Cornerstone: Automating Remote NDN Entity Bootstrapping

Tianyuan Yu
UCLA
USA
tianyuan@cs.ucla.edu

Xinyu Ma
UCLA
USA
xinyu.ma@cs.ucla.edu

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

Dirk Kutscher
Hong Kong University of Science and Technology (GZ)
China
dku@ust.hk

Lixia Zhang
UCLA
USA
lixia@cs.ucla.edu

ABSTRACT

To secure all communications, Named Data Networking (NDN) requires that each entity joining an NDN network go through a bootstrapping process first, to obtain its initial security credentials. Several solutions have been developed to bootstrap IoT devices in localized environments, where the devices being bootstrapped are within the physical reach of their bootstrapper. However, distributed applications need to bootstrap remote users and devices into an NDN-based system over *insecure* Internet connectivity. In this work, we take Hydra, a federated distributed file storage system made of servers contributed by multiple participating organizations, as a use case to drive the design and development of a remote bootstrapping solution, dubbed Cornerstone. We describe the design of Cornerstone, evaluate its effectiveness, and discuss the lessons learned from this process.

CCS CONCEPTS

• **Networks → Security protocols**; • **Security and privacy → Authentication**.

KEYWORDS

Named Data Networking (NDN), Authentication, Security Bootstrapping

ACM Reference Format:
Tianyuan Yu, Xinyu Ma, Hongcheng Xie, Dirk Kutscher, and Lixia Zhang. 2023. Cornerstone: Automating Remote NDN Entity Bootstrapping. In *Asian Internet Engineering Conference (AINTEC '23), December 12–14, 2023, Hanoi, Vietnam.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3630590.3630598

1 INTRODUCTION

The existing TCP/IP architecture was originally designed to provide resilient infrastructure connectivity [5]. IP views a networked system as made of interconnected nodes identified by IP addresses. IP's bootstrapping process, generally carried out by the DHCP service, provides each newly joined node with a set of parameters to enable it to send and receive IP packets. Over time, a number of security solutions, such as IPSec, TLS, BGPSec, etc., have been introduced into today's Internet to fence off various attacks. Security solutions also need a bootstrapping step, i.e. getting a set of security parameters to enable nodes to sign and verify each piece of data. However, today's security bootstrapping, by and large, relies on manual configurations. For example, TLS relies on out-of-band installation of root certificate files into end hosts [12].

As a new network architecture design, Named Data Networking (NDN) [2, 9] views a networked system as made of named entities that have various trust relations among each other. These named entities can be devices, servers, application instances, or anything that produces and/or consumes named data packets [24]. These entities communicate by fetching secured data using their semantic names, and this use of semantic names makes it possible to build security directly into all communications. To communicate securely, however, requires that all entities go through a *security bootstrapping* process first when they join an NDN network, in order to obtain their initial security credentials and parameters, namely trust anchor, certificate, and security policies.

Several systematic and automatic NDN bootstrapping solutions have been developed in recent years, such as those reported in [10, 13, 14, 16]. Each of them takes a different technical approach to implement security bootstrapping, however a shared feature among all is that they provide bootstrapping for devices within the physical reach of the bootstrapper. Therefore, they can utilize the physical vicinity to securely perform the device authentication, such as using visual or acoustic signals, scanning barcodes or QR codes attached to the devices to be bootstrapped; one could also let the bootstrapper install all the required security parameters into devices through the local, secured communication channel.

Recently, a new class of NDN-based remote collaborative applications has emerged [6, 7, 11, 15]. These applications need to perform security bootstrapping for *remote* devices and users. We refer to this problem as *Remote Bootstrapping*. Due to the collaborative nature of these applications, NDN entities usually run on devices that are in the control of different organizations. Therefore, it is infeasible for an application administrator to directly access remote devices and manually install trust anchors, certificates, and trust schemas into remote application instances or devices. Existing NDN security

AINTEC '23, December 12–14, 2023, Hanoi, Vietnam
© 2023 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0939-5/23/12…$15.00
https://doi.org/10.1145/3630590.3630598

bootstrapping solutions that utilize physical vicinity [10] or require direct access [13, 14, 16] no longer work.

In this study, we take a first step towards addressing NDN remote bootstrapping problems by developing a bootstrapping solution for a specific application use case, Hydra. Hydra [11, 15] is a distributed, federated file storage system. It is composed of storage servers contributed by the Hydra user community from multiple organizations. Using Hydra bootstrapping as a case study, we make three contributions to remote bootstrapping by developing Cornerstone. First, we explore the approach of using *existing unique identifiers* in the Internet, such as DNS names and email addresses, to authenticate remote entities. Following the direction sketched out by Yu et. al. in remote server authentication [19], this work develops a protocol for remote server and user authentication, and makes a concrete implementation. Second, to simplify the definition of trust schemas for Hydra, we develop naming conventions to systematically convert the existing identifiers to Hydra specific NDN names. Third, we decouple authentication and naming from certification in the bootstrapping process, which simplifies the implementation and avoids potential conflicts of functionality.

