Hydra app on the file server, and bootstraps the Hydra app using the SSL certificate.

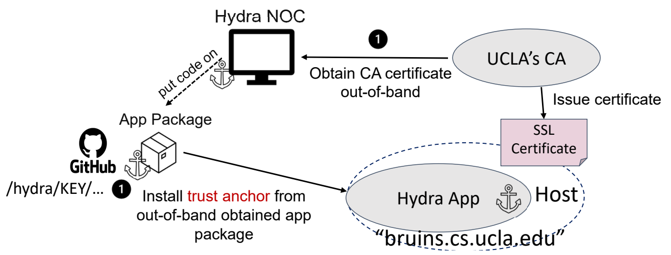


Fig. 5: Achieving mutual authentication between Hydra app and Hydra NOC

Figure 5 illustrates the process of accomplishing the mutual authentication. The trust zone controller part of mutual authentication (①) is achieved through software distribution. Hydra NOC embeds its trust anchor into the application package, and Alice downloads the application package manually from trusted sources. For instance, Alice can learn the git URL of the Hydra application package through the Hydra project communication channel, and securely download it (see more discussions in Section V). As a result, the trust anchor "`/hydra /KEY/...`" is installed into the file server at the app installation time.

To authenticate E_{new}, a remote Hydra app instance (①), Hydra NOC operators obtain the certificate of UCLA's CA out-of-band. Because UCLA's CA authenticates the file server's DNS name in the SSL certificate, and the Hydra app is securely installed on the file server, Hydra NOC can view the app is authenticated when it authenticates the identifier of the file server signed by the known CAs. Therefore, the mutual authentication is accomplished.

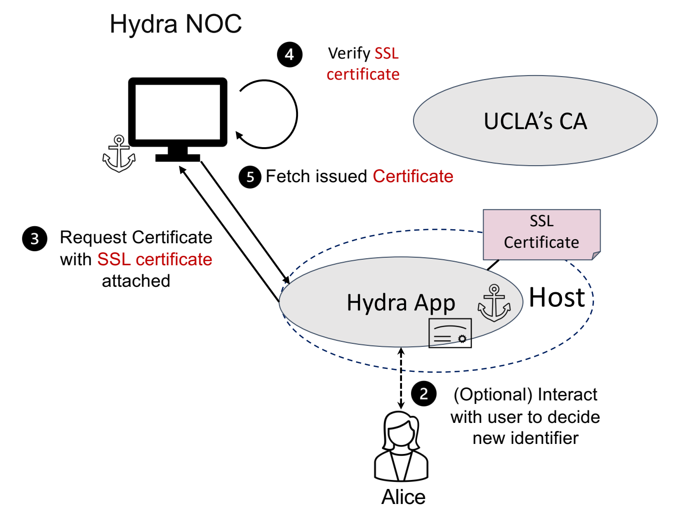

Hydra NOC
4 Verify SSL certificate
UCLA's CA
5 Fetch issued Certificate
3 Request Certificate with SSL certificate attached
SSL Certificate
Hydra App
Host
2 (Optional) Interact with user to decide new identifier
Alice

Fig. 6: Hydra app requests a certificate from the Hydra NOC

On the Hydra app side, it prepares for self-obtaining a name, and applying certificate for the name after Alice starts the app on the file server. Because Hydra app learns the system prefix "`/hydra`" from the trust anchor, which is obtained from the app package at installation time, it loads the SSL certificate for identifier "`bruins.cs.ucla.edu`" to generate an identity name "`/hydra/bruins.cs.ucla.edu`" based on pre-defined naming conventions[8]. Afterwards, it constructs the name of its certificate to apply "`/hydra/bruins.cs.ucla.edu /KEY/...`" from the identity name.

As shown in Figure 6, the Hydra app generates a key pair, and self-signs public key as a certificate signing request for "`/hydra/bruins.cs.ucla.edu/KEY/...`". Then it initiates the certificate issuing process by following the NDNCERT protocol. It sends an Interest with name "`/hydra/CA/NEW/...`" and having the certificate request attached. The response to this Interest brings back *identity verification chanllenge* from the Hydra NOC.

The Hydra app uses the "Proof of SSL Certificate Possession" as response to the authentication challenge, and sends a second Interest "`/hydra/CA/CHALLENGE/...`" with the SSL certificate attached as the challenge material. Hydra NOC processes this Interest, and verifies the SSL certificate with the CA root certificate previously obtained out-of-band [9]. A successful signature verification confirms the remote Hydra app's authenticity. Then Hydra NOC issues the app an NDN certificate with the name "`/hydra/bruins.cs.ucla.edu/KEY /...`" and sends it in the reply to this second Interest from the remote Hydra app. The NDNCERT protocol ensures the security of the above communication process.