The remainder of this paper is organized as follows. §2 provides an overview of NDN and its trust domain model. §3 introduces our target application scenario, Hydra, and our design goals. §4 describes the overview and design details of Cornerstone§5 reports the evaluation results of our Python implementation of Cornerstone on its performance and effectiveness. Finally we conclude our work in §8.

2 BACKGROUND AND RELATED WORK

In this section, we first give an overview of the NDN networking model, then describe previous works in this area, and a few core concepts our work is based on.

2.1 NDN Networking Model

An NDN-based system is made of connected, *semantically named* entities, with various trust relations among them. Because their names are decoupled from network addresses, these entities can utilize all available connectivity options to exchange named and secured data packets. NDN uses semantics in the names to write trust schema, which express security policies by defining the relations between the names of data and the names of crypto keys used to sign and encrypt data [20]. All NDN entities use defined trust schema to authenticate all received data.

NDN enables each administrative domain to perform its own authentication and set its own security policies through a hierarchical trust model. The work by Nichols [13] defines the concept of *trust domain* as a network governed by a single authority identified as the trust root for the domain. This trust root can be cryptographically identified by a self-signed certificate, dubbed *trust anchor*, and this trust anchor is installed into all the entities in the trust domain. The concept of trust domains helps precisely define the control scope of a trust anchor and the trust schema.

In order to produce authenticatable data, each entity E in a trust domain 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 namespace, enabling the enforcement of finer-grained security policies for authentication, authorization, and access control.

2.2 NDN Bootstrapping

Our Cornerstone model is built upon the results from a few previous works in NDN bootstrapping. Zhang *et al.* [21, 24] identified the necessary security components that must be installed into each new entity during its bootstrapping process. Following the trust domain concept introduced in [13], NDN bootstrapping is about bootstrapping an NDN entity into a specific trust domain. Yu *et al.* [19] further introduced the concept of a *trust domain controller* as a trust domain's governing entity, and articulated the following four necessary logical steps in the security bootstrapping process: (1) achieving mutual authentication between the bootstrapper and the entity E to be bootstrapped, (2) installing the domain trust anchor, (3) installing the trust schema defined for E, and (4) issuing a NDN certificate to E.

Our work in this paper adopts the concept of trust domain and trust domain controller, and develops a solution for remote bootstrapping scenarios.

2.3 Utilizing Existing Internet Identifiers for Authentication

In [19], Yu et. al. make a proposition that the establishment of a new trust relation relies on some pre-existing trust relation(s), which is based on some pre-existing identifiers. Therefore, NDN Bootstrapping requires two pieces of information to be provided to a trust domain D: the unique semantic identifier of a remote entity E, and the trust relation between D and E. In the context of adding a new entity E into a trust domain D, this trust relation denotes authorize E into D.

Assuming E has a unique, semantically meaningful identifier in the Internet, this identifier may also carry sufficient information to help the trust domain controller to assign E an NDN name that is meaningful in the specific NDN application context as we show later. The trust relation between D and E needs to come out-of-band, *e.g.* defined by the operator of trust domain D. As an example, Hydra design assumes that the network operation center (NOC) is configured with a list of pre-authorized user identifiers (the users' email addresses) and server identifiers (the servers' DNS names).

When new NDN entities are *remote* thereby only reachable through the network, the authentication process must be carried out through unsecured Internet connectivity. A practical approach is to utilize existing trust relations in today's systems to establish trust relations with remote entities. As a system that has been in operation over 40 years, today's Internet has deployed a number of authentication systems that associate semantic identifiers with defined trust relations [3, 4, 8, 17]. We argue that these authenticated, semantic identifiers from today's Internet can serve as a starting point to bootstrap remote NDN entities. As we describe in §4.1, these existing identifiers and trust relations are only used for authentication purposes during the bootstrapping procedure. Once bootstrapped into a trust domain, NDN entities no longer rely on those identifiers; instead they will be assigned NDN names and certificates that will be used in secure NDN communications.

3 REMOTE BOOTSTRAPPING

In this section, we provide a brief introduction to our target application Hydra and its security assumptions, then we explain why Hydra needs remote bootstrapping.

Hydra [15] is a distributed and federated data storage system designed for sharing genomics data among researchers from multiple campuses. It operates on the basis of a federated trust model, where different organizations contribute storage servers which collectively form a distributed data repository. To ensure secure communications within the system, Hydra establishes a Networking Operating Center (NOC) responsible for establishing an NDN trust domain for Hydra system and maintaining the trust relations within the domain.

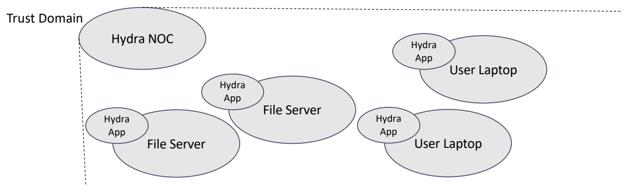


Figure 1: Hydra trust relationshop. Not shown here is the trust relation among storage servers, once they are installed with the same trust anchor.

Figure 1 shows a simplified trust relationship of the Hydra system, in which the Hydra NOC self-signs a certificate as the trust anchor, and inputs the trust schema configuration from human administrators to start a trust domain. The NOC uses two lists to control the domain membership: the *server list* and the *user list*. The former contains the DNS names of all the contributed storage servers, while the latter is made of the email addresses of invited users that are authorized to upload data files to Hydra. The files stored on Hydra contain public data, therefore there is no control on the users who fetch files from Hydra. Hydra administrators manually input the NOC with these two lists, which serve the role of authorizing storage servers and file-uploaders to join Hydra. The lists need to be kept up-to-date whenever a server or user joins or leaves the system. Developing an automated update tool is one of our future work.

Security Assumptions: Hydra operates under the assumption that each organization has *complete* control over the servers they contribute, and that the authorized Hydra servers are considered trustworthy. However, the communication between Hydra servers, as well as between servers and the NOC, occurs over unsecured networks. Similarly, the authorized Hydra users are assumed to be trustworthy, and they communicate with the NOC and servers over unsecured networks. Additionally, it is assumed that each contributed Hydra server possesses a DNS name (as all institutional hosts generally do), and each Hydra user has a valid email address.

Restrictions Leading to Remote Bootstrapping: Since the remote owners have complete control over their devices, Hydra NOC cannot remotely access and configure individually owned devices. Meanwhile, the owners of the contributed storage servers and user devices may not be familiar with NDN, thereby Hydra also cannot rely on human at the remote end configuring NDN security components. A systematic solution is needed for Hydra to remotely bootstrap user and server entities.

4 THE DESIGN OF CORNERSTONE

In this section, we describe the design of Cornerstone. First, we discuss the design goals of our remote bootstrapping solution, followed by an overview of Cornerstone (§4.1). We then discuss the detailed design of Cornerstone by the four tasks done in the bootstrapping: installing the trust anchor and the trust schema (§4.2), authenticating the new entity (§4.3), assigning an NDN name to the new entity (§4.4), and issuing certificates (§4.5).

Design Goals: To enable secure data communications in Hydra, the remote security bootstrapping solution must fulfill the following two design goals. First, enabling remote Hydra servers and users to *securely* obtain all necessary trust parameters, including the trust anchor, certificate, and trust schema, from the Hydra NOC. Second, automating the remote bootstrapping as much as possible to minimize manual intervention.

4.1 Overview

As Figure 2 shows, the security bootstrapping starts with *mutual authentication* between remote NDN entity and the NOC. The Cornerstone makes use of the trust relations that already exist in today's Internet operational environment to let both sides authenticate each other. A new Hydra user or server authenticates the NOC through Hydra software distribution chain (§4.2), from which a new entity *E* is also able to install the Hydra trust anchor and trust schema.

Afterwards, the new entity runs *Name Authentication and Assignment Protocol (NAAP)* (§4.3) with the NOC to request authentication and name assignment by using of its *existing identifier* (§4.3), that could be an X.509 certificate owned by a Hydra server, or an email identifier assigned by some recognized external identity provider to a new Hydra user. For example, a server with domain name "`node1.medicalx.com`" can be authenticated by its X.509 certificate. Alice, a user with account "`alice`" at "`medicalx.com`", can be authenticated with its *OpenID Connect*'s ID token signed by "`medicalx.com`". During the authentication process, each new entity uses a temporary key pair to uniquely identify itself from other new entities.

After the authentication step, the NOC assigns an NDN name to the new entity based on its external identity used for authentication (§ 4.4). The name assignment rules are configurable by the NOC deployment. In the example above, if Hydra runs at the name prefix "`/hydra`", the server may obtain the name "`/hydra/servers/com/medicalx/node1`" and Alice may obtain "`/hydra/users/com/medicalx/alice`". The assigned name is given to Alice in the format of *Proof-of-Possession (PoP)*, a NOC-signed NDN Data packet that binds Alice's temporary public key to her assigned name.

Lastly, Alice utilizes the NDNCERT protocol [23] to interact with the Hydra Certificate Authority (CA), an entity delegated by the Hydra NOC for certification. Alice presents the PoP signed by the NOC to the CA as her name assignment. Upon verifying Alice's PoP, the CA issues Alice the final NDN certificate that she can use in Hydra (§4.5).

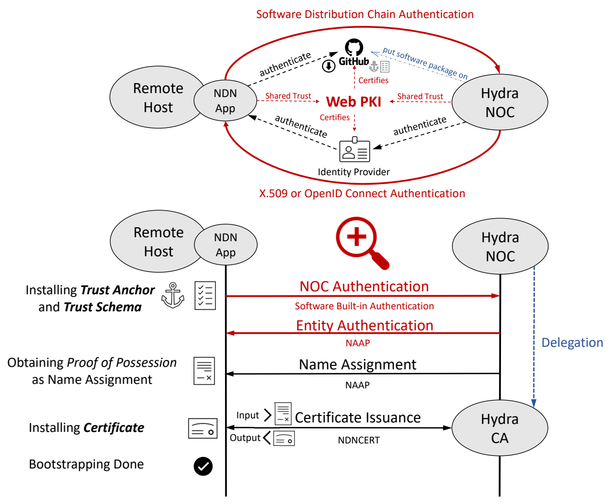


Figure 2: Overall workflow of Cornerstone. The magnifier focuses on the mutual authentication process and the upper part of this figure reveals more details of this two-way authentication.