Learning the certificate name from Hydra NOC's reply, Hydra app fetches its identity certificate from Hydra NOC and expresses Interest "`/hydra/bruins.cs.ucla.edu/POLICY`" to learn its trust policies. After this last step in the bootstrapping process, the Hydra app on the file server is plugged into the "`/hydra`" system and ready to communicate securely with all the other entities in the same system.

[8]Alternatively, Alice can also enter a new identifier for the Hydra app in this process.

[9]In order to prove the certificate possession, Hydra NOC will ask the Hydra app signing a nonce with the corresponding private key.

V. Discussion

While examing the designs under different scenarios, we make the following observations on design considerations.

A. Authentication Based on Existing Trust Relations

As discussed in Section III, the first logical step of NDN bootstrapping is to achieve mutual authentication. In each of the three cases we investigated, the trust zone controller and E_{new} take *different* approaches to authenticate each other. However, we believe that they share one commonality: they can all be viewed as authentication via some trust relation that already exists.

In Section IV-A, the E_{new} authentication is naturally achieved by the trust in a physically sealed communication environment. In Section IV-B, where a QR code is used to accomplish authentication, the user has to trust the sensor manufacturer for the purpose of deriving the trust to its manufactured product, and the device (a sensor) trusts the physical possession by the user. In the third example (Section IV-C), where the SSL certificate is used to authenticate the remote Hydra app, this authentication relies on the Hydra NOC trusting the campus's CA out-of-band, which already verified the file server that the Hydra app runs on.

Applications implemented upon the TCP/IP architecture utilize the same pre-established trust relations to achieve security. Today's practice of registering new accounts to websites, which typically includes verifying the user's identity by email addresses, puts trusts into the authenticity of the existing email systems.

On the other side of mutual authentication, a trust anchor also requires out-of-band authentication. The first two cases in Section IV consider the trust anchor is authenticated through either the secure environment or trusted physical channels. Whereas in the third case, such assumption depends on the users obtain the Hydra app package from trusted sources (*e.g.* pre-shared Github repository).

In today's TCP/IP networks practice, trust anchors come from software releases (e.g., OSes and browsers) that contain built-in lists of root certificates from CAs, selected by the software vendors. The authentication of these trust anchors relies on end-users' implicit trust on OS and browser vendors as well as the correct operation of these software.

B. Local Trust Anchors and Trust Policies

As we discussed in Section II, NDN makes use of local trust anchors instead of building security upon the existing CAs. Any named entity can make itself a trust anchor for all the other entities under its control, and the same trust relationship can be build recursively. Further more, each trust anchor defines trust policies for the entities under its control, which specifies exactly what a given entity is allowed to do. In short, NDN supports the *least privilege principle* by limiting every trust anchor, even the ones at the higher level in the namespace, to play a confined rule, and every cryptographic key to be assigned as little control power as engineeringly feasible.

In contrast, the current CA infrastructures do not allow the certificate receiving entities to establish local trust anchors for their own system, because the root certificates are pre-installed into all entities, which can only authenticate the certificates that are directly issued by the CAs. As a result, for software distribution, new entities can only be security-bootstrapped with CA root certificate as trust anchors. Moreover, data receiving entities (*e.g.* browser) build their trust solely on CA issued certificates; there is no notion of "trust polices" for finer granularity security.

VI. Conclusion

Decades of IP deployment experiences have made people familiar with the process of connecting IP nodes into a network, which has been largely automated over time. In addition, there is no need to configure either names or trust relations in this IP host plug process, because IP uses its own IP address space to communicate, and names belong to application layer and hidden from network, and because today's Internet security solutions are built on the trust relations established through third parties (i.e. the Certificate Authorities) which are external to the communicating parties. Consequently, people who are new to NDN are not generally aware of the necessity of name and trust configurations in deploying new entities in an NDN network; the lack of systematic solutions and clear documentation further adds onto perceived difficulties in getting NDN-enabled applications deployed.

Our efforts and results reported in this work are an initial step towards addressing the above important open issues. In this work, we clarified that the process of NDN configuration is to "plug" an entity into the application layer namespace, the namespace that NDN uses for network layer communication. Therefore it requires name assignments and security bootstrapping. We developed solutions for three use case scenarios where new entity configurations are realized using context specific approaches. We further articulate that the three different cases share a meta level commonality by utilizing some existing trust relations that are exhibited in different forms, that either exist directly between the trust zone controller and the new entities to be bootstrapped (e.g. our first use case), or need additional bridging to close the loop (e.g. our third use case where Hydra NOC needs to learn campus's CA out-of-band).