4.2 Installing Trust Anchor and Trust Schema

Hydra embeds the trust anchor and trust schema into its software distribution, therefore all users and servers will install the same trust schema. By doing so, we assume that every Hydra user and server can receive these security components securely through the existing software distribution channel, such as GitHub, which can provide an authenticated communication channel that Hydra administrators use to distribute Hydra software package.

As a result, when Hydra users and servers receive the software package, they can be confident that the embedded trust anchor and trust schema are authentic and have not been tampered.

4.3 Authenticating New Hydra Entity

Cornerstone leverages *existing* identifier authentication systems to authenticate server and users.

If a storage server has a DNS name, the existing authentication systems include DNSSEC [8] and X.509 certificates issued by certificate authorities [4]. For user authentication, Cornerstone utilizes identity verification protocols such as OpenID Connect (OIDC) [17]. OIDC allows an application to verify the identity of users through identity providers such as Google, Facebook, or GitHub. Hydra NOC can request OIDC authentication from user's identity provider, and this process involves validating a provider-signed token, which indicates the user's account name and authentication status.

As Figure 3 depicts, Cornerstone's overall new entity authentication workflow is a two-phase protocol NAAP (Name Authentication and Assignment Protocol), executed between the new entity, which is either a new user or a new server to be bootstrapped, and the Hydra NOC. The first phase is Authentication Request and Response (*A*), where the new entity requests authentication from the NOC, providing necessary parameters, including a public key to uniquely identify the new entity during the bootstrapping process. The second phase is Identity Proof Request and Response (*P*), where the new entity proves its identity to the NOC and receives a PoP which contains its name assignment (discuss shortly in Section 4.4).

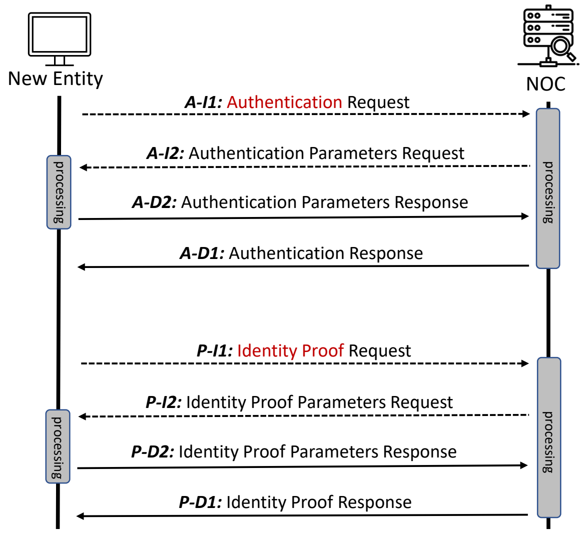

New Entity
NOC
A-I1: Authentication Request
A-I2: Authentication Parameters Request
A-D2: Authentication Parameters Response
A-D1: Authentication Response
P-I1: Identity Proof Request
P-I2: Identity Proof Parameters Request
P-D2: Identity Proof Parameters Response
P-D1: Identity Proof Response
processing
processing
processing
processing

Figure 3: NAAP Overview

Each phase consists of the following Interest-Data packet exchanges:

- *I1*: an Interest sent by the new entity to notify the NOC.[1]
- *I2*: an Interest sent by the NOC to the new entity to fetch necessary parameters.
- *D2*: a Data packet sent by the new entity in response to *I2*, carrying parameters requested by the NOC.
- *D1*: a Data packet sent by the NOC in response to *I1*, carrying the result of this phase.

The rest of this section will describe the server authentication workflow and user authentication workflow, respectively. The detailed packet exachange are presented in Appendix A.

Server Workflow: In the server authentication flow, the server is assumed to have a valid X.509 certificate, and intends to use the same key pair to bootstrap in Cornerstone. The NOC verifies the X.509 certificate chain, and authenticates the server entity by requiring it to sign a random number using the certified key pair.

Specifically, in *A* phase, the authentication parameter *A-D2* is the server's X.509 certificate chain. Then in *A-D1*, the NOC generates a random number and requires new server's signature to be sent back in *P-D2*. Receiving the signature, the NOC examines that the X.509 certificate chain is properly signed by a known CA, and the random number is properly signed by the certified key.

User Workflow: In the user authentication workflow, new user presents an identity from an OIDC provider known to the NOC (discuss shortly), proves the ownership, and gets authenticated. In *A* phase, the authentication parameters *A-D2* are the OIDC provider's name, the user's username at the provider, and a newly generated public key. Then in *A-D1*, the NOC gives a URI which the user can

[1] *I1* contains the new entity's temporary NDN name prefix in its content. The NOC uses this prefix to construct *I2* and fetch parameters. For example, a user Alice connected to the MedicalX network may use the name prefix "`/Medical/Alice/HydraBoot`" in this authentication.

use to perform an OIDC authentication. Generally, this URI embeds the application's credentials registered at the OIDC provider. In *P* phase, the *P-D2* is the OIDC ID token provided by the OIDC provider. Receiving *P-D2*, the NOC examines that the OIDC ID token is properly signed by the key of user's OIDC provider.

Automation: The automation of server authentication relies on two factors. First, the Hydra NOC has established trust relationships with CAs through out-of-band installation of CA root certificates. This requirement is typically fulfilled by the operating system's software distribution chain, as modern operating systems come pre-installed with CA root certificates. Second, the servers need to be able to request X.509 certificates and obtain them automatically. To address this, Cornerstone leverages Let's Encrypt [1], which provides a free and automated X.509 certificate issuance service for servers.

The automation of user authentication relies on Hydra NOC having pre-established trust relationships with the relevant identity provider. If the provider is certified by a NOC trusted CA, the NOC-Provider trust relationship has established during the operating system installation.

4.4 Naming New Hydra Entity

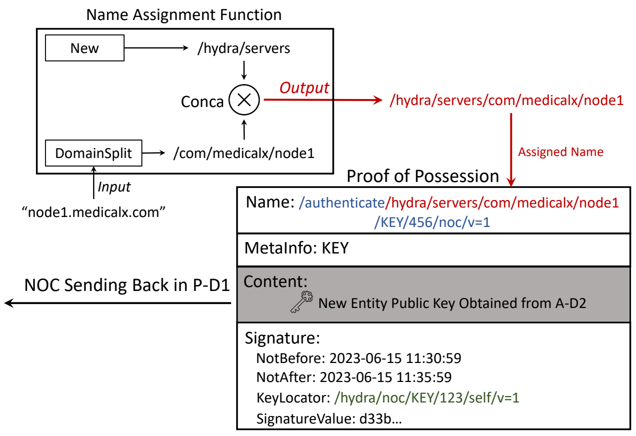
Name assignment happens between the NOC receiving *P-D2* and sending *P-D1*. Note that the NOC only assigns name to new entities that are authorized by the user list and server list.

To simplify trust schema rule writing and ensure semantically meaningful NDN names in the system, it is desirable to assign names that leverage existing identifiers used for authentication.

Based on our communications with Hydra administrators, we have learned that they prefer managing trust relationships with organizations groups. Therefore, the semantics of DNS names and email addresses are suitable for reuse in the name assignment process. In the assignment of names for Hydra entities, we split the domain names into multiple name components and concatenate them with a name prefix that indicates whether the entity is a server or a user (which will be discussed shortly). This approach allows us to preserve the existing identifiers' semantics while providing unique and semantically meaningful NDN names within the system.

As shown in figure 4, the entity naming convention is configurable by administrators programming name assignment functions using Cornerstone's predefined rules. For example, the rule *New* creates a base name "`/hydra/servers`", while the *DomainSplit* rule converts a DNS name *node1.medicalx.com* into an NDN name "`/com/medicalx/node1`". The *Concat* rule then appends the converted name after "`/hydra/users`", resulting in the NDN name "`/hydra/ servers/com/medicalx/node1`". Similarly, by combining the rules *New*, *EmailSplit*, and *Concat*, Alice's email address can be converted into the NDN name "`/hydra/ users/com/medicalx/ alice`".

The programmable naming rules in Cornerstone empower Hydra administrators to define entity naming conventions without dealing with the low-level implementation details. For advanced administrators who wish to define custom name conversion rules, Cornerstone also provides APIs that allow for the design of customized naming rules. This flexibility enables administrators to tailor the naming process according to their specific requirements.



Name Assignment Function
New
/hydra/servers
Conca
Output
/hydra/servers/com/medicalx/node1
DomainSplit
/com/medicalx/node1
Input
"node1.medicalx.com"
Assigned Name
Proof of Possession
Name: /authenticate/hydra/servers/com/medicalx/node1 /KEY/456/noc/v=1
MetaInfo: KEY
NOC Sending Back in P-D1
Content:
New Entity Public Key Obtained from A-D2
Signature:
NotBefore: 2023-06-15 11:30:59
NotAfter: 2023-06-15 11:35:59
KeyLocator: /hydra/noc/KEY/123/self/v=1
SignatureValue: d33b…

Figure 4: Example of name assignment based on domain names

Proof-of-Possession: After the NOC assigns an NDN name to a new entity, it generates a Proof of Possession (PoP) by signing a Data packet with its private key, and sends PoP back in Data *P-D1*. The PoP can be considered as a temporary certificate used to bind the entity's public key with the NDN name which the NOC assigned to the new entity. The content of the Data packet includes the new entity's public key obtained during the authentication process (Data *A-D2*).

The PoP follows a specific naming convention that starts with the keyword prefix "`/authenticate`", followed by the assigned entity name, the identifier of the corresponding public key, issuer information, and a version number. In the example of Figure 4 in which the NOC assigns the name "`/hydra/servers/com/medicalx/node1`" to a new server with the public key identifier "`456`", the NOC signs a PoP with the name "`/authenticate/hydra/servers/com/ medicalx/node1/KEY/456/noc/v=1`". The "`noc`" component indicates that the PoP is issued by the NOC, and "`v=1`" represents the version number.