We hope that the lessons learned from our efforts can serve as a starting point for future development of easy-to-use configuration solutions to facilitate NDN application deployments. As our future work, we plan to investigate solutions that enable configured entities invalidating compromised trust anchors, and further automate the developed solutions to minimize manual operations during NDN configurations.

References

[1] L. Zhang, A. Afanasyev, J. Burke, V. Jacobson, k. claffy, P. Crowley, C. Papadopoulos, L. Wang, and B. Zhang, "Named data networking," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, p. 66–73, Jul. 2014. [Online]. Available: https://doi.org/10.1145/2656877.2656887

[2] E. Newberry, T. Yu, Z. Zhang, J. Dellaverson, L. Zhang, "NDN Plug and Play," https://www.nist.gov/news-events/events/2020/09/ndn-community-meeting, September 2020, presentation at NDN Community Meeting 2020.

[3] H. Zhang, Y. Li, Z. Zhang, A. Afanasyev, and L. Zhang, "Ndn host model," *SIGCOMM Comput. Commun. Rev.*, vol. 48, no. 3, p. 35–41, Sep. 2018. [Online]. Available: https://doi.org/10.1145/3276799.3276804

[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," *IEEE Communications Magazine*, vol. 56, no. 11, pp. 62–68, November 2018.

[5] Oxford University Press, "plug-and-play, adj. and n.: Oxford English Dictionary," 2020, accessed: 2021-08-03. [Online]. Available: https://www.oed.com/view/Entry/247195?rskey=rOQGJa

[6] H. Zhang, Z. Wang, C. Scherb, C. Marxer, J. Burke, L. Zhang, and C. Tschudin, "Sharing mhealth data via named data networking," in *Proceedings of the 3rd ACM Conference on Information-Centric Networking*, 2016, pp. 142–147.

[7] Y. Yu, A. Afanasyev, D. Clark, k. claffy, V. Jacobson, and L. Zhang, "Schematizing trust in named data networking," in *Proceedings of the 2nd ACM Conference on Information-Centric Networking*, ser. ACM-ICN '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 177–186. [Online]. Available: https://doi.org/10.1145/2810156.2810170

[8] K. Nichols, "Trust schemas and icn: Key to secure home iot," ser. ICN '21. New York, NY, USA: Association for Computing Machinery, 2021.

[9] J. Shi, E. Newberry, and B. Zhang, "On broadcast-based self-learning in named data networking," in *2017 IFIP Networking Conference (IFIP Networking) and Workshops*, June 2017, pp. 1–9.

[10] A. Padmanabhan, L. Wang, and L. Zhang, "Automated tunneling over ip land: Run ndn anywhere," in *Proceedings of the 5th ACM Conference on Information-Centric Networking*, ser. ICN '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 188–189. [Online]. Available: https://doi.org/10.1145/3267955.3269023

[11] Named Data Networking Project, "ndn-autoconfig," https://named-data.net/doc/NFD/current/manpages/ndn-autoconfig.html.

[12] A. K. M. M. Hoque, S. O. Amin, A. Alyyan, B. Zhang, L. Zhang, and L. Wang, "Nlsr: Named-data link state routing protocol," in *Proceedings of the 3rd ACM SIGCOMM Workshop on Information-Centric Networking*, ser. ICN '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 15–20. [Online]. Available: https://doi.org/10.1145/2491224.2491231

[13] Y. Yu, A. Afanasyev, Z. Zhu, and L. Zhang, "Ndn technical memo: Naming conventions," *NDN, NDN Memo, Technical Report NDN-0023*, 2014.

[14] Z. Zhang, Y. Yu, A. Afanasyev, and L. Zhang, "Ndn certificate management protocol (ndncert)," *NDN, Technical Report NDN-0054*, 2017.

[15] Named Data Netweorking Project, "ndnsec," https://named-data.net/doc/ndn-cxx/current/manpages/ndnsec.html.

[16] K. Nichols, "Trust schemas and icn: key to secure home iot," in *Proceedings of the 8th ACM Conference on Information-Centric Networking*, 2021, pp. 95–106.

[17] Named Data Networking (NDN) project, "NDN Packet Format Specification version 0.3," 2021, accessed: 2021-07-19. [Online]. Available: https://named-data.net/doc/NDN-packet-spec/current/

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

[19] S. K. Ramani, P. Podder, and A. Afanasyev, "Ndnviber: Vibration-assisted automated bootstrapping of iot devices," in *2020 IEEE International Conference on Communications Workshops (ICC Workshops)*. IEEE, 2020, pp. 1–6.

[20] E. Osterweil and L. Zhang, "Interadministrative Challenges in Managing DNSKEYs," *IEEE Security and Privacy*, vol. 7, no. 5, pp. 44–51, 2009.