Note that the public key contained in the PoP is supposed to be an ephemeral key that serves to uniquely identify the new entity during the security bootstrapping process. It may not be the same public key that will be included in the final certificate issued to the entity.

PoP also includes a validity period that restricts the lifetime of the key-name binding. Within the validity period of the PoP, the new entity must complete the security bootstrapping process by requesting a certificate from a Hydra CA (as described in Section 4.5) using this PoP. If the entity fails to complete the security bootstrapping within the validity period, a new PoP must be obtained through re-executing the entire protocol.

In summary, the PoP serves as a proof of ownership for the assigned name and facilitates the security bootstrapping process by establishing a binding between the entity's public key and its assigned NDN name.

4.5 Certification

One key aspect that sets Cornerstone apart from previous bootstrapping solutions is the decoupling of authentication and naming from certification. This is achieved by allowing the Hydra NOC to delegate the certification process to NDNCERT [22], a CA entity within Hydra. Hydra NOC configures the Hydra CA in the trust domain setup time, by delegating a name prefix to host NDNCERT CA, signing the CA certificates, and running the CA software.

New entity learns the Hydra CA prefix by naming convention. Once the new Hydra entity obtains its PoP, it runs the NDNCERT protocol and initiates the *Proof-of-Possession Challenge*. During this challenge, the CA validates the entity's PoP and requests the entity to provide proof of ownership of the corresponding private key. Upon successful verification, the CA issues the final NDN certificate for the entity under the name appointed by the PoP.

This approach offers several advantages. Firstly, it simplifies the implementations of both the NOC and the CA, as their roles and responsibilities are clearly defined. Secondly, it helps avoid potential conflicts in functionality between the NOC and the CA. By separating the authentication and naming process from certification, each component can focus on its specific tasks, leading to a streamlined Cornerstone design.

5 IMPLEMENTATION

We have implemented Cornerstone for Hydra security bootstrapping by using python-ndn libraries [18] and other tools. The tools include python scripts that download Hydra software from a trusted URL, then use the Cornerstone library to run the NAAP protocol with the Hydra NOC and NDNCERT protocol with the Hydra CA.

In order to evaluate the efficiency of Cornerstone itself, we eliminate the effects of network delay by running Hydra new entity, Hydry NOC, and Hydra CA, all on the same machine equipped with Apple M1 Pro and 16GB RAM. Our preliminary performance evaluation focuses on the time between the Hydra software download completion and the installation of the NDN certificate, which we refer to as the *bootstrapping time cost*. To simulate the users' daily work environment, we assume that users have already logged in with their OIDC providers in browsers. During our experiments, the authors acted as users to be bootstrapped into Hydra, providing only their email addresses and OIDC providers to the user bootstrapping tool. The servers participated in security bootstrapping with their actual X.509 certificates. As shown in Table 1, Cornerstone took $0.27s$ to bootstrap a Hydra server, while the user bootstrapping process took $2.34s$ to complete. This user bootstrapping time cost was primarily due to necessary user interactions in the browser. Specifically, the bootstrapping tools opened a browser prompt and requested the user's consent to authorize Hydra to authenticate their email.[2] Therefore, Cornerstone still achieved the design goals mentioned in § 3.

Table 1: Bootstrapping Time Cost

Bootstrapping Time Cost	User Entity	Server Entity
Total	**2.34s**	**0.27s**
User Interaction	85.89%	N/A
Miscellaneous	14.11%	100%

6 RELATED WORK

The focus of previous works [10, 13, 14, 16] has primarily been on bootstrapping devices that do not have existing identifiers. As a result, these works have to rely on other authentication approaches such as direct access or physical confirmation (*e.g.* visual or acoustic) to establish trust.

In contrast, our work addresses a different problem, that is remotely bootstrap entities that have already been authenticated and named in today's Internet. This key distinction allows us to leverage existing authentication systems and reuse the authenticated identifiers in NDN names.

Although [19] described the idea of using TLS certificates to authenticate Hydra servers, this work moves forward to propose a specific protocol design to authenticate Hydra servers and users thus focus more on technical details.

7 DISCUSSION

A popular misconception is that NDN applications communicate securely unconditionally. NDN applications do require security bootstrapping, as the crucial step, to configure the initial security components to start secured communications.

One might view the Cornerstone's approach to security bootstrapping, which depends on existing identities, as ad hoc. However, as [19] pointed out, authentication always relies on pre-existing trust relations. Before NDN gets widely deployed, one needs to identify isolated remote entities through existing identity system. In today's Internet, application platform identity providers (*e.g.* Google) offer the most available unique identifiers for end users, and Web Public Key Infrastructure (PKI) offers the security solution for almost all applications that people use daily. Therefore, leveraging those trust relations to authenticate new NDN entities is a practical solution in NDN Bootstrapping. Once NDN becomes widely deployed, each remote entity would possess NDN assigned name and certificate by a remote trust domain D; therefore when a trust domain D_1 wants to bootstrap a remote entity E into its own domain, D_1 can utilize its trust relation with E (the pre-existing trust relation) to authenticate E.

8 CONCLUSION

Security bootstrapping plays a critical role in building secure NDN-based systems. Although different application scenarios may require bootstrapping approaches tailored to the specific needs, the shared goal of all bootstrapping solutions is the installation of trust anchors, certificates, and trust schema into new entities joining NDN-based networks and applications.

Cornerstone is the first complete design and implementation for bootstrapping remote users and devices. Our preliminary evaluation shows that Cornerstone successfully achieves its design goals.

[2]We acknowledge that the user's response time to the browser prompt may vary from one person to another, and our experiments did not involve multiple user interviews. However, the user's reaction time to provide consent is an essential time cost that cannot be optimized.

Although it is designed for a specific federated system Hydra, Cornerstone offers an initial exploration to the solution space of remote bootstrapping. Our design experience suggests that performing remote security bootstrapping necessarily makes use of the existing identifier authentication systems, and that a separate authorization step (checking the authorization list provided by human administrators) follows the authentication. We hope that these insights could help the development of future bootstrapping solutions.

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] Alexander Afanasyev, Tamer Refaei, Lan Wang, and Lixia Zhang. 2018. A Brief Introduction to Named Data Networking. In *Proc. of MILCOM*.

[3] Richard Barnes, Jacob Hoffman-Andrews, Daniel McCarney, and James Kasten. 2019. Automatic Certificate Management Environment (ACME). RFC 8555. https://doi.org/10.17487/RFC8555

[4] Sharon Boeyen, Stefan Santesson, Tim Polk, Russ Housley, Stephen Farrell, and David Cooper. 2008. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 5280. https://doi.org/10.17487/RFC5280

[5] David Clark. 1988. The Design Philosophy of the DARPA Internet Protocols. In *Symposium Proceedings on Communications Architectures and Protocols (SIGCOMM '88)*. Association for Computing Machinery.

[6] Saurab Dulal, Nasir Ali, Adam Robert Thieme, Tianyuan Yu, Siqi Liu, Suravi Regmi, Lixia Zhang, and Lan Wang. 2022. Building a secure mhealth data sharing infrastructure over ndn. In *Proceedings of the 9th ACM Conference on Information-Centric Networking*. 114–124.

[7] 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.

[8] Paul E. Hoffman. 2023. DNS Security Extensions (DNSSEC). RFC 9364. https://doi.org/10.17487/RFC9364

[9] Van Jacobson, Diana K Smetters, JD Thronton, Michael F Plass, Nicholas H Briggs, and RL Braynard. 2009. Network Named Content. *CoNEXT* (2009).

[10] Yanbiao Li, Zhiyi Zhang, Xin Wang, Edward Lu, Dafang Zhang, and Lixia Zhang. 2019. A secure sign-on protocol for smart homes over named data networking. *IEEE Communications Magazine* 57, 7 (2019), 62–68.

[11] Siqi Liu, Varun Patil, Tianyuan Yu, Alexander Afanasyev, Frank Alex Feltus, Susmit Shannigrahi, and Lixia Zhang. 2021. Designing hydra with centralized versus decentralized control: A comparative study. In *Proceedings of the Interdisciplinary Workshop on (de) Centralization in the Internet*. 4–10.

[12] Zane Ma, James Austgen, Joshua Mason, Zakir Durumeric, and Michael Bailey. 2021. Tracing Your Roots: Exploring the TLS Trust Anchor Ecosystem. In *21st ACM Internet Measurement Conference (IMC)*. Virtual Event.

[13] 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.

[14] Davide Pesavento, Junxiao Shi, Kerry McKay, and Lotfi Benmohamed. 2022. PION: Password-based IoT onboarding over named data networking. In *ICC 2022-IEEE International Conference on Communications*. IEEE, 1070–1075.

[15] Justin Presley, Xi Wang, Tym Brandel, Xusheng Ai, Proyash Podder, Tianyuan Yu, Varun Patil, Lixia Zhang, Alex Afanasyev, F Alex Feltus, et al. 2022. Hydra–A Federated Data Repository over NDN. *arXiv preprint arXiv:2211.00919* (2022).

[16] 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.

[17] N. Sakimura, J. Bradley, M.B. Jones, B. de Medeiros, and C. Mortimore. 2014. OpenID Connect Core 1.0 incorporating errata set 1.

[18] NDN Project Team. 2023. *python-ndn documentation*.

[19] 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.

[20] 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*. 177–186.

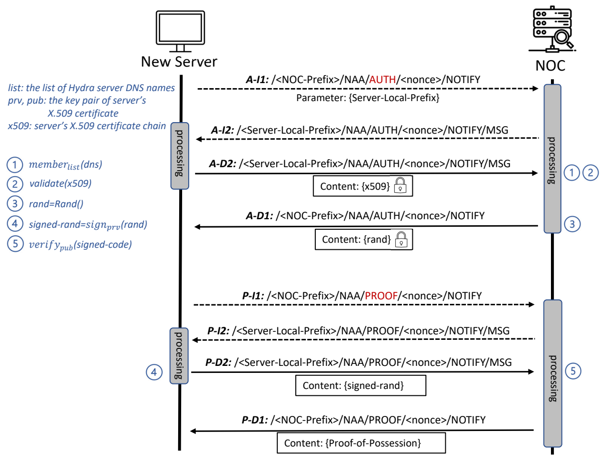
[21] Haitao Zhang, Yanbiao Li, Zhiyi Zhang, Alexander Afanasyev, and Lixia Zhang. 2018. Ndn host model. *ACM SIGCOMM Computer Communication Review* 48, 3 (2018), 35–41.

[22] Zhiyi Zhang, Alexander Afanasyev, and Lixia Zhang. 2017. NDNCERT: Universal Usable Trust Management for NDN. In *Proceedings of the 4th ACM Conference on Information-Centric Networking* (Berlin, Germany) *(ICN '17)*. Association for Computing Machinery, New York, NY, USA, 178–179. https://doi.org/10.1145/3125719.3132090

[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).

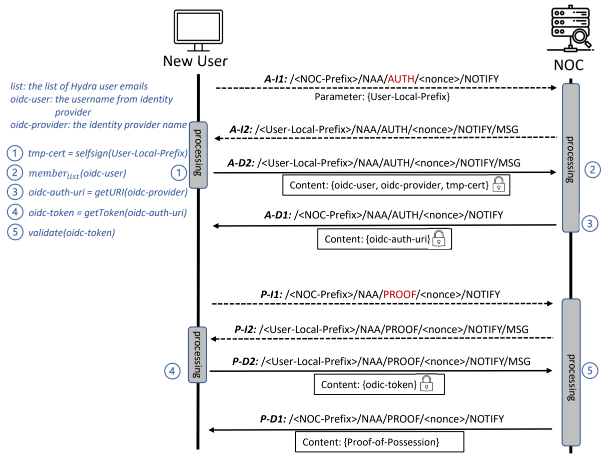
[24] Zhiyi Zhang, Yingdi Yu, Haitao Zhang, Eric Newberry, Spyridon Mastorakis, Yanbiao Li, Alexander Afanasyev, and Lixia Zhang. 2018. An overview of security support in Named Data Networking. *IEEE Communications Magazine* 56, 11 (2018), 62–68.

A NAAP PROTOCOL DETAILS



New Server
NOC
list: the list of Hydra server DNS names
prv, pub: the key pair of server's X.509 certificate
x509: server's X.509 certificate chain
1 $member_{list}$(dns)
2 validate(x509)
3 rand=Rand()
4 signed-rand=$sign_{prv}$(rand)
5 $verify_{pub}$(signed-code)
A-I1: /<NOC-Prefix>/NAA/AUTH/<nonce>/NOTIFY
Parameter: {Server-Local-Prefix}
A-I2: /<Server-Local-Prefix>/NAA/AUTH/<nonce>/NOTIFY/MSG
A-D2: /<Server-Local-Prefix>/NAA/AUTH/<nonce>/NOTIFY/MSG
Content: {x509}
A-D1: /<NOC-Prefix>/NAA/AUTH/<nonce>/NOTIFY
Content: {rand}
P-I1: /<NOC-Prefix>/NAA/PROOF/<nonce>/NOTIFY
P-I2: /<Server-Local-Prefix>/NAA/PROOF/<nonce>/NOTIFY/MSG
P-D2: /<Server-Local-Prefix>/NAA/PROOF/<nonce>/NOTIFY/MSG
Content: {signed-rand}
P-D1: /<NOC-Prefix>/NAA/PROOF/<nonce>/NOTIFY
Content: {Proof-of-Possession}
processing

Figure 5: Server authentication workflow



New User
NOC
list: the list of Hydra user emails
oidc-user: the username from identity provider
oidc-provider: the identity provider name
1 tmp-cert = selfsign(User-Local-Prefix)
2 $member_{list}$(oidc-user)
3 oidc-auth-uri = getURI(oidc-provider)
4 oidc-token = getToken(oidc-auth-uri)
5 validate(oidc-token)
A-I1: /<NOC-Prefix>/NAA/AUTH/<nonce>/NOTIFY
Parameter: {User-Local-Prefix}
A-I2: /<User-Local-Prefix>/NAA/AUTH/<nonce>/NOTIFY/MSG
A-D2: /<User-Local-Prefix>/NAA/AUTH/<nonce>/NOTIFY/MSG
Content: {oidc-user, oidc-provider, tmp-cert}
A-D1: /<NOC-Prefix>/NAA/AUTH/<nonce>/NOTIFY
Content: {oidc-auth-uri}
P-I1: /<NOC-Prefix>/NAA/PROOF/<nonce>/NOTIFY
P-I2: /<User-Local-Prefix>/NAA/PROOF/<nonce>/NOTIFY/MSG
P-D2: /<User-Local-Prefix>/NAA/PROOF/<nonce>/NOTIFY/MSG
Content: {odic-token}
P-D1: /<NOC-Prefix>/NAA/PROOF/<nonce>/NOTIFY
Content: {Proof-of-Possession}
processing

Figure 6: User authentication workflow